New Reconfigurable Middleware for Feasible Adaptive RT-Linux

Imen Khemaissia[1], Olfa Mosbahi[2], Mohamed Khalgui[2] and Walid Bouzayen[1]
[1]*Faculty of Sciences, Tunis El-Manar University, Tunis, Tunisia*
[2]*National Institute of Applied Sciences and Technology, INSAT, University of Carthage, Tunis, Tunisia*

Keywords: RT-Linux, Middleware, Reconfiguration, Low-power and Real-time, Agent-based Architecture.

Abstract: This paper is interested in reconfigurable real-time embedded OS for microprocessors. Our study concerns specifically RT-Linux. Since the latter is not designed to be reconfigurable, we propose to develop 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 hardware execution 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 a such scenario, an agent-based-architecture is defined to modify the parameters of the tasks. The different services offered by this middleware are developed. A simulation study is done in order to highlight the performance of our proposed solutions.

1 INTRODUCTION

Recently, the low-power became a major concern in embedded real-time operating systems RTOSs since the new generations of these systems are addressing a new criteria such as the low-power consumption. For the low power schedule, many research activities suggest the modification of the temporal parameters of tasks and the dynamic voltage scaling (DVS) of the processors. For the high performance of these systems, reducing the power consumption on the OS level and offering optimal real-time services are necessary. Many academic research are dedicated to the low-power reconfigurable RTOS (X. Wang and Li, 2011) (C. Angelov and Marian, 2005) (M. Khalgui and Hanisch, 2011) (Khalgui and Hanisch, 2011) (K. Thramboulidis and Frantzis, 2004) (George and Courbin,). A fair amount of interesting work for the schedule of tasks that share resources propose many algorithms and protocols (Liu and Layland,) (Burns and Wellings, 2001) (T.P.Baker, 1990). The main of this study is devoted to minimize the power consumption and to satisfy the real-time constraints of the RTOS after applying a reconfiguration scenario where the tasks are with precedence constraints and share resources. A reconfiguration can be hardware or software. A software reconfiguration is defined by the addition/removal/update of the tasks at run-time but the hardware one means the acivation/deactivation of a processor. This paper is original since there's no work have treated the low-power-reconfiguration of an embedded RTOS to be composed of periodic and aperiodic tasks under precedence constraints and sharing resources. This paper aims to reconfigure dynamically an RTOS by applying dynamic window-constrained scheduling (DWCS) (West and Schwan, 1999) (West and Poellabauer, 2000) (P. Balbastre and Crespo, 2002) (Niu, 2011). To our best knowledge, there is no other published paper utilizing DWCS to dynamically reconfigure a RTOS with periodic/aperiodic and real-time OS tasks under precedence and resources constraints. This study presents an intelligent agent that is a software unit controlling the evolution in both the system and its environment. It can dynamically decompose $Sys_{(i)}$ into three virtual processors, VP_1 , VP_2 and VP_3. These virtual processors are utilized to execute the periodic tasks, aperiodic tasks with hard deadlines, and aperiodic tasks with soft deadlines, respectively. All these tasks share resources and with precedence constraint. A new ap-

Khemaissia I., Mosbahi O., Khalgui M. and Bouzayen W..
New Reconfigurable Middleware for Feasible Adaptive RT-Linux.
DOI: 10.5220/0004752901580167
In *Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2014), pages 158-167
ISBN: 978-989-758-000-0

proach is developed to locate the different tasks according to their types. The tasks of different virtual processors are with different set-priorities (*SP*), which are assigned by labels. The tasks in VP_1 , VP_2 and VP_3 are assigned with labels $SP = 1$, $SP = 2$, and $SP = 3$, respectively. The label 1 represents the highest *SP*. The task with a lower *SP* can be preempted by the task with a higher *SP*, which can be used to switch the context. After any reconfiguration, some deadlines of the periodic/aperiodic tasks may be violated. The agent modifies the deadlines in order to reach the systems's feasibility. This work assumes that *Sys* can be reconfigured repeatedly. Before any reconfiguration scenario, Sys is feasible with low-power consumption, which is considered as $Sys^{(0)}$. Sys evolves into $Sys^{(i)}$ after the i_th reconfiguration. After any reconfiguration, power consumption may increase and real-time constraints may be violated. A presented intelligent control agent provides run-time solutions to reduce the power consumption while satisfying the real-time constraints dynamically. The solutions consider: Periods or WCETS modification of periodic tasks, λ_C modification of aperiodic tasks. Also, the agent can apply the m,k firm-based degraded reconfiguration where some deadlines can be allowed to be violated (Hamdaoui and Ramanathan, 1995) (Ramanathan, 1999). The agent also will calculate the idle periods of VP_1 to locate the aperiodic tasks. To our best knowledge, no work address the problem of a reconfigurable RT-Linux to be composed of periodic and aperiodic tasks sharing data and under real-time and low-power constraints. The contribution of this paper will be applied to RT-Linux. The different services offered by this middleware are developed. The rest of the paper is as follows: Section 2 deals with the state of the art. A background is defined in section 3. The case study is mentioned in section 4. Section 5 presents the task model and formalize the RT-Linux. Section 6 describes the agent-based architecture. The different proposed solutions are described in section 7. The architecture is implemented, simulated and analyzed in section 8. Finally, we summarize our work by a conclusion.

