Reconf-Pack: A Simulator for Reconfigurable Battery-Powered Real-Time Systems

Aymen Gammoudi, Adel Benzina, Mohamed Khalgui, Daniel Chillet, Aicha Goubaa

Reconf-Pack: A Simulator for Reconfigurable Battery-Powered Real-Time Systems

Aymen Gammoudi
LISI Lab, INSAT
Tunisia Polytechnic School
University of Carthage, Tunisia
IRISA Lab, ENSSAT
University of Rennes1, France
aymen.gammoudi@irisa.fr

Adel Benzina
LISI Lab, INSAT
Tunisia Polytechnic School
University of Carthage, Tunisia
Adel.Benzina@isd.rnu.tn

Mohamed Khalgui
LISI Lab, INSAT
University of Carthage, Tunisia
SystemsControl Lab
Xidian University, China
khalgui.mohamed@gmail.com

Daniel Chillet
IRISA Lab, ENSSAT
University of Rennes1, France
daniel.chillet@irisa.fr

Aicha Goubaa
LISI Lab, INSAT
University of Carthage, Tunisia
goubaa.aicha@gmail.com

KEYWORDS

Embedded system; Real-time and low-power scheduling; Reconfiguration; OS software optimization; Simulation.

ABSTRACT

This paper addresses the management of independent periodic OS tasks running on battery-powered real-time systems which can change their parameters at run-time for each reconfiguration scenario. A reconfiguration is assumed to be any run-time automatic addition, removal, or also update of software tasks according to external events or also user requirements. After any scenario, the system can become not feasible and can also lead to a shortage of energy before the next recharge. To resolve these problems, we propose a dynamic strategy to be applied at run-time based on grouping tasks in packs. This strategy offers five different solutions to ensure the system remains up until the next battery recharge while remaining feasible. In this context, we present, in this paper, a new simulator *Reconf-Pack* for analyzing a reconfiguration and applying the proposed strategy for real-time systems. It is based upon another tool *Task-Generator* which generates random tasks. According to the state of the system after a reconfiguration, *Reconf-Pack* calculates dynamically a deterministic solution. Moreover, it compares our pack-based solutions to related works.

Introduction

Nowadays, in academy and manufacturing industry, many research works have been made dealing with real-time scheduling of embedded systems. In these systems, the correctness of functions depends not upon only the results of computation but also on the times at which they are produced Gharsellaoui et al. (2012). The new generations of these systems are addressing today new criteria such as flexibility and agility Gharsellaoui and Ahmed (2014). To reduce their cost, these systems have to be changed and adapted to their environment without any disturbance Brennan et al. (2001). This paper aims to study the influence of the reconfiguration for adaptive real-time systems. A reconfiguration scenario means the addition, removal or update of software tasks in order to save the whole system on the occurrence of hardware/software faults. In the literature, two reconfiguration policies exist, (i) static reconfigurations Allen et al. (1998) to be generally applied offline and (ii) dynamic reconfigurations that can be applied at run-time. We generally define two solutions for the second case: Manual reconfigurations Rooker et al. (2007) to be applied by users at run-time and automatic reconfigurations Khalgui et al. (2011) which are generally handled by software autonomous agents Khalgui (2010). We are interested in this paper in dynamic reconfigurations and assume that the system executes n real-time tasks initially feasible towards real-time scheduling. We also assume that the system battery is recharged periodically with a recharge period *RP*. Such a system may face several scenarios: (i) Increased power consumption that, in the worst case, may surpass the available energy budget, (ii) Increased computing demand, which may lead to the violation of real-time constraints, and (iii) Increased memory demand, potentially exceeding the provided memory capacity. Several research studies Wang et al. (2014) and Wang et al. (2011) have focused on resolving these problems. The authors offer different solutions that are mainly based on the modification of the periods (T_i) or the WCETs (C_i) of software tasks in order to ensure that the system will run correctly until the next battery recharge after each reconfiguration scenario and to satisfy the real-time feasibility and memory constraints. We also proposed in a previous paper Gammoudi et al. (2015) a methodological strategy, according to the system and battery state, that proposes quantitative techniques to modify periods, reduce execution times of tasks or remove some of them to ensure real-time feasibility, avoiding memory overflow and ensuring a rational use of remaining energy until the next recharge. The general goal of this paper is to evaluate the different approaches and show the gain for each execution theory. Therefore, we developed the tool *Reconf-Pack* to support and evaluate these approaches that we apply to a running example. We compare our approach in Gammoudi et al. (2015) with the approach proposed in Wang et al. (2014) using the same case study. After the execution of *Reconf-Pack*, we notice that our strategy Gammoudi et al. (2015) has a lower cost in terms of delay than Wang et al. (2014). To generalize the performance evaluation of our strategy, another tool called *Task-Generator* has been developed to generate random systems and to randomly plan reconfiguration scenarios. The organization of this paper is as follows. Section 2 presents well known concepts in a real-time embedded systems. Section 3 gives a useful background. We detail the different solutions Gammoudi et al. (2015) in Section 4. Section 5 shows the *Reconf-Pack* tool and presents the case study. In Section 6 we present the performance evaluation of the compared techniques through *Reconf-Pack*. We conclude and present our future work in Section 7.

