New Reconfigurable Middleware for Adaptive RTOS in Ubiquitous Devices

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

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

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

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

***Abstract*—Energy management is a central problem in battery powered real-time systems design, in particular for periodically reconfigurable embedded wireless devices. This kind of systems can be more or less intensive in computing, but must remain alive until the next recharge. They are not always critical, or at least some treatments are not critical. In this case, modification on tasks parameters of non-critical parts of the system can be done to increase the autonomy of the battery. The objective of this work is to develop a software plugin, called *Reconf-Middleware*, which corresponds to a software layer to be placed above the Operating System (OS). The main role of this software layer is to manage tasks execution for reconfigurable architecture when the battery recharges are done periodically. We integrate also a new scheduling strategy to ensure that the system will run correctly, after any reconfiguration scenario, under memory, real-time and energy constraints until the next recharge. This software component is designed to execute and evaluate the performance, reliability and correctness of some real-time scheduling approaches, which are theoretically validated. The middleware can be integrated into many operating systems and provides good quality both in terms of execution time and energy consumption. We discuss the paper's contribution by analyzing the experimental results that we did on a running example. We propose in this paper a new middleware to be placed above the operating system.**

Keywords–RTOS; Ubiquitous device; Reconfiguration; Real-Time and Low-Power Scheduling; Energy-aware.

I. INTRODUCTION

Reconfigurable real-time embedded systems are used in many application domains, manufacturing process control, telecommunications, robotics, sensor networks, ubiquitous devices and consumer electronics. In all of these areas, there is rapid technological progress, yet, energy concerns are still the bottleneck. In this context, we focus on reconfigurable real-time embedded systems when the battery recharges are done periodically. The minimization of energy consumption is an important criterion for development of rechargeable real-time embedded systems due to limitations in the capacity of their batteries. In addition, battery life can be extended by reducing power consumption [1]. When undergoing a reconfiguration, to reduce the energy consumption, these systems have to be changed and adapted to their environment without any disturbance. Any reconfiguration scenario may increase energy consumption and/or cause some software tasks to violate their deadlines. Concerning the reconfiguration, two policies are defined in the literature: i) Static reconfigurations [2] to be generally applied off-line; ii) Dynamic reconfigurations [3] that can be applied at run-time. Dynamic reconfiguration is important in embedded systems, where one does not necessarily have the luxury to stop a running system. For these reasons, we consider here dynamic reconfiguration and we 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*. However, development and design of high quality of scheduling middleware for real-time environment is difficult and complex, as it demands several requests such as system implementation, validation and optimization. The recent advance of middleware technologies, that enables communication and coordination in a computing system provides the perfect way to implement real-time middleware solutions. The general goal of this paper is to design a new middleware, called *Reconf-Middleware*, able to reconfigure the execution of the application tasks and to ensure that any reconfiguration scenario changing the implementation of the embedded platform does not violate real-time constraints and does not result in fatal energy over consumption or in memory saturation. A middleware is a software layer located above the operating system. It communicates with it and exploits its functionality to support the development of many reliable solutions. However, the architectures of most of the middleware solutions, do not offer the predictability required to support the real-time behavior in new complex systems, or the reconfigurability required for these middleware solutions to be integrated into various Real-Time Operating System (RTOS). To manage tasks on a reconfiguration architecture, RTOS plays an important role in the system.

As a major contribution of this paper, to respect the memory, real-time and energy constraints, a new middleware is defined where after each reconfiguration scenario, suitable

 ISBN: 978-1-61208-505-0

and acceptable modifications are performed on parameters of tasks. *Reconf-Middleware* presents a middleware implemented in RTLinux and describes the transition from the theory to the actual implementation. We implement in *Reconf-Middleware* a new original methodological strategy 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 paper is organized as follows. Section II presents the background of RTOSs for embedded architectures. Section III explains the strategy formalization. The fourth section presents the reconfiguration of tasks with the proposed run-time strategy and the operating mode of *Reconf-Middleware*. We present the Unified Modeling Language (UML) design models for *Reconf-Middleware* in Section V. In Section VI, we present the performance evaluation of the compared techniques presented by Wang et al. [3] and Wang et al. [1]. Finally, we conclude and present our future work in Section VII.

