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

Mounir Chadli, Jin Hyung Kim, Axel Legay, Louis-Marie Traonouez, Stefan Naujokat, Bernhard Steffen

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

Mounir Chadli Jin Hyun Kim Axel Legay Louis-Marie Traonouez
Inria Rennes
firstname.lastname@inria.fr

Stefan Naujokat Bernhard Steffen
Technische Universität Dortmund
{stefan.naujokat,steffen}@cs.tu-dortmund.de

## Abstract

For decades, schedulability analysis of Cyber-Physical Systems (CPS) has been conducted by analytical methods rather than model-based methods. However, CPS are getting more and more complicated, beyond the capability of analytical methods, as more sophisticated scheduling mechanisms are used. This encourages the use of model-based and automated verification techniques. These techniques must be flexible enough to be adapted to any system, and easy to use by system designers, without deep knowledge of formal verification. In this paper, we present a flexible model-based framework for specifying hierarchical scheduling systems and performing automated formal verification. It allows to easily specify complex scheduling mechanisms, with hierarchical scheduling units that can be analyzed efficiently in a compositional manner. Formal verification using statistical techniques is performed automatically by generating on-the-fly the formal models. Finally, the framework returns comprehensible feedback from the results of formal verification in the design tool.

## 1. Introduction

Cyber-Physical Systems (CPS) are software-implemented control systems that control physical objects in the real world. These systems are being increasingly used in many critical systems, such as avionics and automotive systems, and they are now integrated into high performance platforms, with shared resources. This motivates the development of efficient design and verification methodologies to guarantee the safety and reliability of the systems.



One of the trends in developing a CPS is to integrate many heterogeneous computational components into a high-performance platform in order to maximize the utilization of hardware resources. The heterogeneous components in the same platform are managed to be completely partitioned such that errors caused by one component are alienated from the other components. For instance, heterogeneous operating platforms in avionics and automotive systems, that manage various and different integrity-level applications, are integrated using a high-performance hardware platform supported by multi-core processors, advanced memory, and multi-level cache architectures.

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, which is essential to evaluate the safety and reliability of mission-critical systems. For this reason, designers are still reluctant to use lower-price hardware components with higher capabilities, such as multi-core processors, for these 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*) [2] for instance is a well known formal model that can be used to perform schedulability analysis of real-time systems [12, 13]. 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 [9].

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 statisti-

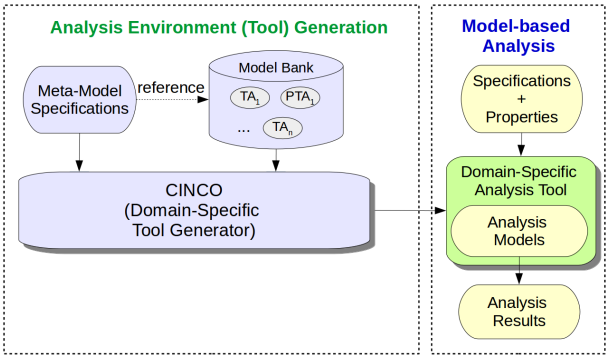


Figure 1: Our Model-based Formal Analysis Engineering

cal methods such as statistical model-checking (*SMC*) [14]. This approach is much more efficient for analyzing complex systems, that are often intractable with exhaustive methods.

Currently, many models and tools are successfully used to analyze properties of CPS, but they are domain-specific, and thus 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.

This paper demonstrates a flexible and formal analysis engineering approach for analyzing scheduling properties of CPS. Fig. 1 depicts the principles of our model-based engineering framework.

We first introduce a new model-based compositional framework with stochastic real-time tasks in a hierarchical scheduling system. This framework is designed with timed automata and probabilistic timed automata that constitute a model bank to describe hierarchical scheduling systems. In particular we introduce new probabilistic timed automata (PTA) models to instantiate stochastic tasks where task real-time attributes, such as deadline, execution time or period, are characterized by probabilities. This allows to design generic models that cover more cases of CPS.

We then encapsulate this formal framework into CINCO [26, 27], 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 design the specifications of a hierarchical scheduling system and the properties it 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 [15] to perform the analysis. This approach allows to completely hide the formal models being used from the system designer that can concentrate on the structure and the parameters of the hierarchical 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. It is even more difficult to understand the results when the models used by these tools have been automatically generated. Nevertheless the most recent version of CINCO introduced an API for model transformations that allows to program actions that can update the model. We have used these new functionalities to parse the results of the analyses output by UPPAAL and to show graphically the most important information.

To sum up, the contributions of this paper are to provide

1. A flexible formal engineering approach for specifying and analyzing scheduling systems with a generated graphical interface.
2. A new model of real-time stochastic tasks for the analysis of real-time properties of scheduling systems supported by sophisticated operating environment.
3. A model-based compositional statistical analysis for the new model with real-time stochastic tasks.

The rest of this paper is organized as follows: Section 2 discusses related works on scheduling techniques for real-time systems. Section 3 presents the background theories about formal models and hierarchical scheduling that we are using in this paper. Section 4 presents our framework for model-based analysis of hierarchical scheduling system, with our contributions in extending the models with stochastic tasks. Section 5 describes our implementation of the framework in the tool CINCO, which provides a user-friendly approach capable of formal analysis. We illustrate this framework with a case-study in Section 6. Section 7 concludes the paper.

## 2. Related Work

### 2.1 Analytical Methods for Analysis of Sporadic Tasks

Sporadic tasks were first introduced in [3, 25] as an extension of the Liu and Layland [19] task model. The authors in [3] proposed an exact schedulability analysis by providing some necessary and sufficient conditions for a sporadic task system. In [31], the authors propose a framework for the schedulability analysis of real-time systems, where they define a generalized model for sporadic tasks to more precisely characterize the task arrival times. Each task is characterized by two constraints: *higher instantaneous arrival rate*, which bounds the maximum number of task arrivals during some small time interval; *lower average arrival rate*, which is used to specify the maximum number of arrivals over some longer time interval. The work of [24] considers systems with probabilistic execution times and probabilistic inter-arrival times. However it does not handle dynamic scheduling policies. Moreover, the method is a numerical analysis technique whose complexity is exponential in proportion to the number of samples and tasks. In [30], the authors propose a method to control the preemptive behavior

of real-time sporadic task systems by the use of CPU frequency scaling. They introduced a new sporadic task model in which the task arrival may deviate, according to a *discrete* time probability distribution, from the minimum inter-arrival time. Based on the probability of arrivals, the authors propose an on-line algorithm computing CPU frequencies that guarantee non-preemptiveness of task behavior while preserving system schedulability.

### 2.2 Model-based Analysis of Stochastic Sporadic Tasks

In the context of model-based analysis, the authors in [11] present a symmetric multi-core framework where a flat scheduling system can be described in the Prelude language. The schedulability can be checked using generated UPPAAL models.