Fundamental Concepts

In this section we recall known concepts for real-time embedded systems that are useful to present our proposition.

Known Concepts in the EDF and RM Theory

A hard real-time system comprises a set of n independent real-time tasks τ_1, τ_2, ..., τ_n. Each task consists of an infinite or finite stream of jobs or requests which must be completed before their deadlines. A uniprocessor system can only execute one process at a time and must switch between processes, for this reason context switching will add more time to the overall execution time when preemption is used. According to Liu and Layland (1973), we present the following well-known concepts in the theory of real-time scheduling: A periodic task τ_i (C_i, T_i, D_i, MF_i) is an infinite collection of jobs that have their request times constrained by a regular interarrival time T_i, a worst case execution time (WCET) C_i, a relative deadline D_i and a memory footprint MF_i. A real-time scheduling problem is said feasible if there is at least one scheduling policy able to meet the deadlines of all the tasks. A task is valid with a given scheduling policy if and only if no job of this task misses its deadline.
EDF is the earliest deadline First policy for scheduling real-time tasks. EDF schedules tasks according to their deadlines: The task with the shortest deadline has the highest priority. Let $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ be the processor utilization factor. In the case of synchronous, independent and periodic tasks such that their deadlines are equal to their periods, $U \leq 1$ is a necessary and sufficient condition for this set of tasks to be feasible according to the EDF-based scheduling.
RM is the rate monotonic policy for scheduling real-time tasks. RM schedules tasks according to their periods: The task with the shortest period has the highest priority. A sufficient condition for a set of n tasks to be feasible according to the RM scheduling algorithm $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$ Liu and Layland (1973). We use as a notation for this real-time feasibility analysis : $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \alpha_{policy}$,where α_{policy}=1 for EDF scheduling and α_{policy}=$n(2^{\frac{1}{n}} - 1)$ for RM scheduling.

Energy Model

We consider the following energy model as described in Wang et al. (2014) and Gammoudi et al. (2015). Each rechargeable embedded system is characterized by i) A quantity of energy available at full recharge E_{max}, ii) An energy available at time t : $\Delta E(t)$, iii) A recharge period RP and iv) A time remaining until the next recharge Δt. The power consumption P is proportional to the processor utilization U Shin and Choi (1999). Then the power consumption is calculated by:

$$P = k.U^2 = k.(\textstyle\sum_{i=1}^{n} \frac{C_i}{T_i})^2 \qquad (1)$$

We assume in this paper that $k = 1$. To ensure that the system will run correctly until the next recharge, it is necessary that at time t:

$$P(t).\Delta t \leq \Delta E(t) \qquad (2)$$

where $P(t)$ is the power consumption at t, that means $P(t) \leq \frac{\Delta E(t)}{\Delta t}$. We define $P_{limit}(t) = \frac{\Delta E(t)}{\Delta t}$. After each reconfiguration scenario, we have to ensure that $P(t) \leq P_{limit}(t)$: This is the Energy Constraint.

Memory Model

We suppose that the memory model in a real-time embedded system is characterized by a memory size MS. Each task occupies at run-time MF_i amount of memory. After each reconfiguration scenario, we must ensure that: $\sum_{i=1}^{n} MF_i < MS$. This is the memory constraint.

Problem Position

We suppose that the system Sys is initially composed of n tasks and assume that $Sys(t0)$ is feasible. A system is feasible if and only if it satisfies the three constraints (real-time, energy and memory constraints). We assume in the following that the system Sys is dynamically reconfigured at run-time at $t1$ such that its new implementation of tasks is $Sys(t1) = \{\tau_1, \tau_1, ..., \tau_n \, \tau_{n+1}, ..., \tau_m\}$. The subset $\{\tau_{n+1}, ..., \tau_m\}$ is added to the initial implementation $\{\tau_1, \tau_2, ..., \tau_n\}$. To ensure that the system will run correctly after this reconfiguration scenario, at a particular time, it is necessary to check whether the new configuration satisfies the following constraints:

1. Real-time scheduling feasibility constraint, Sys must verify:

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i} \leq \alpha_{policy}$$

.