II. Background

Rechargeable reconfigurable embedded systems are composed of a variety of different processing elements, memories, Input/Output devices, sensors, battery, and so forth. The choice of processing elements includes instruction-set processors, application specific fixed-function hardware, and reconfigurable hardware devices. Several distinguished studies deal with rechargeable reconfiguration systems [4][5][6][7]. This type of systems can execute different reconfiguration scenarios at a particular time *t*. A reconfiguration scenario means the addition, removal or update of tasks in order to manage the whole system at the occurrence of hardware/software faults, or also to improve its performance at run-time. When such a scenario is applied, the system risks a fatal increase in energy consumption, a violation of real time constraints or a memory saturation. In this context, the real-time operating system is required to provide services for memory management, energy management, task scheduling and reconfiguration. To ensure that the system will run correctly until the next recharge, several interesting studies have been proposed in recent years for this kind of real-time operating systems. In this section, we decompose the state of the art into groups, the first corresponding to the work on scheduling of embedded systems with rechargeable periods, the second group concerns scheduling algorithm of embedded system without battery recharges.

A. Real-Time Scheduling

1) Real-Time Scheduling for Embedded Systems with Rechargeable Battery: Real-time scheduling has been extensively studied in the last three decades [8]. These studies propose several feasibility conditions of the dimensioning of real-time systems. These conditions are defined to enable a designer to guarantee that time constraints associated with an application are always met for all possible configurations. Two main classical scheduling are generally used in real-time embedded systems: Rate Monotonic (RM) and Earliest Deadline First (EDF). Several studies have been performed in this context, such as the research works reported in [9][10][11]. Chetto et al. [5][11] are interested in a real-time embedded system that is powered through a renewable energy storage device. They present a scheduling framework called $EDeg$ (Earliest Deadline with energy guarantee) and an exact feasibility test that decides for periodic task sets. EDeg is a variation of EDF able to cope with energy constraints. These studies are interesting, but the authors didnt consider neither the reconfiguration problems, nor the aperiodic tasks. In addition, the authors of the papers [5] and [11] have not studied the rechargeable systems with a well-defined period of recharge.

2) Real-Time Scheduling for Reconfigurable Architectures: Nowadays, a fair amount of research has been done to develop reconfigurable embedded systems. Wigley and Kearney [12] present one of the first attempts to develop an OS dedicated to the management of reconfigurable resources. Steiger et al. [13] discuss the design issues for reconfigurable hardware operating systems and the problem of on-line scheduling of hard real-time tasks for partially reconfigurable devices. They also developed two on-line scheduling heuristics in order to ensure that the system will respect the real-time feasibility. Merino et al. [14][15] split the reconfigurable area into an array of predefined subareas, so-called slots. The operating system schedules tasks to these slots based on a task allocation table that keeps track of currently loaded tasks. As each task fits into one slot, there is again no placement problem involved. Wang et al. [3] propose a study for feasible low-power dynamic reconfiguration of real-time systems where additions and removals of real-time tasks are applied at run-time. They aim to minimize the energy consumption after any reconfiguration scenario. This effort is continued by Khemaissia et al. [16] who propose an intermediate layer to play the role of middleware that will be in interaction with the kernel Linux. This layer will manage the addition/removal/update of the periodic and also aperiodic tasks sharing resources and with precedence constraints. These tasks should respect their deadlines after any reconfiguration scenario. The proposed middleware will divide the tasks into several virtual processors as time slots. The decomposition is done based on the task's category. The first virtual processor executes dependent periodic tasks, the second one executes dependent aperiodic tasks with hard deadlines and the third virtual processor executes dependent aperiodic tasks with soft deadlines. After applying a reconfiguration scenario, some tasks may miss their deadlines and the power consumption may increase. In order to re-obtain the feasibility of the system after such scenario, an agent-based-architecture is defined to modify the parameters of the tasks. The studies of [3][16] present a simple run-time strategy that reduces the energy consumption. They propose to modify the tasks period T_i, assigning a single value to all tasks, which is not reasonable in practice [1]. Another solution proposed is to reduce the Worst Case Execution Time (WCETs) C_i assigning a single value to all tasks, which is not reasonable in practice [1]. The formulas proposed by Wang et al. [1] and Wang et al. [3] are simple with soft calculation, 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 requirement. Moreover, if tasks have very diverse periods T_i, tasks that have small periods will be too much affected if they will be aligned with tasks that have large periods. Although these rich and useful contributions provide interesting results, no one is reported to address the problem of dynamic reconfigurations of real-time systems under battery with a periodic recharges and memory constraints. To address this problem, we propose a new middleware *Reconf-Middleware* that implements the

