Real-Time Scheduling of Energy Harvesting Embedded Systems with Timed Automata

Yasmina Abdeddaïm, Damien Masson

Real-Time Scheduling of Energy Harvesting Embedded Systems with Timed Automata

Yasmina Abdeddaïm and Damien Masson
Université Paris-Est,
LIGM UMR CNRS 8049,
Département Systèmes Embarqués, ESIEE Paris,
2 bld Blaise Pascal, BP 99, 93162 Noisy-le-Grand CEDEX,
France
Email: {y.abdeddaim/d.masson}@esiee.fr

June 14, 2012

Contents

Abstract

In this paper, we propose feasibility and schedulability tests for a real-time scheduling problem under energy constraints. We first introduce the problem and show how to model it using timed automata. We then propose a feasibility test based on CTL model checking and schedulability tests for EDF and Preemptive Fixed Priority algorithms (PFP). Our approach also permits to generate a feasible schedule if one exists or otherwise to find how to correct battery characteristics to make the problem feasible. It is finally possible to generate schedules that optimize some criteria, such as the number of context switches between the battery recharging and discharging modes, the minimal and the maximal energy levels reached during the execution, or the number of preemptions. The approach is illustrated by some experiments using the model checking tool Uppaal [1].

1 Introduction

In this work, we investigate a real-time system model for embedded systems that collect and store energy from their environment. Such systems are composed, in addition to traditional embedded system components, by an energy collector unit (e.g. a solar panel) and by an energy storage unit (a battery or a capacitor).

One common hypothesis in real-time system theory is to consider that the CPU is always available to execute real-time tasks, whereas in the studied systems, known as energy harvesting systems, the CPU has to be switched off at some points in order to permit to recharge the energy storage unit. These harvesting embedded systems are more and more present in our lives: sensor networks in structures such as bridges that collect vibration energy, medical implants that collect energy from the human body, mobile or fix devices with solar panel or windmill etc. Despite their energy supply particularity, some of these systems need to satisfy strict timing constraints. Their particularity is that the energy resource is not limited, but the energy available at a given instant is. The energy harvesting and storage process takes time. Therefore it is important to consider both the time and energy needs of a task in other to schedule it, since both the energy and CPU time resources of the system have to be shared among the tasks.

From a scheduling point of view, the time intervals needed for the energy scavenging will result in inserting gaps in the schedule. So, an energy-aware scheduler will not be a work-conserving one. Assuming this, traditional feasibility analysis algorithms are no longer relevant. Moreover, we can easily show that commonly known optimal scheduling policies[1], such as Earliest Deadline First (EDF), Rate Monotonic (RM) or Deadline Monotonic (DM) are no more optimal for these systems.

Another particularity of the studied embedded systems is that they often need to be as cheap as possible (e.g. for networked sensors widely dispersed in an area, some of them will stay unused), as tiny as possible, and as light as possible. The size of the energy storage unit must so be minimized. So, the goals of a real-time scheduler for energy harvesting systems will not be only to warranty timing constraints, but also to take account of these systems

[1] in the sense that they produce a non feasible schedule iff it does not exist any algorithm in the same class that can correctly schedule the system.

specificities to minimize the energy management overheads. For example, it is known that energy storage processes have not linear rates. For certain devices, the less the level of energy is, the faster the charging process will be. Hence, it can be important to try not to let the energy level be too high if there is no need to respect timing constraints. On the contrary, a too low energy level can damage some devices. Trying not to let the energy level get too low when it is not needed can also be important. Another example is that depending on the battery technology, it can be of importance to let the battery have complete cycles as long as it is possible regarding the timing constraints.

This work investigates several open problems related to the scheduling of such harvesting systems:

- providing a feasibility test,
- providing a schedulability test under Preemptive Fixed Priority (PFP) and Earliest Deadline First (EDF),
- find the minimal energy collector size that permits the system to be feasible,
- find a schedule that optimize the energy consumption profile on several criteria, such as the minimal and maximal authorized energy levels, or the number of context switches between the battery recharging mode and discharging one.

Doing so, three hypotheses are made. First, the tasks' energy consumption is not related to their execution time [2]. Second, the energy consumption profile of the tasks are not known and the worst case is so assumed: all the energy budget of a task is considered used as soon as the task has begun its execution. Third, it is assumed that the energy is collected and stored linearly. Moreover, the set of solutions for the feasibility problem is restricted to fixed priority at job level schedules where no idle times are allowed when a task is under preemption, except if these idle times correspond to a battery recharge. Even in this case, the recharge is only possible at the beginning of a preempting task. In a fixed priority at job level schedule, when the relative priority assignment between two jobs has been decided, it cannot change. EDF is an example of fixed priority at job level scheduling algorithm. Least Laxity First (LLF) is a well known counter example.

We review the related works in Section 2. Then Section 3 formalizes the problem. Section 4 introduces the timed automaton model. Section 5 exposes how to check the feasibility and the schedulability with both PFP and EDF and how to generate a scheduler. Section 6 presents experiments and finally we conclude in Section 7.

2 Related work

Even if energy issues are more and more popular in real-time systems theory, most of the research to date has concentrated on reducing the power consumption. Mainly previous efforts have focused on predictive shutdown techniques [3] and varying speeds of processors [4, 5, 6, 7].

To our knowledge, the first paper to address the problem of harvesting systems, is [8]. However, the task model considered in this paper is the frame-based model: all tasks have the same release and the same period and deadlines.

In [9], the LSA algorithm is proposed. The context of this algorithm is a little bit different then the one we address. The authors consider tasks for which the execution time will depend on the energy given to them. Then they propose algorithms that optimally assign power to arriving tasks in order to minimize the battery size while guaranteeing temporal constraints. In that work, a task energy consumption is related to its execution time, that is not a realistic hypothesis. Indeed, in practice, the total energy which can be consumed by a task is not related to its worst case execution time, as stated in [2].

The first work considering task models where energy consumptions are not linked to CPU demand are the ones of Chetto [10, 11], for dynamic priority systems. Some heuristics for fixed priority systems are also considered in [12]. To our knowledge, the existence of feasibility or schedulability tests, and the existence of an optimal scheduling algorithm for these systems are open problems. We address in this paper this problem using a timed automata approach.

The timed automata approach has been already used in the literature to model and solve some scheduling problems. In [13, 14], the approach has been used to solve the job shop scheduling problem. The goal was to find optimal schedules in the sense of minimal execution time. Then in [15], the authors present a model based on timed automata to solve real-time scheduling problems. However, this model does not consider the tasks' energy consumption. The principal benefits of the timed automata approach is first that it proposes a model for both the scheduling and the formal verification of the system, and second that it manages to handle open problems, where no results are currently known. For example we used this approach to address the scheduling problem of self-suspending tasks in [16].

3 The problem statement

We define our real-time problem as a pair $\mathcal{P} = (\Sigma, \mathcal{B})$ where $\Sigma = \{\tau_1, \dots, \tau_n\}$ is a set of real-time tasks and $\mathcal{B}$ a battery.

