# New Optimal Solutions For Real-Time Reconfigurable Periodic Asynchronous OS Tasks with Minimizations of Response Times

**Authors:** Hamza Gharsellaoui[(1)], Mohamed Khalgui [(2)], Olfa Mosbahi[(1)], Samir Ben Ahmed[(3)]
[(1)]National Institute of Applied Sciences and Technology, INSAT Institute - University of Carthago, Tunisia
[(2)]National Institute of Applied Sciences and Technology, INSAT Institute - University of Carthago, Tunisia, ITIA Institute - CNR Research Council, Italy,
Systems Control, Xidjian University, China.
[(3)] FST Faculty - University of Tunis El Manar, Tunisia
{gharsellaoui.hamza@gmail.com, khalgui.mohamed@gmail.com, olfamosbahi@gmail.com Samir.benahmed@fst.rnu.tn }

## ABSTRACT

This research work deals with Reconfigurable Uniprocessor embedded Real-Time Systems to be classically implemented by different OS tasks that we suppose independent, asynchronous and periodic in order to meet functional and temporal properties described in user requirements. We define in the book chapter a schedulability algorithm for preemptable, asynchronous and periodic reconfigurable task systems with arbitrary relative deadlines, scheduled on a uniprocessor by an optimal scheduling algorithm based on the EDF principles and on the dynamic reconfiguration. Two forms of automatic reconfigurations which are assumed to be applied at run-time: Addition-Remove of tasks or just modifications of their temporal parameters: WCET and/or Periods. Nevertheless, when such a scenario is applied to save the system at the occurrence of hardware-software faults, or to improve its performance, some real-time properties can be violated. We define a new semantic of the reconfiguration where a crucial criterion to consider is the automatic improvement of the system's feasibility at run-time by using an Intelligent Agent that automatically checks the system's feasibility after any reconfiguration scenario to verify if all tasks meet the required deadlines. Indeed, if a reconfiguration scenario is applied at run-time, then the Intelligent Agent dynamically provides otherwise precious technical solutions for users to remove some tasks according to predefined heuristic (based on soft or hard task), or by modifying the worst case execution times (WCETs), periods, and/or deadlines of tasks, that violate corresponding constraints by new ones, in order to meet deadlines and to minimize their response time. To handle all possible reconfiguration solutions, we propose an agent-based architecture that applies automatic reconfigurations in order to re-obtain the system's feasibility and to satisfy user requirements. Therefore, we developed the tool *RT-Reconfiguration* to support these contributions that we apply to a Blackberry Bold 9700 and to a *Volvo* system as running example systems and we apply the Real-Time Simulator *Cheddar* to check the whole system behavior and to evaluate the performance of the algorithm (detailed descriptions are available at the website: http://beru.univ-brest.fr/~singhoff/ cheddar). We present simulations of this architecture where we evaluate the agent that we implemented. Also, we present and discuss the results of experiments that compare the accuracy and the performance of our algorithm with others.

## INTRODUCTION

Real-Time systems are playing a crucial role in our society, and in the last two decades, there has been an explosive growth in the number of real-time systems being used in our daily lives and in industry production. Systems such as chemical and nuclear plant control, space missions, flight control systems, military systems, telecommunications, multimedia systems, and so on all make use of real-time technologies. The most important attribute of real-time systems is that the correctness of such systems depends on not only the computed results but also on the time at which results are produced. In other words, real-time systems have timing requirements that must be guaranteed. Scheduling and schedulability analysis enables these guarantees to be provided.

Common denominators for these embedded systems are real-time constraints. These systems are often safety critical and must react to the environment instantly on an event. Imagine for example the airbag of a car not going off instantly as a crash occurs; reaction time delay would be disastrous (H.Gharsellaoui, M.Khalgui, S.BenAhmed, 2011) [34]. Several interesting academic and industrial research works have been made last years to develop reconfigurable systems (A.-L. Gehin and M. Staroswiecki, 2008). We distinguish in these works two reconfiguration policies: static and dynamic reconfigurations where static reconfigurations are applied off-line to apply changes before the system cold start (C. Angelov, K. Sierszecki, and N. Marian, 2005) whereas dynamic reconfigurations are applied dynamically at run-time. Two cases exist in the last policy: manual reconfigurations applied by user (M. N. Rooker, C. Sunder, T. Strasser, 2007) and automatic reconfigurations applied by Intelligent Agents (M. Khalgui, 2010); (Al-Safi and V. Vyatkin, 2007).
Also, today in academy and manufacturing industry, many research works have been made dealing with real-time scheduling of embedded control systems. The new generations of these systems are addressing today new criteria as flexibility and agility. For this reason many reconfigurable embedded control systems have been developed in recent years.
In this book chapter, we are interested in the automatic reconfiguration of embedded real time Systems.
We define at first time a new semantic of this type of reconfiguration where a crucial criterion to consider is the automatic improvement of the system's feasibility at run-time. We propose thereafter an Agent-based architecture to handle all possible reconfiguration scenarios. Therefore, nowadays in industry, new generations of embedded real time systems are addressing new criteria as flexibility and agility. A disturbance is defined in this current book chapter as any internal or external event allowing the addition or removal of tasks to adapt the system's behavior. A reconfiguration scenario means the addition, removal or update of tasks in order to save the whole system on the occurrence of hardware/software faults, or also to improve its performance when disturbances happen at run time. To reduce their cost, these systems have to be changed and adapted to their environment without any disturbance. It might therefore be interesting to study the temporal robustness of real-time system in the case of a reconfiguration where the reconfiguration results in a change in the value of tasks parameters: WCET, deadline and period. This new reconfiguration semantic is considered in our work and we will present its benefits. We are interested in this work in automatic reconfigurations of real-time embedded systems that should meet deadlines defined in user requirements (S. Baruah and J. Goossens, 2004). A task is synchronous if its release time is equal to 0. Otherwise, it's asynchronous. These systems are implemented by sets of tasks that we assume independent, periodic and asynchronous (e.g. they are activated at any t time units). According to (Liu and Layland, 1973) we characterize each task , $\tau_i$ by a period to be denoted by $T_i$, by a deadline denoted by $D_i$, by an initial offset $S_i$ (a release time), and by a Worst Case Execution Time (WCET) denoted by $C_i$. We assume that the relative deadline of each task can be different from its corresponding period. We assume also that the whole system is scheduled by the earliest Deadline First (EDF) scheduling policy [9]. In single processor system, the Earliest Deadline First (EDF) scheduling algorithm is optimal [1], in the sense that if a task set is feasible, then it is schedulable by EDF. Therefore, the feasibility problem on single processor systems can be reduced to the problem of testing the schedulability with EDF. For this reason, the usage of Earliest Deadline First (EDF) scheduling policy is starting to catch the attention of industrial environments, given its benefits in terms of increased resource usage. EDF is now present at different layers of a real-time application such as programming languages, operating systems, or even communication networks. So, it is available in real-time languages like Ada 2005 [2] or RTSJ [3], and in real-time operating systems such as SHaRK [4]. It has been also implemented at the application level in OSEK/VDX embedded operating systems, and there are real-time networks using EDF for scheduling messages too; for instance in general purpose networks, or in the CAN Bus [5]. A feasible schedule is a schedule in which all tasks meet their deadlines. The feasibility problem for a set of independent periodic tasks to be scheduled on a single processor has been proven to be co-NP-complete in the strong sense [6, 7]. Leung and Merril [6] proved that it is necessary to analyze all deadlines in the hp = [0, 2*LCM+$\max_k(A_{k,1})$], where LCM is the well-known Least Common Multiple of all task periods and ($A_{k,1}$) is the earliest offset (starting time) of each task $\tau_k$. Baruah et al. [7] proved that, when the system utilization U is strictly less than 1, the Leung and Merrils condition is also sufficient. Reconfiguration policies are classically distinguished into two strategies: static and dynamic reconfigurations. Static