methodological strategy proposed by Gammoudi et al. [17]. The principle of this strategy is to evaluate system and battery states and 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.

B. RTLinux

The contribution of this paper can be applied for a large number of RTOS. We choose to implement it on RT-Linux since it is an open source. It is a hard real-time RTOS microkernel that runs the entire Linux operating system as a fully preemptive process. Fig. 1 depicts the design of the RT-Linux system. Important aspects are displayed in Figure 1:

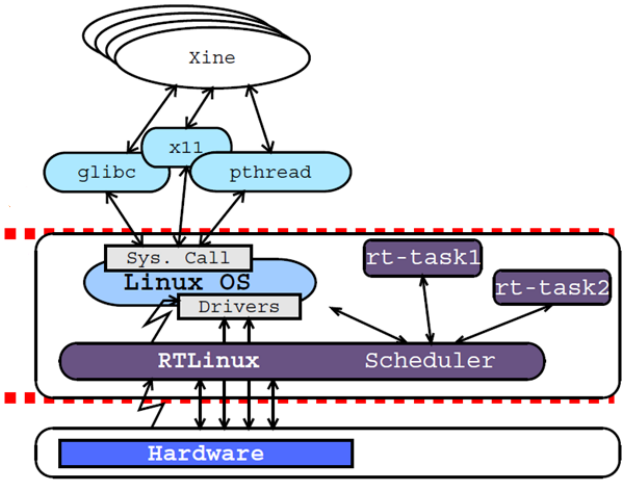

Figure 1. RTLinux system [18].

1) RT-Linux sits between the real hardware and the kernel,
2) It acts as the hardware for the kernel,
3) It treats the kernel as a single big process. RT-Linux receives the interruptions of the hardware layer and sends them to the kernel linux after converting them into software interruptions [18]. Also, the RT-LINUX manages the scheduling of the real-time tasks. According to [19], the kernel is not designed to be reconfigurable.

In this research, we suggest to create a middleware layer *Reconf-Middleware* between the RT-Linux and the hardware. *Reconf-Middleware* will be in interaction with the kernel Linux.

The basic functionality of RTLinux is initialized in a system by inserting five modules into the kernel: rtl.o, rtl_time.o, rtl_posixio.o, rtl_fifo.o and rtl_sched.o.

III. FORMALIZATION

We recall in this section the task and energy models with their characteristics and scheduling constraints [17]. We continue with a description of the memory model. Finally, we define the reconfiguration problem and state our goals.

A. Task Model

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 processor. For this reason, the context switching will add more time to the overall execution time when preemption is used. According to [20], 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 $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$ [20]. 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.

B. Energy Model

We consider the following energy model as described by Wang et al. [3] and Gammoudi et al. [17]. 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 [21]. Then, the power consumption is calculated by:

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

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:

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

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.

 ISBN: 978-1-61208-505-0

C. 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.

D. Problem Formalization

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.