2. Energy constraint, Sys must verify:

$$P(t) \leq P_{limit}(t)$$

3. Memory constraint, Sys must verify:

$$\sum_{i=1}^{m} MF_i < MS$$

After each reconfiguration scenario, one or more of these constraints can be violated. We have to find the suitable solution to bring back the system to the feasibility conditions.

Background

We define in this section a brief summary about different approaches proposed by Wang et al. in Wang et al. (2014) and Wang et al. (2011). We present also the strategy presented by the authors in Gammoudi et al. (2015). This study is necessary to show the interest of the approach in Gammoudi et al. (2015) compared to Wang et al. (2014). Wang et al. in Wang et al. (2011),Wang et al. (2014), present a simple run-time strategy to ensure that the system runs correctly after any reconfiguration scenario. They propose to modify the tasks period T_i, assigning a single value to all tasks which is not reasonable in practice. Another solution proposed is to reduce WCETs C_i of all tasks. These solutions are interesting, but the main disadvantage is that it is not acceptable for a real-time system to change the period of tasks more than a certain limit according to user requirements. To improve these solutions and implement more suitable values, we propose in a previous paper Gammoudi et al. (2015) a new strategy based on the definition of packs of tasks and the management of their parameters. We propose to group the tasks that have "similar" periods in several packs, denoted Pk, by assigning a unique new period T^{New} to all tasks of the first pack Pk_1. Moreover all new periods affected to pack Pk_j are multiples of T^{New}, the period affected to tasks belonging to pack Pk_1. We have only to compute in this case the suitable T^{New}. This solution controls the complexity of the problem. We compare this strategy in Gammoudi et al. (2015) to Wang et al. (2014) and Wang et al. (2011) and show that the cost of delaying tasks is significantly improved.

Pack Oriented Solutions for Feasible Adaptive Real-Time Systems

We present in this section the different solutions and the proposed strategy detailed in Gammoudi et al. (2015). As explained in Section 3, we consider that each time a new period T^{New} is affected to a task that has originally a period T_i, the cost is a delay penalty for this task of T^{New} - T_i. This is applicable for tasks of pack Pk_1. For other packs Pk_j, the new period is $j * T^{New}$. So the cost for each task of the system is: (T^{New}-(T_i mod T^{New})) mod T^{New}. The total cost for the approach is the sum of all these costs.
To ensure that the system is feasible after each reconfiguration scenario, we present the following five solutions detailed and justified in Gammoudi et al. (2015). For each solution we adjust the new period T^{New} or the new WCET C^{New} to fulfill the real-time or the energy constraints. For each solution the value of T^{New} or C^{New} is calculated by minimizing the total cost of the solution in terms of delaying tasks.

Solution A: Modification of Periods under Real-Time Scheduling Constraint:

In order to respect the real-time scheduling constraint according to a scheduling policy "α_{policy}": $\sum_{i=1}^{m} \frac{C_i}{T_i} \leq \alpha_{policy}$. We propose to extend T_i the period of task τ_i as a multiple of T_{RT}^{New}:

$$T_{RT}^{New} = \left\lceil \frac{\sum_{Pk_1} C_i + \sum_{Pk_2} \frac{C_i}{2} + \ldots + \sum_{Pk_j} \frac{C_i}{j}}{\alpha_{policy}} \right\rceil \quad (3)$$

where T_{RT}^{New} is the new period affected to the tasks of pack Pk_1, $j * T_{RT}^{New}$ to tasks of Pk_j in order to respect the real-time scheduling (RT).

Solution B: Modification of WCETs under Real-Time Scheduling Constraint:

To ensure that the system is feasible after any reconfiguration scenario, we propose to reduce WCETs C_i of tasks as multiples of:

$$C_{RT}^{New} = \left\lfloor \frac{\alpha_{policy}}{\sum_{Pk_1} \frac{1}{T_i} + \sum_{Pk_2} \frac{2}{T_i} + \ldots + \sum_{Pk_j} \frac{j}{T_i}} \right\rfloor \quad (4)$$

We assign C_{RT}^{New} to tasks of Pk_1, $j * C_{RT}^{New}$ to tasks of Pk_j. After the modification of the WCETs, the processor utilization of tasks is reduced and can satisfy the real-time scheduling (RT).