reconfigurations are applied offline to modify the assumed system before any system cold start [25], whereas dynamic reconfigurations can be divided into two cases: manual reconfigurations applied by users [26] and automatic reconfigurations applied by intelligent agents [27, 28]. This paper focuses on the dynamic reconfigurations of assumed asynchronous real-time embedded control systems that should meet deadlines defined according to user requirements [29]. This work is a complete generalization of [21, 22] work where the special case, of synchronous real-time OS tasks with EDF algorithm has been studied. Here, we define the dynamic reconfiguration as any change in software to lead the whole embedded system into a better safe state at run time. We define a new semantics of reconfigurations that allow automatic improvements of system performances at run-time even if there are no hardware faults [23]. The general goal of this paper is to be reassured that any reconfiguration scenario changing the implementation of the embedded system does not violate real-time constraints: i.e. the system is feasible and meets real-time constraints even if we change its implementation and to correctly allow the minimization of the response time of this system after any reconfiguration scenario. We define an automatic reconfiguration as any operation allowing additions-removes or updates of tasks at run-time. Therefore the system's implementation is dynamically changed and should meet all considered deadlines of the current combination of tasks. Nevertheless, when a reconfiguration is applied, the deadlines of new and old can be violated. We define an agent-based architecture that checks the system's evolution and defines useful solutions when deadlines are not satisfied after each reconfiguration scenario and the Intelligent Agent handles the system resources in such way that, meeting deadlines is guaranteed. Three cases of suggestions are possible to be provided by the agent: remove of some tasks from the new list, modification of periods or/and deadlines, and modification of worst case execution times of tasks. For this reason and in this original work, we propose a new algorithm for optimization of response time of this system. To obtain this optimization, we propose an intelligent agent-based architecture in which a software agent is deployed to dynamically adapt the system to its environment by applying reconfiguration scenarios. Before any reconfiguration scenario, the initial real-time embedded control system is assumed to be feasible. The problem is that when a scenario is applied and new tasks are added, the processor utilization U will be increased and/or some deadlines can be violated. We propose an agent that applies new configurations to change the periodicity, WCET of tasks or also to remove some of them as a worst case solution. The users should choose the minimum of these solutions to re-obtain the system's feasibility and to guarantee the optimality. The problem is to find which solution proposed by the agent that reduce the response time. To obtain these results, the intelligent agent calculates the processor utilization U before and after each addition scenario and calculates the minimum of those proposed solutions in order to obtain $R_k$ optimal noted $R_k^{opt}$, where $R_k^{opt}$ is the minimum of the response time of the current system under study given by the following equation:

$R_k^{opt}$ = min ( $R_{k,1}$, $R_{k,2}$, $R_{k,3}$ ). To calculate these previous values $R_{k,1}$, $R_{k,2}$ and $R_{k,3}$, we proposed a new theoretical concepts S (t, d), $\hat{S}(t,d)$ and W (t, d) for the case of real-time periodic asynchronous OS tasks. Where S(t, d) is the new function of job arrival with deadline at t time, $\hat{S}(t,d)$ is the new function of major job arrival with deadline at t time and W(t, d) is the amount of workload in wait of treatment of which the execution must be ended before the deadline d at t time. A tool named RT-Reconfiguration is developed in our research laboratory at INSAT University to support all the services offered by the agent. The simulator Cheddar [8] is used to verify and to prove the schedulability analysis of the considered tasks given by our tool. We give in the following section a useful background before we detail thereafter the book chapter problems and we present our contributions.

## BACKGROUND

The study of real-time embedded systems is growing at an exponential rate. Widespread deployment and complexity Software is becoming an important component of embedded systems, even the training manpower on the design and implementation of embedded software is becoming increasingly important. This section provides a review of the research related to our work. Users of this technology face a set of challenges: the need for fast, predictable, and bounded responses to events such as interrupts and messages, and also the ability to

manage system resources to meet processing deadlines. However, the Real-time and Embedded Systems Forum intends to plug this gap by bringing together system developers and users to build upon existing standards where they exist, evolve Product Standards that address market requirements, and develop Testing and Certification Programs that deliver products meeting these requirements. Ultimately the vision of the Forum is to grow the marketplace through the development of standardized systems based on real software solutions. Industry sectors that will benefit from the Forum include aerospace/defense, telecommunications, manufacturing, automotive, and medical/scientific research. This will advance standards development based on real software solutions. It will also establish test tools for suppliers to use to establish Confidence that their products conform. Consequently, the impact of software on the customer and, hence, on market shares and competition will be enormous. So, we can conclude that software is established as a key technology in the domain of real-time embedded systems (H.Gharsellaoui, M.Khalgui, S.BenAhmed, 2011) [34].