IV. Pack Oriented Solution

A. Pack Model

In this section, we present a brief summary about different approaches proposed by Wang et al. [3] and Wang et al. [1] to solve this problem. We also discuss the strategy presented by the authors in [17]. This study is necessary to show the interest of the approach in [17] compared to [3]. Wang et al. [1][3], 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 proposed in a previous paper [17] 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. Let us note 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 period is $j * T^{New}$. So the cost for each task of Pk_j is: (T^{New}-(T_i mod T^{New})) mod T^{New}. The total cost for the approach is the sum of all these costs. We compare this strategy in [1][3][17] and show that the cost of delaying tasks is significantly improved. To ensure that the system is feasible after each reconfiguration scenario, we present the following five solutions (Sol A, Sol B, Sol C, Sol D and Sol E) detailed and justified in [17]. The first two solutions can be applied in order to ensure that the system satisfies the real-time constraint. Sol C and Sol D are used if the energy constraint is not satisfied. 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. Sol E is used to remove less important tasks according to the importance factor I_i in order to minimize the energy consumption.

B. Operating Mode

Thanks to *Reconf-Middleware*, the system is able to be adapted after any reconfiguration scenario. To satisfy the memory, real-time and energy constraints after any reconfiguration scenario, *Reconf-Middleware* 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, then this algorithm ensures a deterministic choice between the solutions A, B, C, D and E. Fig. 2 explains this strategy step-by-step.

V. Contribution: Reconfigurable Middleware

The objective of this section is to integrate the pack-based solution in an RTLinux in order to ensure that the system runs correctly after any reconfiguration scenario.

A. Architecture

We present in Fig. 3 the different services of a classical OS and the additional reconfiguration services. Let us explain some services: i) Memory service: Keeps track of the status of each memory location, either allocated or free. It determines how memory is allocated by processes, decides which one gets memory. When memory is allocated, it determines which memory locations will be assigned. It tracks when the memory is freed or unallocated and updates the status. ii) Garbage collector or just collector: It attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program. iii) Scheduler and Dispatcher: A scheduler is a component that schedules the system's tasks on a processor. Another component that is involved in the CPU-scheduling function is the dispatcher, which is the module that gives control of the CPU to the process selected by the short-term scheduler. It receives control in kernel mode as the result of an interrupt or system call. iv) Battery service: It is a service that communicates with the battery component and retrieves the level of the system's battery. v) We add another service *Reconfiguration Service* that communicates with the other OS services to apply the proposed strategy that will satisfy the three constraints.

B. Middleware Design

Several research studies [22][23][24] have focused on the UML model of reconfiguration operating system. We present