Solution C: Modification of Periods under Energy Constraint:

If the system risks a fatal increase in energy consumption, it is necessary that the current power $P(t) = k.U^2$

should be less than the critical power P_{limit}. To resolve this problem, we propose to extend the periods of tasks as follows:

$$T_{Eg}^{New} = \left\lceil \frac{\sum_{Pk_1} C_i + \sum_{Pk_2} \frac{C_i}{2} + \ldots + \sum_{Pk_j} \frac{C_i}{j}}{\sqrt{\frac{P_{limit}(t)}{k}}} \right\rceil \qquad (5)$$

where T_{Eg}^{New} is the new period assigned to the tasks of Pk_1 to satisfy the energy constraint (Eg), $j * T_{Eg}^{New}$ is the period assigned to the tasks of Pk_j.

Solution D: Modification of WCETs under Energy Constraint:

The extended WCET C_i of task τ_i is multiple of C_{Eg}^{New}:

$$C_{Eg}^{New} = \left\lfloor \frac{\sqrt{\frac{P_{limit}(t)}{k}}}{\left(\sum_{Pk_1} \frac{1}{T_i} + \sum_{Pk_2} \frac{2}{T_i} + \ldots + \sum_{Pk_j} \frac{j}{T_i}\right)} \right\rfloor \qquad (6)$$

We assign C_{Eg}^{New} to the tasks of Pk_1, $j * C_{Eg}^{New}$ to the tasks of Pk_j. After the modification of the WCETs, the processor utilization of tasks is reduced and can respect the energy constraint (Eg).

Solution E: Removal Of Tasks

This solution proposes the removal of less important tasks according to an importance factor I_i defined in Gammoudi et al. (2015) in order to minimize the energy consumption.

Operating Mode

In this subsection, we present the operating mode of a system after any reconfiguration scenario. We explain here how the system is able to be adapted after any reconfiguration scenario. To satisfy the memory, real-time and energy constraints after any reconfiguration scenario, the system should start by checking the memory availability. If this constraint is respected, then the energy and also the real-time constraints have to be checked. If one or more constraints are violated, this program ensures a deterministic choice between the solutions A, B, C, D and E. To understand these steps, Figure 1 explains this strategy after a reconfiguration scenario.

Case Study

We present in this section a case study through which we test the quality of the proposed solutions in Gammoudi et al. (2015) by comparing them to Wang et al. (2014) and Wang et al. (2011). Table 1 presents the same 50 tasks presented in Wang et al. (2014) and we apply the defined approach in Gammoudi et al. (2015) in order to show the gain. *Reconf-Pack* implements all the

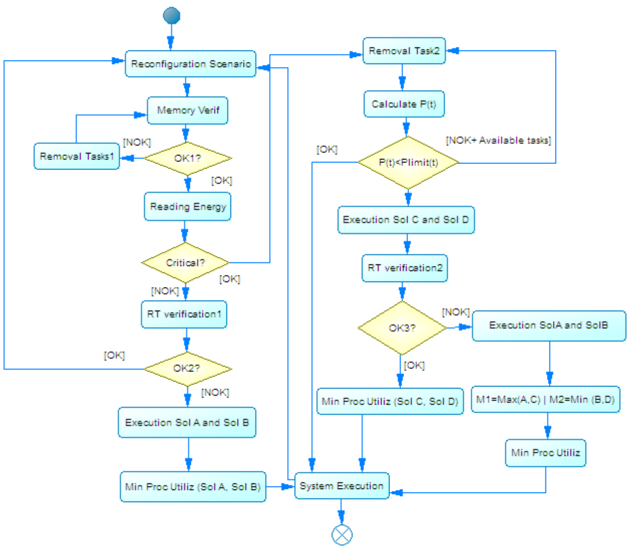
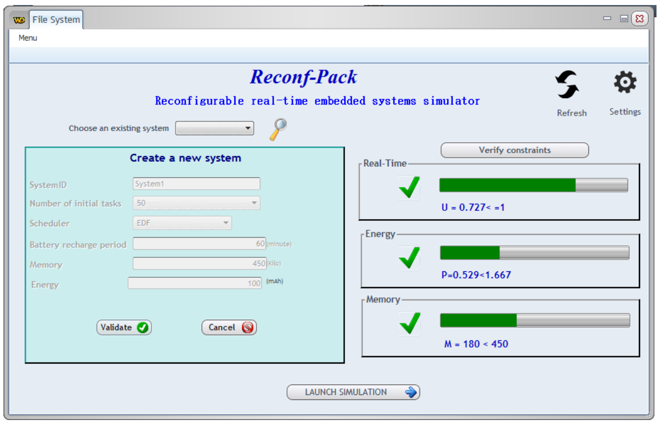


Figure 1: Decision strategy.