**Real-Time Scheduling**

**The Definition of "Real-Time"**

We consider a computing system or operating system to be a *real-time* one to the extent that: time-physical or logical, absolute or relative- is part of the system's logic and in particular, the completion time constraints of the applications' computations are explicitly used to manage the resources, whether statically or dynamically.

Time constraints, such as deadlines, are introduced primarily by natural laws- e.g., physical, chemical, biological -which govern an application's behavior and establish acceptable execution completion times for the associated real-time computations.

Real-time scheduling theory provides a formal framework for checking the schedulability of a tasks configuration and finding feasible, as well as optimal, scheduling. The aim of this section is to give a brief overview of this framework, and afterwards to introduce the notion of functional determinism. Real-time scheduling has been extensively studied in the last thirty years (S. Baruah and J. Goossens, 2004). Several Feasibility Conditions (FC) for the dimensioning of a real-time system are defined to enable a designer to grant that timeliness constraints associated to an application are always met for all possible configurations. Different classes of scheduling algorithm are followed nowadays: (i) Clock-driven: primarily used for hard real-time systems where all properties of all jobs are known at design time. (ii) Weighted round-robin: primarily used for scheduling a real-time traffic in high-speed, (iii) Priority-driven: primarily used for more dynamic real-time systems with a mixture of time-based and event-based activities. Among all priority driven policies, Earliest Deadline First (EDF) or Least Time to Go is a dynamic scheduling algorithm used in real-time operating systems. It places processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution. EDF is an optimal scheduling algorithm on preemptive uniprocessor in the following sense: if a collection of independent periodic jobs characterized by arrival times equal to zero and by deadlines equal to corresponding periods, can be scheduled by a particular algorithm such that all deadlines are satisfied, then EDF is able to schedule this collection of jobs.

**Reconfigurable Scheduling**

The nature of real-time systems presents us with the job of scheduling tasks that have to be invoked repeatedly. These tasks however may range from simple aperiodic tasks with fixed execution times to dynamically changing periodic tasks that have variable execution times.

Periodic tasks are commonly found in applications such as avionics and process control requiring data sampling on a continual basis. On the other hand, sporadic tasks are associated with event driven processing such as user response and non-periodic devices. Given a real-time system the goal of a good scheduler is to schedule the system's tasks on a processor, so that every task is completed before the expiration of the task deadline. These and some of the other issues like stability and feasibility are examined here.

**Scheduling Policies:**

A scheduling strategy consists in organizing the execution of a tasks set under constraints. Usually, scheduling strategies are classified as preemptive versus non-preemptive, and off-line versus on-line policies. In non-

preemptive case, each task instance, when started, completes its execution without interruptions. Conversely, in preemptive case, the scheduling unit can suspend a running task instance if a higher priority task asks for the processor. Off-line scheduling is based on a schedule which is computed before run-time and stored in a table executed by a dispatcher. One of the most popular off-line scheduling strategies is cyclic executive approach. With this method, tasks are executed in a predefined order, stored in a cyclic frame whose length is the least common multiple of the tasks periods. Each task can then be executed several times in the frame according to its period.

In the Round Robin scheduling algorithm at each instant, a scheduling policy chooses among the set of all active instances exactly one instance for being executed on the processing unit. In a uniprocessor system there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled. At any one time, a process can only be in one state and will continue to change states until it terminates. Figure 1 shows a state diagram of a process. In figure 1 the scheduler uses the Round-Robin scheduling algorithm that is designed especially for time-sharing systems. To implement the Round-Robin scheduling, we keep the ready queue as a FIFO (First In First Out) queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

Conversely, the idea of on-line scheduling is that scheduling decisions are taken at run-time whenever a running task instance terminates or a new task instance asks for the processor. The three most popular on-line scheduling strategies are Rate Monotonic (RM), Deadline Monotonic (DM) and Earliest Deadline First (EDF) (Liu and Layland, 1973). RM is an on-line preemptive static priority scheduling strategy for periodic and independent tasks assuming that T = D (period equals deadline) for each task t. The idea is to determine fixed priorities by task frequencies: tasks with higher rates (shorter periods) are assigned higher priority. DM is a generalization of RM with tasks such that $T_t = D_t$. In that case, tasks with shorter deadlines are assigned higher priority. EDF is a more powerful strategy. It is an on-line preemptive dynamic priority scheduling approach for periodic or aperiodic tasks. The idea is that, at any instant, the priority of a given task instance waiting for the processor depends on the time left until its deadline expires. Lower is this time, higher is the priority.

**Earliest Deadline First (EDF) Policy**

Earliest Deadline First (EDF) or Least Time to Go is a dynamic scheduling algorithm used in real-time operating systems. It places processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution. EDF is an optimal scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent *jobs*, each characterized by an arrival time, an execution requirement, and a deadline, can be scheduled (by any algorithm) such that all the jobs complete by their deadlines, the EDF will schedule this collection of jobs such that they all complete by their deadlines. In other hand, if a set of tasks is not schedulable under EDF, then no other scheduling algorithm can feasibly schedule this task set. So, compared to fixed priority scheduling techniques like rate-monotonic scheduling, EDF can guarantee all the deadlines in the system at higher loading. With scheduling periodic processes that have deadlines equal to their periods, EDF has a utilization bound of 100%. The necessary and sufficient condition for the schedulability of the tasks follows that for a given set of n tasks, $\tau_1$, $\tau_2 \ldots \tau_n$ with time periods $T_1$, $T_2 \ldots T_n$, and computation times of $C_1$, $C_2 \ldots C_n$, the deadline driven schedule algorithm is feasible if and only if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \qquad EXP1$$

Where U is the CPU utilization, $C_i$ is the worst-case computation-times of the *n* processes (Tasks) and the $T_i$ is their respective inter-arrival periods (assumed to be equal to the relative deadlines), (Liu and Layland, 1973).

We assumed that the period of each task is the same as its deadline. However, in practical problems the period of a task may at times be different from its deadline. In such cases, the schedulability test needs to be changed. If $T_i > D_i$, then each task needs $C_i$ amount of computing time every $min\ (T_i, D_i)$ duration of time. Therefore, we can rewrite $EXP1$ as:

$$\sum_{i=1}^{n} \frac{C_i}{\min(T_i, D_i)} \leq 1, \qquad EXP2$$

However, if $p_i < d_i$, it is possible that a set of tasks is EDF schedulable, even when the task set fails to meet the $EXP2$. Therefore, $EXP2$ is conservative when $T_i < D_{i,}$ and is not a necessary condition, but only a sufficient condition for a given task set to be EDF schedulable.

