Component-Based Analysis of Hierarchical Scheduling using Linear Hybrid Automata

Youcheng Sun*, Giuseppe Lipari*†, Romain Soulat†, Laurent Fribourg†, Nicolas Markey†
*Scuola Superiore Sant'Anna
{firstname.lastname}@sssup.it
†LSV - ENS Cachan and CNRS,
{firstname.lastname}@lsv.ens-cachan.fr

***Abstract*—Formal methods (e.g. Timed Automata or Linear Hybrid Automata) can be used to analyse a real-time system by performing a reachability analysis on the model. The advantage of using formal methods is that they are more expressive than classical analytic models used in schedulability analysis. For example, it is possible to express state-dependent behaviour, arbitrary activation patterns, etc.**

In this paper we use the formalism of Linear Hybrid Automata to encode a hierarchical scheduling system. In particular, we model a dynamic server algorithm and the tasks contained within, abstracting away the rest of the system, thus enabling component-based scheduling analysis. We prove the correctness of the model and the decidability of the reachability analysis for the case of periodic tasks. Then, we compare the results of our model against classical schedulability analysis techniques, showing that our analysis performs better than analytic methods in terms of resource utilisation. We further present two case studies: a component with state-dependent tasks, and a simplified model of a real avionics system. Finally, through extensive tests with various configurations, we demonstrate that this approach is usable for medium-size components.

I. Introduction

The complexity of modern embedded real-time applications, like automotive and avionics systems, is steadily increasing. Until recently, complexity was addressed by using physical separation: each different functionality was implemented by a different application module on a different ECU (Electronic Control Unit), and all ECUs were connected by a real-time control network.

The pressure to reduce the design costs and the number of ECUs is forcing developers to integrate different applications on the same computational platform. The IMA (Integrated Modular Avionics) [1], [2] is a set of standard specifications for simplifying the development of avionic software; among other requirements, it allows different independent applications to share the same hardware and software resources [3].

To avoid interference between independently developed applications that share the same processor, the underlying RTOS must support the concepts of *temporal partitioning* and *hierarchical scheduling* [4]–[6]. Hierarchical scheduling consists in using two (or more) levels of scheduling: the global one performs the temporal partitioning among the applications; whereas the local ones are specific for each application and dictate which task to execute. In Fig. 1 we show a pictorial representation of a hierarchical scheduling system consisting of three applications that coexist in the same single processor system, each one with its own scheduler. Each application and its corresponding local scheduler are "wrapped" into an entity that we denote as *scheduling server* (or simply *server*) which acts as a mediator between the global scheduler and the application. The global scheduler "sees" the entire application as it were a single task to be scheduled according to its specific scheduling parameters; the application "sees" the platform on which it is executing as a virtual processor of slower speed.

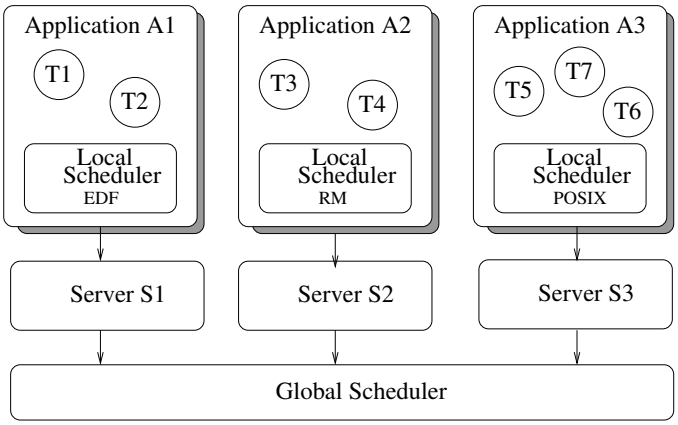

Fig. 1: An example of hierarchical scheduling system.

Therefore, the combination of temporal partitioning and hierarchical scheduling makes it possible to define a virtual processor for each application, and to perform schedulability analysis on the virtual processor rather than on a single dedicated processor.

If the applications running on a system are independent of each other, then it is possible to analyse each of them in isolation; in fact, the ability of an application to meet its deadlines depends on the worst-case computation times and the arrival patterns of its tasks, and on the temporal partition that the global scheduler (and the server) allocate to it, but *it does not* depend on the presence of other applications in the system.

Such property enables *component-based* schedulability analysis, a research topic largely investigated in recent years. In particular, many different schedulability analyses have been

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7 under grant agreement n. 246556, "RBUCE-UP"

proposed for independent applications of periodic or sporadic tasks, even when they share mutually exclusive resources. However, such analyses usually rely on simple task models, like periodic or sporadic tasks: introducing a new arrival pattern or a new task model requires to derive new equations for the analysis.

Schedulability analysis is also possible using formal methods, like Timed Automata (TA) [7] or Linear Hybrid Automata (LHA) [8]. The advantage of using formal methods is that they are much more expressive than classical models used in schedulability analysis [9]. Since these methods are extensions of state-based models with time, it is very easy to express state and time-dependent behaviours and arbitrary arrival patterns. The schedulability problem is usually encoded as a reachability problem on the state space of the model: a deadline-miss condition is modelled as a *bad location* that should never be reached by the model. However, few researchers [10]–[14] until now have proposed to use formal methods for modelling hierarchical systems.

Contributions of this paper: In this paper we use the formalism of LHA [8] to model a dynamic periodic server algorithm for hierarchical scheduling, and the application contained within. We prove the correctness of the model with respect to server schedulability. We also prove the decidability of the schedulability analysis for the case of periodic independent tasks.

We compare the results of our model against the schedulability test proposed by Lipari and Bini in [6] on a simple case-study. The results are reported in Section VIII and show that our test based on LHA finds many more schedulable points than the Lipari-Bini test, and thus permits to achieve a better resource utilisation. In Section VIII-B, we show how to encode the schedulability analysis of a state-dependent application, thus demonstrating the power of our methodology. We also use our model in the hierarchical design of a complex avionics system. Finally, a set of experiments has been conducted to evaluate the complexity of our model-based component analysis for practical use.

II. State of the Art

The ARINC 653 standard [3] defines temporal partitioning for avionics applications. The global scheduler is a simple Time Division Multiplexing (TDM), in which the time is divided into *frames* of fixed length, each frame is divided into *slots* and each slot is assigned to one application. Besides TDM, more dynamic time partitioning algorithms are possible: for example the periodic resource model [5] and the periodic server [6].

Dynamic server algorithms have some advantages over TDM. First of all, the temporal interface of a periodic server consists only of two parameters: the budget Q and the period P; the server guarantees that the application will receive Q time units every period P, but unlike the TDM, it does not specify *at which precise instants* the application will receive the allocation. This means that, once the application has been guaranteed feasible on a server with certain parameters Q and P, during the integration phase the designer has much more freedom in the allocation of the budget. The second advantage is that a dynamic server algorithm can better take advantage of the dynamic behaviour of the application and adapt itself at run-time to different conditions.