SystemID System1
Number of initial tasks 50
Scheduler EDF
Battery recharge period 60 (minute)
Memory 450 (Kilo)
Energy 100 (mAh)

Real-Time U = 0.727< =1
Energy P=0.529<1.667
Memory M = 180 < 450

Figure 2: *Reconf-Pack* main interface.

tasks and executes the different reconfiguration scenarios. If the system violates one or more constraints, then *Reconf-Pack* applies the solutions proposed by Wang et al. Wang et al. (2014) and applies the defined strategy in Gammoudi et al. (2015). In addition, we intend to develop *Reconf-Pack* which is an open source environment that supports and performs required comparisons to related work. We implement in Figure 2 these tasks in order to verify the system feasibility: The real-time, energy and memory constraints are satisfied.

Reconfiguration 1:

At a particular time, the system undergoes the first reconfiguration by adding 10 tasks that are presented in Table 2:

After the first reconfiguration, we notice that the three constraints are satisfied ($U = 0.942 \leq 1$, $P = 0.888 \leq 1.667$ and $M = 216 \leq 450$).

Table 1: Initial tasks

Task	C_i	T_i	MFi	Task	C_i	T_i	MFi
$A1$	3	500	2	$F1$	3	900	2
$A2$	4	500	6	$F2$	4	920	6
$A3$	5	720	3	$F3$	3	940	3
$A4$	6	740	4	$F4$	2	50	4
$A5$	7	760	3	$F5$	3	980	3
$B1$	3	780	2	$G1$	4	980	2
$B2$	4	500	6	$G2$	5	800	6
$B3$	5	660	3	$G3$	6	50	3
$B4$	6	100	4	$G4$	4	700	4
$B5$	4	740	3	$G5$	8	680	3
$C1$	3	600	2	$H1$	4	700	2
$C2$	4	620	6	$H2$	5	720	6
$C3$	5	740	3	$H3$	6	50	3
$C4$	6	680	4	$H4$	4	760	4
$C5$	7	680	3	$H5$	8	780	3
$D1$	3	700	2	$I1$	4	800	2
$D2$	4	50	6	$I2$	6	840	6
$D3$	5	740	3	$I3$	6	50	3
$D4$	6	760	4	$I4$	7	860	4
$D5$	7	780	3	$I5$	8	880	3
$E1$	3	800	2	$J1$	4	900	2
$E2$	4	820	6	$J2$	5	920	6
$E3$	5	840	3	$J3$	3	940	3
$E4$	6	860	4	$J4$	3	100	4
$E5$	3	900	3	$J5$	4	980	3

Table 2: Reconfiguration 1: added tasks

Task	C_i	T_i	MFi	Task	C_i	T_i	MFi
$AA1$	3	205	2	$AB1$	3	215	2
$AA2$	7	245	6	$AB2$	5	235	6
$AA3$	4	215	3	$AB3$	6	245	3
$AA4$	7	255	4	$AB4$	6	235	4
$AA5$	5	225	3	$AB5$	4	225	3

Reconfiguration 2:

At a particular time, the system undergoes the second reconfiguration by adding of 10 tasks that are presented in Table 3:

After this reconfiguration scenario, the real-time constraint is not satisfied because U is greater than 1 ($U = 1.185 \geq 1$, $P = 1.402 \leq 1.667$ and $M = 252 \leq 450$). So, to correct this, we explore the solution A and the solution B detailed in Gammoudi et al. (2015).

If we apply solution A: After the modification of the periods T_i, the processor utilization is reduced and can satisfy the real-time scheduling: **U= 0,99**. Figure 3 illustrates the considered system of 70 tasks after changing periods by our solution A in Gammoudi et al. (2015) and by the proposed solution in Wang et al. (2014). Both solutions provide a change on the period of tasks. We define the cost of a solution as the total delay introduced to all periods. To evaluate the performance of our solution compared to the approaches in Wang et al. (2014), we present the following curves (Figure 3): The curve in blue is our solution and in red color is when applying the solution presented in Wang et al. (2014).