**Example**
Consider 3 periodic Tasks scheduled using **EDF**, the following acceptance test shows that all deadlines will be met.

| Tasks | Execution Time = C | Period = T=D |
|---|---|---|
| T1 | 1 | 8 |
| T 2 | 2 | 5 |
| T 3 | 4 | 10 |

***Table 0.1: A first task set example***

The utilization will be:

$$\frac{1}{8} + \frac{2}{5} + \frac{4}{10} = 0.925 = 92.5\%$$

The theoretical limit for any number of processes is 100% and so the system is schedulable.

Consider now 3 periodic Tasks scheduled using **EDF**, the following Figure (Figure 2) shows that all deadlines will be met.

| Tasks | Execution Time = C | Period = T | Deadline = D |
|---|---|---|---|
| T1 | 3 | 8 | 7 |
| T2 | 1 | 3 | 3 |
| T 3 | 1 | 7 | 6 |

***Table 0.2: A second task set example***

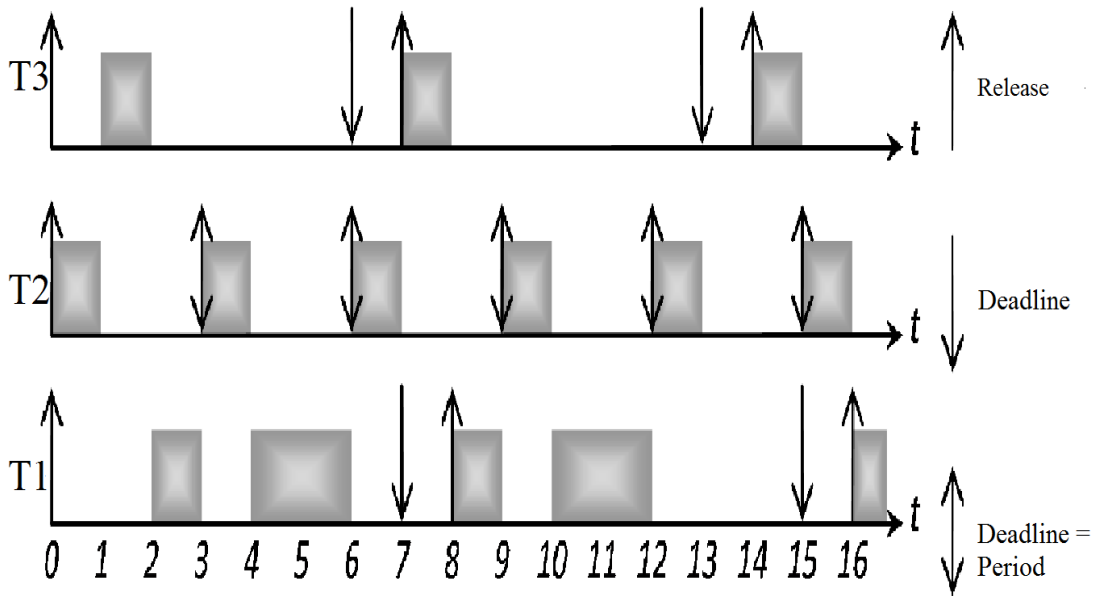


***Figure 1. Scheduling of the system described in Table 0.2 by EDF***

However, when the system is overloaded, the set of processes that will miss deadlines is largely unpredictable (it will be a function of the exact deadlines and time at which the overload occurs). This is a considerable disadvantage to a real time systems designer. The algorithm is also difficult to implement in hardware and there is a tricky issue of representing deadlines in different ranges (deadlines must be rounded to finite amounts, typically a few bytes at most). Also, the limitation of the EDF is that we cannot tell which tasks will fail during a transient overload. Even though the average case CPU utilization is less than 100%, it is possible for the worst-case utilization to go beyond and thereby the possibility of a task or two being aborted. It is desirable to have a control over which tasks fail and which does not; however, this is not possible in EDF. Therefore EDF is not commonly found in industrial real-time computer systems. The situation is somewhat better in RM because it is the low priority tasks that are preempted.

**Rate Monotonic Algorithm (RM Policy)**

This is a fixed priority algorithm and follows the philosophy that higher priority is given to tasks with the higher frequencies. Likewise, the lower priority is assigned to tasks with the lower frequencies. The scheduler at any time always chooses the highest priority task for execution. By approximating to a reliable degree the execution times and the time that it takes for system handling functions, the behavior of the system can be determined before. The rate monotonic algorithm can successfully schedule tasks in a static priority environment but it has bound of less that 100% efficiency. The CPU utilization of tasks $\tau_i$ where $1 \leq i \leq n$, is computed as the ratio of worst case computing time $C_i$ to the time period $T_i$. The total utilization of the CPU is computed as follows:

$$\mathbf{Un} = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad (1).$$

Here the frequency of the task is the reciprocal of the time period of the particular task. For the RM algorithm the worst-case schedulable time bound $W_n$ for a set of n tasks was shown to be:

$$\mathbf{Wn} = n*(2^{1/n} - 1) \quad (2).\ (\text{Liu and Layland, 1973})$$

From (2), we can observe that $W_1$ = 100%, $W_2$ = 83%, $W_3$ = 78% and as the task set grow in size, $W_n$ = 69% (ln2). Thus for a set of tasks for which the total CPU utilization is less than 69% means that all the deadlines will be met. The tasks are guaranteed to meet their deadlines if $U_n \leq W_n$. If $U_n > W_n$, then only a subset of the original task set can be guaranteed to meet the deadline which forms the upper echelon of the priority ordering. This set of tasks will be the critical set (Liu and Layland, 1973). Another problem that exists is the inability for RM to support dynamic changes in periods, which is a regular feature of dynamically configurable systems. For example, consider a task set of three $\tau_1$, $\tau_2$, and $\tau_3$, with time periods $T_1$=30 ms, $T_2$=50 ms and $T_3$=100 ms respectively. The priorities assigned are according to the frequency of occurrence of these tasks and so $\tau_1$ is the highest priority task. If the period for the first task changes to $T_1$=75ms, we would then under RM require that the priority orderings be changed to, $\tau_2$, $\tau_1$, and $\tau_3$. This change is detrimental to the completion of the scheduled jobs, which have to finish before their deadlines expire. The problem with RM encouraged the use of dynamic priority algorithms.