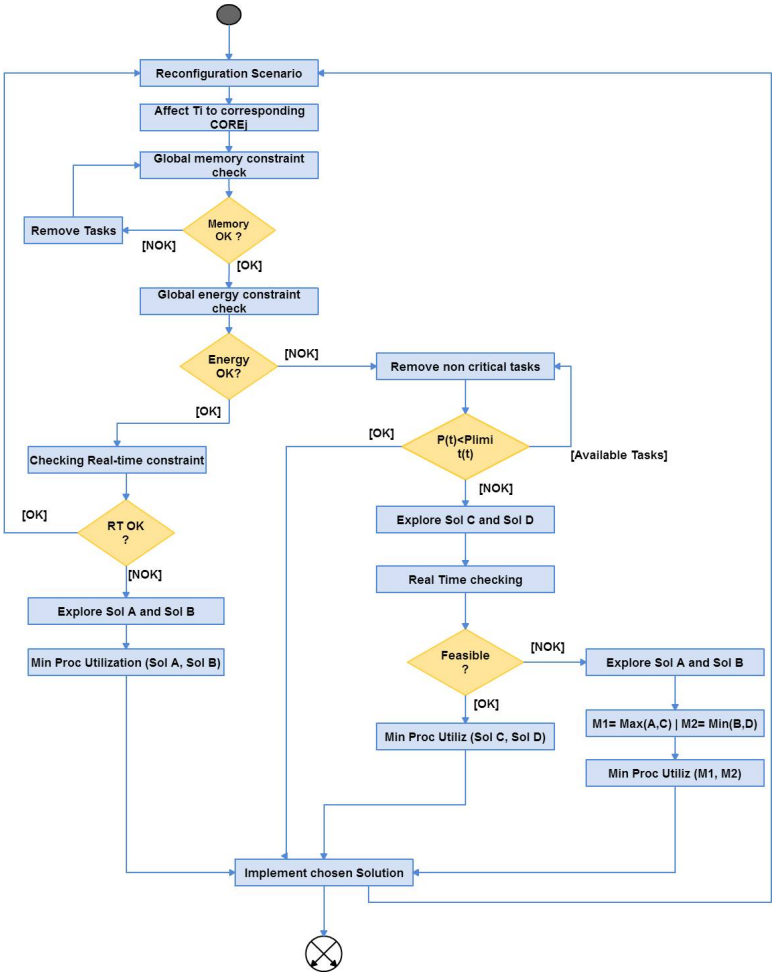

Figure 2. Activity diagram of strategy.

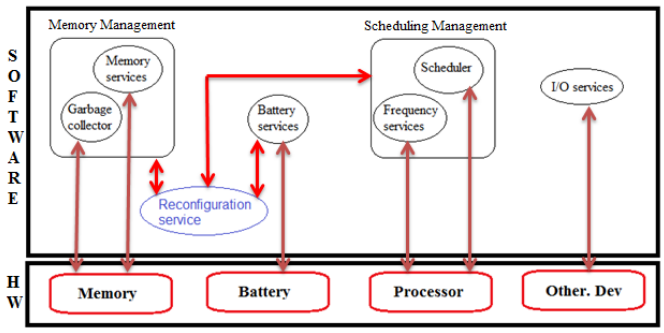

Figure 3. OS services.

in this section the UML [25] class diagram for our middleware: The hardware and software elements and the interactions between them. The diagram (Fig. 4) is divided into three layers: the software layer, the hardware layer and the reconfiguration layer.

Software Layer: It presents the different tasks (periodic and aperiodic tasks). The task scheduling is ensured by this layer. In this paper, we focus on RM and EDF policies. Task class is specified through the typical parameters used in the real-time context: Identification, worst case execution time, importance factor, memory footprint and whether the task is periodic or not. In order to store any interaction between tasks, we define DataAccess class that ensures data storage. The amount of memory needed by each task is indicated in parameter *Data* and its size in *Size*.

Hardware Layer: It contains classes representing the hardware components that are physically implemented in the platform. It includes the processor, battery and memory.

Reconfiguration Layer: It ensures that the system will run correctly after any reconfiguration scenario. Reconfiguration class receives different reconfiguration scenarios taht can violate one or more of the three constraints: real-time feasibility, memory and energy constraints. The Manger class proposes quantitative techniques to modify periods, reduce execution times of tasks or remove some of them to ensure the real-time feasibility, avoiding memory overflow and ensuring a rational use of remaining energy until the next recharge. The PackConst class ensures the grouping of tasks that have “similar” periods or WCETs in several Packs. This idea is formalized in [17].

C. Middleware Overview

The proposed middleware will have the role to reconfigure the OS. Many routines are added to RTLinux in order to apply