2 STATE OF THE ART

We summarize in this section the previous researches dealing with the reconfiguration of real-time embedded systems, the real-time scheduling, the low-power scheduling, the dynamic window-constrained scheduling and (m,k) firm model.

2.1 Real-time Embedded Reconfigurations

Several studies have been dedicated to develop reconfigurable real-time embedded control systems. We can find two reconfiguration policies in the available literature: static reconfigurations (C. Angelov and Marian, 2005) and dynamic reconfigurations (M. Khalgui and Hanisch, 2011; Khalgui and Hanisch, 2011). The research in (X. Wang and Li, 2011) focus on low-power dynamic reconfigurations of synchronous real-time embedded control systems. The author have proposed an agent-based architecture in order to reduce the power consumption after applying a reconfiguration scenario. The work in (K. Thramboulidis and Frantzis, 2004) deals with real-time unified modeling language (UML)-based meta-models that will be used between design models of tasks and their implementation models to support dynamic user-based reconfigurations of control systems. The research in (C. Angelov and Marian, 2005) suggests reusable tasks to implement a broad range of systems where each task is statically reconfigured without any re-programming. Different solutions for the reconfigurations of WCETs, deadlines, and periods of the tasks that are scheduled by FP or EDF are proposed in (George and Courbin,). To our best knowledge, no work addresses the problem of reconfigurable RT-Linux for dependent tasks under real-time and low-power constraints.

2.2 Real-time Scheduling

There is a lot of successful researches addressing the problem of scheduling of real-time tasks. The author in (Liu and Layland,) suggests the EDF and RM for the scheduling of periodic tasks. The well-known EDF is used to schedule the independent tasks. The original priority ceiling protocol OPCP and immediate priority ceiling protocol IPCP are presented in (Burns and Wellings, 2001) in order to manage the tasks that share resources. The author of (T.P.Baker, 1990) propose the stack resource policy SRP that allows processes with different priorities to share a single run-time stack. In this paper, we will use the EDF for the scheduling of periodic tasks. For the dependant tasks, we will order them according to their priorities in order to be independent. Then, we will use the EDF for the scheduling of periodic tasks and FIFO for the aperiodic tasks. In order to manage the sharing resources, we will use the IPCP protocol since it is more efficient. Moreover, this protocol is available in cheddar (F. Singhoff and Marce, 2004) that will be used to verify the feasibility of the system after any

reconfiguration scenario.

2.3 Dynamic Window-constrained Scheduling

R. West and K. Schwan are the first to propose the scheduling DWCS in (West and Schwan, 1999). The author in (West and Poellabauer, 2000) has proposed a DWCS-based algorithm which can be utilized to guarantee real-time constraints. The work in (P. Balbastre and Crespo, 2002) provide a window constrained-based method that aims to achieve the upper bound of the computation time of a task under EDF scheduling. A dynamic approach is proposed in (Niu, 2011) in order to minimize the power consumption of soft real-time systems by considering the QoS-guarantee. To our best knowledge, there is no other published paper that uses DWCS to dynamically reconfigure real-time aperiodic embedded control systems with asynchronous and periodic real-time OS tasks as the background tasks. In this work, DWCS is utilized in the RT-Linux to decompose the tasks within the system into different virtual processors.

2.4 Low-power Scheduling

In the literature, many algorithms are proposed to address the problem of low-power scheduling. Power-reduction techniques can be restricted into two categories: static (Shin and Choi, 1999), and dynamic (Quan and Hu, 2003). The work in (Ishihara, 2010) proposes an energy-efficient processor that can be used as an alternative design for the DVS processors in embedded system design. Based on the concept of the Nash bargaining solution in (George and Courbin, 2011), a processor's clock speed and supply voltage are dynamically adjusted to satisfy the conflicting performance metrics. The principles that can be implemented in periodic/aperiodic task sets considering battery-aware voltage scheduling algorithms are proposed in (T. Yokoyama and Takada, 2010). The work in (George and Courbin,) concludes the most achievement solutions for the reconfiguration of WCETs, deadlines, and periods of tasks scheduled in FP or EDF. Although all these related studies are interesting, the current research addresses low-power reconfigurations of real-time embedded control systems when dynamic reconfigurations of the periodic and aperiodic tasks are applied at run-time to save or improve the performance. This work aims to reconfigure dynamically real-time embedded control systems to be composed of dependent periodic/aperioic tasks sharing resources by applying dynamic window-constrained scheduling (DWCS).

2.5 (m,k)-firm Model

The (m,k)-firm model requires at least that m requests meet their deadline for any k consecutive requests of the task according to (Hamdaoui and Ramanathan, 1995). In (Ramanathan, 1999), Ramanathan proposes a scheduling approach that defines mandatory and optional partitions of tasks. Also this scheduling provides deterministic (m,k)-firm guarantee to each task. In this work, (m,k)-firm model will be used as a solution to get a feasible schedule after applying a reconfiguration scenario.

3 BACKGROUND

The definitions of several important terms used in the following chapters such as processor demands, workload demands, and busy periods are listed below (L. George and Spuri, 1996).