**Example**

| Tasks | Execution Time = C | Period = T | Deadline = D |
|---|---|---|---|
| T1 | 3 | 11 | 11 |
| T2 | 4 | 15 | 15 |
| T 3 | 1 | 5 | 5 |

***Table 0.3: A task set example***

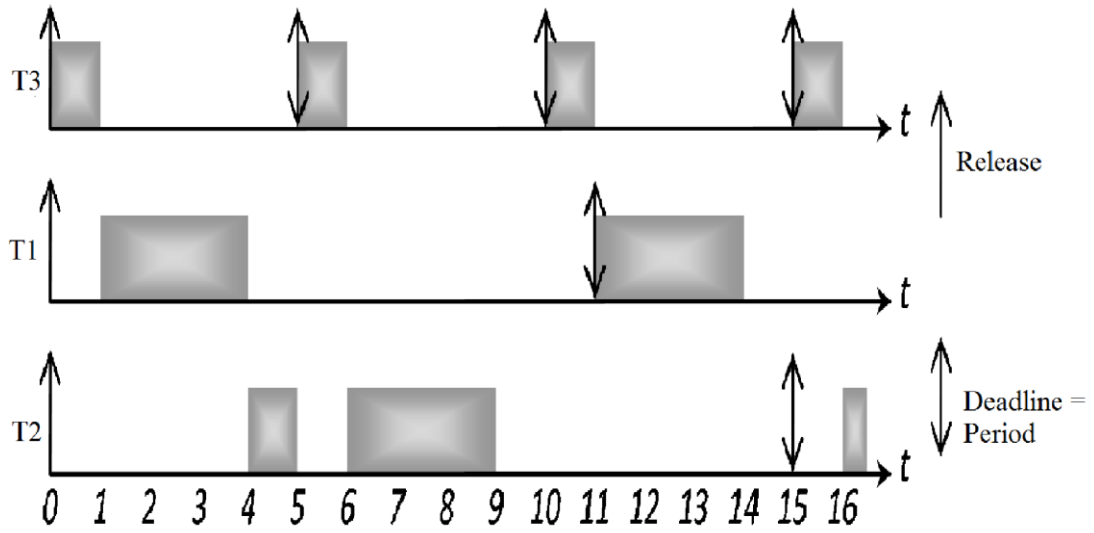


***Figure 2. Scheduling of the system described in Table 0.3 by RM***

**Deadline Monotonic Algorithm (DM Policy)**

The priority of a task under RM is proportional to the rate at which jobs in the task are released while the priority of a task under DM is inversely proportional to the relative deadline of the task. Also, priorities may also be assigned dynamically: One of the problems with RM is that many systems will need job deadlines shorter than the job's period which violates the assumption mentioned earlier. A solution to this problem arrived in 1982 with the introduction of the Deadline Monotonic (DM) algorithm (Leung W82). With DM, a job's

priority is inversely proportional to its relative deadline. That is to say, the shorter the relative deadline, the higher the priority. RM can be seen as a special case of DM where each job's relative deadline is equal to the period. However, the similarities end there. The "69%" feasibility test which we saw earlier doesn't work with DM. The DM feasibility test involves calculating how long it takes a job to go from the start of its period to the point where it finishes execution. We'll call this length of time the response time and denote it with R. After calculating R we then compare it with the job's relative deadline. If it is shorter then this job passes the test, otherwise it fails because a deadline can be missed. We have to check the feasibility of every job we define. New schedulability tests have been developed by the authors for the deadline monotonic approach (Audsley, 1990). These tests are founded upon the concept of *critical instants* (Liu and Layland, 1973). These represent the times that all processes (Tasks) are released simultaneously. When such an event occurs, we have the worst-case processor demand. Implicitly, if all processes can meet their deadlines for executions beginning at a critical instant, then they will always meet their deadlines. Thus, we have formed the basis for a schedulability test: check the executions of all processes for a single execution assuming that all processes are released simultaneously [34].

**Example**

| Tasks | Execution Time = C | Period = T | Deadline = D |
|---|---|---|---|
| T1 | 3 | 12 | 10 |
| T2 | 2 | 15 | 8 |
| T 3 | 1 | 5 | 5 |

***Table 0.4: A task set example***

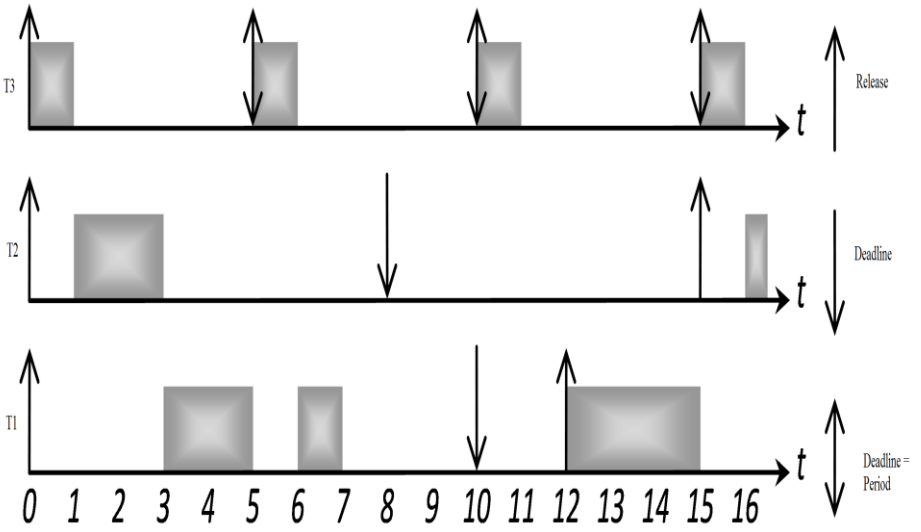


***Figure 3. Scheduling of the system described in Table 0.4 by DM***

**Least Laxity First (LLF) Algorithm**

Least laxity first algorithm (LLF) assigns priority bases upon the slack time of a task. The laxity time is temporal difference between the deadline, the remaining processing time and the run time. LLF always schedules first an available task with the smallest laxity. The laxity of a task indicates how much the task will

be scheduled without being delayed. LLF is a dynamic scheduling algorithm and optimal to use an exclusive resource. LLF is commonly used in embedded systems. Since the run time is not defined, laxity changes continuously. The advantage of allowing high utilization is accompanied by a high computational effort at schedule time and poor overload performance [34].