A real-time task is a tuple $\tau_i = (r_i, C_i, T_i, D_i, e_i)$ where r_i is the release time of the task, C_i the execution time, T_i the period, D_i the relative deadline ($D_i \leq T_i$) and $e_i \geq 0$ the energy consumption rate of the task per time unit. An active task can be started iff there is enough energy in the battery to execute it completely. We suppose that the energy consumption profile of the task is unknown. We so assume the most unfavorable case and consider that the whole energy budget of a task is consumed as soon as the task has begun its execution.

A battery is defined as a tuple $\mathcal{B}(E_{max}, e_{bat})$ where E_{max} is its maximal capacity and e_{bat} its charge rate, i.e. the number of energy units it collects per time unit. For the sake of clarity, we consider in the examples the scenario where the battery is full before the first task release. However our model works whatever the energy initial state level is. If a task is executing, the battery is in a *consuming mode*, i.e. a mode where the battery is not charged. A task τ_i can be executed completely if the battery energy level is greater than $e_i \times C_i$, the total consumption of the task. If the processor is idle, the battery can move to the charging mode s.t.: if the level of energy is less than its maximal capacity

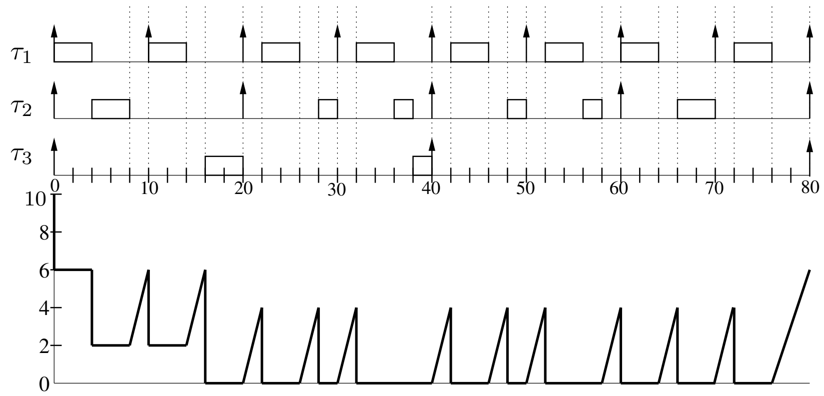

Figure 1: EDF_{asap}: Battery $\mathcal{B}(10, 2)$ and $e_1 = e_2 = e_3 = 1$

E_{max} for a duration t, the energy is augmented by $e_{bat} \times t$ otherwise the level of energy does not increase.

A schedule for $\mathcal{P} = (\Sigma, \mathcal{B})$ is a sequence of tasks produced by an algorithm that at each time:

1. assigns the processor to an instance of a pending task, or

2. lets the processor idle and does not charge the battery, or

3. lets the processor idle and charges the battery.

The real-time problem $\mathcal{P} = (\Sigma, \mathcal{B})$ is schedulable iff there exits a feasible schedule: a schedule where no task misses its deadline and where the battery energy level is always included in $[0, E_{max}]$.

Illustrative example

Let $\mathcal{P}_1$ be a real-time problem defined by a battery $\mathcal{B}_1(10, 2)$ and a set $\Sigma_1 = \{\tau_1, \tau_2, \tau_3\}$ of real-time tasks where, $\tau_1 = (0, 4, 10, 10, 1)$, $\tau_2 = (0, 4, 20, 20, 1)$ and $\tau_3 = (0, 6, 40, 40, 1)$. Note that if we relax the constraints on energy consumption of the tasks ($\forall i, e_i = 0$), this problem is schedulable using both EDF and RM priority driven scheduling algorithms.

We first apply a naive non work conserving algorithm to this problem. In this algorithm, if no more energy is available to execute new instances of tasks, the battery is charged until there is enough energy to execute the next highest priority task according to EDF. Thus, the tasks are executed as soon as possible according to the EDF policy. We call this algorithm "as soon as possible EDF" and note it EDF_{asap}.

The EDF_{asap} schedule of the illustrative example is represented in Figure 1 for the interval $[0, 80]$. At the beginning of the execution, the battery is completely charged and the tasks τ_1 and τ_2 can be executed. At $t = 8$, the battery energy level is equal to 2, then the task τ_3 which consumes 6 units of energy cannot be executed. Thus, the processor is idle and the battery is in the charging mode until $t = 10$. Then at $t = 10$ there is enough energy to execute a new instance of task τ_1 and so on. Note that at $t = 36$ and $t = 38$, the tasks

τ_2 and τ_3 are executed even if the battery is empty. Indeed, they have already consumed the necessary energy at the beginning of their execution. At $t = 40$, all the tasks are active and none of them has missed any deadline. However, at $t = 80$, the task τ_3 misses its deadline. Thus, this problem is not schedulable using EDF_{asap}.

4 The Modeling step

4.1 Timed Automata

A Timed automaton [17] is a model extending the classical automaton model with a set of variables, called clocks. Clocks are real variables evolving continuously and synchronously with time. Thanks to these variables, it is possible to express constraints over delays between transitions. Indeed, each transition of a timed automaton can be labeled by a clock constraint called guard which controls the firing of a transition. Clocks can be reset to zero in a transition and each location is constrained by a staying condition called invariant.

Formally, let $\mathcal{X}$ be a set of real variables called *clocks* and $\mathcal{C}(\mathcal{X})$ the set of clock constraints ϕ over $\mathcal{X}$ generated by $\phi ::= x\sharp c \mid x - y\sharp c \mid \phi \wedge \phi$ where $c \in I\!N$, $x, y \in \mathcal{X}$, and $\sharp \in \{<, \leq, \geq, >\}$. A *clock valuation* is a function $v : \mathcal{X} \rightarrow \mathcal{R}_+ \cup \{0\}$ which associates to every clock x its value $v(x)$.

Definition 1 (Timed Automaton) *A* timed automaton *(TA) is a tuple $\mathcal{A} = (\mathcal{Q}, q_0, \mathcal{X}, \mathcal{I}, \Delta, \Sigma)$ where $\mathcal{Q}$ is a finite set of states, q_0 is the initial state, $\mathcal{X}$ is a finite set of clocks, $\mathcal{I} : Q \rightarrow \mathcal{C}(\mathcal{X})$ is the invariant function, $\Delta \subseteq \mathcal{Q} \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{X}} \times \mathcal{Q}$ is a finite set of transitions and Σ is an alphabet of actions augmented with the action $\perp$ that represents the empty action.*

A configuration of a timed automaton is a pair $(q, \mathbf{v})$ where q is a state and $\mathbf{v}$ a vector of clock valuations. The semantic of a timed automaton is given as a timed transition system with two kinds of transition: timed transitions representing the elapse of time in a state, and discrete transitions representing the ones between states. A timed transition is enabled if clocks valuations satisfy the invariant of the state and a discrete one is enabled if clocks valuations respect the guard on the transition. Then, we define a run in a timed automaton as a sequence of timed and discrete transitions. Given a network of timed automata, synchronous communication between timed automata can be done using input actions denoted $a?$ and output actions denoted $a!$.