- **busy period:** A busy period is defined as a time interval $[a,b)$ such that there is no idle time in $[a,b)$ (the processor is fully busy) and such that both a and b are idle times, and
- **idle period:** An idle time t of a processor is defined as a time where no tasks released before time t are unfinished at time t. An interval of successive idle times is classically called an idle period.

The calculation of the processor demand and the workload demand in time intervals $[0,t)$ and $[0,t]$ are proposed in (S. Baruah and Rosier, 1990) as follows:

workload demand in $[0,t]$**:** $\overline{W}(t) = \sum_{i=1}^{n}(1 + \left\lfloor \frac{t}{T_j} \right\rfloor)C_j$ Moreover, according to (Baker, 1991), the processor utilization of periodic and aperiodic tasks that share resources is given by:

$$U = (\sum_{i=k}^{n} \frac{C_i}{D_i}) + \frac{B_k}{D_k}, \forall k \in [1..n] \quad (1)$$

4 CASE STUDY

The contribution of this paper can be applied for a large number of RTOS. We choose 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 by the figure:

1) RT-Linux sits between the real hardware and the kernel, 2) it acts as the hardware for the kernel and

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 to convert them into software interruptions (FSM Labs, 2001). Also, the RT-LINUX manages the schedule of the real-time tasks. According to (D. Faggioli and Scordino,), the kernel is not designed to be reconfigurable. Thus, RT-Linux is not reconfigurable. In this research, we suggest to create a middleware layer between the RT-Linux and the hardware. The middleware will be in interaction with the kernel Linux.

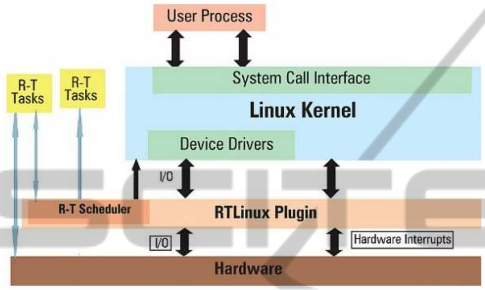

Figure 1: RT-LINUX system.

5 MODEL OF TASKS

The RT-Linux is assumed to be composed of a set of periodic and aperiodic tasks and a set of precedence relations and resource to be shared by these tasks. The different sets are as follows:

- Set_{per}: Set of the initial periodic tasks.
- Set_{pro}: Set of the initial aperiodic tasks.
- $Set_{precedpre}$: Set of precedence constraints between the periodic tasks
- $Set_{precedpro}$: Set of precedence constraints between the aperiodic tasks
- $List_{resource}$: List of resources in system *sys*

Each periodic task τ_i may produce many jobs (Liu and Layland,). It is described by:

- A release time R_i : It is the time when a job becomes available for execution. Since the tasks are synchronous, $R_i = 0$.
- A period T_i: is the regular inter$-$arrival time.
- A deadline D_i: The absolute deadline is equal to the release time plus the relative deadline.
- A WCET C_i: It is the time needed to compute a job.
- A static priority S_i: The highest static priority is equal to 1, i.e., $S_i = 1$ represents τ_i with the highest static priority.

Note that we consider that T_i=D_i. For the aperiodic tasks, we can characterize them by a deadline d_i, a release time r_i and a WCET c_i. Each aperiodic set tasks arrives with two rates λ_C and λ_r (I.Khemaissia, 2012). Since T_i=D_i, the processor utilization U_{per} of Set_{per} is given by:

$$U_{per} = (\sum_{k=1}^{n} \frac{C_k}{T_k}) + \frac{B_k}{T_k} \quad (2)$$

According to (I.Khemaissia, 2012), the processor utilization U_{pro} is formalized by $U_{pro} = \frac{\lambda_r}{\lambda_C}$ The initial processor utilization is the sum of U_{per} and U_{pro} as given by the following equation:

$$U_{bef} = U_{per} + U_{pro} \quad (3)$$

We assume that the system is scheduled by the well known EDF algorithm. For the tasks with precedence constraints, we turn them into independent ones according to (H. Chetto, 1989). To manage the precedence constraints, we assign firstly a priority Pr_i for each task. The priority of the tasks that have a precedence constraint are assigned as follows:

- $Pr_i < Pr_j$

The deadline D_i is equal to:

$$D_i = minD_i, (D_j - C_j), andD_j = D_j \quad (4)$$

Where τ_i and τ_j are dependant and τ_i is before τ_j. By Eq. (4), we guarantee that the precedence constraint will not be violated.

For the tasks that share resources, we have used the immediate priority ceiling protocol IPCP (Burns and Wellings, 2001).