The authors in [21] formally characterize stochastic tasks for various platforms and presents a model-based analysis technique to check the schedulability of the tasks. The main ides is to compute the probability distribution of a task termination time by a convolution of the probability density functions of the task starting time and execution time. However, it is restricted to non-preemptive stochastic tasks, and the analysis complexity is also exponential.

Using the statistical model checking technique in UPPAAL, the work in [8] proposes a way of estimating the "degree of schedulability" of sporadic tasks and also presents the UPPAAL models used to implement the concepts as well as an avionics case-study.

The analysis technique of our work is based on [7, 9, 10]. Extending the models of Timed Automata (TA) and Stopwatch Automata (SWA) in [7, 9], we present a model of hierarchical scheduling systems, based on the stochastic sporadic tasks of [10] but with dynamic stochastic updates of real-time attributes.

## 3. Background

### 3.1 Timed Automata

Timed Automata (*TA*) [2] are a classical formal model to design real-time systems. A *TA* consists in a finite automaton extended with real-time clocks. It is composed of locations and transitions between locations associated with guards and resetting of clocks.

For a set of real-time clocks $C$, a function $v : C \to \mathbb{R}_{\geqslant 0}$ is called a *clock valuation*. For a delay $d \in \mathbb{R}_{\geqslant 0}$, let $v+d$ denote the clock assignment that maps all $x \in C$ to $v(x)+d$. A *clock constraint* on $C$ is a finite conjunction of expressions of the form $x \sim k$ where $x \in C$, $\sim = \{<, \leqslant, >, \geqslant\}$, and $k \in \mathbb{N}$. Let B(C) denotes the set of all clock constraints on $C$. A clock valuation $v$ satisfies the clock constraint $g \in \mathcal{B}(C)$, written $v \models g$, *iff* $g$ holds after all the clocks in $g$ have been replaced by their value $v(c)$.

**DEFINITION** 1. *A **Timed Automata (TA)** is a tuple $(L, l_0, \Sigma, C, \to, I)$, where*

- *$L$ is a finite set of locations,*
- *$l_0 \in L$ is the initial location,*
- *$\Sigma$ is an alphabet of actions,*
- *$C$ is a finite set of clocks with valuation in $\mathbb{R}_{\geqslant 0}$.*
- *$\to \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is the set of transitions.*
- *$I : L \to \mathcal{B}(C)$ associates an invariant constraint on clocks in $C$ to each location.*

A state $s = (l, v)$ of a *TA* consists in a location $l$ and a clock valuation $v$. The semantics of a *TA* is defined by an infinite transition system with two types of transitions:

- Continuous transition updates the clock valuation in a location by the same value $d \in \mathbb{R}_{\geqslant 0}$ for all clocks, provided that the invariant of the location remains satisfied $(v + d \models I(l))$.
- Discrete transition switches from one location to another, if there exists a transition $(l, a, g, r, l') \in E$, whose guard $g$ is enabled $(v \models g)$, and it resets the clocks in $r$ to 0.

An execution of the *TA* is an alternating sequence of discrete and continuous transitions.

A simple example of *TA* is shown in Figure 2. It depicts an abstract model for a real-time task that starts execution when receiving the event schedule?. It sends an event done! as soon as the clock x has reached the best case execution time (bcet) and before reaching the worst case execution time (wcet), or goes to location MissingDeadline if the clock exceeds the deadline. Finally it returns to location Ready for the next execution round. The running task at location Executing can be preempted when receiving the event not_schedule?, and then it returns to the Ready location.

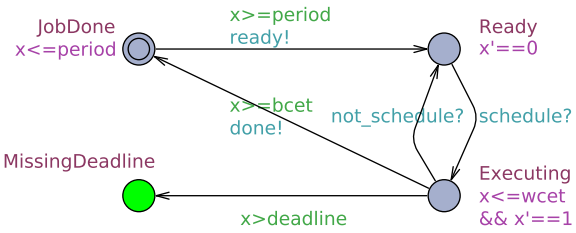


Figure 2: *TA* for a simple real-time task

Several *TA* can be combined in a network of *TA*. The semantics of the network is then defined using the parallel composition of *TA* such that transitions labeled with the same action are synchronized.

### 3.2 Probabilistic Timed Automata

Probabilistic Timed Automata (*PTA*) [4] are a stochastic extension of *TA*, such that probabilities are attached to the transitions of the *TA*. Formally, the transition relation of *TA* is replaced by the following definition:

$$\to \subseteq L \times \mathcal{B}(C) \times \Sigma \times Dist(2^C \times L)$$

where $Dist(2^C \times L)$ is a discrete probability distribution over the clocks reset and the next location. This definition generates a set of possible transitions in the semantics, whose choice is governed by the probability distribution. A *PTA* still exhibits non-deterministic behaviors, since the time elapsing is non-deterministic and several transitions may be enabled at the same time.

However, we can provide a fully stochastic semantics for *PTA* using the one defined in [14] for priced timed automata. First, we associate a continuous probability distribution for the time elapsing in each location. If the invariant of a location $l$ is unbounded, meaning that for all state $(l, v)$ there exists no delay $d \in \mathbb{R}_{\geqslant 0}$ such that $v + d \not\models I(l)$, then this probability distribution is an exponential distribution with a rate $\lambda(l)$ assigned to the location $l$. Otherwise we consider a uniform distribution over the timed bounds defined by the invariant. Second, we consider a uniform distribution between all the transitions enabled from the same state.

Using these probability distributions the set of executions of a *PTA* is the same as the one of the underlying *TA*, but it also defines a measurable space, with a probability over the set of executions. This will enabled to use statistical model-checking as an analysis technique for *PTA*.

### 3.3 Statistical Model-Checking

Correctness of the system is specified using formal logics that defines which are the admissible executions of the system. We will use a subset of the Computational Tree Logic (*CTL*) as defined by the model-checker UPPAAL. The grammar of this subset is $\varphi ::= \mathsf{A[]}P \mid \mathsf{A{<}{>}}P \mid \mathsf{E[]}P \mid \mathsf{E{<}{>}}P$. A and E are paths operators, meaning respectively "for all the paths" and "there exists a path". [] and <> are state operators, meaning respectively "all the 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 deadlock" specifies that in all the paths and all the states on these paths we will never reach a deadlock state, defined as a state where the system is permanently blocked.

*Model-checking* is an automated verification technique that explores all the possible executions of a *TA* to verify if it satisfies a property expressed in a logic like *CTL*. Probabilistic model-checking can also be used to compute the probability to satisfy a *CTL* property. However these technique are limited by state-space explosion problems when the model is too large, which can prevent the analysis due to a lack of memory.