4.2 Modeling the Task

For the sake of clarity, we start by showing how to model a real-time task without taking into account the energy consumption.

Let $\tau_i(r_i, C_i, T_i, D_i)$ be a real-time task. We model this task using a timed automaton $Task_i$ with a set $\mathcal{Q} = \{ini_i, act_i, exe_i, pre_i, stop_i\}$ of states and two clocks c_i and d_i. This automaton is synchronized with an automaton $Period_i$ using the action $release_i$. This action is launched by the automaton $Period_i$ every period T_i. These automata are presented in Figure 2. The automaton $Task_i$ starts its execution at state ini_i, where no instance of task τ_i is active. When an action $release_i$ is captured, the automaton moves to state act_i and

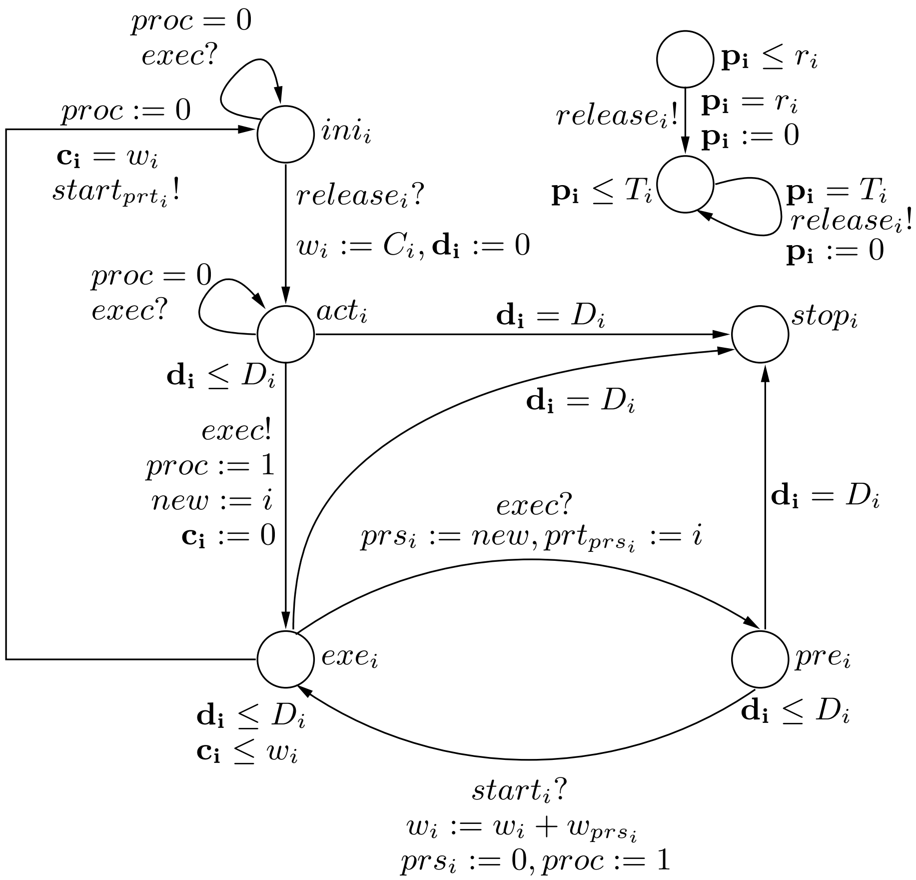

Figure 2: Timed Automaton Model for a Task τ_i

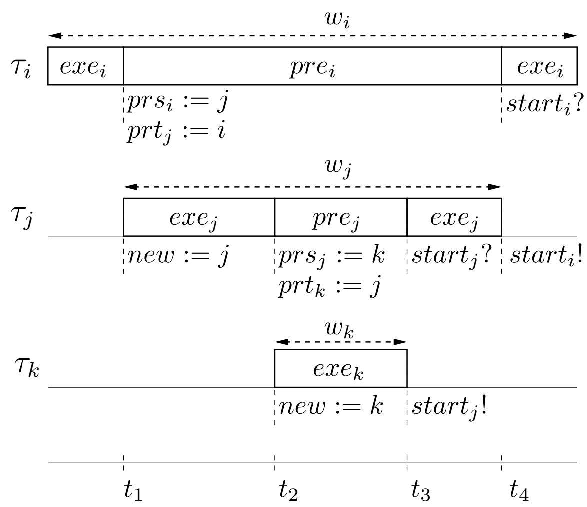

Figure 3: Restrictions on Preemption

the clock d_i is reset to zero. The clock d_i is used to measure the elapsed time since the activation of the task. When the clock d_i reaches the deadline D_i the automaton moves to state $stop_i$.

In state act_i, the task is active but not yet executed. If the task starts its execution, the automaton moves to state exe_i and a global variable *proc* is reset to one indicating that the processor is not idle. When a task τ_i starts, the clock c_i is reset to zero, this clock is used to measure w_i the response time of task τ_i. The response time w_i of a task is set initially to C_i, the execution time of the task. The automaton stays in state exe_i exactly w_i time units, which is modeled using an invariant $c_i \leq w_i$ on state exe_i and a guard $c_i = w_i$ from state exe_i to state ini_i.

To handle preemptions using timed automata, we restrict ourselves to a class of schedules that meet the two restrictions cited bellow. Indeed, modeling a task where preemptions can occur at every instant is not possible using timed automata. Preemption could however be modeled using stopwatch automata, a model where clocks can be stopped. Unfortunately, model checking is known to be undecidable on this model in the general case [18, 19]. That is why we have the following restrictions:

1. Restriction 1: we restrict ourselves to fixed priority at job level schedules. As a consequence, if a task τ_i is preempted by a task τ_j, τ_i cannot be resumed until τ_j has finished. As mentioned before, note that EDF is part of this class of scheduling algorithms.

2. Restriction 2: the processor can be idle only if no task is under preemption.

Under these restrictions, it is possible to handle preemptions, using the following property:

Proposition 1 *Let τ_i and τ_j be two real-time tasks of a schedule respecting Restriction 1 and Restriction 2. If τ_i is preempted by τ_j then, the preemption duration of task τ_i is equal to w_j the response time of τ_j.*

An illustration of Proposition 1 is given in Figure 3. This figure shows an example of three tasks τ_i, τ_j, τ_k that respect the two restrictions. Task τ_j preempts task τ_i, and task τ_k preempts task τ_j. We can see in the figure that the time of preemption of each task is equal to the response time of its preempting task. This example is easily transposable to the case where a task job is preempted several times.

