Reconfigurable Priority Ceiling Protocol
Under Rate Monotonic Based Real-time Scheduling

Maroua Gasmi[1,2], Olfa Mosbahi[2], Mohamed Khalgui[2] and Luis Gomes[3]
[1]*Computer Science Departement, Faculty of Science of Tunis, University Tunis el Manar, Tunis, Tunisia*
[2]*Computer Science Departement, LISI Lab, INSAT Institute, University of Carthage, Tunis, Tunisia*
[3]*Departamento de Engenharia Electrotcnica, Faculdade de Cilncias e Tecnologia,*
Universidade Nova de Lisboa, Caparica, Portugal

Keywords: Real-time System, Reconfiguration, Scheduling, Resource Sharing, Priority Ceiling Protocol.

Abstract: This research paper deals with reconfigurable real-time systems to be adapted to their environment under user requirements. A reconfiguration scenario is a run-time software operation that allows the addition, removal and update of real-time OS tasks which can share resources and should meet corresponding deadlines. We propose a new Reconfigurable Priority Ceiling Protocol (denoted by RPCP) that avoids deadlocks after any reconfiguration scenario and changes the priorities of tasks in order to reduce their response and blocking times, and to meet their deadlines. This protocol requires the use of two virtual processors in order to guarantee the non-interruption of execution during the reconfiguration step. We develop a tool that encodes this protocol which is applied to a case study.

1 INTRODUCTION

Real-time constraints (Stankovic, 1996) are common bases to most of the actual embedded systems (Lee et al., 2007), since the latter have many time requirements imposed on their activities. These systems follow a definite classification (Colnaric and Verber, 2007). The functions performed by the real-time systems, are consistently, executed by a fixed number of tasks. Nevertheless, the notion of time is what makes the difference between real-time and non-real-time systems. The main rule is that the preeminent parameter, the deadline, has to be met under even the worst circumstances (Kalinsky, 2003). In the case where several tasks share a specific number of resources, many issues can occur preventing these tasks from meeting their deadlines. In the perspective of solving these problems, Rate Monotonic schedule (Lehoczky et al., 1989), is a scheduling algorithm that assigns priorities on the basis of the period of the task. Although, this algorithm, solves the mentioned problems, others can occur as a consequence. In fact, a high priority task can be interrupted by a lower priority one, inverting the priorities of the two tasks (Sha et al., 1990). This problematic scenario, called priority inheritance, is solved by dint of a synchronization protocol called priority ceiling protocol (PCP). Furthermore, a real-time system has the ability to be reconfigured according to its surroundings (Brennan et al., 2002). In fact, a reconfiguration consists on modifying the behavior of the system depending of the modifications that occurred in its environment (Wang et al., 1995). The reconfiguration can either be static, where it is only applied offline before the starting of the system, or dynamic (Stewart et al., 1997). The dynamic form of reconfiguration can be either manual (applied by a user) or automatic (applied by intelligent agents within the system). In the literature, the concept of reconfiguration that we are introducing in this paper is indicated as a mode change since a system is able to move from one mode of execution to another. A mode change is defined as the removal of tasks, the addition of new ones and the change of their parameters (Sha et al., 1989). As a matter of fact, the particularity of the work that we propose in this paper lies, essentially, in the possibility of reconfiguring the resources as well as the set of tasks, optimizing the blocking times and lowering the response times after

[1]This research paper is a collaboration between LISI Lab (INSAT at University of Carthage in Tunisia), PASRI (Ministry of High Study and Research in Tunisia) and Universidade Nova de Lisboa in Portugal. It is financially supported as a MOBIDOC grant from the European Commission. We thank Mr. Wael Bouslama for his fruitful collaboration in the experimental part.

DOI: 10.5220/0005060900420052
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 42-52
ISBN: 978-989-758-039-0

each scenario of reconfiguration. However, several authors treated the mode change, proposing different techniques. None of these techniques offers the advantages previously mentioned. In (Real and Crespo, 2004) the authors present a classification and an evaluation of mode change protocols for single-processor, fixed priority, preemptively scheduled real-time systems. The contribution in (Stoimenov et al., 2009) consists on presenting a method for timing analysis of single-processor multi-mode systems with earliest deadline first (EDF) or fixed priority (FP) scheduling of tasks. In (Tindell et al., 1992) the mode changes are defined either as operations increasing a task sets processor utilization, or operations that decrease it. The analysis approach in the latter work is improved and extended to deadline-monotonic scheduling in (Pedro and Burns, 1998). The model is augmented with transition offsets in (Pedro and Burns, 1998), which permits to avoid overload situations. In the idle time protocol (Tindell and Alonso, 1996), when a mode change request occurs, the activation of the new tasks is not done until the next idle instant takes place. Although its implementation is simple, the latter protocol is considered to be poor when it comes to promptness. The ceiling protocol in Multi-Moded Real-Time Systems (Farcas, 2006) is an approach that combines the mode changes and permits an important degree of flexibility with immediate inheritance priority ceiling protocol (IIPCP). In the works presented in (Gharbi et al., 2013; Gharbi et al., 2011; Khalgui et al., 2011) the Priority Ceiling Protocol (PCP) is applied as an approach to ensure the scheduling between periodic tasks but the change of priorities of these tasks in order to minimize the response time reconfiguration is not taken into account. Generally speaking, in a random scenario of reconfiguration, the problems of deadlock and exceeding of deadline can occur. No one in the related works treated this situation where we can have activations of resources and tasks. we propose an original solution, denoted as Reconfigurable Priority Ceiling Protocol (RPCP) to the previously defined problems, in addition to the optimization of blocking and response times. To guarantee the non interruption of execution after any reconfiguration scenario, the proposed solution starts by separating the physical processor into two virtual ones. The first continues the regular execution of PCP, while the second one calculates the new periods and therefore priorities that guarantee the previously defined optimizations. We developed a simulation tool at LISI Lab (University of Carthage) which is applied to a case study in order to show the contributions of the paper.
The following section gives an overview on the different axes that create the context of the work. the section case study takes an example of reconfigurable tasks and resources and shows the impact of the random reconfiguration on causing issues in the system. After that, we formalized the elements that form mathematically our environment. Then we explain our contribution step by step and finish by presenting the proposed algorithm and exposing the simulation.