To counter this limitations we will apply *Statistical Model-Checking* (*SMC*) techniques [15]. The principle is to combine formal verification and techniques from the statistic area. For instance, the Monte-Carlo algorithm computes $N$ executions $\rho$ and it estimates the probability $\gamma$ that the system satisfies a logical formula $\varphi$ 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 $\varphi$ is satisfied and 0 otherwise. It guarantees that the estimate $\tilde{\gamma}$ is close enough to the true probability $\gamma$ with a probability of error that is controlled by the number $N$ of simulations.

### 3.4 Hierarchical Scheduling Systems

Hierarchical scheduling systems (HSS) provide a flexible method to design and analyze scheduling systems. Instead of scheduling the entire system, tasks are grouped in *scheduling units* that can be analyzed independently. Scheduling units can themselves be grouped in a hierarchical manner, and the schedulability of the system is guaranteed by the schedulability of all the scheduling units at every levels.

Formally, a scheduling unit $C(W, A)$ is composed of a *workload* $W$ and a *scheduling algorithm* $A$. A workload $W$ is a set of tasks, and each task $T_i(p, e, d)$ is characterized by period ($p_i$), execution time ($e_i$), and deadline ($d_i$). Scheduling algorithms can be *fixed priority* (FP) and *earliest deadline first* (EDF).

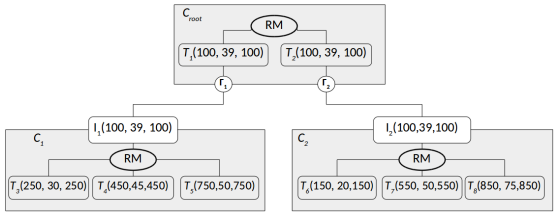


Figure 3: A hierarchical scheduling system

Fig. 3 shows an example of a 2-layer hierarchical scheduling system. At the base level the scheduling units $C_1$ and $C_2$ are composed of real-time tasks. These scheduling units are each associated to an interface $I_{C_i}(p_i, b_i)$, that specifies a period $p_i$ and a budget $b_i$. The budget is a collective timing requirements for the workload of the scheduling unit. At the upper level, the interfaces $I_{C_i}(p_i, b_i)$ are associated to tasks $T_i(p_i, b_i, p_i)$ in the $C_{root}$ scheduling unit. An interface can be viewed as a contract between parent and child units, such that the budget given to the interface is guaranteed by the parent unit. The tasks in a child unit can only execute when the parent task associated to the interface of the scheduling unit is running.

The schedulability analysis of an HSS can be performed in a compositional manner: each scheduling unit is checked to determine if it can schedule its tasks with the budget provided by its interface. We adopt the compositonal framework of [1, 28], where scheduling units are analyzed individually using a *periodic resource model* (PRM). The PRM provides the worst case assignement of the resources. Note that while HSS allow to compose the schedulability analysis of a complex system, this may cost some over-approximation.

### 3.5 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 [16], but with a strong emphasis on simplicity [23], 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 [29] directly. This is achieved by focusing on graph model structures (i.e. models consisting of nodes and edges) and automatically generating the required metamodel as well as the complete corresponding graphical editor from an abstract specification, essentially providing a form of constraint-based variability management [17, 18].

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 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 edges and express cardinality constraints on those connections. The second important file is a specification in the Meta Style Language (MSL), which is used for defining 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.

Those specifications are already enough for CINCO to generate the complete graphical modeling tool. But CINCO also provides mechanisms to integrate code that interprets or transforms the models. It automatically generates APIs specific to the model type and – similar to the appearance providers – seamlessly integrates code implemented against it into a ready-to-run modeling tool, which is realization of the one-thing-approach [22]. 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 HSS modeling tool. Please refer to [26, 27] or the website[1] 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, the most recent version of CINCO introduced a feature that 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.

Summing up, CINCO allows us to concentrate more on the formal model and its analysis instead of how to develop a graphical modeling tool with numerous complicated frameworks.

## 4. Formal Model-based Compositional Framework for HSSs

Our model-based compositional analysis tool implements a model-based analysis framework of HSSs that is flexible enough to represent any scheduling systems.

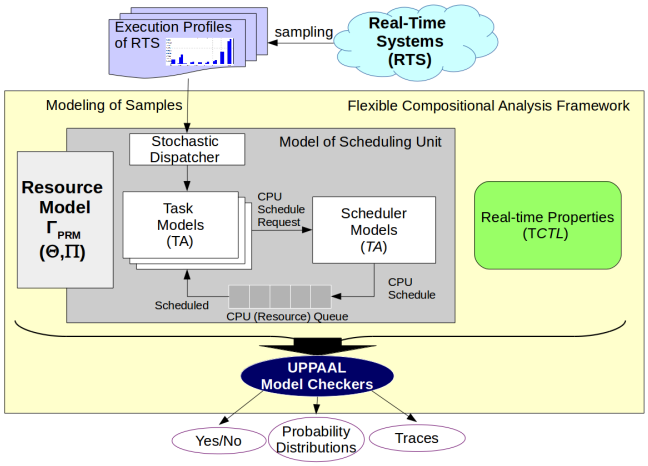


Figure 4: Flexible Compositional Analysis Framework

As shown in Fig. 4, the framework is composed of a set of component models (tasks, a scheduler and a stochastic dispatcher), that are used to configure the scheduling units of a HSS, and a set of real-time properties that must be analyzed.

The configuration of the scheduling units of a HSS are determined by the user, who defines the structure of the

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

HSS and specifies the real-time attributes of individual tasks. Once the configuration of the HSS has been made, our tool enables the designer to check the configuration of the HSS against real-time properties. In our setting, three important real-time properties are checked: the deadlock freedom of a HSS, and the schedulability and the worst-case response time of individual tasks.

### 4.1 Stochastic Task

We introduce in this paper a new model of stochastic tasks whose real-time attributes depend on probability distributions. An execution of a task is characterized by 3 real-time attributes: an execution time, a period, and a deadline. The difference between these stochastic tasks and the previous work [6, 9, 24] is that the three real-time attributes are dynamically configured according to the condition in which the system is running. This dynamic configuration is modeled by a stochastic dispatcher with an extension of Timed Automata with configuration actions that depends on the probability distributions.

A task represents the time spent for executing some computation. Its execution time may vary due to the length of executions of the computation logics and the capability of the execution environments, such as CPU, memory, I/O and caches, etc. Real values can be obtained by sampling the execution times from the real world system. The sampled execution times can then be captured by a probability distribution.

Meanwhile, 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. In a similar way, they can also be represented by probability distributions.

In our stochastic task model we consider discrete probability distributions, defined with a random variable $X$ given by:

$$X = \begin{pmatrix} x_1, ..., x_n \\ p_1, ..., p_n \end{pmatrix} \quad (1)$$

where $\{x_1, .., x_n\}$ are samples, $P(x_i) = p_i$ is the probability of each sample $x_i$ and $\sum_{i=1}^{n} p_i = 1$. The probability of any variable $x$ is given by $P(x)$ if $x \in \{x_1, .., x_n\}$, otherwise $P(x) = 0$.