Using this proposition, it is possible to model preemptions with a timed automaton as follow. When a task is preempted, the automaton moves to state pre_i. To respect Restrictions 1 and 2, a task can be preempted only if a new task is executed. Indeed, a global action $exec?$ synchronizes every preemption with the beginning of a new task. The variable prs_i records the identifier of the preempting task and prt_i the identifier of the task preempted by τ_i. In Figure 3, at $t = t_1$ the task τ_i is preempted by τ_j. Consequently, in our automaton model, the global variable new, recording the identifier of the new task, is equal to j, the variable prs_i is equal to j and prt_j is equal to i.

When the preempting task τ_{prs_i} resumes, the automaton of the preempted task τ_i moves to state exe_i synchronizing with an action $start_i$. Then, the response time w_i of the task τ_i is augmented by the response time of the preempting task τ_{prs_i}.

The first restriction does not limit the generality of our work so much. Indeed, mostly all the commonly known scheduling algorithms respect Restriction 1. Restriction 2 seems stronger in a first glance. However, this is not true in the general context, without considering energy related issues, because most commonly studied schedulers are work-conserving, and are so de facto respecting the restriction.

However, when idle times must be inserted in order to recharge the battery, Restriction 2 appears as a strong restriction. Therefore, we explain in Section 4.3 how to overcome it by letting the model checker insert idle times at the beginning of a preempting task.

4.3 Task and Energy

Let $\mathcal{B}(E_{max}, e_{bat})$ be a battery as defined in Section 3. The battery is modeled using a timed automaton A_B with one clock b and three states. The variable E is used to store the level of energy in the battery.

The states *init* and *charge* represent the consuming mode and the charging mode respectively. The third state is an urgent state (no timed transition in this state) and is used only to separate the two modes. When the battery moves from consuming to charging mode, the clock b is reset to zero and a variable ch is set to 1, indicating that the battery is in the charging mode. The charging of the battery is discretized. Indeed, the automaton stays in the charging mode exactly one time unit using the invariant $b \leq 1$ in state *charge* and the guard $b = 1$ from *charge* state to the urgent state. Then, after each time unit in the charging mode, the battery level increases by e_{bat}. When the battery reaches its maximal capacity E_{max}, the battery level does not increase even if the battery is in the charging mode. The variable *tcharge* represents the total time spent in the charging mode. Figure 4 represents the corresponding automaton.

To take into account the energy consumption of a task, we need to add some modifications to the task automaton. Figure 5 represents the timed automaton modeling a task τ_i where we omit some information already given by Figure 2.

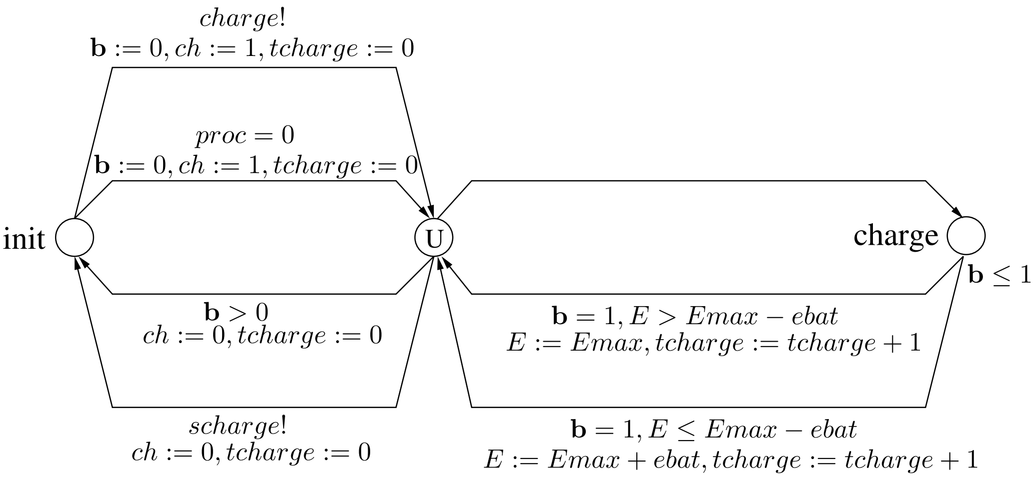

Figure 4: The Automaton for a Battery $\mathcal{B}(E_{max}, e_{bat})$

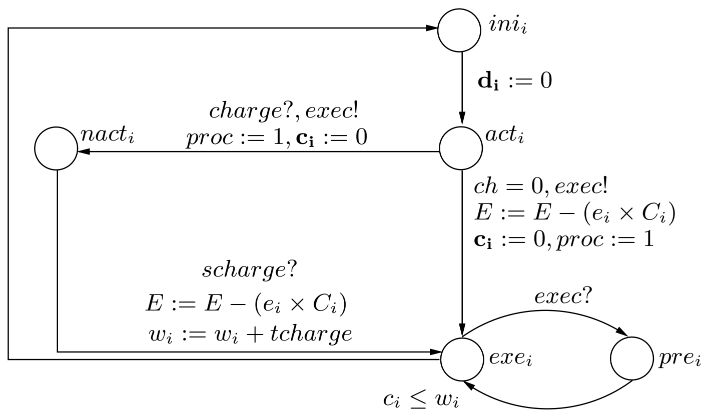

Figure 5: Adding Energy Consumption to the Timed Automaton $Task_i$

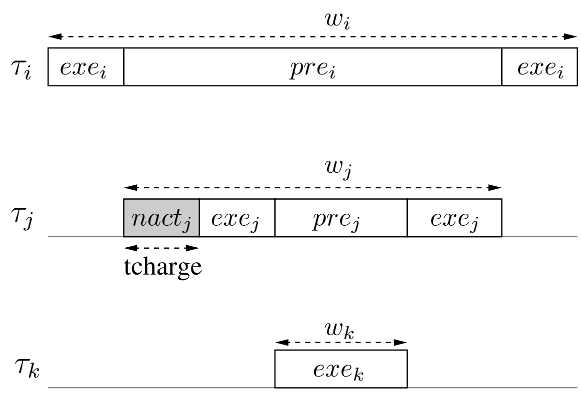

Figure 6: Charging During Preemption

Since we consider that a task consumes all the energy required for its execution at its beginning, when a task starts its execution, the energy available E is reduced by $e_i \times C_i$.

According to Restriction 2, the processor can not be idle if a task is preempted. This restriction does not allow us to model a behavior where the battery is in a charging mode if a task is preempted. To overcome this restriction in the particular case where the inserted idle time is a battery recharge, we add a new state $nact_i$ s.t.: if the automaton is in state $nact_i$, the processor is considered to be busy and is dedicated to the task τ_i. When the automaton moves to state $nact_i$, the battery automaton moves, synchronizing with the action $charge!$, to the charging mode. When the automaton moves from state $nact_i$ to the execution state, the battery moves to the idle mode and $tcharge$, the total time spent in the charging mode, is added to w_i, the response time of the task τ_i, as shown in Figure 6.

5 Scheduling using CTL Model Checking

Model checking is a method for automatic verification where the system is modeled using a formal model M and the correctness property is stated with a formal specification language ϕ. Given a model M and a property ϕ, model checkers are used to automatically decide whether M satisfied ϕ or not.