Table 3: Reconfiguration 2: added tasks

Task	C_i	T_i	MFi	Task	C_i	T_i	MFi
$AC1$	8	225	2	$AD1$	7	205	2
$AC2$	4	245	6	$AD2$	5	245	6
$AC3$	7	215	3	$AD3$	4	235	3
$AC4$	3	225	4	$AD4$	5	255	4
$AC5$	6	205	3	$AD5$	6	255	3

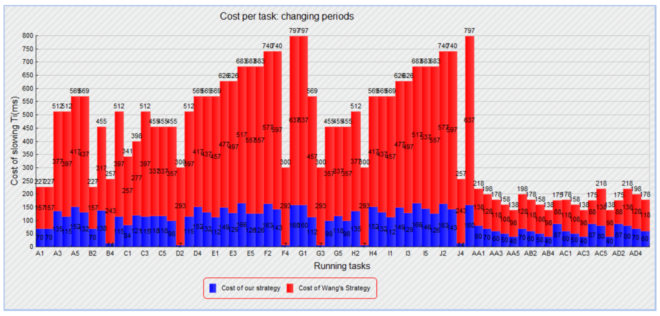


Figure 3: Cost of modification of periods T_i compared with Wang et al. (2014).

We know that the cost is a delay penalty for a task i of T^{New} - T_i. Therefore, the total cost for each approach is the sum of all these costs. After the execution of our strategy in Gammoudi et al. (2015), we note that the total cost is equal to $6940ms$, but the total cost by using the second strategy proposed by Wang et al. (2014) is equal to $23036ms$. Then, our solution is better than the solution presented in Wang et al. (2014). The introduced delay is only 30% ($\frac{6940}{23036} * 100 \simeq 30\%$) of the introduced delay by Wang et al. (2014).

If we apply solution B: After the modification of the WCETs C_i, the processor utilization is reduced and can satisfy the real-time scheduling: **U= 0,636**. According to Figure 4, we can notice that our solution is less costly also in case B than the Wang strategy in Wang et al. (2014).

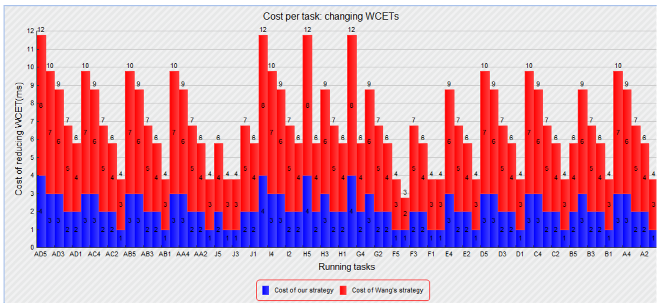


Figure 4: Cost of modification of WCETs C_i (solution B).

The total cost of our strategy is equal to $154ms$ but the total cost by using the second strategy defined in Wang

Table 4: Reconfiguration 3: added tasks

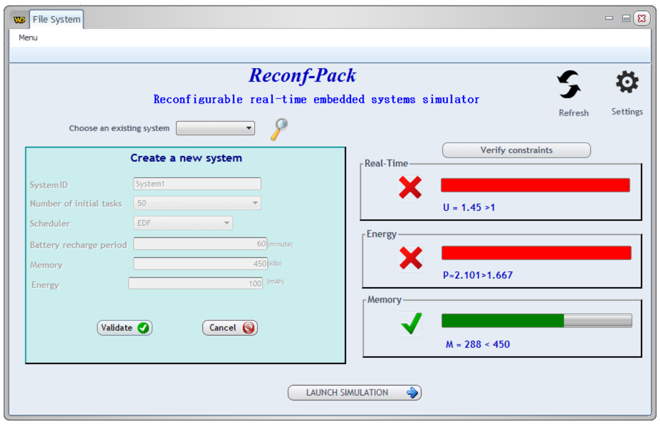
Task	C_i	T_i	MFi	Task	C_i	T_i	MFi	Task	C_i	T_i	MFi
$AA1$	3	205	2	$AB1$	3	215	2	$AE1$	4	225	8
$AA2$	7	245	6	$AB2$	5	235	6	$AE2$	5	205	11
$AA3$	4	215	3	$AB3$	6	245	3	$AE3$	6	215	5
$AA4$	7	255	4	$AB4$	6	235	4	$AE4$	7	225	11
$AA5$	5	225	3	$AB5$	4	225	3	$AE5$	8	235	8
$AC1$	8	225	2	$AD1$	7	205	2	$AF1$	4	205	2
$AC2$	4	245	6	$AD2$	5	245	6	$AF2$	5	215	1
$AC3$	7	215	3	$AD3$	4	235	3	$AF3$	6	225	5
$AC4$	3	225	4	$AD4$	5	255	4	$AF4$	7	235	1
$AC5$	6	205	3	$AD5$	6	255	3	$AF5$	8	245	2

et al. (2014) is equal to $326ms$. Then, our solution is better than the solution presented in Wang et al. (2014): The introduced delay is reduced to 45%.

Let us take the same system described previously (Table 1). We stress the energy constraint in the following. We assume that : P_{limit} $=$ **1,67 Watt**. The system is initially feasible.