Hierarchical scheduling and component based real-time scheduling analysis has been studied extensively in the past years. Feng and Mok [4] proposed the *resource partition model* and schedulability analysis for it. Shih and Lee [5] introduced the concept of temporal interface and the periodic resource model. Lipari and Bini [6] proposed the periodic server model to abstract many different temporal partitioning algorithms, and an algorithm to compute the values of the parameters to make the application schedulable. Davis and Burns [15] proposed a method to compute the response time of tasks running on a local fixed priority scheduler when the tasks periods are synchronised with the server period.

Schedulability analysis using TA has been proposed by many researchers [9]. In [16], Timed Automata are extended with tasks and non-preemptive scheduling is encoded as a reachability problem. Based on [16], Fersman and Yi [17], [18] provide decidability and undecidability results for generic real-time schedulability analysis in Timed Automata, and they formalise TA extended with tasks as *Task Automata*. In [19], self-suspending task schedule is modelled in TA and an off-line schedule is generated. [20] uses TA with stopwatches for modelling a schedule in distributed real-time systems. None of these works take resource partition into account.

A formal model of hierarchical scheduling systems using TA has been proposed in [10]. The goal of the authors is to verify the correctness of the two-level scheduler and generate C code for the scheduler and the tasks. Moreover, their analysis is *global* in the sense that they verify the correctness of the whole system rather than a single application. In this paper, instead, we aim at component-based schedulability analysis of a single application. Another formal model of hierarchical scheduling using Parametric Timed Automata (PTA) has been proposed in [11], [12]. The authors restrict themselves to a TDM global scheduler and perform a global analysis (rather than a component-based one).

A component based analysis of hierarchical real-time systems is proposed in [13], [14]. The authors use the model of Preemptive Timed Petri Nets (pTPN), to model a hierarchical systems, and are able to perform analysis of independent applications. They show that component based analysis considerably reduces the complexity of analysing a system. The main difference with this paper is that they model a TDM global scheduler, whereas in this paper we model a dynamic periodic server algorithm.

III. System Model

In this paper we assume that an application is a set of periodic real-time tasks $\mathcal{A} = \langle \tau_1, \tau_2, \ldots, \tau_n \rangle$. The system consists of a set of applications to be scheduled using hierarchical scheduling and temporal partitioning on a single processor system. We assume that all applications are independent of each other thus that we can analyse them in isolation.

Each application is executed upon a *virtual processor platform*, which is provided by a *server*. In this paper we consider the periodic server proposed in [6]. Each server is assigned a budget Q and a period P, and the global scheduler guarantees

that the application will receive Q units of execution time every P time units. The global scheduler performs Earliest Deadline First (EDF) [21] among the servers: each server is considered as a periodic task with period and relative deadline P, and worst-case computation time Q. Therefore, it must hold true that $\sum_i \frac{Q_i}{P_i} \leq 1$.

A periodic task τ_i is a tuple $\langle O_i, C_i, T_i, D_i, p_i \rangle$, where O_i is the initial phase, C_i is the Worst-Case Execution Time (WCET), T_i is the period, D_i is the relative deadline, and p_i is an integer that represents the task's priority (lower p_i means higher priority). A task is activated every period, starting from its phase O_i, at time instants $r_{i,k} = O_i + kT_i$ (where k is a non-negative integer), called *release times*. Each task instance (or job) must execute for C_i units of time within its absolute deadline $d_{i,j} = r_{i,j} + D_i$. In this paper, we consider constrained deadline tasks, i.e., $D_i \leq T_i$.

As for local scheduler, we assume the Fully Preemptive Fixed Priority (FPFP) scheduler: each task is assigned a priority p_i, and the task with the highest priority is chosen to execute, preempting lower priority tasks when necessary.

IV. Server Algorithm

In this section we present the server algorithm that is used to provide the temporal partition necessary for an application to execute. We use the same algorithm proposed in [6], which is a particular case of the Constant Bandwidth Server [22]. We summarise the algorithm here for convenience.

A server S is assigned two parameters: Q is the server maximum budget, and P is the server period. In addition, the server maintains three internal variables: q represents the *current remaining budget*, and d is the current *scheduling deadline*, and an internal state which can be one of the following:

- Idle: the initial state; it represents the situation in which no task is active in the application;
- Active: when there is at least one active task, but the server is not executing because other servers (for other applications) with earlier scheduling deadlines have been selected by the global EDF scheduler;
- Executing: when the server has been selected by the global EDF scheduler, and it is running an application task;
- Recharging: when there is at least one active task in the application, but the server cannot execute because the current budget is zero;
- Empty: when there is no active task, but the server has already consumed part of its budget, so it has to wait before it can become Idle again.

The server variables and the server state are updated according to the following rules:

1) Initially, $q = 0, d = 0$ and the server is Idle.
2) When a task is released at time t, if the server is Idle, then $q := Q$ and $d := t + P$, and the server becomes Active; if the server is already Active, then nothing needs to be done.
3) At any time t, the global scheduler selects an Active server, and a task inside it which will be chosen to run subject to local scheduling policy. The chosen server moves to the Executing state.
4) While some task in the server is running, the current budget q is decremented accordingly.
5) The global scheduler can preempt a server to execute another server. The preempted server moves back to the Active state.
6) If q reaches 0 and some task has not completed execution, then the server will be suspended till time d, and moves to Recharging. During the suspended interval, it cannot be chosen by the global scheduler. At time d, q is recharged to Q and d is set to $d+P$ and the server moves to the Active state.
7) When, at time t, the last task in the server has finished its execution, if $t \geq d - q\frac{P}{Q}$, the server becomes Idle; otherwise, it remains Empty, and will become Idle at time $d - q\frac{P}{Q}$, unless another task arrives before that time point.

The global scheduler performs the Earliest Deadline First policy using the scheduling deadlines of the servers. The following two results are direct consequences of Theorem 1 and Lemma 1 of [22]:

Theorem 1. *Consider a system consisting of n servers, $\{S_1, \ldots, S_n\}$, with $S_i = (Q_i, P_i)$, such that $\sum_{i=1}^{n} \frac{Q_i}{P_i} \leq 1$. Then, no server misses its scheduling deadlines.*

Theorem 2. *Given a server $S_i = (Q_i, P_i)$, let t_s be an instant in which the server moves from the idle state to the active state, and let t_f be the first instant after t_s such that the server becomes idle again. Then the server receives in interval $[t_s, t_f]$ an amount of execution time Δ_{exe} which is bounded by:*

$$\left\lfloor \frac{t_f - t_s}{P_i} \right\rfloor Q_i \leq \Delta_{\mathsf{exe}} \leq \left\lceil \frac{t_f - t_s}{P_i} \right\rceil Q_i$$

V. Linear Hybrid Automata

A hybrid automaton [8], [23] is a finite-state automaton associated with a finite set of variables continuously varying in dense time. In this section, we introduce the basic terminology and the definition of Linear Hybrid Automata.