In this section, we present how we use CTL [20] model checking to test the feasibility of a task set and its schedulability with PFP and EDF.

CTL properties are generated using the following grammar:

$$\phi ::= p|(\neg\phi)|(\phi \wedge \phi)|(\phi \vee \phi)| \\ AX\phi|EX\phi|AG\phi|EG\phi|A[\phi U \phi]|E[\phi U \phi]$$

where p is a set of atomic formulas. CTL formulas are interpreted on a transition system s.t. the initial state s_0 satisfies: $AG\phi$ iff in all the paths starting at s_0 all the states satisfy ϕ, $EG\phi$ iff there exists a path starting at s_0 where all the states satisfy ϕ, $AX\phi$ iff in all the paths starting at s_0 in the next state ϕ is satisfied, $EX\phi$ iff there exits a path starting at s_0 where in the next state ϕ is satisfied, $E[\phi_1 U \phi_2]$ iff there exists a path starting at s_0 where ϕ_1 is satisfied until ϕ_2 is satisfied and $A[\phi_1 U \phi_2]$ iff for all the paths starting at s_0 ϕ_1 is satisfied until ϕ_2 is satisfied.

5.1 Feasibility

Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1, \ldots, \tau_n\}$ and $\mathcal{B} = (E_{max}, e_{bat})$. We model each task τ_i using a timed automaton $Task_i$ and model the battery B using a timed automaton A_B as defined in Section 4.3. We note $\mathcal{A}_\mathcal{P}$ the parallel composition of the automata $Task_1, Task_2, \ldots Task_n$ and A_B. A configuration of $\mathcal{A}_\mathcal{P}$ is a tuple (q, v, E) where $q = (s_1, \ldots s_n, s_b)$ and $\mathbf{v} = (v(c_1), v(d_1), \ldots v(c_n), v(d_n), v(b))$ s.t.:

1. $\forall i \in [1, n], s_i$ is a state of the automaton $Task_i$ and $v(c_i), v(d_i)$ are the clocks valuations of c_i and d_i respectively,
2. s_b is a state of the automaton A_B and $v(b)$ is a clock valuation of clock b,

3. E is the global variable storing the level of energy in the system.

The configurations of the timed transition system of $A_{\mathcal{P}}$ represent the possible configurations of a task (active, executing, preempted, ...) plus the possible battery configurations (mode and level).

The following proposition provides a feasibility test for the scheduling problem $\mathcal{P} = (\Sigma, \mathcal{B})$.

Proposition 2 (feasibility) *Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1 \dots \tau_n\}$. $\mathcal{P}$ is feasible iff the network $A_{\mathcal{P}}$ modeling $\mathcal{P}$ satisfies the CTL Formula 1*

$$\phi_{Sched_1} : EG\neg\Big(\bigvee_{i\in[1,n]} stop_i \vee E < 0\Big) \tag{1}$$

Proposition 2 states that the problem is feasible iff there exists an infinite run ξ in $A_{\mathcal{P}}$ where all the configurations (q, v, E) satisfy the property $\varphi : \neg(\bigvee_{i\in[1,n]} stop_i \vee E < 0)$. We call this run a feasible run. In other words, a run ξ is feasible iff none of its configurations contains a state $stop_i$ or a negative battery level. If such a run exits, it corresponds to a schedule where there is enough energy to execute all the active tasks and none of the tasks misses its deadline. Indeed, an automaton $Task_i$ reaches the state $stop_i$ iff the clock d_i reaches the deadline D_i of the task τ_i (see Figure 2). Since the clock d_i is reset to zero at each activation of a task τ_i, a clock greater than the deadlines implies that a deadline miss has occurred.

Given a real-time problem $\mathcal{P} = (\Sigma, \mathcal{B})$, one can check the optimal characteristics of a battery to make the problem feasible. For example, find the minimal value for E_{max} to make the problem feasible. But also find a maximal value for a lower bound E_{min} under which it is not permitted for the battery to go bellow. Given a minimal value E_{min} and a maximal value E_{max}, Formula 2 checks if the problem is feasible.

$$\phi_{Sched_2} : EG\neg\Big(\bigvee_{i\in[1,n]} stop_i \vee E > E_{max} \vee E < E_{min}\Big) \tag{2}$$

Another application is to find a schedule that minimizes the number of state changes between the battery charging mode and the task execution mode. We add to the model a global variable NbC counting the number of battery mode changes. The variable is initially set to zero, and is incremented when the battery moves from consuming to idle mode. Then, when all the tasks are in their initial state, the variable is reset to zero.

Using Formula 3, we can find the feasible schedule minimizing the number of mode changes. Indeed, by changing the value of NbC_{min}, we can find the minimal value of mode changes for which the problem remains feasible.

$$\phi_{Sched_3} : EG\neg\Big(\bigvee_{i\in[1,n]} stop_i \vee E < 0 \vee NbC > NbC_{min}\Big) \tag{3}$$

The same idea can be used to minimize the number of preemptions.

5.2 Schedulability

To test schedulability according to a given scheduling policy[2], one can model the scheduling policy in the CTL checked formula.

To test PFP schedulability, we have to test if there exists a feasible run where some configurations are forbidden: the ones where a task is executing while a greater priority task is not.

Proposition 3 (PFP Schedulability) *Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1 \dots \tau_n\}$ is sorted according to the priorities of the tasks. $\mathcal{P}$ is schedulable according to PFP iff the network $A_{\mathcal{P}}$ modeling $\mathcal{P}$ satisfies the CTL Formula 4.*

$$\begin{aligned}\phi_{fp} : EG\neg\Big(& \bigvee_{i\in[1,n-1]} (act_i \wedge d_i > 0 \bigwedge_{j\in[i+1,n]} exe_j) \bigvee_{i\in[1,n-1]} \\ & (pre_i \bigwedge_{j\in[i+1,n]} exe_j) \bigvee_{i\in[1,n]} stop_i \vee E < 0\Big)\end{aligned} \tag{4}$$

Formula 4 states that the problem is schedulable according to PFP iff there exists a feasible run where, in all the configurations, a task τ_j cannot be in its execution state exe_j if a highest priority task τ_i is in state act_i or pre_i.

Using this approach, we can also test the EDF schedulability.

Proposition 4 (EDF Schedulability) *Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1 \dots \tau_n\}$. $\mathcal{P}$ is schedulable according to EDF iff the network $A_{\mathcal{P}}$ modeling $\mathcal{P}$ satisfies the CTL Formula 5.*

$$\begin{aligned}\phi_{edf} :EG\neg\Big(& \bigvee_{i\in[1,n]} \bigvee_{j\neq i\in[1,n]} (act_i \wedge d_i > 0 \wedge exe_j \wedge p_{ij}) \\ \bigvee_{i\in[1,n]} \bigvee_{j\neq i\in[1,n]} & (pre_i \wedge exe_j \wedge p_{ij}) \bigvee_{i\in[1,n]} stop_i \vee E < 0\Big)\end{aligned} \tag{5}$$

p_{ij} is a state of an observer automaton reachable when $d_i - d_j > D_i - D_j$ with d_i and d_j the deadline clocks of tasks τ_i and τ_j respectively.