Reconfiguration 3:
At a particular time, the system undergoes a reconfiguration by adding 30 tasks that are presented in the Table 4:

We notice that the value of U is greater than 1 and the value of P is greater than P_{limit} (Figure 5). The system is then no more feasible after this reconfiguration. Moreover, in this case it consumes all the available energy before the next battery recharge. We focus on the energy constraint and we execute solutions C and D to ensure that the system will run correctly until the next recharge.
If we apply solution C: After the modification of the periods of tasks T_i, the power consumption of the system is reduced and can satisfy the energy constraint: **P= 1,601 Watt**. **If we apply solution D:** After



System ID: System1
Number of initial tasks: 50
Scheduler: EDF
Battery recharge period: 60 (minute)
Memory: 450 (Kilo)
Energy: 100 (mAh)

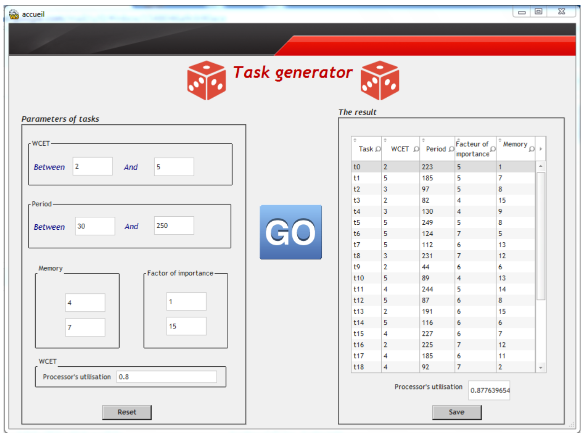
Real-Time: U = 1.45 >1
Energy: P=2.101>1.667
Memory: M = 288 < 450

Figure 5: System1's properties after the reconfiguration.

the modification of the WCETs of tasks C_i, the power consumption of the system is reduced and can satisfy the energy constraint: **P= 0,604 Watt**. Periodically recharged systems have not been studied in Wang et al. (2014).

Performance Evaluation

We presented in Section 5, the *Reconf-Pack* tool that implements the same case study in Wang et al. (2014) using two different theories and we showed that our strategy introduces less delay than the strategy detailed in Wang et al. (2014). Now, in order to generalize the performance evaluation of our strategy in Gammoudi et al. (2015), another tool called *Task-Generator* (Figure 6) has been developed to generate random systems and to randomly plan reconfiguration scenarios. We use then *Reconf-Pack* to apply for each system both strategies. Table 5 summarizes the test set of 50 systems generated by *Task-Generator*. The second column is the total delay cost when we apply our strategy in Gammoudi et al. (2015) and the third column is the total delay cost when applying the solution presented in Wang et al. (2014).



WCET: Between 2 And 5
Period: Between 30 And 250
Memory: 4, 7
Factor of importance: 1, 15
WCET: Processor's utilisation 0.8

Task	WCET	Period	Facteur of importance	Memory
t0	2	223	5	1
t1	5	185	5	7
t2	3	97	5	8
t3	2	82	4	15
t4	3	130	4	9
t5	5	249	5	8
t6	5	124	7	5
t7	5	112	6	13
t8	3	231	7	12
t9	2	44	6	6
t10	5	89	4	13
t11	4	244	5	14
t12	5	87	6	8
t13	2	191	6	15
t14	5	116	6	6
t15	4	227	6	7
t16	2	225	7	12
t17	4	185	6	11
t18	4	92	7	2

Processor's utilisation 0.877639654

Figure 6: Random task generator.

According to Table 5, our strategy introduces less delay in 88% of randomly generated tests. Moreover, the average delay introduced by our strategy to keep the system feasible and up until the next recharge is only 55% of the average delay introduced by Wang's strategy.

Table 5: Randomly generated systems and test results

System	*Total cost of our strategy (ms)*	*Total cost of Wang strategy (ms)*	*System*	*Total cost of our strategy (ms)*	*Total cost of Wang strategy (ms)*
$System1$	978	3524	$System26$	1054	3202
$System2$	630	1103	$System27$	11095	4257
$System3$	1917	2656	$System28$	11095	503
$System4$	1321	2465	$System29$	780	813
$System5$	313	644	$System30$	742	1354
$System6$	806	1828	$System31$	205	996
$System7$	428	395	$System32$	1220	999
$System8$	421	1407	$System33$	525	1003
$System9$	477	1200	$System34$	9306	9306
$System10$	647	1471	$System35$	1627	1697
$System11$	302	1065	$System36$	376	543
$System12$	307	1193	$System37$	327	801
$System13$	526	1229	$System38$	565	1438
$System14$	911	1449	$System39$	1136	2337
$System15$	1209	1209	$System40$	562	1578
$System16$	1171	1702	$System41$	177	228
$System17$	2554	2888	$System42$	495	256
$System18$	2951	2421	$System43$	483	2526
$System19$	1765	1889	$System44$	2048	4116
$System20$	1442	2862	$System45$	683	3526
$System21$	2522	2903	$System46$	1978	4127
$System22$	1065	2788	$System47$	1237	3699
$System23$	653	1805	$System48$	502	1293
$System24$	422	3535	$System49$	1451	1496
$System25$	1435	5093	$System50$	2821	6661