Table 1: Initial tasks.

task	R	C	D/d	T
τ_1	0	1	10	40
τ_2	0	1	8	15
τ_3	0	4	20	30
τ_4	0	8	30	40
τ_{h1}	0	2	26	
τ_{h2}	0	1	30	
τ_{s1}	0	1	40	

The reconfigurable RT-Linux is illustrated through a running example shown by the table 1. It contains four synchronous tasks and an aperiodic task set consisting of three tasks, τ_{h1}, τ_{h2}, and τ_{s1}, with the distributions $\lambda_C = 0.5$ and $\lambda_r = 0.2$. We consider $Pr(\tau_1) < Pr(\tau_2)$, $Pr(\tau_4) < Pr(\tau_3)$ and $Pr(\tau_{h1}) < Pr(\tau_{h2} < Pr(\tau_{s1})$. To manage the dependent tasks, the deadlines of the periodic tasks are calculated as follows:

- $D_1 = min(40, 15-1) = 14$
- $D_2 = 15$
- $D_3 = 30$
- $D_4 = min(40, 30-4) = 26$

The new deadlines of the aperiodic tasks are:

- $D_{h1} = min(26, 30-1) = 26$
- $D_{h2} = min(30, 40-1) = 30$
- $D_{s1} = 40$

All the tasks share a unique resource X. The blocking factor B is assumed to be equal to 1. By applying Eq. (2), the processor utilization of periodic tasks U_{per} is equal to 0.574. The processor utilization of aperiodic tasks is equal to 0.4. Thus, the total processor utilization is equal to 0.974. By applying Cheddar (F. Singhoff and Marce, 2004), Fig. 5 presents the low-power scheduling result of *Sys*. The system is feasible and the real-time constraints are respected

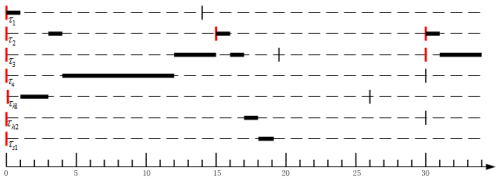

Figure 2: Scheduling of the initial system.

The energy is proportional to the processor utilization according to (X. Wang and Li, 2011).

$$P_i \propto U_i \tag{5}$$

A reconfiguration scenario is the addition/removal/update of periodic and aperiodic tasks. If, we add a dependant task, we should add a precedence relation. If we delete a task, we should delete the correspondent precedence relation.

The initial configuration is denoted as $Sys^{(0)}$. *Sys* evolves into $Sys^{(i)}$ after the i_{th} reconfiguration scenario at run-time by adding/removing/updating some periodic/aperiodic tasks. The new processor utilizations after the reconfiguration scenario $U_{per}^{(ia)}$ and $U_{pro}^{(ia)}$ are given as follows:

$$U_{per}^{(ia)} = \sum_{j=1}^{n^{(i)}} U_{per}^{j} \tag{6}$$

and

$$U_{pro}^{(ia)} = \sum_{j=0}^{i} \frac{\lambda_{rj}}{\lambda_{Cj}} \tag{7}$$

The total processor utilization after the reconfiguration scenario is equal to:

$$U^{(ia)} = U_{per}^{(ia)} + U_{pro}^{(ia)} \tag{8}$$

Table 2: Added tasks.

task	R	C	D/d	T
τ_5	0	5	18	20
τ_6	0	6	20	25
τ_{h3}	0	2	25	
τ_{s2}	0	1	30	

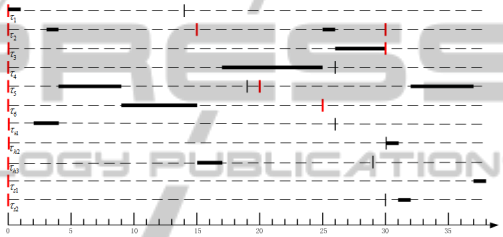

Figure 3: Scheduling of the system after tasks addition.

Table 2 indicates two periodic tasks and two aperiodic tasks satisfying $\lambda_{C1} = 0.4$ and $\lambda_{r1} = 0.1$. τ_6 must be after tau_5. After their addition to *Sys*, U_{per} and U_{ape} increase to be = 1.154 and 0.65. The deadlines of the added periodic tasks are as follow:

- $D_5 = min(20, 25-6) = 19$
- $D_6 = 25$

The deadlines of the added periodic tasks are as follow:

- $D_{h3} = min(25, 30-1) = 29$
- $D_{s2} = 30$

Then, the new processor utilization of $Sys^{(1)}$ is equal to 1.431. Moreover, some tasks don't meet their deadlines for example τ_{h2} completes its execution at 31 and its deadline is equal to 30. Then, it is obvious that the system is infeasible and the task set is not schedulable. The scheduling result at the full speed is shown in Fig. 3.

6 DYNAMIC WINDOW-CONSTRAINED-BASED ARCHITECTURE LOW POWER

In order to re-obtain the feasibility of the system after any reconfiguration scenario, an agent based architecture is defined to propose new technique solutions.

By utilizing DWCS after applying a reconfiguration scenario, three virtual processors are provided

dynamically to reconfigure $Sys^{(i)}$ in order to re-obtain a feasible system with low-power. First of all, this agent will create the virtual processors. Then, modify the parameters of tasks. After that it calculates the busy periods of VP_1. The latter will be used to calculate the idle periods to locate the aperiodic tasks.

6.1 Virtual Processors

The agent decomposes the execution of the hardware processor into into three virtual processors VP_1, VP_2, VP_3. Each virtual processor executes a category of tasks as follows:

- VP_1 : executes the dependent periodic tasks
- VP_2 : executes the dependent aperiodic tasks that share resources.
- VP_3 : executes the dependent aperiodic tasks with hard deadlines.