2 BACKGROUND

A real-time task (Liu and Layland, 1973), designated in this paper as τ_i, is essentially characterized by its: **(i)** Arrival time A when τ_i becomes ready for execution, **(ii)** Computation time C known as Worst Case Execution Time (WCET), this parameter has to be determined previously, **(iii)** Deadline D is the time limit by which τ_i must be accomplished, **(iv)** Starting time S is the moment when the system decides to start τ_i. Indubitably, it cannot be earlier than the arrival time A as before this time the task is totally unknown, **(v)** Finish time E is the time when the execution of τi finishes. It can be depicted by the sum of the starting time S and the computation time C, **(vi)** Period T which serves as a duration of one cycle on a repeating execution of a periodic task and represents the interval between two consecutive activations. It is important to mention that in the case of an aperiodic task, the concept of period is utterly missing, **(vii)** Work Left W is the work left for a task to execute and finally **(vii)** response time R is the length of time from the moment of release to the instant when the task completes its execution. This time is given by the following formula (Tokuda et al., 1990):

$$R_k^0 = 0, R_k^q = C_k + B_k + \sum_{j>k} \left\lceil \frac{R_k^{q-1}}{T_j} \right\rceil C_j$$

The response time of a task, denoted as R_k, is obtained once $R_k = R_k^q = R_k^{q+1}$. During its execution, a task is able to use one or several resources, referring by the latter to any shared hardware or software object (Mok et al., 2001). The execution runs regularly, until the moment when several tasks wish to use a single resource (Lipari and Bini, 2003). It is necessary to mention that a blocking can be caused when several tasks wish to access a single resource (Tokuda et al., 1990). Here comes the role of the real-time scheduling. Its main goal is to assign processors and resources to tasks in such a way that all the imposed constraints are respected. Among the scheduling algorithms, Rate Monotonic scheduling (Liu and Layland, 1973) occupies an important role. It assigns priorities in a static way: the shorter the period of the task the higher its priority. In (Sha et al., 1990) the authors prove that this scheduling protocol is optimal among the rest of static policies . One major limitation of fixed-priority scheduling is that it is not always possible to fully utilize the CPU (Sha et al., 1990). The schedulability test for RMS is:

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti} \leq n * (2^{\frac{1}{n}} - 1)$$

In a system with shared resources, it is impossible to eliminate all priority inversions but it is possible to limit the waiting time to minimize time and predict blocks. For this, several approaches are introduced. PCP prevents the deadlock situation as well as chained blocking (Sha et al., 1990). The rules in PCP aim essentially to prohibit a task to enter the critical section if there were any semaphores that may block this task. This protocol supposes that every task has a fixed priority and the used resources are known before the starting of the execution (Chen and Lin, 1991). In this protocol each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource. Hence, it should be taken into consideration that under the priority ceiling protocol, a task is blocked at most once, by a lower priority task, for the duration of a critical section, no matter how many tasks conflict with it. With the given information in (Liu et al., 2000), computing the maximum blocking time Bi for a task is possible. Above all else, we should point out that blocking time, when using PCP in particular, may arise under 3 possibilities: Directly blocked tasks, Inheritance blocked tasks or Avoidance blocked tasks. Therefore, the proposed protocol RPCP is based upon both RM and PCP since the first is optimal and the second is useful for shared resources; In fact, it is customized to fit the feasibility test and the condition imposed by RM. Besides, the particularities of this protocol lie in its ability to change priorities, reconfigure both tasks and resources and minimize the response as well as the blocking times.

3 CASE STUDY

We present in this section a case study to expose our problem, and to be assumed in the following as a running example. Let us consider a system to be scheduled by both PCP and RM, and to be implemented by OS tasks with shared resources. The details related to these tasks are given by Table1. The task τ_1 for example is periodically executed each 60 time units, and uses the resource R1 for 5 time units.

Table 1: Parameters of the initial tasks.

Tasks	Priorities	Resources	Computation times (Ci)	Periods (Ti)
τ_1	P1	R1	5	60
τ_2	P2	R1	2	55
		R2	3	
τ_3	P3	R2	5	50
τ_4	P4	R5	2	45
		R6	3	
		R8	2	
τ_5	P5	R6	4	40
		R7	3	

According to the simulator Cheddar (Singhoff et al., 2004), the system is feasible since all the tasks meet the related deadlines as depicted in Figure 1. We can prove the system feasibility by applying the RM condition $\sum_{i=1}^{5} \frac{Ci}{Ti} = 60\%$ and is lower than $5 * (2^{\frac{1}{5}} - 1) = 74\%$. We are interested in the current work in the software reconfiguration of tasks and resources. A reconfiguration is assumed to be an operation allowing the addition-removal of tasks or resources. No one in all related works dealing with real-time scheduling treats this form of reconfiguration. Let we assume the following reconfiguration that adds the tasks τ_6 and τ_7, and removes τ_4 and τ_5 under well-defined conditions described in user requirements.

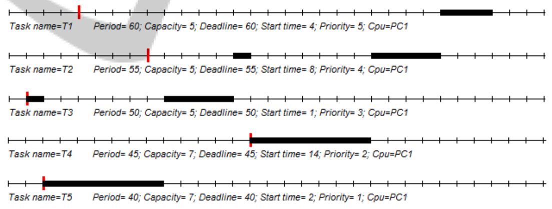


Figure 1: Execution graph of the tasks.

Table 2 depicts in detail the new configuration of the system.

Table 2: Parameter of the tasks after reconfiguration.