Conclusion

In this paper, we study the functional feasibility in an architecture powered by a battery. We are interested in the tasks scheduling after reconfiguration scenarios. To ensure that the system is feasible after any reconfiguration scenario and does not consume its battery before the next recharge, we propose a strategy based on grouping tasks in packs presented in Gammoudi et al. (2015). We developed two tools *Reconf-Pack* and *Task-Generator* to evaluate our solution and compare it to the solution in Wang et al. (2014). We show very interesting gains in terms of minimizing the introduced delay to task periods to keep the system feasible and up till the next recharge. We are planning to develop *Reconf-Pack* and *Task-Generator* open source tools to support and perform required comparisons to other related work. Thanks to these two tools, we make automatic and autonomous simulations. In our future work, we are interested in improving on ameliorating the strategy and generalizing it to multi-core real-time embedded systems.

REFERENCES

Allen R.; Douence R.; and Garlan D., 1998. *Specifying and analyzing dynamic software architectures.* In *Fundamental Approaches to Software Engineering.* Springer, 21–37.

Brennan R.W.; Fletcher M.; and Norrie D.H., 2001. *A holonic approach to reconfiguring real-time distributed control systems.* In *Multi-Agent systems and applications II.* Springer, 323–335.

Gammoudi A.; Benzina A.; Khalgui M.; and Chillet D., 2015. *New Pack Oriented Solutions for Energy-Aware Feasible Adaptive Real-Time Systems.* In *Intelligent Software Methodologies, Tools and Techniques.* Springer, 73–86.

Gharsellaoui H. and Ahmed S.B., 2014. *Real-Time Reconfigurable Scheduling of Sporadic Tasks.* In *Software Technologies: 8th International Joint Conference, ICSOFT 2013, Reykjavik, Iceland, July 29-31, 2013, Revised Selected Papers.* Springer, vol. 457, 24.

Gharsellaoui H.; Khalgui M.; and Ahmed S.B., 2012. *New Optimal Solutions for Real-Time Reconfigurable Periodic Asynchronous Operating System Tasks with Minimizations of Response Time.* In *International Journal of System Dynamics Applications (IJSDA).* IGI Global, vol. 1, 88–131.

Khalgui M., 2010. *NCES-based modelling and CTL-based verification of reconfigurable embedded control systems.* In *Computers in Industry.* Elsevier, vol. 61, 198–212.

Khalgui M.; Mosbahi O.; Li Z.; and Hanisch H.M., 2011. *Reconfigurable multiagent embedded control systems: From modeling to implementation.* In *Computers, IEEE Transactions on.* IEEE, vol. 60, 538–551.

Liu C.L. and Layland J.W., 1973. *Scheduling algorithms for multiprogramming in a hard-real-time environment.* In *Journal of the ACM (JACM).* ACM, vol. 20, 46–61.

Rooker M.N.; Sünder C.; Strasser T.; Zoitl A.; Hummer O.; and Ebenhofer G., 2007. *Zero Downtime Reconfiguration of Distributed Automation Systems: The epsilonCEDAC Approach.* In *HoloMAS.* Springer, 326–337.

Shin Y. and Choi K., 1999. *Power conscious fixed priority scheduling for hard real-time systems.* In *Design Automation Conference, 1999. Proceedings. 36th.* IEEE, 134–139.

Wang X.; Khalgui M.; and Z.Li, 2011. *DYNAMIC LOW POWER RECONFIGURATIONS OF REAL-TIME EMBEDDED SYSTEMS.* In *Proc. 1st Pervas. Embedded Comput. Commu.Syst, Portugal.* 6.

Wang X.; Khemaissia I.; Khalgui M.; and Z.Li, 2014. *Dynamic Low-Power Reconfiguration of Real-Time Systems With Periodic and Probabilistic Tasks.* In *Automation Science and Engineering, IEEE Transactions.* vol. 258-271.