Fig. 5 shows the TA for our stochastic task model. The only difference of this TA model with the periodic TA task model of [9] is that it begins an execution if its job's queue job_q[tid] is not empty.

If a process of the TA stochastic task at the Init location is instantiated, it reads the default attributes by function setTaskAttribute() and initializes a job. Then, the job requests the scheduler to assign a CPU by synchronizing the channel req_sched(pid) and queues at a resource (ready) queue by inserting its id (tid) to the queue.

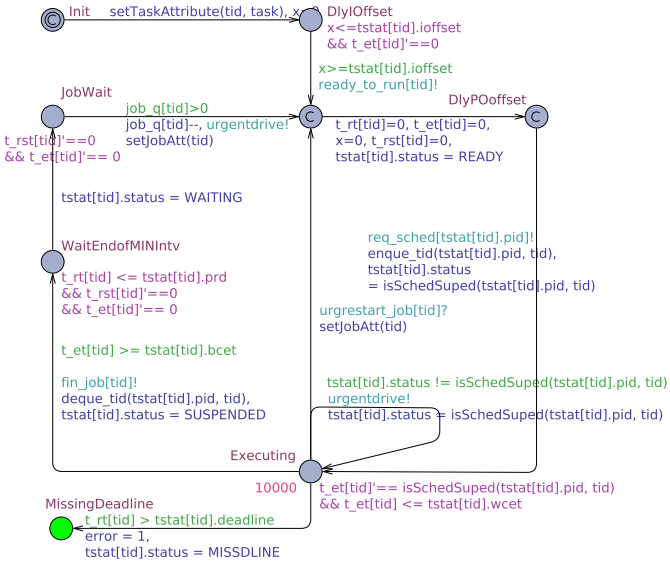


Figure 5: TA template of a stochastic task ($T_i$)

A job process may stay at location Executing as long as job's execution time is not fulfilled and it does not miss the deadline. The process stops and resumes its execution on that location according to the availability of CPU resource, i.e. the job process can make progress when a CPU is available, otherwise, it must stop its execution.

In our model, there is no preemption location to denote that a task is waiting for CPU after it has been preempted but preemption is implemented by a stopwatch clock t_et[tid]. This clock measures the CPU-consuming time of a task since a job of the task has been instantiated; the clock can stop and resume when a CPU is available to the task. At location Executing, the invariant expression t_et[tid]'=isSchedSuped(tstat[tid].pid),tid) is associated to the stopwatch clock. This condition is such that the clock progresses if the function isSchedSuped() returns 1, otherwise, it does not progress.

The process of a task exits from location Executing when it has fulfilled its execution time and it releases the CPU resource using function deque_tid(tstat[tid].pid, tid). Then, it joins the location WaitEndofMINIntv and wait the end of the minimal inter-arrival time. Finally, the process of a task joins the location JobWait to be instantiated by a job dispatcher.

### 4.2 Stochastic Dispatcher

These stochastic tasks are combined with a stochastic dispatcher that configures the real-time attributes of the tasks at each individual execution round. In other words, the stochastic dispatcher determines the configuration of task's real-time attributes at the beginning of each execution round when the task is waiting at the location JobWait.

To represent this dynamic configuration of real-time attributes, we extends the timed automata (TA) with a configuration action. An example of the configuration of a set of

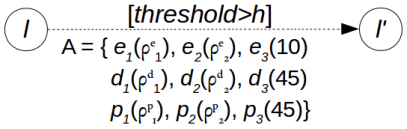


Figure 6: An action to configure stochastic real-time attributes

3 tasks is given in Fig. 6: The transition $l$ to $l'$ is enabled if the condition $threshold > h$ holds. When the transition is taken, the set $A$ of actions are carried out, meaning that the execution $e_1$ of task $T_1$ is chosen randomly according to the probability distribution $p_1^e$. In the similar way, the deadline and period of each individual stochastic tasks are taken from the corresponding probability distributions. Note that $e_3$, $d_3$ and $p_3$ are assigned to constants values, 10, 45 and 45, respectively.

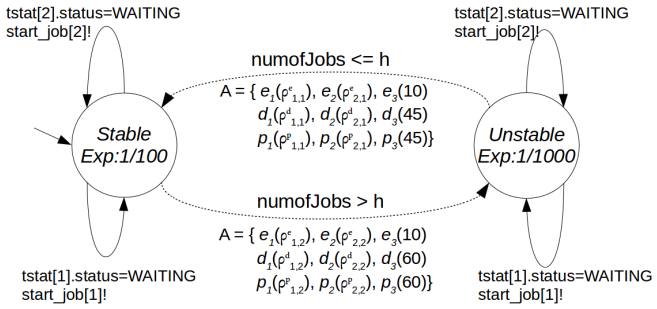


Figure 7: An action to configure stochastic real-time attributes

Fig 7 shows an example of a job dispatcher that uses the configuration actions. On the initial location Stable, two recursive transitions trigger the events start_job[1] and start_job[2] to instantiate the corresponding jobs if the conditions tstat[1].status=WAITING and tstat[2].status=WAITING hold. Even if the transitions are enabled, they are actually taken by the exponential distribution with rate $\lambda = 1/100$. If the condition $numofJobs > h$ holds, the transition heading for location Unstable can be taken. Then, a new configuration on the real-time attributes of $T_1$, $T_2$ and $T_3$ are made and, in particular, the real-time attributes of $T_1$ and $T_2$ are taken from the associated probability distributions, such as $\rho^e_{1,2}$, $\rho^p_{1,2}$, $\rho^d_{1,2}$, etc.

### 4.3 Formal Analysis Model of Scheduling Unit

The approach we pursue is compositional: each scheduling unit is individually analyzed with respect to an interface that abstracts the behavior of the other components. For the analysis of HSSs, the interface we are using is the PRM [28] that assigns the amount $\Theta$ of resources every period $\Pi$.

Fig. 8 depicts the conceptual model of a scheduling unit of a HSS: The scheduling unit is composed of a set of tasks ($T_i$), a scheduler ($A$), a queue ($pq$) and a stochastic

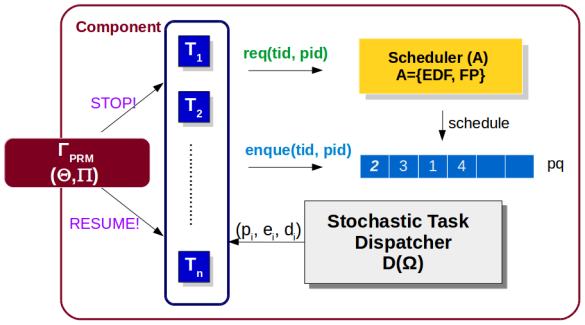


Figure 8: Conceptual model of a scheduling unit of a HSS

dispatcher $D$. The unit is given a PRM ($\Gamma_{PRM}(\Pi, \Theta)$) that is used to analyze the component in a compositional manner. We will call this resource model the *supplier*.