Where:
VP_1 is with the highest priority and VP_3 is with the lowest priority. The different virtual processors are shown in Fig. 4

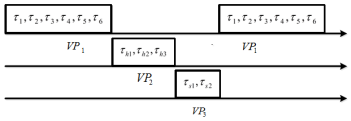

Figure 4: The virtual processors.

6.2 Busy Period and Idle Period Calculation

In this section, we will develop an approach in order to execute the different tasks. We start by calculating, the busy periods of VP_1.

According to (L. George and Spuri, 1996), the processor busy period of $Set_{syn}^{(i)}$ can be calculated by the workload demand in a time interval $[0,t)$ iteratively due to the following equation:

$$\begin{cases} L^{(0)} = \sum_{j=1}^{n^{(i)}} C_j^{(i)} \\ L^{(m+1)} = W(L^{(m)}) \end{cases} \quad (9)$$

The last equation will give us all the busy periods of VP_1. Before the calculation of the busy periods, we should calculate the $C_j^{(i)}$ at a lower speed (Quan and Hu, 2003). Thus, the $C_j^{(i)}$ are equal to the ratio of C_j and the processor speed that it is fixed to be equal to 0.7.

$L_k^{start}(VP_1)$= 0 since the tasks are synchronous. $L_k^{end}(VP_1)$ is calculated by Eq. (9). It is equal to 38. $L_k^{start}(VP_2)$= $L_k^{end}(VP_1)$ $L_k^{end}(VP_2)$ is calculated by:

$$\begin{cases} L^{(0)} = L_k^{end}(VP_1) + \sum_{j=n}^{k^{(i)}} C_k^{(i)} \\ L^{(m+1)} = W(L^{(m)}) \end{cases} \quad (10)$$

In the idle periods of VP_1 , the execution of the other category of tasks is done. They are calculated as follows:

- $IL_k^{start}(VP_2)$ = $L_k^{end}(VP_1)$
- $IL_k^{end}(VP_2)$ is equal to the sum of the WCETs of the tasks that included into VP_2 plus $L_k^{end}(VP_1)$
- $IL_k^{start}(VP_3)$= $IL_k^{end}(VP_2)$.
- $IL_k^{end}(VP_3)$ is equal to the sum of the WCETs of the tasks that included into VP_2 plus $IL_k^{end}(VP_2)$.

For the aperiodic tasks, if the current idle period is not sufficient to support a WCET of task, it must be executed in the next idle period of the relative virtual processor.

After modifying the periods, we calculate the busy periods of VP_1.

- $L_k^{start}(VP_1)$=0
- $L_k^{end}(VP_1)$=38

After that, we calculate the idle periods of VP_1 to locate the aperiodic tasks.

- $IL_k^{start}(VP_2)$= 38
- $IL_k^{end}(VP_2)$=46
- $IL_k^{start}(VP_3)$=46
- $IL_k^{end}(VP_3)$=50

The different virtual processors and busy/idle periods are illustrated via Fig. 5

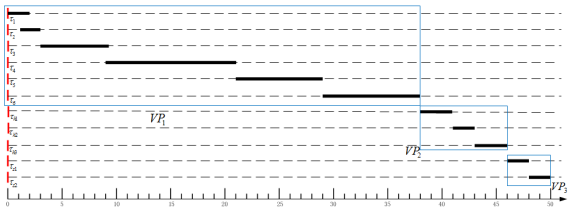

Figure 5: Virtual processors after the reconfiguration.

6.3 Schedule verification

After each reconfiguration scenario, the agent verifies the system's feasibility. If $U^{(i)} \leq 1$, the agent must propose different technique solutions:

- Period modification or WCETS modification

- The modification of the rate λ_C, and
- Degraded reconfiguration by using $(m,k)-$ firm

6.4 Power Consumption Evaluation

The power consumption is evaluated in every virtual processor. According to (X. Wang and Li, 2011), the power consumption is proportional to the processor utilization. If the processor utilization is minimized, then the power consumption is minimized automatically. The power decrease of each virtual processor is calculated as follows:

$$P_{Di} = P^i_{aft} - P^i_{bef} \tag{11}$$

where :

- i: is the i_th reconfiguration
- P^i_{aft}: Power consumption after the reconfiguration
- P^i_{bef}: Power consumption before the reconfiguration

7 PROPOSED SOLUTIONS

In this section, we present the different suggested solutions.

7.1 Parameters Modification

In this work, we propose the modification of the periods or the WCETS of periodic tasks and the rate λ_C of the aperiodic tasks. For the periodic tasks, if we modify the periods or the WCETs, we obtain a feasible system with low-power in both cases.

7.1.1 Period Modification of Periodic Tasks

Proposition 1. *The extended period $T_j^{(i)}$ of a periodic task τ_j is equal to* $T_j^{(i)} = \left\lceil (\sum_{j=1} (\frac{(C_j+B_j)}{U_{per}}) \right\rceil$.