Let $\mathsf{Var} = \{x_1, \ldots, x_n\}$ be a set of variables and $\dot{\mathsf{Var}} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ be the set of variables' derivatives over time. A *linear constraint atom* over Var is of the form $\sum_{i=1}^{n} c_i x_i \sim b$, where c_i (for $1 \leq i \leq n$) and b are rational numbers and $\sim \in \{<, \leq, =, \geq, >\}$. A *linear constraint* is the conjunction of a finite number of constraint atoms. A *valuation* ν over Var is a function that assigns a real value to each element in Var. The same notations can also be defined for $\dot{\mathsf{Var}}$.

Definition 1. *A Linear Hybrid Automaton $H = \langle \mathsf{Var}, \mathsf{Loc}, \mathsf{Init}, \mathsf{Lab}, \mathsf{Trans}, D, \mathsf{Inv} \rangle$ consists of seven components:*

1) *A finite set Var of variables.*
2) *A finite set Loc of locations.*
3) *A labelling function Init that specifies the set of initial valuations for initial locations of the automaton. If l is not an initial location, then $\mathsf{Init}(l) = \emptyset$.*
4) *A finite set Lab of synchronisation labels including a* stutter *label ϵ.*

5) *A finite set Trans of transitions. Each transition is a tuple $\langle l, \gamma, a, \alpha, l' \rangle$ consisting of a source location l, a target location l', a guard γ that is a linear constraint over Var, a synchronisation label $a \in$ Lab, and the transition relation α that assigns values to variables in Var. We say that l is a pre-location of l'. We require that on each location, there is a stutter transition $(l, true, \epsilon, \mathrm{Id}, l)$ where $\mathrm{Id} = \{(\nu, \nu) | \nu \in V(Var)\}$ is the identical transition relation. These are the only ϵ-labelled transitions.*
6) *A labelling function D which assigns to each location a linear constraint over variables' derivatives.*
7) *A labelling function Inv which assigns to each location a constraint, called* invariant*, over variables. The automaton can only stay in location l as long as current valuations of variables satisfy Inv(l).*

Let $H_1 = \langle \mathsf{Var}, \mathsf{Loc}_1, \mathsf{Init}_1, \mathsf{Lab}_1, \mathsf{Trans}_1, D_1, \mathsf{Inv}_1 \rangle$ and $H_2 = \langle \mathsf{Var}, \mathsf{Loc}_2, \mathsf{Init}_2, \mathsf{Lab}_2, \mathsf{Trans}_2, D_2, \mathsf{Inv}_2 \rangle$ be two LHA over a set of variables Var. Their *parallel composition* $H_1 \times H_2$ is the LHA $\langle \mathsf{Var}, \mathsf{Loc}_1 \times \mathsf{Loc}_2, \mathsf{Init}, \mathsf{Lab}_1 \cup \mathsf{Lab}_2, \mathsf{Trans}, D, \mathsf{Inv} \rangle$ such that

- $\mathsf{Init}(l_1, l_2) = \mathsf{Init}_1(l_1) \cap \mathsf{Init}_2(l_2)$.
- $\langle (l_1, l_2), \gamma, a, \alpha, (l_1', l_2') \rangle \in \mathsf{Trans}$ iff
  1) $\langle l_1, \gamma_1, a_1, \alpha_1, l_1' \rangle \in \mathsf{Trans}_1$ and $\langle l_2, \gamma_2, a_2, \alpha_2, l_2' \rangle \in \mathsf{Trans}_2$;
  2) $\gamma = \gamma_1 \cap \gamma_2$.
  3) – either $a_1 = a_2 = a$,
     – or $a_2 = \epsilon$ and $a_1 = a \notin \mathsf{Lab}_1 \cap \mathsf{Lab}_2$,
     – or $a_1 = \epsilon$ and $a_2 = a \notin \mathsf{Lab}_1 \cap \mathsf{Lab}_2$;
  4) $\alpha = \alpha_1 \cap \alpha_2$.
- $D(l_1, l_2) = D_1(l_1) \cap D_2(l_2)$.
- $\mathsf{Inv}(l_1, l_2) = \mathsf{Inv}_1(l_1) \cap \mathsf{Inv}_2(l_2)$.

A state s of the LHA is of the form of (l, ν), where l is a location and ν is a valuation of Var. A state can change in two ways:

- A discrete step : $(l, \nu) \xrightarrow{\mathrm{a}} (l', \nu')$ which means that there exists an edge $\langle l, \gamma, a, \alpha, l' \rangle$ s.t.
$$\nu \models \gamma \wedge \nu' = \alpha(\nu) \wedge \nu' \models \mathsf{Inv}(l)$$
- A time step: $(l, \nu) \xrightarrow{\mathrm{t}} (l, \nu')$ where t is a real value representing time elapse, s.t.
$$\nu \models \mathsf{Inv}(l) \wedge \nu' \in \nu \uparrow^t_{D(l)} \wedge \nu' \models \mathsf{Inv}(l) \wedge t \geq 0$$
$\nu \uparrow^t_{D(l)}$ represents the set of valuations that can be reached by letting variables continuously evolve for t time units, according to derivatives constrained by D, and starting from the valuation ν.

Let d be a linear constraint on $\dot{\mathsf{Var}}$ and S be a set of valuations. We note $S \nearrow d$ the set of valuations that can be reached by starting from a valuation in S, subject to derivatives constraints in d. We are interested in all possible valuations in a location l, written R_l, that satisfies the following equation:

$$R_l = \Big(\big[\mathsf{Init}(l) \cup \bigcup_{(l', \gamma, a, \alpha, l)} \alpha(R_{l'} \cap \gamma) \cap \mathsf{Inv}(l) \big] \nearrow D(l) \Big) \cap \mathsf{Inv}(l) \quad (1)$$

VI. Periodic Server Model in LHA

In this section, we introduce a LHA for modelling the periodic server algorithm described in Section IV. In the model, we need to stop the clocks, since our servers and tasks can be preempted. Also, we need to use arbitrary linear constraints on clock variables. For convenience, we decided to rely on the more general model of LHA rather than restrict ourselves to TA with stopwatches [7].

If we want to precisely model a system of n applications $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, each one served by a server S_i with parameters (Q_i, P_i), we have to build:

- n automata, one per server;
- one automaton for modelling the global EDF scheduler;
- one automaton per task;
- and finally, one automaton per local FPFP scheduler.

The final system can be represented by the parallel composition of all such automata. However, this approach has two main inconveniences: first of all, it is specific for one single system, and it would be necessary to build a new model for each different system. Second, the resulting automaton is very complex even for a small number of applications and tasks (state-space explosion problem).

We assume that applications are independent of each other, thus we can analyse each application in isolation. It is important to underline that such assumption is basically the same used in avionics real-time applications that have been designed according to the IMA architecture: tasks belonging to different applications can only communicate through non-blocking communication primitives. Therefore, we can use appropriate abstractions to build the model of one single server: in particular, we will abstract away the presence of the other servers and the global scheduler.

We make a one-to-one correspondence between states of the algorithm and locations of the LHA. In particular, we use:

- one location for each *state* of the algorithm;
- two different time variables (also called *clocks*): variable x represents the consumed budget, whereas variable y represents the time passed since the beginning of the server period.