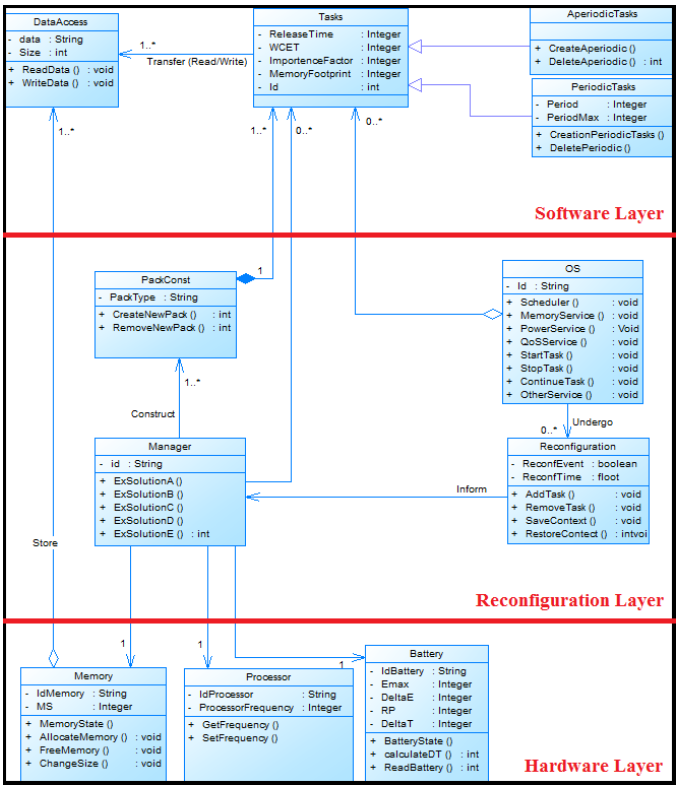

Figure 4. Class Diagram.

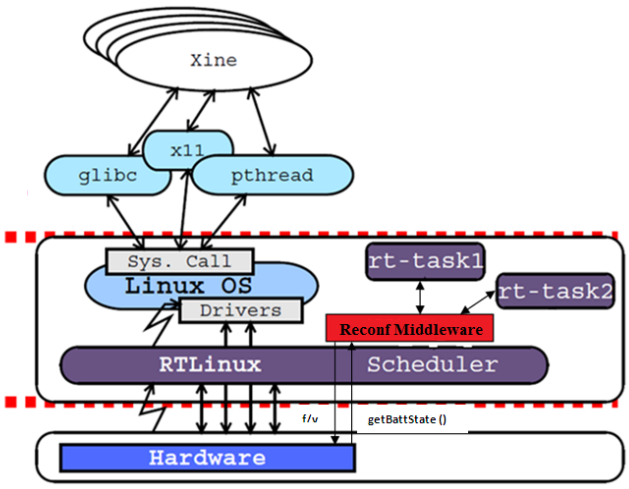

Figure 5. The middleware location.

the different proposed solutions. A routine is a service offered by RTLinux. Fig. 5 represents the Plugin integration. The Plugin will interact with the kernel and the hardware platform.

D. Implementation

The developed middleware *Reconf-Middleware* presents the different possible solutions given by the proposed run-time strategy. After any reconfiguration scenario, our plugin ensures the verification of three constraints (Real-Time, Energy, Memory) using the following functions:

- To obtain the battery level: getBattState()
- To create a task by modifying the period T: int pthread-make-periodic-np(pthread-t thread, hrtime-t start-time, hrtime-t period), Where: pthread-make-periodic-np marks the thread as ready for execution. The thread will start its execution at start-time and will run at intervals specified by a period given in nanoseconds.

The scheduler of RTLinux uses the rtl_sched.h library to ensure that all tasks are schedulable. The data structure of task τ_i is:

```
struct rt_task_struct {
int *stack;
int uses_fp;
int magic;
int state;
int *stack_bottom;
int priority;
RTIME period;
RTIME resume_time;
struct rt_task_struct *next;
RTL_FPU_CONTEXT fpu_regs;
}
```

The data structure that is responsible to store the battery data is:

struct battstate { short unsigned int powerstate, time_hour, time_min; float chargelevel; };

To obtain the current battery state, we use the getBattState() function, which is the pointer on the structure of struct battstate.

VI. Evaluation of Performance

In order to evaluate the performance of *Reconf-Middleware*, we implement the same case study presented by Wang et al. [3]. The initial system is feasible with low-power constraint. Then, we add some periodic tasks in order to violate the real-time constraint. To re-obtain the system feasibility, *Reconf-Middleware* must execute Sol A or Sol B. The cost of a solution is the total delay introduced to periods T_i or to WCETs C_i as explained in IV-A.