Tasks	Priorities	Resources	Computation times (Ci)	Periods (Ti)
τ_1	P1	R1	8	60
		R4	12	
τ_2	P2	R1	15	55
		R2	5	
τ_3	P3	R2	3	50
		R3	17	
τ_6	P6	R3	14	45
		R4	6	
τ_7	P7	R4	2	40
		R1	18	

Note that this reconfiguration scenario can allow the violation of real-time properties or block and destroy the whole system in some situations, since the new tasks have higher priorities and the old ones have to use new resources. We show in Figure 2 the run-time problem that occurs in the system after this reconfiguration scenario. In fact, while τ_1 is holding the resource R_1, the reconfiguration adds the resource R_4 to the list of the resources belonging to the latter task. τ_7, the task added after the application of this scenario, finishes the execution of R_4 and keeps waiting for R_1 as shown in Figure 2. A deadlock happens in this situation.

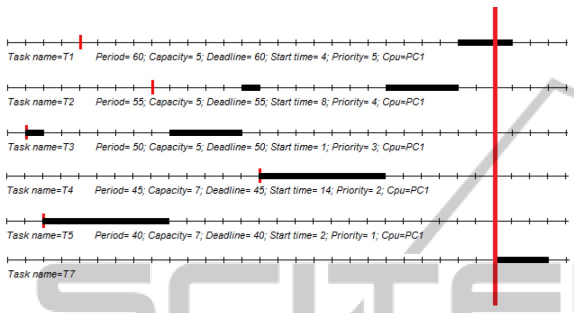


Figure 2: Deadlock due to the reconfiguration scenario.

The random application of the new configuration causes a deadlock leading automatically to the violation of the feasibility conditions. In the related works the deadlock problem was cured but without taking into account the optimization of the computation time neither the possibility of minimizing the blocking times and the response times of the different tasks. The addition and removal of resources within the system as well as changing the priorities of the tasks are original particularities in our work that we cannot find in other works.
In this section, the remarked problem is essentially due to an arbitrary choice of priorities after a reconfiguration scenario. For this reason, we introduce in this paper, a new solution that does not only consist on preventing any deadlocks owing to a sudden change in the set of tasks caused by an incoming reconfiguration, but also on drafting suitable priorities that offer minimal blocking times for each task.

4 FORMALIZATION

In this section, we are interested in mathematically defining the elements of the system and their reactions to any reconfiguration scenario as well as the proposed representation of their characteristics. Hence, in addition to the existing parameters, mentioned in the section background, we propose to add the following new ones to each task. **(i)** π(t): the state of a task within the system (1 if the task is active either executed or not, 0 else). **(i)** σ: the set of possible resources that can be used by the task, **(ii)** *Res*(t): the set of resources used by the task at t, **(iii)** *Cond*(t): state of conditions (1 if the condition that activates the task is met at t, 0 if not) and **(iv)** *Request*(t): the set of resources required by the task at t. Let τ_{Sys} and R_{Sys} respectively be the set of all possible tasks and resources that may be executed within the system independently from the time. Therefore, a general system that describes the global environment, denoted as Sys, is defined by the previously mentioned couple.

$$Sys = (\tau_{Sys}, R_{Sys}) \tag{1}$$

Running example1:
Through the example given in the case study τ_{Sys} and R_{Sys} are expressed as follows:
$\tau_{Sys} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7\}$
$R_{Sys} = \{R_1, R_1, R_3, R_4, R_5, R_6, R_7, R_8\}$

Let $\tau_{Sys}(t)$ and $R_{Sys}(t)$ respectively be the set of active tasks and resources within the system at a given moment t. Pointedly, the set of active tasks , denoted as $\tau_{Sys}(t)$ at the moment t is represented by the group of tasks whose the state is set to be active. This assortment is given by the following formula:

$$\tau_{Sys}(t) = \{\tau_i \in \tau_{Sys} / \tau_i.\pi(t) = 1\} \tag{2}$$

Respectively, the group of resources which are active at t , denoted as $R_{Sys}(t)$, is represented by the resources required by the active tasks at that moment. This set of resources is given by the following formula:

$$\mathcal{R}_{Sys}(t) = \{R_i \in R_{Sys} / \exists \tau_i, \tau_i.\pi(t) = 1 \wedge R_i \in \tau_i.Request(t)\} \tag{3}$$

As a consequence, the general system at that moment, denoted as Sys(t), is defined by the previously mentioned couple.

$$Sys(t) = (\tau_{Sys}(t), R_{Sys}(t)) \tag{4}$$

Running example 2:
Through the example given in the case study, Table1 contains the list of the active tasks and resources at t_0 before the application of the reconfiguration scenario. As a consequence, $\tau_{Sys}(t_0)$ and $R_{Sys}(t_0)$ are expressed as follows:
$\tau_{Sys}(t_0) = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$
$R_{Sys}(t_0) = \{R_1, R_2, R_5, R_6, R_7, R_8\}$

According to user requirements, each reconfiguration scenario is automatically applied to add or remove tasks from a system at a specific moment denoted as t_1. In fact, the couple $\tau_{Sys}(t_0)$ and $R_{Sys}(t_0)$ that takes place at t_0, which is a moment coming right before the reconfiguration, is replaced by $\tau_{Sys}(t_1)$ and $R_{Sys}(t_1)$. Let $\xi_{Sys}(t_1)$,described by the formula (5), be the group of tasks to be added to the system. In fact, a task is ready to be added when the condition that activates it is met at t_1.

$$\xi_{Sys}(t_1) = \{\tau_i \in \tau_{Sys}/\tau_i.Cond(t_1) = 1\} \quad (5)$$

Let $\Delta_{Sys}(t_1)$,described by the formula (6), be the group of tasks to be removed from the system. Similarly, a task is ready to be removed when the condition that deactivates it is met at t_1.

$$\Delta_{Sys}(t_1) = \{\tau_i \in \tau_{Sys}(t_1)/\tau_i.Cond(t_1) = 0\} \quad (6)$$

Thereby, the new set of active tasks at t_1 after reconfiguration is expressed as the addition of the tasks $\xi_{Sys}(t_1)$ and the removal of the tasks $\Delta_{Sys}(t_1)$ from the old set of tasks established at t_0. The formula describing $\tau_{Sys}(t_1)$ is given as follows:

$$\tau_{Sys}(t_1) = \tau_{Sys}(t_0) \cup \xi_{Sys}(t_1) \setminus \Delta_{Sys}(t_1) \quad (7)$$