The Server linear hybrid automaton is depicted in Fig. 2. Idle is the initial location and $\mathsf{Init}(\mathsf{Idle}) = \{x = y = 0\}$. The application tasks served by this server are modelled with two synchronisation labels: **active** notifies a task's activation and **empty** means that the last task in the server has finished its execution.

If a task arrives when the Server automaton is in Idle, the model goes to location Active. The transition from Active to Executing happens when the global scheduler picks the server to execute. The reverse transition from Executing to Active models server preemption. Notice that these two transitions have no synchronisation labels because we want to abstract away the presence of other servers in the system and of the global scheduler.

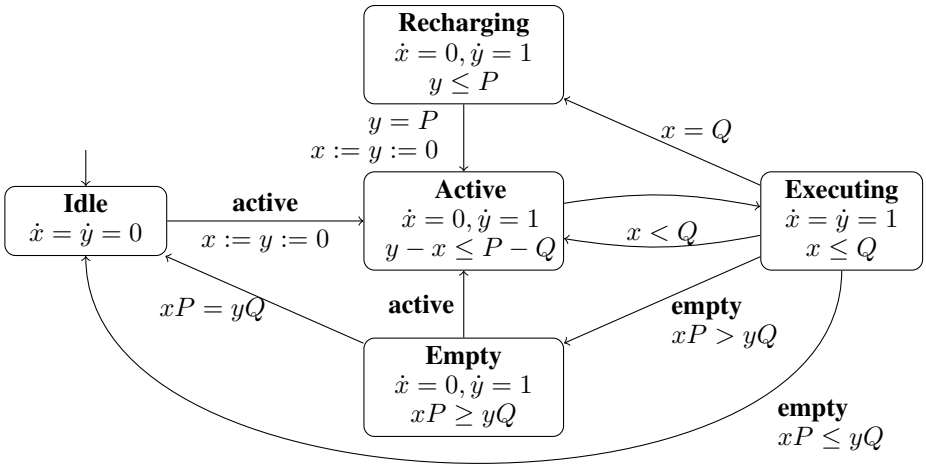

Fig. 2: The Server automaton.

Since we abstracted away the description of the global EDF scheduler and of the other servers, we need to add some constraints to guarantee that the model behaves correctly. We take for granted that Theorem 1 holds, and that therefore the server will meet all its scheduling deadlines. We impose such property by adding invariant $y - x \leq P - Q$ to location Active. This invariant states that, while in Active, there is still enough time to complete the execution of Q units before the end of the period. Therefore, no later than the time when $y - x = P - Q$ the automaton has to move to location Executing.

When the currently used budget reaches $x = Q$, the Server automaton moves from Executing to Recharging. It will stay in location Recharging until the start of a new period, at which point the current consumed budget is reset to $x := 0$. Location Empty models Rule 7 of the algorithm in Section IV: if the server tasks finish executing too early ($xP \leq yQ$), then the automaton directly moves to location Idle. If it is too late ($xP > yQ$), the automaton first moves to location Empty where it waits for the time y to reach the appropriate value before moving to Idle and resetting the model.

A. Proof of correctness

We now prove that the proposed Server automaton correctly models the server algorithm. In particular, we are going to prove that, under the assumption of Theorem 1, the automaton also respects Theorem 2.

Theorem 3. *Let Server be an automaton with parameters (Q, P) that models a dynamic periodic server, and let t_s be an instant in which the automaton moves from location Idle to location Active. Let t_f be the first instant after t_s such that the automaton enters again location Idle. Let $\Delta_{exe}(t_s, t_f)$ be the total amount of time that the server spends in location Executing in interval $[t_s, t_f]$. Then,*

$$\left\lfloor \frac{t_f - t_s}{P} \right\rfloor Q \leq \Delta_{\text{exe}} \leq \left\lceil \frac{t_f - t_s}{P} \right\rceil Q$$

The proof has been removed for space constraints, it can be found in [24].

VII. Schedulability Analysis

In this section, we use the Server automaton to perform the schedulability test of an application on a periodic server. Without loss of generality, we adapted the encoding used in [20] to show how to combine already existing schedulability model with our LHA model, in order to check if a task set in a server will miss its deadline.

A. Scheduler automaton

We now show how to encode an application with a FPFP local scheduler using an example with two tasks $\{\tau_1, \tau_2\}$. In Fig. 3 we show the Scheduler automaton that encodes the FPFP scheduler along with the task execution, and the two automata Arr_1 and Arr_2 that model the arrival times of the two tasks.

Let's start from the latter: each task arrival patterns is modelled with a timed automaton Arr_i with just two locations, and one clock r_i which is always increasing. The first transition from location Phase to Arrival models the first release time at the task offset; the second transition is a loop from Arrival to itself that models subsequent releases. It is easily possible to model different arrival patterns by simple changing the corresponding arrival automaton. Without loss of generality, in this paper we assume periodic tasks with offset.

In the Scheduler automaton, we use two kinds of clock variables: *executing variables*, such as c_1 and c_2, for recording a task's accumulating execution time; and *deadline variables* (d_1 and d_2) for tracking if a task misses its deadline. The synchronisation label **empty** and **active** are the same as the in Server automaton. We accept there exist more than one synchronisation label on a transition in Scheduler automaton. Take the transition from **Idle** to τ_1 **running**, which has two labels τ_1 **release** and **active** on it, as an example. In order to trigger this transition, the Scheduler should first synchronise with Arr_1 through τ_1 **release**, then synchronise with Server through **active**. This can be conveniently implemented by inserting an urgent location [25], where no time elapse is allowed, and decompose the transition into two.