Our framework supports two types of tasks: periodic task and stochastic task. A periodic task instantiates at every the same period. Meanwhile, a stochastic task instantiates with a minimum inter-arrival time by an event. The real-time attributes of stochastic tasks are determined by the stochastic dispatcher $D$ using a set $\Omega$ of probability distributions, as shown in Fig. 8.

Once a job is instantiated by a task, it asks the scheduler for CPU computation time by firing the event req(tid,pid), which inserts the task's id into the ready queue pq. Then, the scheduler sorts task's identities according to a scheduling policy and chooses the id of the task having the highest priority. This task can carry out its jobs until it finishes the jobs or it is preempted.

The model of the scheduling unit is extended with a resource model $\Gamma_{PRM}(\Pi, \Theta)$ in Fig. 8 in order to analyze the HSS in a compositional manner. The resource model $\Gamma_{PRM}$ in Fig. 8 can stop and resume the execution of a running task. It determines when to stop and resume according to the timing requirement $(\Pi, \Theta)$. In this paper, the TA model of resource model $\Gamma_{PRM}$ is created such that it supplies the $\Theta$ amount of resources at every period $\Pi$ in a non-deterministic way, i.e. a task that is scheduled to use CPU is allowed to execute only for $\Theta$ time units at any time within its period.

Such a non-deterministic behavior simulates every resource supplying patterns of a parent task, including the extreme cases when the longest starvation of the resource assignment occurs, as mentioned in Section 3.4.

### 4.4 Resource Model

To speed up the schedulability analysis using model checking techniques, we adapt the PRM to generate the extreme cases more often, as depicted in Fig. 9: The TA model of the PRM uses two clocks, x and y. The clock x is reset every new period and used to measure the current time since a new period has begun. The clock y denotes the time of supplying the resources, so it may progress only when the process resides on the Supply location. A new supply period starts when the clock x reaches $\Pi$ at location PrdDone. Then, one

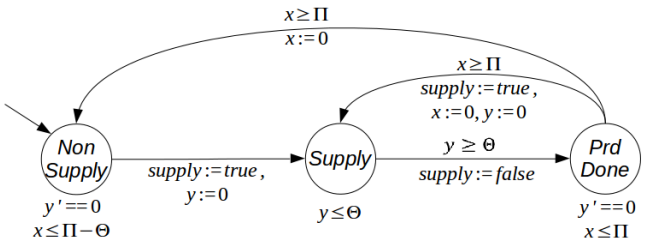


Figure 9: Abstract PRM model in TA

of the two transitions existing from location PrdDone is taken non-deterministically. One of the transitions leads to location Supply where immediately starts the resource supply. Otherwise, the other transition leads to the location NonSupply that postpones the resource supply up to the time $\Pi - \Theta$ that is the laxity time of the resource supply.

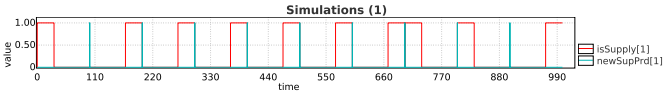


Figure 10: A simulation of PRM behavior model

Fig. 10 shows a simulation of the PRM behavior that provides a resource for 33 time units every 100 time units. The spike in blue denotes a period of the supply and the graph in red denotes a resource supply. Note that the resource supply begins and terminates in synchronization with the beginning and end of a period, which implies that the longest starvation of the supplying resources can occur extremely often.

# 5. Model Checking Tool Generated by CINCO

The formal framework presented in the previous section is generic enough to describe a wide range of HSSs. However, it still requires from the system designer a lot of knowledge to select and configure the components needed for the scheduling analysis. Powerful model-checking tools like UPPAAL and UPPAAL SMC can be used to verify these systems, but the configuration of the tools and the interpretation of the results also requires very specific knowledge. It is therefore of great interest to provide the engineers with more user-friendly interfaces to interact with model-checking tools.

## 5.1 Automated Generation of the Interface

Using the tool generator CINCO we can automatically generate an interface capable of specifying HSSs in a graphical interface, launching formal analyses and displaying relevant information on the graph from the results.

This interface is specified using a few lines of code:

- The MGL file (132 lines) specifies the components (nodes and edges) that can be used to design an HSS and it references the actions that can be performed on these components. Listing 1 presents the specification of a supplier node. This node possesses a set of parameters, some of them being referenced by the style to be displayed (policy, resource, tid, period and budget). It can accept one incoming edge and several outgoing edges. It can contain nodes of the type Query to specify properties to verify.
- The STYLE file (136 lines) specifies the graphical style of these components.
- Custom actions in Java, that are included during the automated generation (cf. Sect. 3.5). These actions allow to 1. generate TA models corresponding to the HSS, 2. launch UPPAAL to verify queries from the model, 3. parse the results of the analyses, 4. display them in the graphical interface, either through message notifications, modifications of the style of model or the display of graphs.

## 5.2 Specification of an HSS in the Tool

We now present how to specify an HSS with the graphical interface generated by CINCO. Fig. 11 shows the generated interface inside the Eclipse environment. The edition zone allows to construct an HSS by selecting components from the *Palette* on the left and linking them together.

***Suppliers*** A supplier describes the interface mechanism of a scheduling unit. Three types of supplier are available: TopSupplier (in blue) for the root of the HSS; ProbSupplier (in red), whose budget is stochastically determined within a minimum and maximum budget; Supplier (in yellow) with a fixed budget. In the parameters of these suppliers we select the scheduling policy and the resource model. Suppliers can contain queries, and they can have task nodes and other supplier nodes as childs.