**Example**

| Tasks | Execution Time = C | Period = T | Deadline = D |
|---|---|---|---|
| T1 | 4 | 8 | 8 |
| T2 | 2 | 6 | 6 |

***Table 0.5: A task set example***

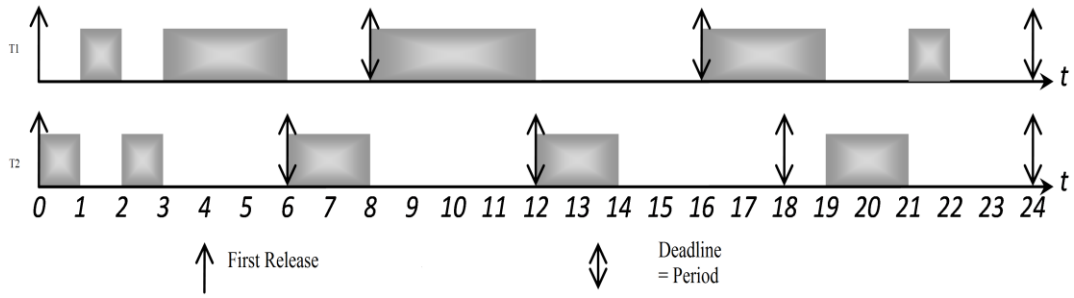


***Figure 4. Scheduling of the system described in Table 0.5 by LLF***

**Round Robin (RR) Algorithm**

Round Robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in circular order, handling all processes without priority. Round Robin scheduling is both simple and easy to implement. Effectiveness and efficiency of RR are arising from its low scheduling overhead of (1), which means scheduling the next task takes a constant time. In Round Robin Scheduling, the time quantum is fixed and then processes are scheduled such that no process get CPU time more than one time quantum in one go. If time quantum is too large, the response time of the processes is too much which may not be tolerated in interactive environment. If time quantum is too small, it causes unnecessarily frequent context switch leading to more overheads resulting in less throughput [34].

**Example**

| Tasks | Execution Time = C | Period = T | Deadline = D |
|---|---|---|---|
| T1 | 6 | 14 | 14 |
| T2 | 3 | 14 | 14 |
| T 3 | 4 | 14 | 14 |

***Table 0.6: A task set example***

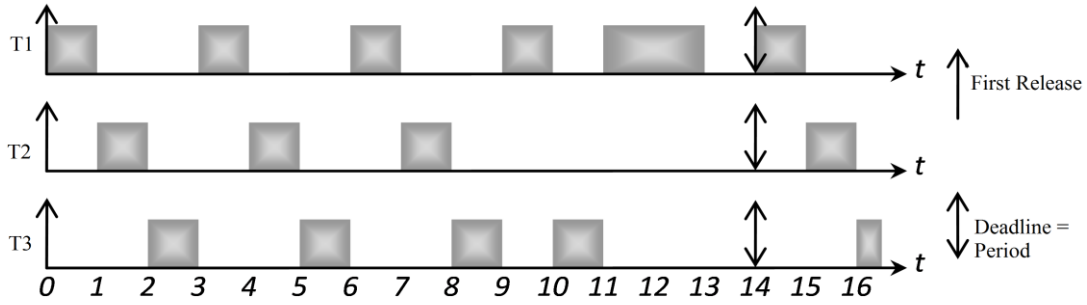


***Figure 5. Scheduling of the system described in Table 0.6 by RR***

## STATE OF THE ART ON RECONFIGURABLE EMBEDDED SYSTEMS

A task is an executable program implementing one and only one functional module. A task may be periodic, sporadic or aperiodic. In most cases, especially in the context of critical systems, tasks are supposed to be periodic. In the following, we only consider periodic tasks. According to (Liu and Layland, 1973), periodic task may be characterized by static parameters (T, r, D, B, W) where T is the task period, r is the release date (first activation), D is the (relative) deadline, and (B and W) are the best and worst case execution time (BCET and WCET). B and W depend on multiple elements: the processor, the compiler, the Memories. Estimation of these parameters is a wide research area which is considered in the scope of this work. Few results have been proposed to deal with deadline assignment problem. In (Baruah, Buttazo, Gorinsky, & Lipari, 1999), the authors propose to modify the deadlines of a task set to minimize the output, seen as secondary criteria of this work.

In (Cervin, Lincoln, & G., 2004), the deadlines are modified to guarantee close-loop stability of a real-time control system. In (Marinca, Minet, & George, 2004), a focus is done on the deadline assignment problem in the distributed for multimedia flows. The deadline assignment problem is formalized in term of a linear programming problem. The scheduling considered on every node is non-preemptive EDF or FIFO with a jitter cancelation applied on every node. A performance evaluation of several deadline assignment schemes is proposed. In (Balbastre, & Crespo, 2006), the authors propose an optimal deadline assignment algorithm for periodic tasks scheduled with preemptive EDF in the case of deadline less than or equal to periods. The goal is to find the minimum deadline reduction factor still meeting all the deadlines of the tasks.

In the case of a variable speed processor, reducing the frequency can create overloads that can result in deadline miss. We identify several approaches to deal with overloads conditions:

- Remove some tasks to come back to a normal load.
- Adapt the task parameters to come back to a normal load
  – Modification of periods (or deadlines),
  – Modification of worst case execution times of tasks,

In the first case, several solutions from the state of the art have been proposed:

- Stop the faulty task or put it in background. This is the solution used by most industrial systems. Probably not the best.
- Use a heuristic to remove some tasks. In (Lock, 1986), the author proposes to remove the task with the lowest importance. The importance is characterized by a Time Value Function (TVF) providing a statistical overload management with no guarantee to solve the overload problem.
- Applied for EDF scheduling, REDF (robust earliest deadline first) described in (Buttazzo, & Stankovic, 1993), where a partitioning of critical real-time tasks and non-critical real-time tasks is proposed. The critical tasks should always meet their deadlines. The non critical tasks are removed if necessary according to their value density. A task $\tau_i$ has a value $v_i$ and a value density $v_i/C_i$. With this mechanism, for an identical value, the task having a long duration will be removed first.
- Applied for EDF scheduling, D-OVER proposed in (Koren, & Shasha, 1992), where the authors assigns a Time Value Function (TVF) to every task. A value equal to 0 is equivalent to a deadline miss. The goal is to obtain the maximum value among all the tasks. They prove that their algorithm is optimal in the sense that is achieves the maximum possible benefit for an on-line algorithm (1/4 of an omniscient algorithm).