If we apply the 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**. Fig. 6 illustrates the considered system of 70 tasks after changing periods by our solution A [17] and by the proposed solution in [3]. Both solutions provide a change on the period of tasks. To evaluate the performance of our solution compared to the approaches in [3], we present the following curves (Fig. 6): The histogram in red is the cost of our solution and the blue one is when applying the solution presented in [3].

As presented in Section IV-A, 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 [17], we note that the total cost is equal to 6940*ms*, but the total cost by using the second strategy [3] is equal to 23036*ms*. Then, our solution is better than that presented in [3]. The introduced delay is only 30% ($\frac{6940}{23036} * 100 \simeq 30\%$) of the introduced delay in [3].

If we apply the 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 (Fig. 7), we

 ISBN: 978-1-61208-505-0

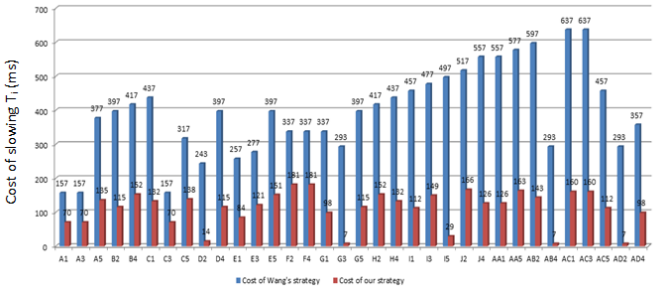

Figure 6. Cost of modification of periods T_i compared with [3].

can notice that our solution is less costly also in case B than the Wang strategy in [3].

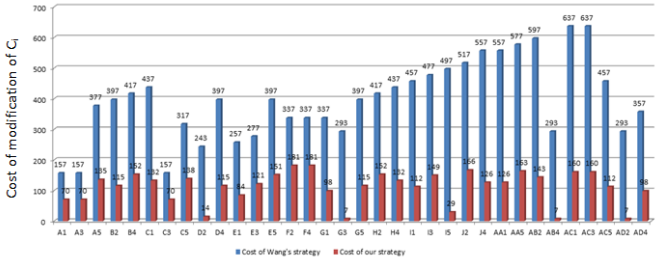

Figure 7. Cost of modification of WCETs C_i (solution B).

The total cost of our strategy is equal to 154*ms* but the total cost by using the second strategy defined in [3] is equal to 326*ms*. Then, our solution is better than the solution presented in [3]: The introduced delay is reduced to 45%.

VII. Conclusion and Future Work

Many solutions and scheduling algorithms have been proposed, however, few of them are implemented in real-time systems. To be used and evaluated, theoretical solutions must be deployed into a real-time system. Unfortunately, just few real-time scheduling middleware have been developed to date and most of them require the use of a specific simulation language. In this paper, we are interested in reconfigurable real-time embedded systems when the battery recharges are done periodically. We propose in this paper a new middleware to be placed above the operating system. Thanks to the strategy implemented in this middleware, the system can run correctly after any reconfiguration scenario. We are interested in improving the strategy and generalizing it to multi-core real-time embedded systems.

References

[1] X. Wang, M. Khalgui, and Z.Li, "Dynamic low power reconfigurations of embedded real-time systems," In: Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems, Portugal, vol. 6, 2010.

[2] C. Angelov, K. Sierszecki, and N. Marian, "Design models for reusable and reconfigurable state machines," in L.T. Yang et al. (Eds.): Proc. of EUC 2005, LNCS 3824, vol. 12, 2005, pp. 152–163.

[3] X. Wang, I. Khemaissia, M. Khalgui, Z.Li, O. Mosbahi, and M. Zhou, "Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks," IEEE Transactions on Automation Science and Engineering, vol. 14, 2014, pp. 258 – 271.