Proof. In order to minimize the processor utilization: $U^{(i)}_{per} = \sum_{j=1} \frac{C_j+B}{T_j^{(i)}} \geq U_{per}$ Then,
$T_j^{(i)} = (\sum_{j=1} (\frac{(C_j+B_j)}{U_{per}})$. Since the periods are integer,

$$T_j^{(i)} = \left\lceil (\sum_{j=1} (\frac{(C_j+B_j)}{U_{per}}) \right\rceil \tag{12}$$

□

After the modification of the periods, the new processor utilization of periodic tasks is given by:

$$U^{(i)}_{perT} = (\sum_{k=1}^{n} \frac{C_k}{T_k^{(i)}}) + \frac{B_k}{T_k^{(i)}} \tag{13}$$

Then, we can formulate the new power consumption of the periodic tasks as follows:

$$P^{(i)}_{perT} = (U^{(i)}_{per})^2 \tag{14}$$

The new periods are equal to 54 according to Eq. (12). By applying Eq. (13), U_{perT} is equal to 0.574. It is obvious that the processor utilization is unchangeable after applying a reconfiguration scenario.

7.1.2 WCETs Modification

Proposition 2. *The extended WCET $C_j^{(i)}$ of a periodic task is* $C_j^{(i)} = \begin{cases} \lfloor \frac{U_{per} - \sum_{j=1} \frac{B_j}{T_j}}{\sum_{j=1} \frac{1}{T_j}} \rfloor \\ 1, if \frac{U_{per} - \sum_{j=1} \frac{B_j}{T_j}}{\sum_{j=1} \frac{1}{T_j}} \leq 0 \end{cases}$

Proof. We followed the same used technique to calculate the new WCETs. After we reconfigure the WCETS, we should get $U^{(i)}_{per} < U_{per}$. Thus
$U^{(i)}_{per} = \sum \frac{C_j^{(i)}+B}{T_j} \geq U_{per}$ So, the new $C_j^{(i)}$ are given by:

$$C_j^{(i)} = \begin{cases} \lfloor \frac{U_{per} - \sum_{j=1} \frac{B_j}{T_j}}{\sum_{j=1} \frac{1}{T_j}} \rfloor \\ 1, if \frac{U_{per} - \sum_{j=1} \frac{B_j}{T_j}}{\sum_{j=1} \frac{1}{T_j}} \leq 0 \end{cases} \tag{15}$$

□

After the modification of the WCETs, the new processor utilization of periodic tasks is given by:

$$U^{(i)}_{perC} = (\sum_{k=1}^{n} \frac{C_k^{(i)}}{T_k} + \frac{B_k}{T_k} \tag{16}$$

Then, we can formulate the new power consumption of the periodic tasks as follows:

$$P^{(i)}_{C} = (U^{(i)}_{perC})^2 \tag{17}$$

All the new WCETS are equal to 2 after applying Eq. (15). According to Eq.(16), the new processor utilization is equal to 0.72. We can conclude that the second solution is more beneficial than the first one since the processor utilization is minimized in a very remarkable way.

7.1.3 λ_C Modification

As similar as in the work (I.Khemaissia, 2012), the rate λ_{Cj} of the aperiodic tasks is modified to be:

$$\lambda_{Cj}^{(i)} = \lambda_{Cj} \times \frac{U^{(ia)}}{U_{bef}}, \forall j \in [0,i] \quad (18)$$

For the rate λ_{Cj} of the aperiodic tasks that share resources is calculated also according to Eq. (18) since all the aperiodic tasks arrive with the same rate λ_{Cj}.

The new rates λ_C are equal to 0.744 and 0.595 respectively. The new processor utilization of aperiodic task $U_{pro}^{(1)}$ is equal to U_{pro}.

After the modification of the parameters of periodic and aperiodic tasks, we will present the new results in Table 3

The reconfiguration result of $Sys^{(1)}$ is shown in Table 3. The system is feasible with low-power according to Fig. 5

Table 3: Reconfiguration Result.

task	$C^{(1)}$	$D^{(1)}/d^{(1)}$	$T^{(1)}$
τ_1	2	54	54
τ_2	2	54	54
τ_3	2	54	54
τ_4	2	54	54
τ_5	2	54	54
τ_6	2	54	54
τ_{h1}	2	46	
τ_{h2}	1	49	
τ_{h3}	2	26	
τ_{s1}	1	40	
τ_{s2}	1	30	

7.1.4 $(m,k)-$ firm

As a third solution, the agent applies the $(m,k)-$ firm model. We assume if we have m instance of each periodic task that respect their real-time constraints between k ones, then we can consider that this system is feasible in a degraded mode.

We can assume that every periodic task has $(2,3)$-firm guarantee. The agent must verify if all the tasks respect this constraint or not. For example, the first and the second instance of T_1 meet their deadlines. Then, T_1 respects the $(2,3)$-firm guarantee. This verification must be done for all the periodic tasks.

8 EXPERIMENTS

This section presents an experimentation study by applying the theoretical contributions of the current paper for the middleware for RT-Linux. We present first of all the implementation of the agent-based architecture before showing the simulations and analysis that are used to evaluate the benefits of our contribution.

8.1 Implementation

The proposed middleware can be utilized to theoretically reconfigure its OS. A tool is developed, which is utilized to minimize the power consumption and satisfy the real-time constraints. Many routines are added to RT-Linux in order to apply the different proposed solutions. A routine is a service offered by RT-Linux.