Running example 3:
Through the example given in the case study, the changes from Table1 to Table2 represent a reconfiguration scenario that occurred in t_1. In fact, the actions of addition and removal of tasks are performed. In our case, the added tasks ($\xi_{Sys}(t_1)$) and the removed ones ($\Delta_{Sys}(t_1)$) are given as follows:

$$\xi_{Sys}(t_1) = \{\tau_6, \tau_7\}\Delta_{Sys}(t_1) = \{\tau_4, \tau_5\}$$

Subsequently, the new set of active tasks after reconfiguration ($\tau_{Sys}(t_1)$) is expressed as follows:

$\tau_{Sys}(t_1) = \tau_{Sys}(t_0) \cup \xi_{Sys}(t_1) \setminus \Delta_{Sys}(t_1) = \{\tau_1, \tau_2, \tau_3, \tau_6, \tau_7\}$

Likewise, the subset of resources can be modified by the reconfiguration. Let $\xi_R(t_1)$,described by the formula (8), be the group of resources to be added to the system. In fact, a resource is considered to be active when it is added to the system as required by a task added through $\xi_{Sys}(t_1)$.

$$\xi_R(t_1) = \{R_i \in R_{Sys}/\exists \tau_j \in \xi_{Sys}(t_1) \wedge R_i \in \tau_j.Request(t_1)\} \quad (8)$$

However, the list of resources which need to be deactivated is described by the ones that are no longer required by any task. It is to mention, that if a resource is shared by several tasks, it cannot be removed when some of them are removed. The group of resources which cannot be removed is denoted by $\overline{\Delta_R}(t_1)$ and described by the formula (9).

$$\overline{\Delta_R}(t_1) = \{R_i \in R_{Sys}(t_1)/\exists \tau_j \in \tau_{Sys}(t_1) \setminus \Delta_{Sys}(t_1), R_i \in \tau_j.Request(t_1)\} \quad (9)$$

Conclusively, the set of resources to be deactivated is defined as the relative complement of $R_{Sys}(t_1)$ in $\overline{\Delta_R}(t_1)$ and described by the following formula:

$$\Delta_R(t_1) = R_{Sys}(t_1) \setminus \overline{\Delta_R}(t_1) \quad (10)$$

Finally the new set of active resources after reconfiguration ($R_{Sys}(t_1)$) is expressed as the addition of resources $\xi_R(t_1)$ and the removal of the tasks $\Delta_R(t_1)$ from the old set of tasks established at t_0. The formula describing $R_{Sys}(t_1)$ is given as follows:

$$R_{Sys}(t_1) = R_{Sys}(t_0) \cup \xi_R(t_1) \setminus \Delta_R(t_1) \quad (11)$$

Running example 4:
Continuing from the previous running example, the added resources ($\xi_R(t_1)$) and the removed ones ($\Delta_R(t_1)$) are given as follows:
$\xi_R(t_1) = \{R_3, \tau_4\}$
$\Delta_R(t_1) = \{R_5, R_6, R_7, R_8\}$ Subsequently, the new set of active resources after reconfiguration ($R_{Sys}(t_1)$) is expressed as follows:
$R_{Sys}(t_1) = R_{Sys}(t_0) \cup \xi_R(t_1) \setminus \Delta_R(t_1) = \{\tau_1, \tau_2, \tau_3, \tau_4\}$

5 CONTRIBUTION RPCP/RM

We propose in this section to resolve the paper's original problems that we detailed in the case study. In fact, the automatic reconfiguration of tasks and/or resources can lead the system to deadlocks or the possible violation of deadlines by new or old tasks. Explicitly, the deadline is violated when a corresponding task has some work left when it reaches it. As for the deadlock, it happens when a task holds resources that another one is waiting for and inversely.This is properly explained in the following formula:

$$Problem: \begin{cases} \exists \tau_i/\tau_i.W > \tau_i.D - t_1 \\ \exists \tau_i, \tau_j/\tau_i.Request(t_1) \cap \tau_j.Res(t_1) \neq \emptyset \\ \wedge \tau_j.Request(t_1) \cap \tau_i.Res(t_1) \neq \emptyset \end{cases} \quad (12)$$

As a consequence to the mentioned problems, the execution of the hardware processor is split into two virtual processors in the purpose of pre-computing the proposed optimizations when applying the reconfiguration at t_1. One of the virtual processors continues the normal execution of old tasks normally without interruption while the other one computes the right set of periods and priorities. The latter mentioned procedure is decomposed in several sub-steps: the blocking time minimization of the new and old tasks, response time minimization, assuring feasibility without deadlock due to the addition of resources and meeting the RM condition.

5.1 Virtual Processors

In order to guarantee the non-interruption of the execution of the system, spreading the physical processor into two distinguished virtual processors, which are time slots, was taken into consideration. The idea behind this, is to gain in terms of computation without having any time gaps during the execution of the old tasks. In this study, two virtual processors are proposed. The first one, denoted as VP_1 takes the responsibility of computing the new appropriate periods and priorities to be assigned to both the old and new tasks after reconfiguration. The second one, VP_2, executes normally the old tasks by using the regular PCP. Figure 3 explains how the two virtual processors operate in order to switch safely from a configuration to another without interrupting the current execution.

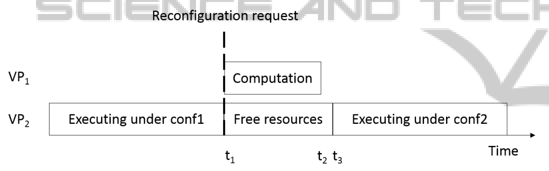


Figure 3: Roles of the virtual processors within the system.