[4] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable batteries."

[5] H. Ghor, M. Chetto, and R. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," Int. Journal of Computers and Electrical Engineering, vol. 26, 2010.

[6] E. Camponogara, A. Oliveira, and G. Lima, "Optimization-based dynamic reconfiguration of real-time schedulers with support for stochastic processor consumption," Industrial Informatics, IEEE Trans. Ind. Informat, vol. 16, 2010, pp. 594–609.

[7] L. George and P. Courbin, "Reconfiguration of uniprocessor sporadic real-time systems: The sensitivity approach," in Book Chapter in IGI Global Knowledge on Reconfigurable Embedded Control systems: Applications for Flexibiliy and Agility, vol. 17, no. 167–189, 2011.

[8] S. Baruah and J. Goossens, "Scheduling real-time tasks: algorithms and complexity," Handbook of Scheduling: Algorithms Models and Performance Analysis, vol. 38, 2003.

[9] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," ACM SIGOPS Operating Systems Review, vol. 37, no. 5, 2003, pp. 149–163.

[10] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," Proceedings of the 2001 international symposium on Low power electronics and design, 2001, pp. 46–51.

[11] M. Chetto and H. El Ghor, "Real-time scheduling of periodic tasks in a monoprocessor system with a rechargeable battery," 2009, p. 45.

[12] G. Wigley and D. Kearney, "The first real operating system for reconfigurable computers," Computer Systems Architecture Conference, vol. 9, no. 1-8, 2001, pp. 130–137.

[13] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks," IEEE Transactions on Computers, vol. 15, 2004, pp. 1393–1407.

[14] P. Merino, J. Lopez, and M. Jacome, "A hardware operating system for dynamic reconfiguration of fpgas," Proc. Int. Workshop Field-Programmable Logic and Applications From FPGAs to Computing Paradigm, vol. 5, 1998, pp. 431–435.

[15] P. Merino, M. Jacome, and J. Lopez, "A methodology for task based partitioning and scheduling of dynamically reconfigurable systems," Proc. IEEE Symp. FPGAs for Custom Computing Machines, vol. 2, 1998, pp. 324–325.

[16] I. Khemaissia, O. Mosbahi, M. Khalgui, and W. Bouzayen, "New reconfigurable middleware for feasible adaptive rt-linux," Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, vol. 10, no. 1–10, 2014.

[17] A. Gammoudi, A. Benzina, M. Khalgui, and D. Chillet, "New pack oriented solutions for energy-aware feasible adaptive real-time systems," The 14th International Conference on Intelligent Software Methodologies, Tools and Techniques, vol. 14, no. 1–14, 2015.

[18] F. Lab, "Getting-started-with-rtlinux," vol. 42, 2001.

[19] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino, "An edf scheduling class for the linux kernel," 2009.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM (JACM), USA, vol. 15, 1973, pp. 46 – 61.

[21] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," Design Automation Conference, 1999. Proceedings. 36th, 1999, pp. 134–139.

[22] F. Verdier, J. Prvotet, a. Benkhelifa, D. Chillet, and S. Pillement, "Exploring rtos issues with a high-level model of a reconfigurable soc platform," vol. 8, 2010.

[23] P. Hastono, S. Klaus, and S. Huss, "An integrated systemc framework for real-time scheduling assessments on system level," In 25th IEEE Int. Real-Time Systems Symposium (RTSS 2004), vol. 4, 2004.

[24] I. Benkermi, "Model and scheduling algorithm for dynamic reconfigurble architectures," Thesis in Computer Science, IRISA, Rennes1 University, France, vol. 140, 2007.

[25] A. B. H. Ali, M. Khalgui, and S. B. Ahmed, "Uml-based design and validation of intelligent agents-based reconfigurable embedded control systems," International Journal of System Dynamics Applications (IJSDA), vol. 1, no. 1, 2012, pp. 17–38.

 ISBN: 978-1-61208-505-0