The middleware will be on interaction with the kernel. Fig. 6 represents the middleware layer location.

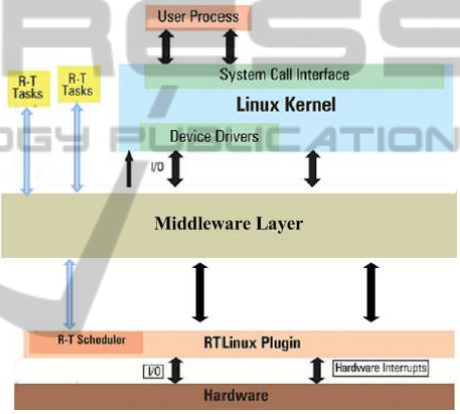

Figure 6: The middleware layer location.

Algorithm 1 is developed in order to reconfigure the RT-Linux. It is with complexity $O(n^2)$ Initially, the agent reads the parameters of the initial tasks. It calculates the processor utilization of the initial system. It reads thereafter the parameters of the added tasks. It assigns the tasks to the different virtual processors according to their categories. Moreover, it manages the shared resources and the precedence constraint. Then, it invokes the developed solutions to minimize the power consumption. At the end, it evaluates the power consumption.

8.2 Simulations and Analysis

The developed tool presents the different possible solutions given by the proposed intelligent agent. Fig. 7 shows a developed software for algorithm 1. We have assumed 50 initial periodic tasks and several aperiodic task sets. All the tasks are with precedence constraint and share resources. The initial system is feasible with low-power. Every time, we add some periodic tasks and an aperiodic task sets. The Simulation

Algorithm 1: Reconfiguration of the RT-LINUX.

1. **Input:** the periodic and the aperiodic tasks;
2. Compute U_{per}, U_{pro};// Eq. (2 and Eq. (5)
3. Calculate U_{bef};// Eq. (3 3. Assign S_{per} and S_{pro} at U_{bef};
4. **Add new tasks?**
   N:
   END;
   Y:
5. **if** category==periodic **then**
   Assign task to VP_1;
   else
   if category==aperiodic with hard deadline **then**
   Assign task to VP_2;
   else
   Assign task to VP_3;
   end if
   end if
6. **if** $(U^{ia} \geq 1)$ or $(U^{ia} \geq U^{bef})$ **then**
   for each periodic task **do**
   if Solution== 'Period modification' **then**
   Modify the periods;//Eq. (12)
   Calculate the new processor utilization of periodic tasks;//Eq. (13)
   else
   Modify the WCETs;//Eq. (15)
   Calculate the new processor utilization of periodic tasks;//Eq. (16)
   end if
   end for
   for each aperiodic task **do**
   Modify the rate λ_C of the tasks that share resources;//Eq. (18)
   Calculate the new processor utilization of aperiodic tasks;//Eq. (7)
   end for
   end if
7. Calculate the total new processor utilization
8. Calculate all the busy periods of the synchronous tasks;
9. Calculate the the idle periods of the tasks of VP_2, VP_3.
10. Calculate the new power consumption
11. Evaluate the power consumption// Eq. (11)
12. Back to **Step 4**;

results are shown on Fig. 8 and Fig. 9. Fig 8 represents the power consumption after modifying the periods. It is between $[0.5949, 05953]$. The variation can be not considered. We can conclude that after applying the period modification, the power consumption is minimized. For Fig. 9, it represents the power consumption after the WCETS reconfiguration. The curve shows a remarkable variations. In a nutshell, we can conclude that by using this tool, the power consumption after applying any reconfiguration scenario is minimized or maintained.

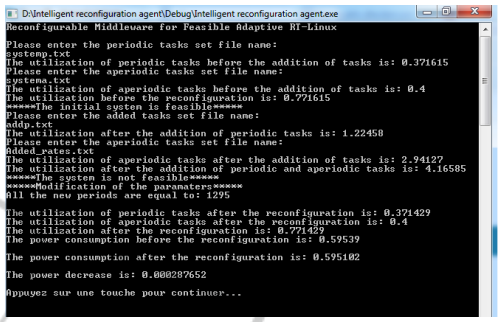

Figure 7: Tool.

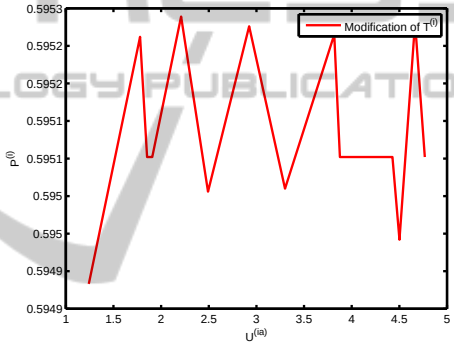

Figure 8: Power consumption decrease by modifying periods.

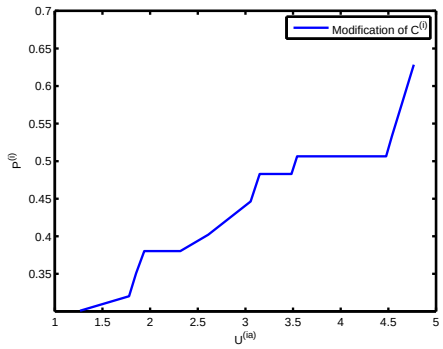

Figure 9: Power consumption decrease by modifying WCETs.