The instant t_1, like we mentioned before, corresponds to when exactly the reconfiguration request occurred, t_2 when the computation ended and t_3 when VP_2 ends the freeing of resources. In fact, VP_2 executes the old tasks and it does not switch to the new configuration until VP_1 finishes its computation and reveals the new assortment of priorities and periods related to each task. The step that proceeds the switch from a configuration to another, consists on allowing the tasks that hold some resources to finish their execution under the previous configuration. Let *Newconf* be the group of tasks saved at t_1 to be used in the computation done by VP_1. It is formulated by using (7). However, at t_1, $\tau_{Sys}(t_1)$ changes and represents the group of tasks which did not finish their execution time when the reconfiguration took place. It is therefore given by the following formula:

$$\tau_{Sys}(t_1) = \{\tau_i \in \tau_{Sys}(t_0) / \tau_i.S < t_1 \wedge \tau_i.W(t_1) \neq 0\} \tag{13}$$

As a consequence, The group of resources $R_{Sys}(t_1)$ within the actual system depends on the change that occurred to $\tau_{Sys}(t_1)$ as remarked in the formula (3). Thus, the moment t_3, previously described as when VP_2 ended freeing the resources, is disposed by the following formula:

$$t_3 = max(\tau_i.E_i / \tau_i \in \tau_{Sys}(t_1)) \tag{14}$$

The phase of computation to be realized by VP_1, consists on finding the right set of periods and priorities. After this phase, the actual virtual processor VP_2 will be able to run under the new configuration. The exact moment when the latter virtual processor starts applying the new configuration is depicted as t_{new} and described as follows:

$$t_{new} = max(t_2, t_3) \tag{15}$$

Running example 5:
Proceeding further in the example given in the case study, the results obtained at t_1, t_2, t_3 and t_{new} are explained in details. At t_1, $Newconf = \{\tau_1, \tau_2, \tau_3, \tau_6, \tau_7\}$ and $\tau_{Sys}(t_1) = \{\tau_1\}$. At t_2, VP_1 finishes the computation and the application of the new periods and priorities of the task set *Newconf*. At t_3, τ_1 finishes executing R_1. At t_{new}, $\tau_{Sys}(t_{new}) = Newconf$.

5.2 Appropriate Set of Periods

At t_{new}, VP_2 is able to run the new set of tasks resulting from the computation done by VP_1. In our contribution, $\tau_{Sys}(t_{new})$, previously defined in (7), is described as the modification of the priorities and periods that belong to the group of tasks *Newconf* generated from the reconfiguration request. This modification that we propose, described in the formula (16), is performed by the recursive function Ψ which is computed in several steps.

$$\tau_{Sys}(t_{new}) = \Psi(Newconf) \tag{16}$$

Furthermore, in order to calculate the new temporal configurations of tasks, some steps need to be followed. the function Ψ is the composition of several other sub-functions such that each of one corresponds to a step of calculation.This is shown by the formula (17).

$$\Psi = \Psi_4 \circ \Psi_3 \circ \Psi_2 \circ \Psi_1 \tag{17}$$

In fact, to calculate the new system configuration $\tau_{Sys}(t_{new})$, we need to compute at first time, $\Psi_1(Newconf)$ which is in charge of finding the right arrangement of priorities that ensures a minimum blocking time. Then we apply the sub-function Ψ_2 to the result of $\Psi_1(Newconf)$ which is responsible for finding the right periods for which the response time of each task is minimum. Ψ_3 is applied to the result of Ψ_2: it is bound to find the periods for which the deadline constraint is respected. And finally, Ψ_4 is applied to the result of Ψ_3: it adjusts the obtained periods to meet the condition of RMS. It is to mention, that these recursive functions are not contradictory and that they are applied to this configuration without proposing apposed values.

5.2.1 Ψ_1: Minimum Blocking Time

For the purpose of identifying the minimum array of blocking times related to the tasks, we use an algorithm that reads through all the possible arrangements of priorities that the tasks may have. Thereafter, it spots the right set for which the blocking times are at their minimum. The number of possibilities of priorities that a vector of tasks can have, is based on the same principle in combinatorics. For each of these arrangement possibilities, the corresponding array of blocking times is computed. Accordingly, the comparison between the resulting vectors is performed by the calculation of the Euclidean norm. As a result, the proposed function Ψ_1 is defined as a n-tuple formed by pairs of priorities and tasks. Let {(P_1,τ_1),..., (P_n,τ_n)} denoted as E_1 be the actual n-tuple corresponding to the group of tasks *Newconf* and {(P_j,τ_1),..., (P_k,τ_n)} denoted as E_2 be the resulting arrangement of tasks. The definition of the function is regarded by the following formula:

$$\begin{gathered}\Psi_1 : E1 \rightarrow E2 \\ /\forall(P_i,\tau_i) \in E_2 : \\ \sqrt{\sum_{i=1}^{n} \tau_i.B^2} = min(\sqrt{\sum_{k=1,(P_k,\tau_k)\in E_1}^{n} \tau_k.B^2}) \end{gathered} \tag{18}$$

Where [P_j,..., P_k] is the same as the vector of priorities [P_1,..., P_n] just in a different order of its elements. It is important to retain that the priority with the least index is the highest among all priorities.

Running example 6:
This example aims to find the right set of priorities that guarantee a minimum blocking time for each task in *Newconf*. The following table contains the resulting minimum blocking time and the corresponding new priority to each task.

Tasks	Initial blocking times	Minimum blocking times	Old Priority	new priority
τ_1	0	6	P_5	P_4
τ_2	12	12	P_4	P_3
τ_3	15	14	P_3	P_1
τ_6	17	0	P_2	P_5
τ_7	15	15	P_1	P_2

The norm of the values of the initial blocking times is 29,71. As for the one corresponding to the Minimum blocking times, its value is 24,51.

5.2.2 Ψ_2: Minimum Response Time

Once the first step dealing with Ψ_1 is done we apply its result to the function Ψ_2. Attaining a specific set of priorities, is only effective when it comes to acquiring the appropriate values of periods. As matter of fact, the next step consists on finding the right periods for which the response time of each task is at its minimum. In this contribution, we can define the minimum response time that a task can have (as long as the priority of the latter is not the maximum) as the sum of its blocking time, its execution time and the execution times of the tasks that are more prioritized. For each task τi, the minimum response time, denoted as $R_{i,min}$, is therefore given by the following formula:

$$R_{i,min} = \begin{cases} C_i + B_i \; if \; P_i = max(P_1, ..., P_n) \\ C_i + B_i + \sum\limits_{P_k > P_i} C_k \; else \end{cases} \tag{19}$$

The obtained response times allow the possibility of defining the boundaries of the period. In fact, the generalization consists on limiting the periods of all the tasks (except the one with the lowest priority) with the maximum of response times among the least prioritized ones. Referring to the previous analysis, let τ_{prior} be the set of tasks except the least prioritized (respecting the order set by Ψ_1). The function Ψ_2 replaces the values of periods of tasks belonging to τ_{prior} with the maximum of response times of the prioritized tasks incremented by one. This is given by the following formula :

$$\Psi_2 : \tau_i.T \rightarrow max(R_k) + 1 / \forall k : P_k < P_i \tag{20}$$

Running example 7:
After assigning new priorities to the given tasks mentioned in the case study, the process of finding the minimum possible periods starts.