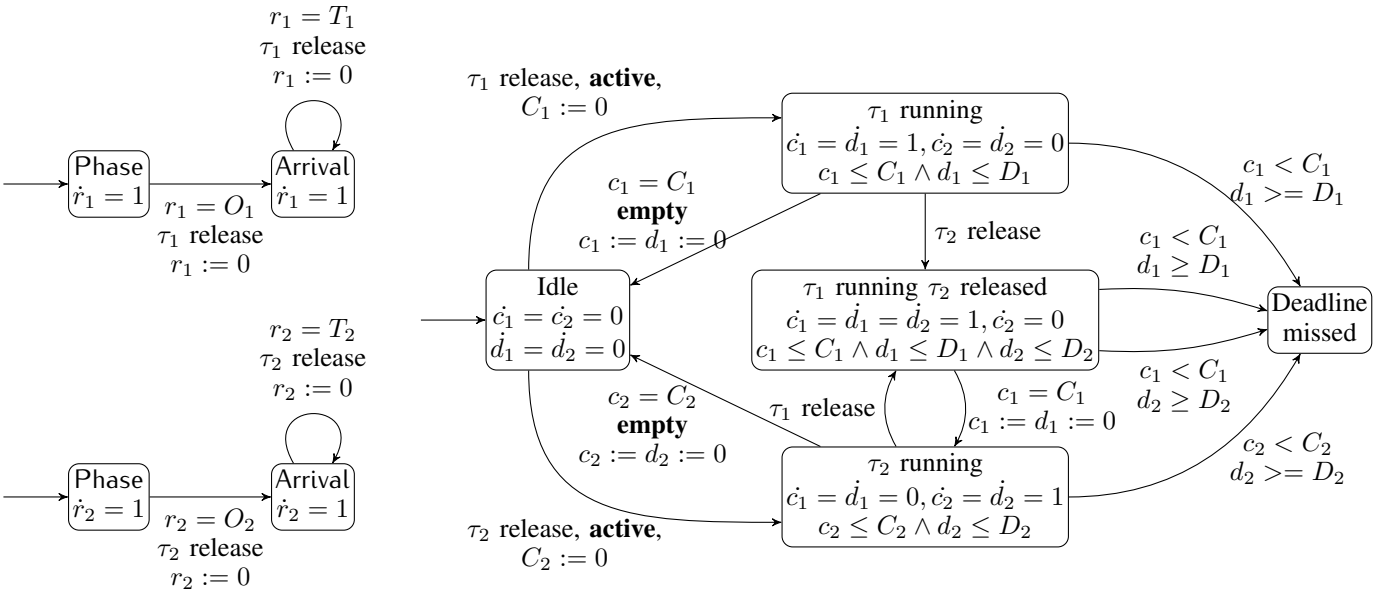

Fig. 3: Model of a FPFP scheduler for two periodic tasks τ_1, τ_2.

Each location in the Scheduler automaton models a different state of the ready queue of the scheduler. Location Idle models an empty ready queue; location "τ_1 running" models the case in which only task τ_1 is active and running; location "τ_1 running, τ_2 released" models the case in which the ready queue contains both τ_1 and τ_2, but τ_1 is running because it has the highest priority. Location "τ_2 running" models the case in which only τ_2 is active and running. Finally, location Deadline Missed models the case in which one of the two tasks misses its deadline. Schedulability can be checked by performing a reachability analysis for location Deadline Missed.

Figure 3 only models the schedule of two tasks. Generating the model for n tasks can be done automatically by generating all possible 2^n configurations of the ready queue. This means that the size of the model is exponential in the number of tasks in the application. However, consider that in most practical cases, the number of tasks inside one application is limited to a few units. Also, component-based analysis abstracts away the rest of the system and hence it is much less complex than analysing the entire system as a whole, as shown in [13].

B. Hierarchical composition

The automata of Fig. 3 models an application consisting of two tasks running on a single processor. We now describe how to compose such model with the LHA model of the server presented in Section VI.

Definition 2. *A **Hierarchical Scheduling Composition** of a task set with a periodic server is defined as the parallel composition of the server automaton Server, the scheduler automaton Scheduler and the task arrival automata* $\mathsf{Arr}_1, \ldots, \mathsf{Arr}_n$*, as defined in Section V:*

$$\mathsf{HSC} = \mathsf{Server} \times \mathsf{Scheduler} \times \mathsf{Arr}_1 \times \ldots \times \mathsf{Arr}_n$$

with the following additional rule:

- *Let* $l \in \mathsf{HSC}$ *be a location of the composed automaton, with* $l = (l_{Ser}, l_{Sched}, l_1, \ldots, l_n)$*, and* $l_{Ser} \in \mathsf{Server}$*,* $l_{Sched} \in \mathsf{Scheduler}$*, and* $l_i \in \mathsf{Arr}_i$*, for all* $i = 1, \ldots, n$*. If* $l_{Ser} \neq \mathsf{Executing}$*, then the derivatives of all execution time variables are set to 0:* $\dot{c}_i = 0$*, for all* $i = 1, \ldots, n$*.*

C. Decidability

Once again, schedulability analysis can be encoded as a reachability analysis over automaton HSC. We now prove that such analysis is decidable for the case of independent periodic tasks.

Lemma 1. *Given a HSC automaton that models a set of periodic tasks* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *executing in a periodic server. The task set is schedulable if, and only if, location Deadline Missed is unreachable in a time interval equal to* $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$*, where* T_i *and* O_i *is the period and the initial offset of task* τ_i*.*

Proof: The proof uses a well-known result by Leung and Whitehead [26]: "A set of periodic tasks with deadlines less than or equal to the periods is schedulable if, and only if, there is no deadline miss in the interval $[0, 2H + \max\{O_i\}]$", where H denotes the hyper period—the least common multiple (lcm) of all tasks' periods. Intuitively, the reason is that the arrival pattern of a set of periodic tasks will repeat every hyperperiod after an initial transitory $2H + \max\{O_i\}$, so the schedule is periodic, and it is sufficient to check in the first periodic instance of the schedule.

Now we also take periodic server into account. We are interested in finding a similar period such that not only task arrival pattern repeat themselves, but also all possible server behaviours will be identical. For finding the upper bound of the length of such an interval, we regard the server as a periodic task with period P, worst-case execution time Q and initial

offset 0. Then we apply Leung and Whitehead test on the extended task set including original task set and the server. Thus, to check if a task misses its deadline in a server, it is sufficient to check all possible paths in the time interval $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$. Notice that, thanks to non-determinism, the HSC automaton already models all possible generated schedules for the servers, therefore it covers all possible scheduling behaviours in the considered interval. ■

From Lemma 1, it follows that our problem can be expressed as a problem of *reachability in bounded time*, in particular, we need to analyse the system and see if a Deadline Miss state is reachable in $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$.

Reachability analysis in bounded time has been proved to be decidable for a particular sub-class of LHA called *Rectangular Automata* [27]. Unfortunately, the HSC automaton does not fall in this class, since the server automaton in Fig. 2 uses diagonal constraints as invariant in locations Empty and Active and as guard in the transitions between Empty and Idle, and between Executing and Idle.

In a nutshell, the problem is that in a LHA there can be an unbounded number of transitions in a finite interval of time. This effect is sometimes referred to as *Zeno effect* [7], as the distance between any two transitions can be made arbitrarily small, hence bounded time does not imply runs of bounded lengths.

Using techniques from [27], it is possible to prove that reachability analysis can be performed on an HSC by only exploring paths of finite and bounded length, hence with a terminating algorithm. Due to space constraints, the complete proof is available in a technical report [24].

VIII. Evaluation

We have implemented the HSC analysis in the the software tool FOrmal Real-Time Scheduler (FORTS) [28]. FORTS is a model checker targeting real-time scheduling problem and it accepts LHA model as input. We use it here for reachability analysis in HSC.

A. Comparison with the Lipari-Bini test

We used the tool to compare the results of our analysis against the test proposed by Lipari and Bini in [6]. We modelled the same application described and analysed in [6]. In Table I we report the parameters of the task set, which has a total utilisation of 47%.

Task	O_i	C_i	D_i	T_i	p_i
τ_1	0	2	8	8	1
τ_2	0	2	20	20	2
τ_3	0	6	50	50	3

TABLE I: Parameters of the example application.