Under the EDF scheduling policy, the processor is assigned to a task if it is the closest to its deadline. Formula 5 states that the problem is schedulable according to EDF iff there exists a feasible run where, in all the configurations, a task cannot be in its execution state if a task with a closer deadline is active or preempted.

To solve the real-time problem $\mathcal{P} = (\Sigma, \mathcal{B})$, a scheduling algorithm has to be non-work-conserving in the sense that the processor can be idle if a task is not executing. Idle times are times where the battery is in a charging mode. A classical scheduling policy (PFP, EDF ...) is not sufficient to solve our real-time problem, because it does not provide the idle times where the battery has to be charged. If the schedulability tests given in Formulas 4 and 5 are satisfied, we only know that there exists a feasible schedule where the order of execution of

[2]note that the scheduling policy has to be a fixed priority at job level one, according to Restriction 1.

tasks respects the priority assignment of PFP or EDF algorithms. For example, if a real-time problem is not EDF_{asap} schedulable, Formula 5 is not sufficient to prove it.

We qualify an algorithm to be *as soon as possible* if the processor is idle only if the highest active priority task can not be executed.

According to the restriction of our model, idle times can be inserted in a schedule only at the beginning of the execution of a task. Thus, to compute an *as soon as possible* schedule, we forbid in our model configurations staying more than necessary in a state act_i or a state $nact_i$.

Proposition 5 provides a schedulability test for PFP *as soon as possible*.

Proposition 5 (PFP As Soon As Possible Schedulability) *Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1 \dots \tau_n\}$ is sorted according to the priorities of the tasks. $\mathcal{P}$ is schedulable according to PFP_{asap} iff the network $A_{\mathcal{P}}$ modeling $\mathcal{P}$ satisfies the CTL Formula 6*

$$\begin{aligned} \phi_{fp_S} : \phi_{fp} \wedge EG\neg\Big(\bigvee_{i \in [1,n]} \big(((act_i \wedge d_i > 0) \vee \\ (nact_i \wedge c_i > 0)) \bigwedge_{j \in [1,i-1]} ini_j \wedge (e_i \times C_i) \geq E\big)\Big) \end{aligned} \tag{6}$$

Formula 6 sates that the problem is schedulable according to an as soon as possible PFP scheduling algorithm iff there exists a feasible run where:

1. all the configurations satisfy formula 4, and
2. there is no configuration where the highest priority active task has enough energy to be executed ($C_i \times e_i \geq E$) and is not scheduled.

An active task τ_i has the highest priority in a configuration $((s_0, \dots s_n, s_b), v, E)$ if all the tasks τ_j with $j < i$ are not active.

The same principle can be used for EDF_{asap} schedulability. In this case, a task has the highest priority in a configuration if it has the closest deadline among all the active tasks.

Proposition 6 (EDF As Soon As Schedulability) *Let $\mathcal{P} = (\Sigma, \mathcal{B})$ be a real-time problem where $\Sigma = \{\tau_1 \dots \tau_n\}$. $\mathcal{P}$ is schedulable according to EDF_{asap} iff the network $A_{\mathcal{P}}$ modeling $\mathcal{P}$ satisfies the CTL Formula 7.*

$$\begin{aligned} \phi_{edf_S} : \phi_{edf} \wedge EG\neg\Big(\bigvee_{i \in [1,n]} \bigvee_{A \subseteq [1,n]-\{i\}} \big(((act_i \wedge d_i > 0) \\ \vee (nact_i \wedge c_i > 0)) \bigwedge_{j \in A} \neg ini_j \bigwedge_{k \notin A, k \neq i} ini_k \bigwedge_{j \in A} p_{ij} \\ \wedge (C_i \times e_i) \geq E\big)\Big) \end{aligned} \tag{7}$$

p_{ij} is a state of an observer automaton reachable when $d_i - d_j > D_i - D_j$ with d_i and d_j the deadline clocks of tasks τ_i and τ_j respectively.

Formula 7 states that if an active task τ_i has the highest priority among the subset A of all other active tasks, then the task has to be executed if there is enough energy for it's completion.

5.3 Scheduling Algorithm

To compute a scheduling algorithm for a feasible real-time problem $\mathcal{P} = (\Sigma, \mathcal{B})$, we first check the appropriate formula to generate a feasible infinite run if one exists. Model checking for timed automata is decidable but PSPACE-complete [21], however, in our approach, the feasible run is computed off line.

Given a run of the network $A_{\mathcal{P}}$, a schedule can be derived. This schedule defines the rules controlling when and how transitions between different configurations occur.

Let ξ be a feasible infinite run satisfying one of the formulas of the previous subsections. This run can be written as: $\xi = (q_0, \mathbf{v}_0, E_0) \to (q_1, \mathbf{v}_1, E_1) \to \ldots (q_k, \mathbf{v}_k, E_k) \ldots$ where $(q_0, \mathbf{v}_0, E_0)$ is the initial configuration with $q_0 = (ini_0, \ldots, ini_n, ini)$, $\mathbf{v}_0$ is an $n+1$ dim valuation vector where all the valuations are equal to 0 and $E_0 = E_{max}$. Since the run ξ is infinite, it contains at least one cycle. We note $(q^*, \mathbf{v}^*, E^*) \ldots (q^*, \mathbf{v}^*, E^*)$ the first cycle of ξ and we call the run $\xi_{sched} = (q_0, \mathbf{v}_0, E_0) \to \ldots (q^*, \mathbf{v}^*, E^*) \ldots \to \ldots (q^*, \mathbf{v}^*, E^*)$ a scheduling run.

An on-line scheduling algorithm can be obtained by simply reading sequentially the configurations of the pre-computed scheduling run. Indeed, using this trace, we can compute a scheduling function $F_{Sched} : \{0 \ldots t^*\} \to \{1 \ldots n\} \cup \{\epsilon_1, \epsilon_2\}$ where t^* is the total length of the run ξ_{sched} s.t. if:

1. $F_{Sched}(t) = i \in \{0 \ldots n\}$ task τ_i is executing at time t,
2. $F_{Sched}(t) = \epsilon_1$ the processor is idle at time t and the battery is not charging,
3. $F_{Sched}(t) = \epsilon_2$ the processor is idle at time t and the battery is charging.

Note that if a scheduling problem is proven to be EDF_{asap} or PFP_{asap} schedulable, there is no use to compute a scheduling function.

The computed scheduling algorithms are sustainable according to the duration of a task. In other words, if a task terminates before its worst case execution time, the schedule remains feasible. Indeed, in our model, a task consumes all its necessary energy at the beginning of its execution. So, if it terminates before its WCET, it simply consumes less than supposed during the verification process, and the new idle times are times where the battery can charge more than proven sufficient for the system to be feasible.