***Tasks*** Tasks are the leaf nodes of the HSS tree. Normal tasks are displayed with rounded rectangle node. They have several integer parameters (deadline, best execution time, worst execution time, period, priority). ProbTasks (rectangle nodes) implements the formal model of stochastic task presented in Section 4. They can specify discrete probability distribution in place of period, deadline and execution time.

```
1  @style(supplier, "${policy}", "${resource}","${tid}",
2  "${period}","${budget}")
3  container Supplier {
4    attr Policy as policy
5    attr Resource as resource
6    attr EInt as tid
7    attr EInt as priority
8    attr EInt as period
9    attr EInt as budget
10   attr EInt as deadline
11   incomingEdges (Transition[1,1])
12   outgoingEdges (Transition[1,*])
13   containableElements (Query)
14 }
```

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

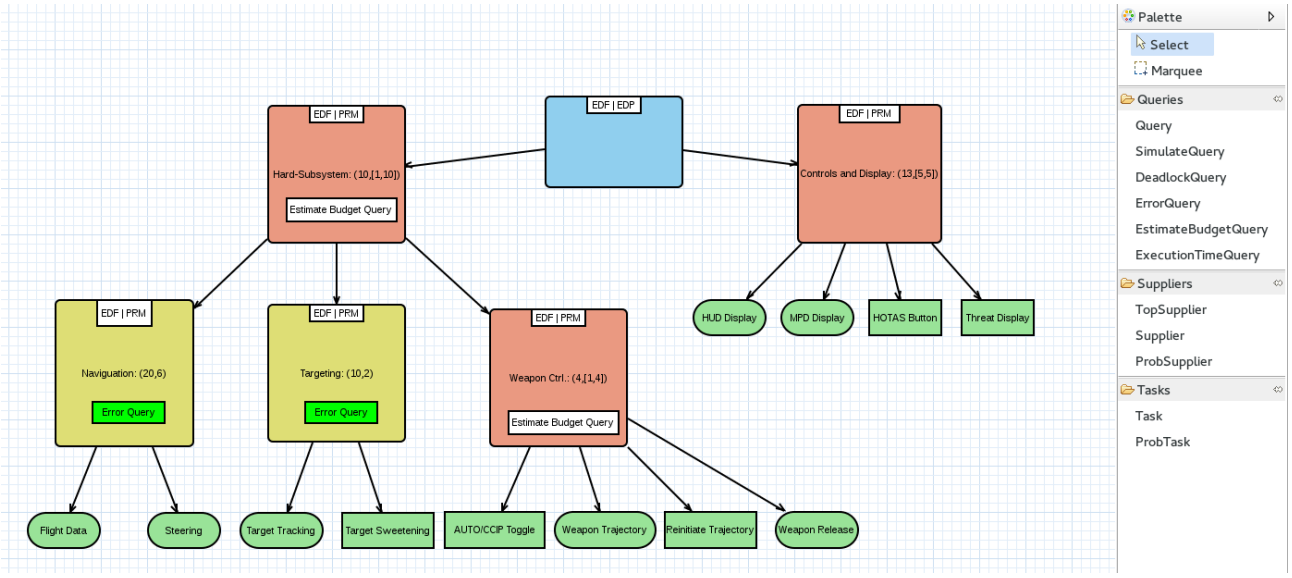


Figure 11: Generated tool interface

***Queries*** Several type of queries can be added to a supplier in order to verify the scheduling unit. The generic type allows to specify any CTL formula, but the tool provides pre-configured type of queries that allow to perform the most usual checks without any knowledge of temporal logics.

*Deadlock Queries* check if the model has been correctly design such that there is no deadlock. *Error Queries* check if a scheduling unit is schedulable, which means that no task ever exceeds its deadline. *Execution Time Queries* compute the maximum response of a task. *Estimate Budget Queries* compute the necessary budget for probabilistic suppliers. *Simulate Queries* run random simulations on a scheduling unit in order to display the scheduling periods provided by the resource model.

With the graphical description of the HSS in our tool, we can generate automatically *TA* models and CTL properties in the format of the tool UPPAAL, using the model bank of our framework described in Section 4. One model is generated for each scheduling unit, and it can be used to check the queries associated to this scheduling unit.

### 5.3 Model-Checking and Feedbacks

Model-checking is launched from a right-click menu on a query. It generates the *TA* model of the scheduling unit and translate the query in a CTL property. Then, depending on the query, either UPPAAL or UPPAAL SMC is used to analyzed the model, and various results are displayed in our tool.

***Model-checking queries*** Generic CTL queries and deadlock queries are checked with UPPAAL model-checker. To this end, deadlock queries are translated to the CTL formula A[] not deadlock. the result of teh analysis is displayed in the tool with a message saying whether the property has been satisfied, and the color of the query is changed accordindly to green if satisfied and to red if not.

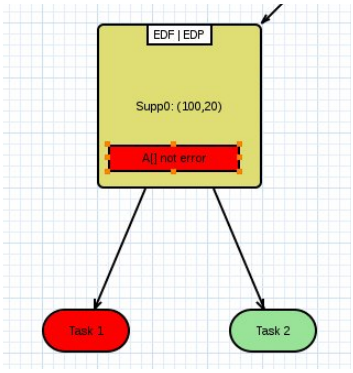


Figure 12: Task that has missed a deadline

***Error queries*** Error queries can be checked with UPPAAL model-checker for an exhaustive analysis. In that case, they are translated to the CTL formula A[] not error. For better performances, an option enables to use UPPAAL SMC for the analysis. In that case the query is translated to a probabilistic property Pr[<=runTime](<>error), which computes the probability to reach an error state within runTime time units.

The results of the analysis are parsed by the tool: if an error state is found, it detects which task missed its deadline, and the color of the query is changed to red as well as the faulty task, as shown in Fig. 12. Otherwise, the color of the query is changed to green.

***Execution time queries*** The query estimate the maximum response time of a task using UPPAAL SMC. To this end the tool translates it to a CTL formula like E[<=100000; 1000](*max* :

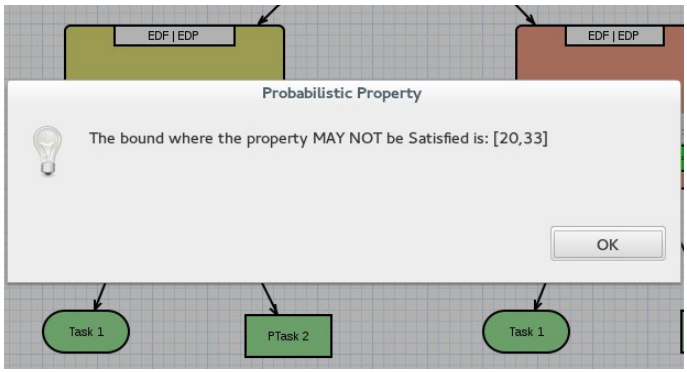


Figure 13: Minimum budget of probabilistic supplier

$t_rst[2]$), which estimates the maximum response time of the task with id 2, over a simulation time of 100'000 t.u. and using 1000 simulations. The result is displayed in the tool with a message box.

***Estimate budget queries*** Using probabilistic suppliers we can determine what is the minimum budget needed by a scheudling unit such that all the tasks satisfied their deadline. This supplier is given a minimum and maximum budget and the *TA* model randomly select a value within this range. The query is translated to the probabilistic property:

$$\Pr[\text{estBudget}[1]<=\text{runTime}]\ (<>\text{globalTime}>=\text{runTime and error})$$

This property is analyzed with UPPAAL SMC that runs random simulations on the model and estimates for each value of the budget the probability that a task misses a deadline.

The tool analyses the results of UPPAAL SMC and determine the minimum budget, as shown in Fig. 13.