In this experiment, we checked the schedulability of the task set in servers with different values of (Q, P). In particular, we tested all integer values of $P \in [1, 27]$, and all values of $Q \in [1, P]$. The results are shown in Fig. 4: the crosses are the results of the Lipari-Bini test, whereas the green triangles

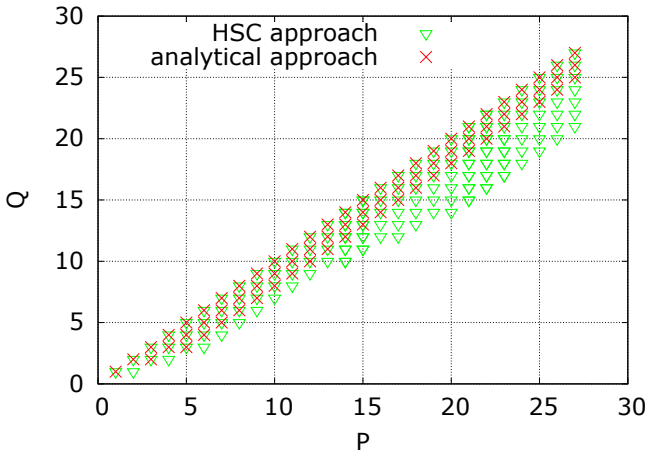

Fig. 4: Feasible server parameters. Crosses are the schedulable pairs (Q, P) found by the Lipari-Bini test [6]; triangles are the ones found by our analysis.

are the results of the HSC model. As you can see, the latter found many more schedulable points. In particular, point $(Q = 3, P = 6)$, which leads to an utilisation of 50% (only 3% larger than the task set utilisation) was found by the HSC, while it was not found by the analytic model.

The reason for this difference is that the Lipari-Bini test makes worst-case assumptions on the maximum delay that an application task can experience. In particular, this test assumes that, when the highest priority task is activated, it may have to wait up to $2(P - Q)$. Since the highest priority task has computation time $C_1 = 2$ and relative deadline $D_1 = 8$, it necessarily follows that $(P - Q) \leq 3$. In Fig. 4, it is possible to note that this is always true for the Lipari-Bini test. However, the worst case initial delay may never happen: the HSC model shows that in many cases $(P - Q) > 3$, and for large Ps this can be as large as 6. How is it possible?

To understand what happens, consider the case of $P = 22$ and $Q = 16$. According to the analytic method, the worst case happens when the highest priority task arrives and, at the same instant, the server has just exhausted its budget. Apparently, this seems to be the case of time $t = 16$, when the server budget $Q = 16$ has just been exhausted. However, notice that the first busy period starting at time $t = 0$ lasts only for 12 units: therefore, at time 12 the server moves to location Empty, and from there, it moves to location Active at time $t = 16$ (due to the arrival of τ_1), where it can spend at most two units of time before moving to location Executing and completing the requested $C_1 = 2$ units of execution time. In other words, it never happens that the application can completely deplete the budget of the server. This fact is not taken into consideration by the analytic method, which then produces pessimistic results.

B. External service test

As discussed in the previous sections, the LHA formalism has the advantage of being more expressive, in the sense that it allows designers to model and analyse complex scheduling scenarios that cannot be expressed easily as a set of independent periodic real-time tasks.

To demonstrate the expressiveness of the model, consider an application consisting of just two real-time periodic tasks, whose parameters are reported in Table II. Each task provides a service that is requested by the external environment (i.e. by other applications, or by an interrupt). Each task τ_i has an incoming queue of requests: at its periodic activation time it checks the contents of the queue: if there is no request, it executes for very little time C_i'; if there is one request, it will execute for C_i''; if there are two or more requests it will execute for C_i'''. Therefore, the actual load generated by the application depends on the number of external requests per task.

Task	C_i'	C_i''	C_i'''	D_i	T_i	p_i
τ_1	1	2	3	12	12	1
τ_2	1	3	5	15	15	2

TABLE II: Parameters of the external service application.

In our example, we model the two request queues with simple counters w_1, w_2, both initialised to be 0. At its arrival time, each task reads its counter, sets its computation time to the corresponding value, and resets the counter to 0. In the LHA formalism, a discrete variable like w_i can be automatically encoded in the location signature.

The arrival of external requests is modelled by the Service automaton shown in Fig. 5. Initially, the automaton waits non deterministically for an interval of time between 0 and its maximum initial offset O_r. Then, every T_r units of time, it produces one request for either τ_1 or τ_2, and the choice is again non deterministic.

It is not easy to compute the worst-case load produced by the application: if we want to use classical schedulability analysis, we need to analyse all possible combinations of requests to the two tasks. In fact, the Service automaton can request only one service at time, and depending on the values of T_r, several possible combinations of service requests may generate the worst-case load.

However, our HSC automaton does exactly this: it checks all possible combinations of service requests, and verifies if the system is schedulable under all possible cases. By setting $T_r = 10$ and $O_r = 0$ and applying the analysis for different values of the pairs (Q, P), we obtained the results shown in Fig. 6. Notice that the worst-case utilisation of the task set, without considering the Service automaton, is

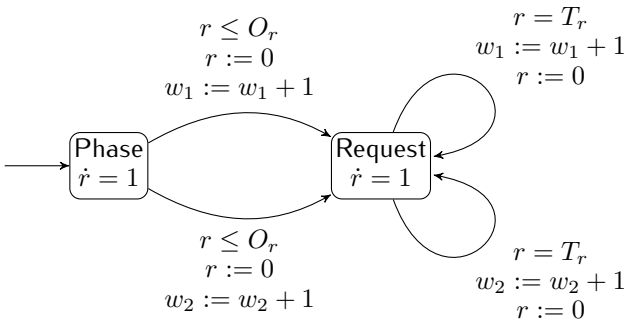

Fig. 5: The service request automaton

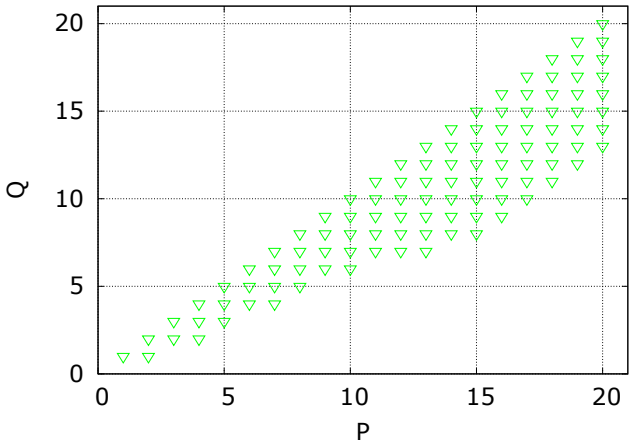

Fig. 6: Feasible server parameters for external service test

$\frac{C_1'''}{T_1} + \frac{C_2'''}{T_2} = 58.333\%$. However, the minimum fraction $\frac{Q}{P}$ found by our analysis is 50%, corresponding to the two pairs $(Q = 1, P = 2)$ and $(Q = 2, P = 4)$. In fact, by analysing all possible combination of service requests, we found that the largest utilisation needed is indeed 50%, exactly equal to what our analysis found. This value corresponds to the case when the first task τ_1 executes for $C_1'' = 2$ (serving one request), whereas the second task τ_2 executes for $C_2''' = 5$ (serving two requests).