9 CONCLUSIONS

This paper focuses on reconfigurable real-time embedded OS for microprocessors to be implemented

by sets of periodic/aperiodic tasks sharing resources and with precedence constraints. It is assumed that the system under consideration can be reconfigured repeatedly. In order to offer low-power consumption before and after the application of any reconfiguration scenario, a DWCS architecture is defined. An intelligent software control agent is proposed accordingly to evaluate the power consumption and reconfigure the system. First of all the agent manages the precedence constraints. Then, it creates the different virtual processors. After that, it modifies the parameters of periodic/aperiodic tasks. And finally it evaluates the power consumption.

To our best knowledge, no work dealing with a middleware is found to dynamically reconfigure RT-Linux.

REFERENCES

Baker, T. (1991). Stack-based scheduling of realtime processes. Journal of Real-Time Systems.

Burns, A. and Wellings, A. (2001). Scheduling algorithms for multiprogramming in a hard real time environment. In *Addison Wesley Longmain.*

C. Angelov, K. S. and Marian, N. (2005). Design models for reusable and reconfigurable state machines. Proc. of Embedded Ubiquitous Comput.

D. Faggioli, F. Checconi, M. T. and Scordino, C. An edf scheduling class for the linux kernel. Germany. Proc. of the 11th Real-Time Linux Workshop.

F. Singhoff, J. Legrand, L. N. and Marce, L. (2004). Cheddar: A flexible real time scheduling framework. Assoc. Comput. Mach.

FSM Labs, I. (2001). Getting-started-with-rtlinux.

George, L. and Courbin, P. Reconfiguration of uniprocessor sporadic real-time systems: the sensitivity approach. In *chapter in IGI-Global Knowledge on Reconfigurable Embedded.*

George, L. and Courbin, P. (2011). Reconfiguration of uniprocessor sporadic real-time systems: the sensitivity approach. IGI-Global Knowledge on Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility.

H. Chetto, M. C. (1989). *Some Results of the Earliest Deadline Scheduling Algorithm*. IEEE Tr. on Software Engineering.

Hamdaoui, M. and Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m,k)firm deadlines. IEEE Transactions on Computers.

I.Khemaissia (2012). *master'sthesis: "Low-power reconfigurations of real-time embedded systems.* Tunisia.

Ishihara, T. (2010). A multi-performance processor for reducing the energy consumption of real-time embedded systems. IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences.

K. Thramboulidis, G. D. and Frantzis, A. (2004). Towards an implementation model for fb-based reconfigurable distributed control applications. Vienna. Proc. IEEE 7th Int. Symp. Object-Oriented Real-Time Dist. Comput.

Khalgui, M. and Hanisch, H.-M. (2011). Reconfiguration protocol for multi-agent control software architectures. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.

L. George, N. R. and Spuri, M. (1996). Preemptive and non-preemptive scheduling real-time uniprocessor scheduling. INRIA Research Report.

Liu, C. L. and Layland, J. W. Scheduling algorithms for multiprogramming in a hard real time environment. In *J. Assoc. Comput. Mach.*

M. Khalgui, O. Mosbahi, Z. W. L. and Hanisch, H.-M. (2011). Reconfiguration of distributed embedded-control systems. IEEE/ASME Trans. Mechatronics.

Niu, L. W. (2011). Energy efficient scheduling for real-time embedded systems with qos guarantee. Real-Time Syst.

P. Balbastre, I. R. and Crespo, A. (2002). Schedulability analysis of window-constrained execution time tasks for real-time control. Proc. 14th Euromicro Conf. Real-Time Syst.

Quan, D. and Hu, X. S. (2003). Minimal energy fixed-priority scheduling for variable voltage processors. IEEE Trans. Comput.-Aid. Des. Integ. Circu. Syst.

Ramanathan, P. (1999). Overload management in real-time control applications using (m,k)-firm guarantee. IEEE Trans:arallel and Distributed Systems.

S. Baruah, R. H. and Rosier, L. (1990). Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. Real-Time Syst.

Shin, Y. and Choi, K. (1999). Power conscious fixed priority scheduling for hard real-time systems. New Orleans. Proc. IEEE Des. Autom. Conf.

T. Yokoyama, G. Zeng, H. T. and Takada, H. (2010). Static task scheduling algorithms based on greedy heuristics for battery-powered dvs systems. IEICE Trans. Inform. Syst.

T.P.Baker (1990). A stack-based resource allocation policy for realtime processes. In *Real-Time Systems Symposium*.

West, R. and Poellabauer, C. (2000). Analysis of a window-constrained scheduler for real-time and best-effort packet streams. Proc. 21st IEEE Real-Time Syst. Symp.

West, R. and Schwan, K. (1999). Dynamic window-constrained scheduling for multimedia applications. IEEE, 6th Intern. Conf. Mult. Comp. and Systems.

X. Wang, M. K. and Li, Z. W. (2011). Dynamic low power reconfigurations of real-time embedded systems. In *in: Proc. 1st Pervas. Embedded Comput. Commu. Syst*, Portugal.