Tasks	Minimum response times	new periods
τ_1	11	23
τ_2	22	23
τ_3	19	45
τ_6	22	0
τ_7	44	23

Since τ_6 has the lowest priority, no period value is affected to it yet.

5.2.3 Ψ_3: Feasibility Test

Once the application of Ψ_2 is done, he results of the previous steps are applied to Ψ_3. Going further in finding the periods, the respect of the constraint of feasibility should be promulgated. In fact, bearing in mind the feasibility condition imposed by the system can allow limiting the period. So far, let $Boundary_k$ be the inferior limit of the resulting period.

$$Boundary_k = \begin{cases} A_k + R_k \; if \; \tau_k \; is \; the \; least \; prioritized \; task \\ \quad max(A_k + R_k, \tau_k.T)) \; if \; not \end{cases} \tag{21}$$

Thus, the definition of our submitted function Ψ_3 is:

$$\Psi_3 : \forall \tau_k : \tau_k.T \rightarrow Boundary_k + 1 \tag{22}$$

Running example 8:
Continuing in the example of the case study, the process of finding the possible periods that permit the respect of the feasibility continues.

Tasks	Starting times (A)	A+R	Periods obtained from Ψ_2	new Periods
τ_1	3	14	23	24
τ_2	1	23	23	24
τ_3	5	24	45	46
τ_6	2	28	0	29
τ_7	4	48	23	49

The new periods obviously correspond to the maximum between the sum of A and R, and the periods previously obtained from Ψ_2.

5.2.4 Ψ_4: RM Condition Test

Once Ψ_3 is well executed,we apply its result to Ψ_4. Basically, the procedure is done by incrementing the values of the periods until fulfilling the RM condition. Therefore, we make a place for a system that minimizes the response time, allows the feasibility and respects the condition imposed by the Rate Monotonic Scheduling (RMS). Hence, the function Ψ_4 is proposed to guarantee the respect of the latter condition which is expressed by the following formula:

$$\Psi_4 : \forall \tau_j : \tau_j.T \rightarrow \tau_j.T / \sum_{j=1}^{n} \frac{Cj}{Tj} \leq n * (2^{\frac{1}{n}} - 1) \tag{23}$$

Running example 9:
Finally, the following table describes the list of periods obtained after running a loop of incrementation that allows to obtain the required periods of tasks that respect the RM condition.

Tasks	Execution times	new Periods
τ_1	5	32
τ_2	5	31
τ_3	5	53
τ_6	7	36
τ_7	7	56

It is to mention that the obtained value of τ_1 and τ_2 is the same in this example. But, since τ_1 is less prioritized than τ_2, we incremented it in order to point out the distinct priorities. The value of $\sum_{j=1}^{5} \frac{Cj}{Tj}$ is around 73% which is less than 74,35% (the value of $5 * (2^{\frac{1}{5}} - 1)$).

5.2.5 Solution

The global function Ψ allowing the correct reconfiguration of the real-time system (applied to both the old and new tasks) is composed of Ψ_1, Ψ_2, Ψ_3 and Ψ_4. It permits to have a group of tasks that implement the system while satisfying the following items: (i) avoiding the deadlock anomaly, (ii) respecting the RM condition as well as minimizing (iii) the response time of the tasks and (iv) their blocking times. Subsequently, the resulting group of tasks that implement the system are free from the problems mentioned in (12) and characterized as follow :

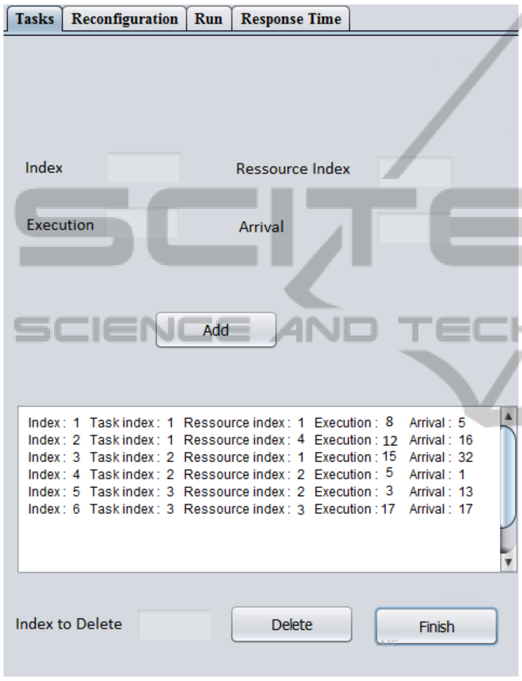
$$Solution : \begin{cases} (VirtualProcessors) \forall \tau_i, \forall \tau_j / \tau_i \neq \tau_j \\ \wedge \tau_i.Request(t) \cap \tau_j.Rest(t) = \emptyset \textbf{(i)} \\ (\Psi_1) \;\; \forall \tau_i / \tau_i.B = Minimum(\tau_i.B) \textbf{(iv)} \\ (\Psi_2) \;\; \forall \tau_i / \tau_i.R = Minimum(\tau_i.R) \textbf{(iii)} \\ (\Psi_3) \;\; \forall \tau_i / \tau_i.W < \tau_i.D - t_1 \textbf{(ii)} \\ (\Psi_4) \;\; \forall \tau_i / \sum_{i=1}^{n} \frac{Ci}{Ti} \leq n * (2^{\frac{1}{n}} - 1) \textbf{(ii)} \end{cases} \tag{24}$$

6 SIMULATION

Defining the procedures mentioned in the formalization in an algorithmic way consists on running two distinguished threads Algorithm1. The first one executes the actually active tasks with regular PCP. The second computes the blocking times and right arrangements of priorities , then starts the procedure of calculating the response times and the periods. Finally it checks the feasibility and the RM Condition in order to deliver the new information to the first thread. For the purpose of simulating the RPCP and showing its contribution compared to random behavior towards reconfiguration, we developed a tool at LISI Laboratory of INSAT Institute (Figure 4) that allows the user to fill in with the desired tasks parameters. Afterward, it is possible to fill with the parameters of the tasks that had been added to the system after the reconfiguration through the interface presented in Figure 5.