Also, notice that the pair $(Q = 8, P = 15)$ provides schedulability with server utilisation equal to 53.34% and a relatively large P. In general, a large P is desirable because it reduces the overhead of switching between different applications. In this case, our analysis shows that we can set a period larger than the smaller period in the application, and still achieve a relatively low resource utilisation.

C. A real case study of an avionics system

The case study we use here was originally described in [29] and [30]; it was later adapted to hierarchical scheduling in [14]. It consist of fifteen real-time tasks with very different values of periods. Carnevali *et al.* [14] partitioned the task set into five components and verified the schedulability of each partition under TDM with a pre-defined pattern of time slot assignment. The tasks and the components are reported in Table III, where time is expressed in milliseconds.

In our analysis, we used the same components as in [14]. We modelled each partition using a HSC automaton. Then, we performed the analysis of the entire system in two steps: we first analysed each component individually. The servers' parameters with minimal utilisation for each component are listed in the fourth column of Table IV.

In the second step, we performed the "integration" by selecting the combination of pairs (P, Q) for each component so that the overall utilisation is less than 100%. Notice that, by using dynamic periodic servers, we can easily select different values for the periods and the budget of the different applications so to minimise some cost function. For example, one objective could be to maximise the values of the servers

Component	Task	O	C	T	D	p	U
$\mathcal{A}_1$	τ_{11}	0	1	10	5	1	0.2
	τ_{12}	0	1	40	40	2	
	τ_{13}	10	2	40	40	3	
	τ_{14}	20	1	40	40	4	
$\mathcal{A}_2$	τ_{21}	0	1	40	40	1	0.305
	τ_{22}	0	5	50	50	2	
	τ_{23}	10	4	50	50	3	
	τ_{24}	16	5	50	50	4	
$\mathcal{A}_3$	τ_{31}	2	4	80	80	1	0.06
	τ_{32}	15	1	100	100	2	
$\mathcal{A}_4$	τ_{41}	0	4	100	100	1	0.045
	τ_{42}	10	1	200	200	2	
$\mathcal{A}_5$	τ_{51}	10	1	200	200	1	0.019
	τ_{52}	3	4	400	400	2	
	τ_{53}	0	4	1000	1000	3	

TABLE III: Specification of the avionics case study

	Orig. Util.	Min Utilisation		Reduced Overhead	
$\mathcal{A}_1$	0.2	(5,1)	$Q/P = 0.2$	(5,1)	$Q/P = 0.2$
$\mathcal{A}_2$	0.305	(50,16)	$Q/P = 0.32$	(50,16)	$Q/P = 0.32$
$\mathcal{A}_3$	0.06	(80,5)	$Q/P = 0.0625$	(100,24)	$Q/P = 0.24$
$\mathcal{A}_4$	0.045	(22,1)	$Q/P = 0.0455$	(100,5)	$Q/P = 0.05$
$\mathcal{A}_5$	0.019	(100,2)	$Q/P = 0.02$	(200,9)	$Q/P = 0.045$
U_{tot}	0.629		0.648		0.855

TABLE IV: Server parameters for the avionics case study.

periods: in fact, small periods imply a more frequent switch between components, and hence a greater overhead. One possible choice for each parameters is reported in the fifth and last column of Table IV.

D. Scalability of the analysis

A full-fledged analysis of the run-time complexity of our model is out of the scope of this paper. Nevertheless, it is important to briefly discuss the scalability of our analysis with respect to the size of the model. First of all, a few *caveat*. It is well-known that formal methods suffer from the so called *state-space explosion* problem: the number of states to analyse is exponential in the size of the input. Therefore, an exponential dependency from the number of tasks in the application is unavoidable. The important issue here is to understand to which extent the analysis is still doable with modern computer systems.

The experiment was conducted on a common MacBook with Intel(R) Core(TM) i5 CPU @ 2.5GHz and 8 GB of RAM. We run tests for different task sets and different values of the parameters. Each test is specified by a tuple (N, U, P) , where $N \in \{8, 10\}$ is the task set size, U is task set utilisation, and $P \in \{20, 40, 60\}$ is the period of a server.

N	8			10		
P	20	40	60	20	40	60
max time (s)	209	443	483	938	2120	2244
ave time (s)	63	85	104	245	371	384
max memory (M)	79	145	178	229	444	527
ave memory (M)	26	35	44	69	107	126

TABLE V: Run-time results of HSC

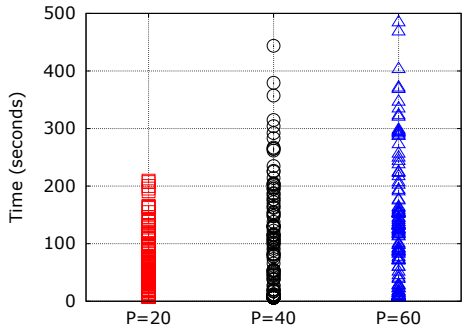

Fig. 7: Run-time of FORTS on a 8 task model

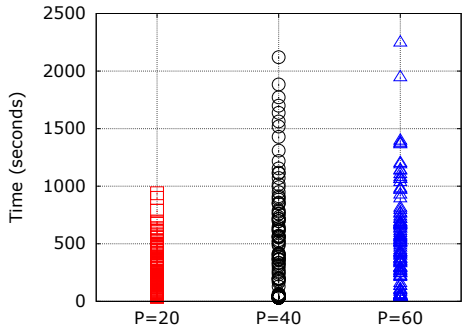

Fig. 8: Run-time of FORTS on a 10 task model

We also fix $\frac{Q}{P} = 0.6$. For each pair (N, P) 150 task sets would be randomly generated according to the Randfixedsum algorithm [31]. For a task set, its utilization U is randomly sampled in the range $[0.2, 0.6]$. Task periods are selected in the range $[10, 100]$ using log-uniform sampling and the minimum granularity of periods is 10. Each task's initial phase is uniformly distributed between 0 and its period. We know that the complexity of reachability analysis in a HSC is directly related to the hyperperiod length of tasks and server. We constrain this hyperperiod length to be less than 2000. Task priorities are assigned by Rate Monotonic scheduling: a task with shorter period will be given higher priority; the priority relation between two tasks with the same period is randomly selected. Furthermore, we avoid the generation of task τ_i, with $D_i - C_i < P - Q$, which will trivially miss their deadline.

For each task set, we measured the time (in seconds) and memory (in MB) needed to decide schedulability. The run-time results are reported in Fig. 7 and Fig. 8; more detailed statistical results are in Table V. A first observation is that larger P will result in higher cost of HSC analysis, due to the non-determinism in the Server automaton. And we get the worst-case scenario when $P = 60$ and $N = 10$, under which circumstance the longest running time for one HSC is around 37 minutes, whereas the memory cost is always less than 530 MB. We believe that, by introducing parallelism in the analysis

tool and by carefully optimising the code we could achieve higher performance.