6 Experiments

To validate our approach, we use the timed model checker Uppaal [1] to implement and test our model on some examples presented in this section. Materials are available on line at [22].

We consider for all the examples the same set of tasks $\Sigma = \{\tau_1, \tau_2, \tau_3\}$ where, $\tau_1 = (0, 4, 10, 10, e_1)$, $\tau_2 = (0, 4, 20, 20, e_2)$ and $\tau_3 = (0, 6, 40, 40, e_3)$.

6.1 $\mathcal{P}_1 = (\Sigma, \mathcal{B}_1(10, 2))$ with: $e_1 = e_2 = e_3 = 1$

We first test our approach on the illustrative example $\mathcal{P}_1$ with $B_1(10, 2)$, $\tau_1 = (0, 4, 10, 10, 1)$, $\tau_2 = (0, 4, 20, 20, 1)$ and $\tau_3 = (0, 6, 40, 40, 1)$. The model checker Uppaal states that Formula 1 is not satisfied, we conclude that the problem $\mathcal{P}_1$ is not feasible.

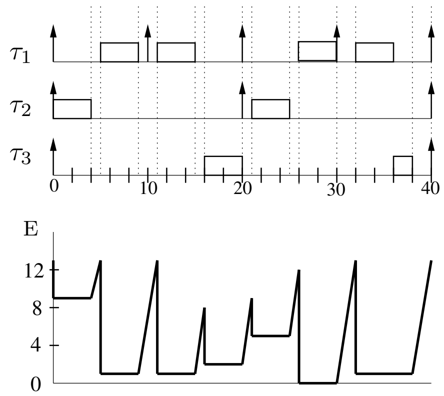

Figure 7:$PFP_{2,1,3}$ Schedule: Battery $\mathcal{B}(13,7)$ and $e_1 = 3, e_2 = e_3 = 1$

6.2 $\mathcal{P}_2 = (\Sigma, \mathcal{B}_2(10,3))$ with: $e_1 = e_2 = e_3 = 1$

We augment the battery charge rate to $e_{bat} = 3$ and prove using Formula 1 that the problem is feasible for a battery $\mathcal{B}_1(10,3)$.

We also prove, using Formulas 6 and 7, that the problem is schedulable for RM_{asap}, $PFP_{2,1,3}$ (a fixed priority as soon as possible algorithm where τ_2 has the highest priority, and τ_3 the lowest one) and EDF_{asap}.

We show that the minimal value of the maximal capacity for which the problem remains schedulable for EDF_{asap} and RM_{asap} is $E_{max} = 6$. And for $PFP_{2,1,3}$ the minimal capacity for schedulability is $E_{max} = 8$.

6.3 $\mathcal{P}_3 = (\Sigma, \mathcal{B}_3(14,7))$ with: $e_1 = 3, e_2 = e_3 = 1$

Then, we increase the rate of consumption of τ_1 to 3 and use a battery $\mathcal{B}_3(14,7)$.

We prove that this problem is schedulable for EDF_{asap}, RM_{asap} and $PFP_{2,1,3}$.

6.4 $\mathcal{P}_4 = (\Sigma, \mathcal{B}_4(13,7))$ with: $e_1 = 3, e_2 = e_3 = 1$

We decrease the maximal capacity of the battery to $E_{max} = 13$ and prove that the problem is no more schedulable using any EDF or RM algorithm, but is schedulable for $PFP_{2,1,3}$.

Figure 7 represents the schedule using $PFP_{2,1,3}$ in the interval $[0,40]$. We can see that in $[0,40]$ no task misses its deadline and the energy of the system E is always greater than 0. At $t = 40$, all the tasks are active and the level of the battery is $E = E_{max} = 13$ as in the initial configuration. That confirms the result obtained with the model checking tool.

6.5 $\mathcal{P}_5 = (\Sigma, \mathcal{B}_5(12,7))$ with: $e_1 = 3, e_2 = e_3 = 1$

We then decrease again the maximal capacity of the battery to $E_{max} = 12$. Using our approach, we prove that the problem is no more schedulable using any fixed priority algorithm and of course is still not schedulable using any EDF algorithm.

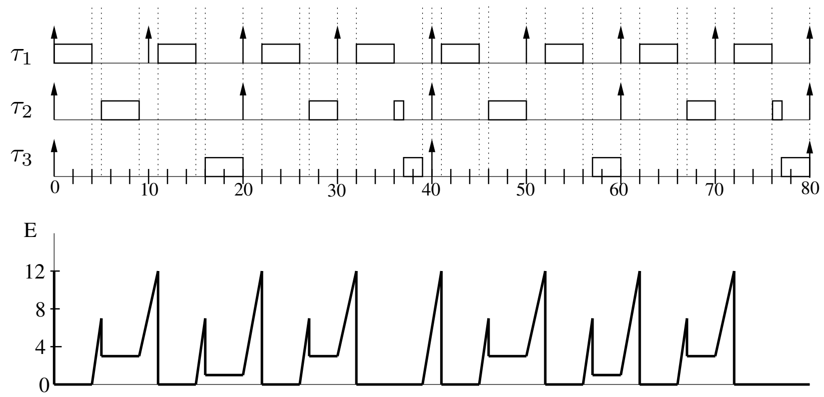

Figure 8:Not Schedulable for EDF_{asap}: Battery $\mathcal{B}(12,7)$ and $e_1 = 3, e_2 = e_3 = 1$

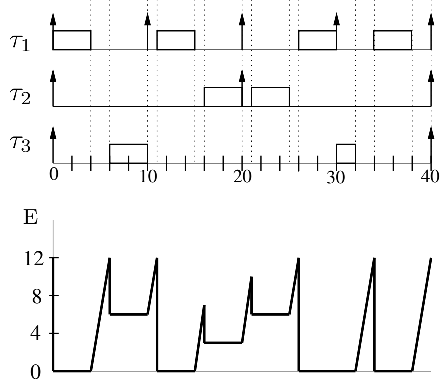

Figure 9:UPPAAL Feasible Schedule: Battery $\mathcal{B}(12,7)$ and $e_1 = 3, e_2 = e_3 = 1$

In Figure 8, an EDF_{asap} schedule for the problem $\mathcal{P}_5$ with $B_5(12,7)$ is represented in the interval $[0,80]$. We can see that at $t = 40$ all the tasks are active but the available energy is equal to 7, then at $t = 80$ there is no more energy in the battery. Thus, at $t = 80$ we have to charge the battery to continue the execution, and task τ_3 will miss its deadline at $t = 120$.

However, using Formula 1, we prove that the problem is feasible. Thus there exits a feasible schedule for this problem. Using the feasible trace generated by the tool UPPAAL, we compute the schedule represented in Figure 9. In this schedule, at $t = 6$, the first instance of task τ_2 has a lower priority than task τ_2, while it is the contrary at $t = 11$. At $t = 40$, all the tasks are active and the battery level is equal to $E_{max} = 12$, thus this schedule can be repeated infinitely with no deadline miss and an energy level always positive.