Algorithm 1 : Proposed response to Reconfiguration request.

```
Data: New Reconfiguration Request
$Tau\_Sys1 \leftarrow$ tasks holding resources
$NewConf \leftarrow$ tasks resulting from reconfiguration
while VP1.end = False And VP2.end = False do
    $Tau\_Sys2 \leftarrow VP1.Execute(NewConf)$
    VP2.Execute(Tau_Sys1)
end
VP2.Execute(Tau_Sys2)
```

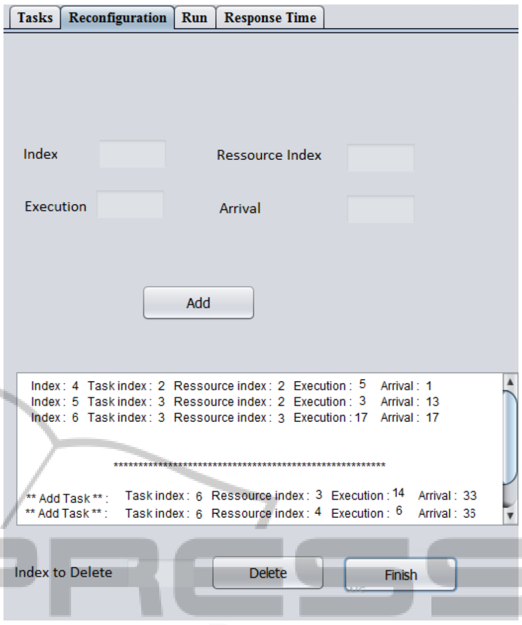


```
Index : 1  Task index : 1  Ressource index : 1  Execution : 8   Arrival : 5
Index : 2  Task index : 1  Ressource index : 4  Execution : 12  Arrival : 16
Index : 3  Task index : 2  Ressource index : 1  Execution : 15  Arrival : 32
Index : 4  Task index : 2  Ressource index : 2  Execution : 5   Arrival : 1
Index : 5  Task index : 3  Ressource index : 2  Execution : 3   Arrival : 13
Index : 6  Task index : 3  Ressource index : 3  Execution : 17  Arrival : 17
```

Figure 4: Initial tasks parameters interface.

The testing of the behavior of the system before and after the application of RPCP is pin pointed through the interface depicted in Figure 6. It is possible to notice that a blocking occurred when using the random reaction to the reconfiguration and how this problem was solved by using RPCP and the system continues its execution smoothly. The response time is then computed for each of the tasks and an average response time for both before and after the application of the RPCP (Figure 7). We show the gain in terms of response time due to RPCP. Through the test done over the case study, the improvement is noticeably obvious. In fact the blocking time is managed to get reduced to almost 80%. Consequently, the response time decreased to 75% compared to the initial procedures.

We note finally that no one in the related works treated the problem of reconfigurable resources while meeting deadline constraints for reconfigurable tasks.

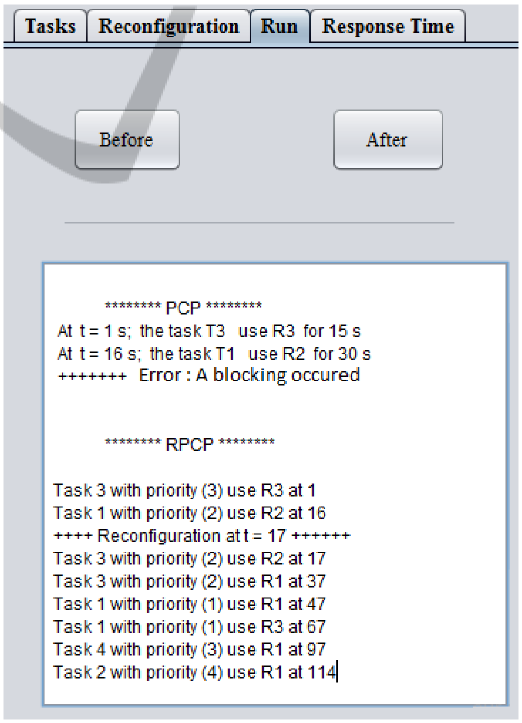


```
Index : 4  Task index : 2  Ressource index : 2  Execution : 5   Arrival : 1
Index : 5  Task index : 3  Ressource index : 2  Execution : 3   Arrival : 13
Index : 6  Task index : 3  Ressource index : 3  Execution : 17  Arrival : 17

            ************************************************************

** Add Task ** :   Task index : 6  Ressource index : 3  Execution : 14  Arrival : 33
** Add Task ** :   Task index : 6  Ressource index : 4  Execution : 6   Arrival : 35
```

Figure 5: Reconfiguration tasks parameters interface.