***Simulate queries*** Using UPPAAL SMC we can also perform random simulations and display the results in a graph. This is useful to look at the variation of selected parameters from the model. For instance in Fig. 14 the graph displayed in our tool shows the value of 4 parameters: the beginning of a supply period, the status of the supplier and the status of the 2 tasks.

## 6. Case Study

We to apply our framework to model and verify an avionic scheduling system. We consider the specification of avionic tasks presented in [20]. This is a mixed-critical system with multiple tasks of varius criticality running together. We arrange these tasks in a hierarchical scheduling system by grouping tasks from similar functions and critically (Navigation, Targeting, Weapong control and Controls and displays). Each function is associated to scheduling unit. The three scheduling units of the most critical functions (navigation, targeting and weapong control) are further grouped

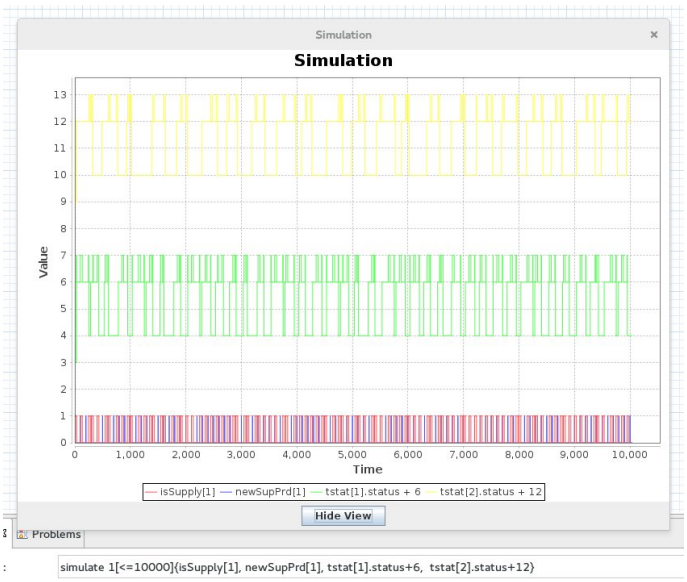


Figure 14: Variations of parameters along a random simulation

under a "Hard-Subsystem" scheduling unit. This results in the hierarchical scheduling systems presented in Fig. 15.

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.

### 6.1 Modeling with Cinco

We design the HSS in Cinco. Sporadic tasks are modeled with stochastic task nodes and are associated to probability distributions. To estimate their necessay budget, each scheduling unit is modeled using a probabilistic supplier.

### 6.2 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.

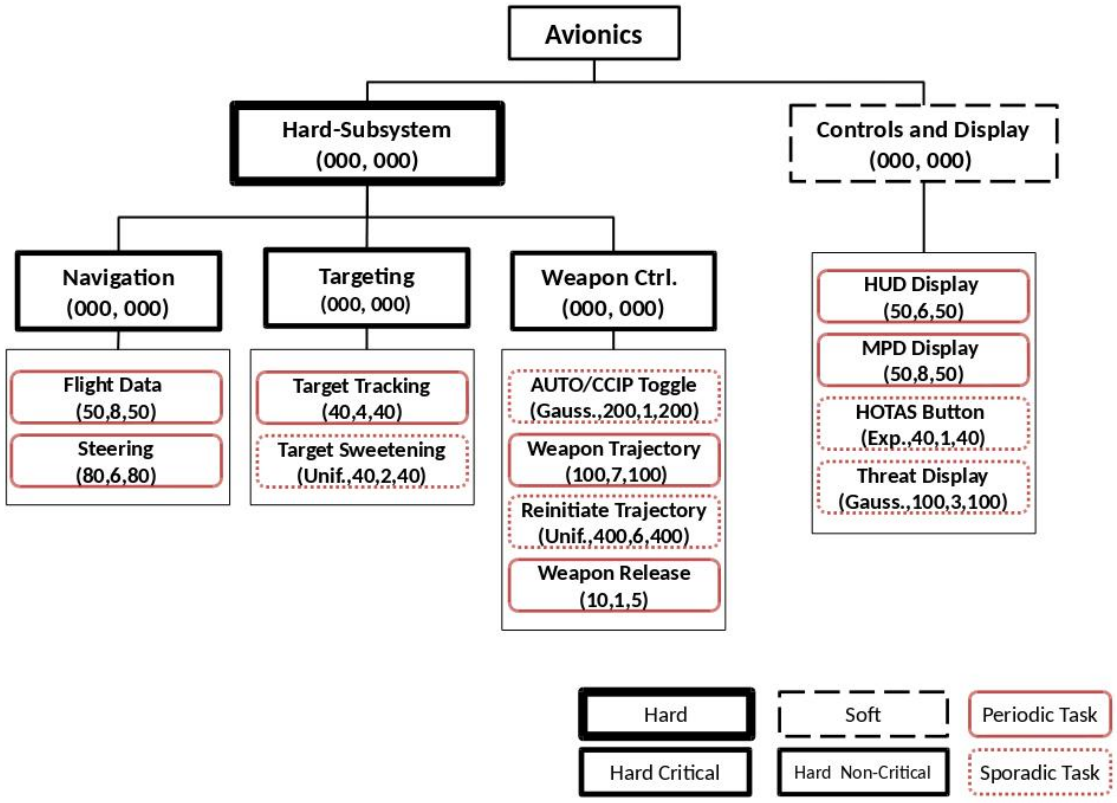


Figure 15: Hierarchical scheduling of avionic tasks

| 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

### 6.3 Results

We present in the Fig. 16 the results obtained from the analysis of the 3 bottom scheduling units (Navigation, Targeting, Weapong 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 error query, that is checked either with model-checking or SMC.

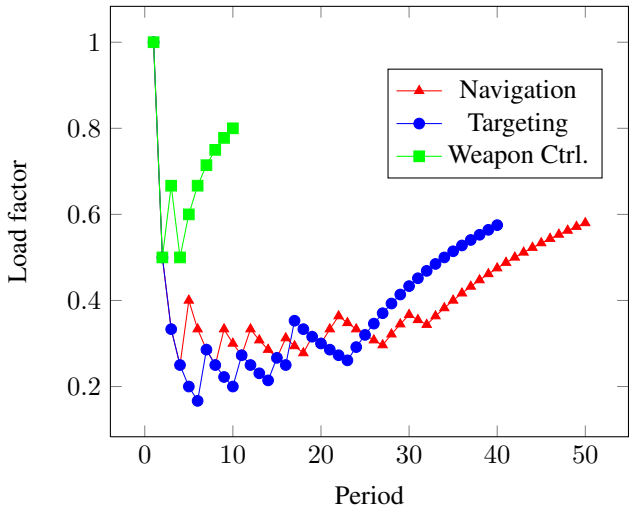


Figure 16: Budget estimation for Navigation, Targeting and Weapon control

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 error 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 ressources.