6.6 Minimize the Number of Battery Mode Change

Using Formula 3, we built the schedule that minimizes the number of battery change modes for the problem $\mathcal{P}_5$. Figure 10 represents the schedule produced

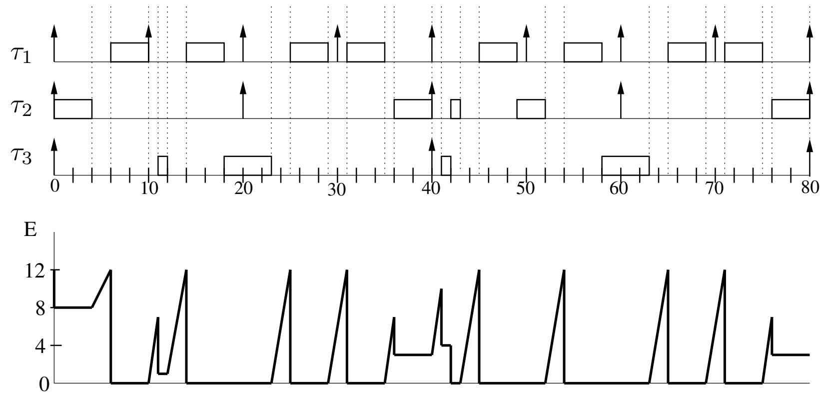

Figure 10: UPPAAL Feasible Schedule With Minimal Change Mode of the Battery: Battery $\mathcal{B}(12, 7)$ and $e_1 = 3, e_2 = e_3 = 1$

by the tool UPPAAL. In this schedule, the battery moves from the consuming to the charging mode 6 times, while it was 7 times in the schedule of Figure 9. To execute the system, one has first to follow the schedule of the interval $[0, 40]$, and then to repeat infinitely the schedule of the interval $[40, 80]$.

6.7 $\mathcal{P}_6 = (\Sigma, \mathcal{B}_6(14, 7))$ with: $e_1 = 3, e_2 = e_3 = 1$ and $e_{min} = 2$

Finally, we prove that neither EDF nor PFP algorithms can schedule the problem $\mathcal{P}_6$ where $E_{max} = 14$ and the minimal tolerated battery level is $e_{min} = 2$. However, using Formula 2, we prove that the problem is feasible and a scheduling algorithm can be computed. The schedule computed using the feasible trace produced by UPPAAL is similar to the schedule of Figure 9.

7 Conclusion

In this paper, we have presented how to use model checking to solve a scheduling problem under energy constraints. We first formalized the problem and then provided a feasibility test and schedulability tests for PFP and EDF. We then showed how to compute a feasible schedule if one exits. Our approach also permits to derive the optimal characteristics of a battery to make a given real-time problem feasible. The studied characteristics are the minimal and maximal reached energy levels, the number of battery mode switches between charging and discharging, and finally the number of preemptions. One can extend this study to other criteria by proposing an appropriate CTL formula. Using the tool UPPAAL, we experimented our model on some examples to validate the approach.

To be able to model preemptions using timed automata, we had to restrict the authorized schedules to fixed priority at job level ones and to allow recharging of the battery only at the beginning of a task if this task is preempting another one.

As future work, we have to formalize the memory complexity of generated on-line schedulers. For that, we first have to characterize the worst execution case of this scheduling problem.

Modeling more realistic recharging behaviors must also be studied. Both in terms of energy availability from the environment (e.g. sunshine previsions), and in terms of the physical capabilities of the storage unit used. For example, at equivalent power supply, a chemical battery will not store energy at the same rate and following the same process as a capacitor.

References

[1] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.

[2] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the worst-case energy consumption of embedded software," in *IEEE Real Time Technology and Applications Symposium.* IEEE Computer Society, 2006, pp. 81–90.

[3] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996.

[4] A. D. F. Yao and S. Shankar., "A scheduling model for reduced cpu energy," *IEEE Annual Foundations of Computer Science*, pp. 374 – 382, 1995.

[5] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors," in *Proceedings of the IEEE Real-Time Systems Symposium*, ser. RTSS '98. IEEE Computer Society, 1998, pp. 178–.

[6] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, ser. DAC '99. New York, NY, USA: ACM, 1999, pp. 134–139.

[7] C. M. Krishna and Y. H. Lee., "Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems," in *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium*, ser. RTASâĂŹ00, 2000.

[8] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable," in *Proceeding of Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS)*, 2001.

[9] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Systems*, vol. 37, no. 3, pp. 233–260, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1007/s11241-007-9027-0

[10] M. Chetto and H. El Ghor, "Real-time Scheduling of periodic tasks in a monoprocessor system with a rechargeable battery," in *WIP Session Proceedings of The 30th IEEE Real-Time Systems Symposium*, IEEE,

Ed., Washington, États-Unis, Dec. 2009, p. 45. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00542182

[11] H. EL Ghor, M. Chetto, and R. H. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," *Computers and Electrical Engineering*, vol. 37, pp. 498–510, July 2011. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2011.05.003

[12] M. Chetto, D. Masson, and S. Midonnet, "Fixed priority scheduling strategies for ambient energy-harvesting embedded systems," in *GreenCom 2011. IEEE/ACM International Conference on Green Computing and Communications*, Chengdu, Sichuan, China, du 4 au 5 AoÃżt 2011, pp. 50–55.

[13] Y. Abdeddaïm, E. Asarin, and O. Maler, "Scheduling with timed automata," *Theoretical Computer Sciences*, vol. 354, no. 2, 2006.

[14] A. Fehnker, "Scheduling a steel plant with timed automata," in *RTCSA*, 1999, pp. 280–286.

[15] E. Fersman, P. Krcál, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Inf. Comput.*, vol. 205, no. 8, pp. 1149–1172, 2007.

[16] Y. Abdeddaïm and D. Masson, "Self-suspending periodic real-time tasks using model checking," in *Work-in-Progress Session of 32nd IEEE Real-Time Systems Symposium*, ser. WIP-RTSS'11, 2011.

[17] R. Alur and D. Dill, "Automata for modeling real-time systems." in *International Colloquium on Automata, Languages and Programming (ICALP)*, Warwick, England, 1990.

[18] K.Cerans, "Algorithmic problems in analysis of real time system specifications," Ph.D. dissertation, University of Latvia, 1992.

[19] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine, "Decidable integration graphs," *Inf. Comput.*, vol. 150, no. 2, pp. 209–243, 1999.

[20] D. Kozen, Ed., *Logics of Programs, Workshop*, ser. Lecture Notes in Computer Science, vol. 131. Springer, 1982.

[21] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[22] Y. Abdeddaïm and D. Masson, "Uppaal implementations." [Online]. Available: http://igm.univ-mlv.fr/~masson/Softwares/EnergyAutomata/