In the second case, the task parameters must be adapted on-line to cope with the overload. The idea is to adapt the periods of the tasks when needed to reduce the processor utilization. This approach has been proposed in the case of equally important tasks by gracefully adjusting the task periods. Other related papers are detailed in (Buttazzo & al., 2004). In this paper, they introduce a novel scheduling framework to propose a flexible workload management a run time. They present the concept of elastic scheduling (introduced in Buttazzo, G., Lipari, & Abeni, 1998). The idea behind the elastic model is to consider the flexibility of a task as a spring able to increase or decrease its length according to workload conditions. The length of a spring is associated to the current processor utilization of its tasks. For a periodic task $\tau_i$, the period $T_i$ is the actual period and is supposed to range from $T_i^{min}$ to $T_i^{max}$. The processor utilization of $\tau_i$ is $C_i/T_i$. The period adaptation is done with a new parameter: $E_i$ defining an elastic coefficient. The greater $E_i$, the more elastic the task. Decreasing processor utilization result is applying a compression force on the spring that results in a period decrease. This model is well adapted to the case of deadlines equal to periods as it is possible in this case to derive sufficient feasibility for Fixed Priority (FP) with Rate Monotonic algorithm (Liu and Layland, 1973); (Bini, & Buttazzo, 2003), and necessary and sufficient feasibility conditions for EDF (Liu and Layland, 1973) based on the processor utilization $U$. In this case, determining if a task set is still schedulable after a task period change is not complex and can be done at run time. In (Buttazzo, 2006), the author proposes to use also the elastic model to adapt the period of the tasks to reach high processor utilization in the case of discrete voltage levels in variable speed processors. In soft real-time systems, another approach has been proposed, to bound the number of deadline miss. The (m, k)-firm approach introduced in (Hamdaoui & Ramanathan, 1995), can be used to specify that a task should have at least $m$ consecutives instances over $k$ meeting their deadlines. This algorithm, first conceived in the context of message transmission, is a best effort algorithm. In (Bernat, Burns & A., L. 2001), the authors propose to extend the *(m, k)-firm* model with the *Weakly-hard* model, considering non consecutives deadline miss.

In (Balbastre, & Ripoll, 2002), the authors show how much a task can increase its computation time still meeting the system feasibility when tasks are scheduled EDF. They consider the case of only one task increasing its WCET.

**Theorem 6** (Bini, E. & Di Natale, 2005)**:**
*Let $\tau^C$ be the task set where every WCET is multiplied by a scaling factor $\alpha$. Let $\tau^T$ be the task set where every Task period is divided by $\alpha$. The task set $\tau^C$ is schedulable if and only if the task set $\tau^T$ is schedulable.*

Laurent George and Pierre Courbin considered in their works the benefits of sensitivity analysis for the reconfiguration of sporadic tasks when only one task parameter can evolve (WCET, Period or Deadline). In the case where applications are defined by several modes of execution, a reconfiguration consists of a mode change. A mode is defined its task set. Changing the mode of an application changes the task set run by the system. The problem of mode change is to study if it is possible end a mode and start a new one still preserving all the timeliness constraints associated to all the tasks in both modes. Mode change is a current active research area and has been considered for e.g. in (Nelis, Goossens, & Andersson, 2009), Sensitivity analysis could be used to determine if a mode change results in acceptable WCET, period or deadline changes. Finally, we believe in the limitation of all these related works in particular cases and we note that all these related works consider the reconfiguration of only one task parameter which can evolve (WCET, Period or Deadline). The only research work dealing with multi-parameters reconfiguration is that we propose in the current book chapter in which we give solutions to the user for all these problems presented by the tool RT-*Reconfiguration.*

**PROBLEM**

Embedded systems architecture is classically decomposed into three main parts. The control software is often designed by a set of communicating functional modules, also called tasks, usually encoded with a high level programming language (e.g. synchronous language) or a low level one (e.g. Ada or C). Each functional module is characterized by real-time attributes (e.g. period, deadline) and a set of precedence constraints. The material architecture organizes hardware resources such as processors or devices. The scheduler decides in which order functional modules will be executed so that both precedence and deadline constraints are satisfied. Behavioral correctness is proved as the result of the logical correctness, demonstrated with the use of formal verification techniques (e.g. theorem proving or model-checking) on the functional part, and the real-time correctness which ensures that all the computations in the system complete within their deadlines. This is a non trivial problem due both to precedence constraints between tasks, and to resource sharing constraints. This problem is addressed by the real-time scheduling theory which proposes a set of dynamic scheduling policies and methods for guaranteeing/proving that a tasks configuration is schedulable. However, in spite of their mutual dependencies, these two items (functional verification and schedulability) are seldom addressed at the same time: schedulability methods take into account only partial information on functional aspects, and conversely the verification problem of real-time preemptive modules has been shown undecidable. To overcome this difficulty, a third property is often required on critical systems, especially for systems under certification: determinism, i.e. all computations produce the same results and actions when dealing with the same environment input. The benefit of this property, if ensured, is to limit the combinatorial explosion, allowing an easier abstraction of real-time attributes in the functional view. For instance, preemptive modules may be abstracted by non preemptive ones characterized by fixed beginning and end dates. The interesting consequence is to allow separated functional and real-time analyses. For ensuring determinism, two ways can be followed: either to force it, or to prove it. Several approaches were proposed in order to guarantee determinism. One of the simplest manners is to remove all direct communications between tasks. This seems quite non realistic but it can be achieved by developing an adequate architecture, for instance, current computed data are stored in a different memory while consumed input are the ones produced in a precedent cycle. The execution order between tasks within each cycle does not impact the produced values. However, the main disadvantage is to lengthen the response time of the system. This solution is then not suitable for systems requiring short response time.