## 7. Conclusions

We have presented a formal framework that performs model-based analysis of the schedulability of CPS. This framework is based on the hierarchical scheduling concept that allows to decompose the schedulability analysis into subsystems, each associated to its own scheduler. In this paper, we have extended these HSS with a model for stochastic task that allows to abstract several timing behaviors or represent unknown behaviors of sporadic tasks. In order to make these techniques more accessible we have embedded our framework in a domain-specific tool generated by the tool generator CINCO. This front end allows a system designer to specify an HSS using simple graphical components and it can launch formal analyses using the model-checking and SMC algorithms of the tool UPPAAL. Finally, our framework can interpret the results of these analyses and output comprehensible information to the system designer.

In future works we would like to increase the accessibility of our approach by generating automatically the formal properties from a collection of queries that can be checked on HSS.

## References


[1] *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*, 2003. IEEE Computer Society. ISBN 0-7695-2044-8.

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

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *In Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190. IEEE Computer Society Press, 1990.

[4] D. Beauquier. On probabilistic timed automata. *Theor. Comput. Sci.*, 292(1):65–84, Jan. 2003. ISSN 0304-3975. . URL `http://dx.doi.org/10.1016/S0304-3975(01)00215-8`.

[5] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, pages 125–126. IEEE Computer Society, 2006. ISBN 0-7695-2665-9. . URL `http://dx.doi.org/10.1109/QEST.2006.59`.

[6] A. Boudjadar, J. H. Kim, A. David, K. G. Larsen, M. Mikučionis, U. Nyman, A. Skou, I. Lee, and L. T. X. Phan. Flexible framework for statistical schedulability analysis off probabilistic sporadic tasks. In *18th International Symposium of Real-Time Distributed Computing (ISORC)*. To be appeared.

[7] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In *FACS 2013*, volume 8348 of *LNCS*, pages 61–78. Springer, 2013.

[8] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Degree of schedulability of mixed-criticality real-time systems with probabilistic sporadic tasks. In *Theoretical Aspects of Software Engineering Conference (TASE), 2014*, pages 126–130, Sept 2014. .

[9] A. Boudjadar, A. David, J. Kim, K. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Widening the schedulability of hierarchical scheduling systems. In I. Lanese and E. Madelaine, editors, *Formal Aspects of Component Software*, volume 8997 of *Lecture Notes in Computer Science*, pages 209–227. Springer International Publishing, 2015. ISBN 978-3-319-15316-2. . URL `http://dx.doi.org/10.1007/978-3-319-15317-9_14`.

[10] J. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, A. Skou, I. Lee, L. P. Xuan, and U. Nyman. Quantitative schedulability analysis of continuous probability tasks in a hierarchical context. Springer International Publishing, 2015. URL `http://dx.doi.org/10.1007/978-3-319-15317-9_14`. To be appeared.

[11] M. Cordovilla, F. Boniol, J. Forget, E. Noulard, and C. Pagetti. Developing critical embedded systems on multicore architectures: the PRELUDE-SCHEDMCORE toolset. In S. Faucou, A. Burns, and L. George, editors, *RTNS 2011*, pages 107–116, 2011. URL `http://rtns2011.irccyn.ec-nantes.fr/files/rtns2011.pdf`.

[12] A. David, J. I. Rasmussen, K. G. Larsen, and A. Skou. *Model-based Framework for Schedulability Analysis Using Uppaal 4.1d*.

[13] A. David, K. G. L. Larsen, A. Legay, and M. Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, volume 7610 of *Lecture Notes in Computer Science*, pages 293–307. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34031-4. . URL `http://dx.doi.org/10.1007/978-3-642-34032-1_28`.

[14] A. David, K. G. Larsen, A. Legay, and D. B. Poulsen. Statistical model checking of dynamic networks of stochastic hybrid automata. *ECEASST*, 66, 2013. URL `http://journal.ub.tu-berlin.de/eceasst/article/view/893`.

[15] A. David, K. Larsen, A. Legay, M. Mikučionis, and D. Poulsen. Uppaal smc tutorial. *International Journal on*

*Software Tools for Technology Transfer*, pages 1–19, 2015. ISSN 1433-2779. . URL http://dx.doi.org/10.1007/s10009-014-0361-y.

[16] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, Boston, MA, USA, 2008.

[17] S. Jörges, A.-L. Lamprecht, T. Margaria, I. Schaefer, and B. Steffen. A Constraint-based Variability Modeling Framework. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(5):511–530, 2012. .

[18] A.-L. Lamprecht, S. Naujokat, and I. Schaefer. Variability Management Beyond Feature Models. *IEEE Computer*, 46 (11):48–54, 2013. ISSN 0018-9162. .

[19] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973. ISSN 0004-5411. . URL http://doi.acm.org/10.1145/321738.321743.

[20] D. Locke, L. Lucas, and J. Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990. URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11181.

[21] S. Manolache, P. Eles, and Z. Peng. Analysis of monoprocessor systems. In *Real-Time Applications with Stochastic Task Execution Times*, pages 27–60. Springer Netherlands, 2007. ISBN 978-1-4020-5505-8. . URL http://dx.doi.org/10.1007/1-4020-5509-9_4.

[22] T. Margaria and B. Steffen. Business Process Modelling in the jABC: The One-Thing-Approach. In J. Cardoso and W. van der Aalst, editors, *Handbook of Research on Business Process Modeling*. IGI Global, 2009.

[23] T. Margaria and B. Steffen. Simplicity as a Driver for Agile Innovation. *Computer*, 43(6):90–92, 2010. ISSN 0018-9162. .

[24] D. Maxim and L. Cucu-Grosjean. Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters. In *RTSS 2013 - IEEE Real-Time Systems Symposium*, Vancouver, Canada, 2013.

[25] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Cambridge, MA, USA, 1983.

[26] S. Naujokat, L.-M. Traonouez, M. Isberner, B. Steffen, and A. Legay. 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 2014)*, number 8802 in LNCS, pages 463–480. Springer, 2014. .

[27] S. Naujokat, M. Lybecait, D. Kopetzki, and B. Steffen. CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools. 2015. to appear.

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

[29] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley, Boston, MA, USA, 2008.

[30] A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Probabilistic preemption control using frequency scaling for sporadic real-time tasks. In *The 7th IEEE International Symposium on Industrial Embedded Systems*, June 2012. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=2888.

[31] Y. Zhang, D. K. Krecker, C. Gill, C. Lu, and G. H. Thaker. Practical schedulability analysis for generalized sporadic tasks in distributed real-time systems. In *Proceedings of the 2008 Euromicro Conference on Real-Time Systems*, ECRTS '08, pages 223–232, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3298-1. . URL http://dx.doi.org/10.1109/ECRTS.2008.33.