IX. Conclusions

We presented a formal model of a dynamic server algorithm for hierarchical scheduling that can be used for component-based analysis of hierarchical real-time systems. The model is based on the very expressive formalism of Linear Hybrid Automata. We have shown that the model provides more precise results than classical analytic schedulability formulas, and allows to model components with complex dependencies. We have run extensive simulation to demonstrate that the model can be analysed efficiently for components with 10 real-time periodic tasks.

The proposed model is very general but it does not account yet for the overhead of context switch between components. Also, the impact of memory access and caches on the execution time of the tasks has been neglected. We are currently working on a more accurate model that can account for the scheduling overhead and the cache-related preemption delay caused by other components in the system.

References

[1] C. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," in *Proc. 26th Digital Avionics Systems Conference (DASC'07)*, pp. 2.A.1–1–2.A.1–10, 2007.

[2] ARINC, *ARINC 651: Design Guidance for Integrated Modular Avionics*. Airlines Electronic Engineering Committee (AEEC), November 1991.

[3] ARINC, *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*. Airlines Electronic Engineering Committee (AEEC), June 1996.

[4] X. Feng and A. K. Mok, "A Model of Hierarchical Real-Time Virtual Resources," in *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, (Austin, TX USA), pp. 26–35, Dec. 2002.

[5] I. Shih and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," in *Proc. 24th IEEE Real-Time Systems Symposium (RTSS'03)*, (Cancun, Mexico), pp. 2–13, Dec. 2003.

[6] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *J. Embedded Computing*, vol. 1, no. 2, pp. 257–269, 2005.

[7] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[8] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*, vol. 736 of *LNCS*. Springer, 1993.

[9] Y. Abdeddaïm, E. Asarin, and O. Maler, "Scheduling with timed automata," *Theoretical Computer Science*, vol. 354, no. 2, pp. 272–300, 2006.

[10] M. Åsberg, P. Pettersson, and T. Nolte, "Modelling, Verification and Synthesis of Two-Tier Hierarchical Fixed-Priority Preemptive Scheduling," in *Proc. 23rd Euromicro Conference on Real-Time Systems (ECRTS'11)*, 2011.

[11] É. André, L. Fribourg, U. Kühne, and R. Soulat, "IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems," in *Proc. 18th International Symposium Formal Methods (FM'12)* (D. Giannakopoulou and D. Méry, eds.), vol. 7436 of *LNCS*, (Paris, France), pp. 33–36, Springer, Aug. 2012.

[12] L. Fribourg, D. Lesens, P. Moro, and R. Soulat, "Robustness Analysis for Scheduling Problems using the Inverse Method," in *Proc. 19th International Symposium Temporal Representation and Reasoning (TIME'12)* (M. Reynolds, P. Terenziani, and B. Moszkowski, eds.), (Leicester, UK), pp. 73–80, IEEE Comp. Soc. Press, Sept. 2012.

[13] L. Carnevali, G. Lipari, A. Pinzuti, and E. Vicario, "A Formal Approach to Design and Verification of Two-Level Hierarchical Scheduling Systems," in *Proc. 16th Ada-Europe International Conference Reliable Software Technologies (Ada-Europe'11)* (A. Romanovsky and T. Vardanega, eds.), vol. 6652 of *LNCS*, pp. 118–131, Springer, 2011.

[14] L. Carnevali, A. Pinzuti, and E. Vicario, "Compositional Verification for Hierarchical Scheduling of Real-Time Systems," *IEEE Trans. on Software Engineering*, vol. 39, no. 5, pp. 638–657, 2013.

[15] R. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proc. 26th IEEE Real-Time Systems Symposium (RTSS'05)*, pp. 10 pp.–398, 2005.

[16] C. Norström, A. Wall, and W. Yi, "Timed Automata as Task Models for Event-Driven Systems," in *Proc. 6th International Workshop Real-Time Computing Systems and Applications (RTCSA'99)*, IEEE Comp. Soc. Press, 1998.

[17] E. Fersman, P. Pettersson, and W. Yi, "Timed Automata with Asynchronous Processes: Schedulability and Decidability," in *Proc. 8th International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, vol. 2280 of *LNCS*, pp. 67–82, Springer, 2002.

[18] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability Analysis Using Two Clocks," in *Proc. 9th International Conference ools and Algorithms for the Construction and Analysis of Systems (TACAS'03)* (H. Garavel and J. Hatcliff, eds.), vol. 2619 of *LNCS*, pp. 224–239, Springer, 2003.

[19] Y. Abdeddaïm and D. Masson, "The scheduling problem of self-suspending periodic real-time tasks," in *Proc. 20th International Conference Real-Time and Network Systems (RTNS'12)*, (New York, NY, USA), pp. 211–220, ACM, 2012.

[20] Y. Sun, R. Soulat, É. Lipari, Giuseppe André, and L. Fribourg, "Parametric schedulability analysis of fixed priority real-time distributed systems," in *Second International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13)* (C. Artho and P. Ölveczky, eds.), vol. 419 of *Communications in Computer and Information Science*, Springer, October 2013. To appear.

[21] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, pp. 46–61, Jan. 1973.

[22] L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," in *Proc. 19th IEEE Real-Time Systems Symposium (RTSS'98)*, (Madrid, Spain), Dec. 1998.

[23] N. Halbwachs, Y.-E. Proy, and P. Roumanoff, "Verification of Real-Time Systems using Linear Relation Analysis," *Formal Methods in System Design*, vol. 11, no. 2, pp. 157–185, 1997.

[24] Y. Sun, G. Lipari, R. Soulat, L. Fribourg, and N. Markey, "A linear hybrid automata model for component-based analysis of hierarchical scheduling systems," Research Report LSV-13-14, Laboratoire Spécification et Vérification, ENS Cachan, France, Oct. 2013. 25 pages.

[25] G. Behrmann, R. David, and K. G. Larsen, "A tutorial on uppaal," pp. 200–236, Springer, 2004.

[26] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[27] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J. Raskin, and J. Worrell, "On Reachability for Hybrid Automata over Bounded Time," in *Proc. 38th International Colloquium on Automata, Languages and Programming (ICALP'11)*, vol. 6756 of *LNCS*, pp. 416–427, Springer, 2011.

[28] Y. Sun and G. Lipari, "FOrmal Real-Time Scheduler (FORTS)." Web page: https://github.com/glipari/forts, January 2014.

[29] R. Dodd, "An Analysis of Task-Scheduling for a Generic Avionics Mission Computer," tech. rep., DTIC Document, 2006.

[30] R. Dodd, "Coloured petri net modelling of a generic avionics mission computer.," tech. rep., DTIC Document, 2006.

[31] P. Emberson, R. Stafford, and R. I. Davis, "Techniques For The Synthesis Of Multiprocessor Tasksets," in *Proc. 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*, (Brussels, Belgium), pp. 6–11, July 2010.