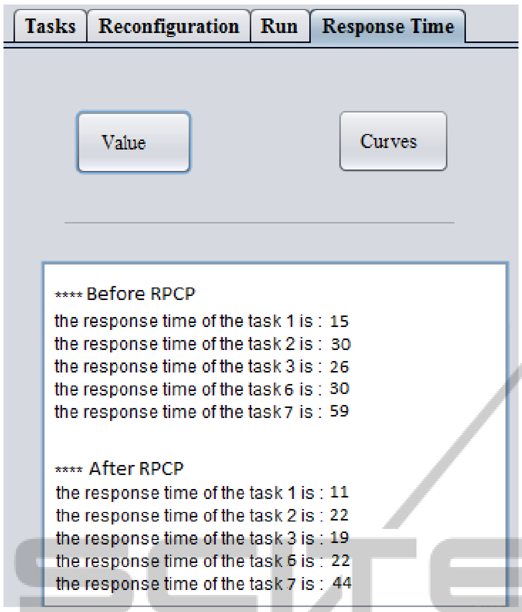
```
     ******** PCP ********
At t = 1 s;  the task T3   use R3  for 15 s
At t = 16 s;  the task T1   use R2  for 30 s
+++++++  Error : A blocking occured


     ******** RPCP ********

Task 3 with priority (3) use R3 at 1
Task 1 with priority (2) use R2 at 16
++++ Reconfiguration at t = 17 ++++++
Task 3 with priority (2) use R2 at 17
Task 3 with priority (2) use R1 at 37
Task 1 with priority (1) use R1 at 47
Task 1 with priority (1) use R3 at 67
Task 4 with priority (3) use R1 at 97
Task 2 with priority (4) use R1 at 114
```

Figure 6: Execution before and after RPCP.

This tool is original and the first to treat this original problem.



```
**** Before RPCP
the response time of the task 1 is : 15
the response time of the task 2 is : 30
the response time of the task 3 is : 26
the response time of the task 6 is : 30
the response time of the task 7 is : 59

**** After RPCP
the response time of the task 1 is : 11
the response time of the task 2 is : 22
the response time of the task 3 is : 19
the response time of the task 6 is : 22
the response time of the task 7 is : 44
```

Figure 7: Response Time details.

7 CONCLUSION

In this paper, we introduce RPCP as a protocol that solves well-defined real-time problems due to random reaction to reconfiguration. In fact, the power within this protocol lies on two different bases. The first one, corresponds to the choice of well-based scheduling methods and their ability to solve problems and optimize the parameters of the system. Surely, the use of a solid scheduling algorithm such as Rate Monotonic and an efficient protocol like Priority Ceiling Protocol reflects an important benefit to conclude from the proposed solution. Since the first one is known for its utility and optimality in the industrial field and the second one is able to prevent deadlocks as well as chained blocking. The second advantage of the proposed protocol RPCP, is its ability to fix the deadlock problems and to prevent exceeding the deadlines. Moreover, this protocol works on minimizing the blocking and the response times by changing the priorities of the tasks, leading to an optimal system that runs effectively. We plan in the future to apply this protocol to real complex case studies in order to evaluate the contributions of the current paper.

REFERENCES

Brennan, R. W., Fletcher, M., and Norrie, D. H. (2002). An agent-based approach to reconfiguration of real-time distributed control systems. *Robotics and Automation, IEEE Transactions on*, 18(4):444–451.

Chen, M.-I. and Lin, K.-J. (1991). A priority ceiling protocol for multiple-instance resources. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 140–149. IEEE.

Colnaric, M. and Verber, D. (2007). *Distributed Embedded Control Systems: Improving Dependability with Coherent Design*. Springer.

Farcas, E. (2006). *Scheduling multi-mode real-time distributed components*. PhD thesis, PhD thesis, Department of Computer Sciences, University of Salzburg.

Gharbi, A., Gharsellaoui, H., Khalgui, M., and Valentini, A. (2011). Safety reconfiguration of embedded control systems.

Gharbi, A., Khalgui, M., and Ben Ahmed, S. (2013). The embedded control system through real-time task. In *Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on*, pages 1–8. IEEE.

Kalinsky, D. (2003). Basic concepts of real-time operating systems. *LinuxDevices magazine, Nov*.

Khalgui, M., Mosbahi, O., Li, Z., and Hanisch, H.-M. (2011). Reconfiguration of distributed embedded-control systems. *Mechatronics, IEEE/ASME Transactions on*, 16(4):684–694.

Lee, I., Leung, J. Y., and Son, S. H. (2007). *Handbook of real-time and embedded systems*. CRC Press.

Lehoczky, J., Sha, L., and Ding, Y. (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE.

Lipari, G. and Bini, E. (2003). Resource partitioning among real-time applications. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 151–158. IEEE.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Liu, F., Narayanan, A., and Bai, Q. (2000). *Real-time systems*. Citeseer.

Mok, A. K., Feng, X., and Chen, D. (2001). Resource partition for real-time systems. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 75–84. IEEE.

Pedro, P. and Burns, A. (1998). Schedulability analysis for mode changes in flexible real-time systems. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 172–179. IEEE.

Real, J. and Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal. *Real-time systems*, 26(2):161–197.

Sha, L., Rajkumar, R., Lehoczky, J., and Ramamritham, K. (1989). Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9):1175–1185.

Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2004). Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM.

Stankovic, J. A. (1996). Real-time and embedded systems. *ACM Computing Surveys (CSUR)*, 28(1):205–208.

Stewart, D. B., Volpe, R. A., and Khosla, P. K. (1997). Design of dynamically reconfigurable real-time software using port-based objects. *Software Engineering, IEEE Transactions on*, 23(12):759–776.

Stoimenov, N., Perathoner, S., and Thiele, L. (2009). Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *proceedings of the Conference on Design, Automation and Test in Europe*, pages 99–104. European Design and Automation Association.

Tindell, K. and Alonso, A. (1996). A very simple protocol for mode changes in priority preemptive systems. *Universidad Politécnica de Madrid, Tech. Rep.*

Tindell, K. W., Burns, A., and Wellings, A. J. (1992). Mode changes in priority preemptively scheduled systems. In *Real-Time Systems Symposium, 1992*, pages 100–109. IEEE.

Tokuda, H., Nakajima, T., and Rao, P. (1990). Real-time mach: Towards a predictable real-time system. In *USENIX Mach Symposium*, pages 73–82.

Wang, J.-C., Chiang, H.-D., and Darling, G. R. (1995). An efficient algorithm for real-time network reconfiguration in large scale unbalanced distribution systems. In *Power Industry Computer Application Conference, 1995. Conference Proceedings., 1995 IEEE*, pages 510–516. IEEE.