A second approach is based on off-line non preemptive strategies, such as cyclic scheduling. Provided that functional modules are deterministic, the global scheduled behavior will also be deterministic. This solution is frequently followed by aircraft manufacturer for implementing critical systems such as a flight control system. However this strategy has two main several drawbacks. Firstly this scheduling leads to a low use of resources because tasks are supposed to use their whole worst case execution time (WCET). To overcome this first

problem, tasks are required to be as small as possible. Secondly, off-line scheduling strategies often need for over-dimensioning the system in order to guarantee acceptable response times to external events. For that purpose, tasks periods are often to be reduced (typically divided by 2) compared to the worse period of the polled external events. The guaranty that WCET and BCET (resp. worst and best case execution times) coincide provides a third interesting context. Any off-line scheduling is then deterministic and it is possible to modify the task model to produce a deterministic on-line scheduling. Unfortunately, warranting that BCET is equal to WCET is hardly possible. This can limit the programming task (no use of conditional instruction or on the contrary use of dead code to enforce equality). Other more recent approaches are based on formal synchronous programming languages. Systems are specified as deterministic synchronous communicating processes, and are implemented either by a sequential low level code which enforces a static execution order, or by a set of tasks associated with static or dynamic scheduling policies. Implementation is correct-by-construction, i.e., it preserves the functional semantics (and then determinism). These approaches are interesting, for they allow to by-pass the determinism verification problem. Previous solutions are not suitable for highly dynamic non synchronous systems with high Workload.

On-line preemptive scheduling strategies are often optimal, easy to implement, but deeply non deterministic when associated to asynchronous communication models. Problematic reconfiguration appears when there are temporal indeterminism on execution time and preemption. Consequently, if on-line preemptive scheduling policies are needed (for performance reasons for instance), it is the necessary to verify determinism. The aim of this book chapter is to answer the question is a scheduling deterministic for a particular multi-periodic tasks model and a given policy? The result is that the determinism problem is decidable even in case of preemptive on-line scheduling policies. So, Due to the increasing complexity of the developed systems it is necessary to model correctly and to implement the chosen design in a correct manner. In the rest of this Book Chapter, we only consider single-processor systems.

## CONTRIBUTIONS

In addition, before the main contributions are explained in this book chapter, we are interested in automatic reconfigurations of real-time embedded systems that should meet deadlines defined in user requirements. These systems are implemented sets of tasks that we assume independent, periodic and synchronous (e.g. they are simultaneously activated at time t = 0 time units). We assume also that the deadline of each task is equal to the corresponding period. We define an agent-based architecture that checks the system's evolution and defines useful solutions when deadlines are not satisfied after each reconfiguration scenario and the Intelligent Agent handles the system resources in such way that, meeting deadlines is guaranteed. The resulting contributions of this Book Chapter can be divided into five cases of suggestions are possible to be provided by the agent:

- ✓ Remove of some tasks from the new list,
- ✓ Modification of periods and/or deadlines,
- ✓ Modification of worst case execution times of tasks,

The general problem of our project is to be reassured that any reconfiguration scenario changing the implementation of the embedded system does not violate real-time constraints: i.e. the system is feasible and meets real-time constraints even if we change its implementation.

**Formalization of Reconfigurable Real-Time Embedded Systems**

Nowadays, manual and automatic reconfigurations are often useful technical solutions to change the system's behavior at occurrences of software/hardware faults or also to improve the performance. Let *Sys* be such system to be classically composed of a set of real-time tasks that support all different functionalities. We mean by a dynamic reconfiguration any operation allowing addition, removal or also update tasks at run-time. Let *Sys* be the set of all possible tasks that can implement the system, and let us denote by $Current_{Sys}(t)$ the current set of tasks implementing the system *Sys* at *t* time units. These tasks should meet all required deadlines defined in user requirements. In this case, we note that $Feasibility(Current_{Sys}(t)) \equiv True$.

**Example:**
Let us suppose a real-time embedded system (*Volvo system)* to be initially implemented by 5 characterized tasks (Table 0.7). These tasks are feasible because the processor utilization factor U = 0.87 < 1. These tasks should meet all required deadlines defined in user requirements and we have Feasibility ($Current_{V\ olvo}(t)$) ≡ True.
We suppose that a reconfiguration scenario is applied at t1 time units to add 3 new tasks C; G; H.
The new processor utilization becomes U = 1.454 > 1 time units. Therefore the system is unfeasible.
Feasibility ($Current_{V\ olvo}(t)$) ≡ False.

| Task | $T_i$ | $C_i$ | $D_i$ | $U^{100}$ | $U_{asy}$ | $U_{OPT}$ |
|---|---|---|---|---|---|---|
| **A** | 10 | 2 | 10 | 20% | 20% | 4.7% |
| **B** | 20 | 2 | 5 | 10% | 40% | 4% |
| **D** | 50 | 6 | 50 | 12% | 12% | 1.6% |
| **E** | 100 | 8 | 100 | 8% | 8% | 5.6% |
| **F** | 2000 | 7 | 100 | 7% | 7% | 9% |
| **C** | 50 | 1 | 2 | 2% | 50% | 1% |
| **G** | 2000 | 8 | 100 | 8% | 8% | 18.6% |
| **H** | 2000 | 8 | 2000 | 8% | 0.4% | 18.6% |

***(Table 0.7): The Volvo case study***

**Agent-based architecture for Reconfigurable Embedded Control Systems**

We define in this section an agent-based architecture for reconfigurable real-time embedded systems that should classically meet different deadlines defined in user requirements. The agent controls all the system's evolution and provides useful solutions for users when deadlines are violated after any dynamic (manual or automatic) reconfiguration scenario.

*Running Example:*
In our real-time embedded system *Volvo* to be initially implemented by 5 characterized tasks which are feasible because the processor utilization factor U = 0.87 < 1. We suppose that a reconfiguration scenario is applied at t1 time units to add 3 new tasks C; G; H. The new processor utilization becomes U = 1.454 > 1 time units. Therefore the system is unfeasible. Feasibility ($Current_{V\ olvo}(t)$) ≡ False.

To check the whole system behavior of this system *Volvo*, we present simulations given by the real-time simulator Cheddar in Figure 6.

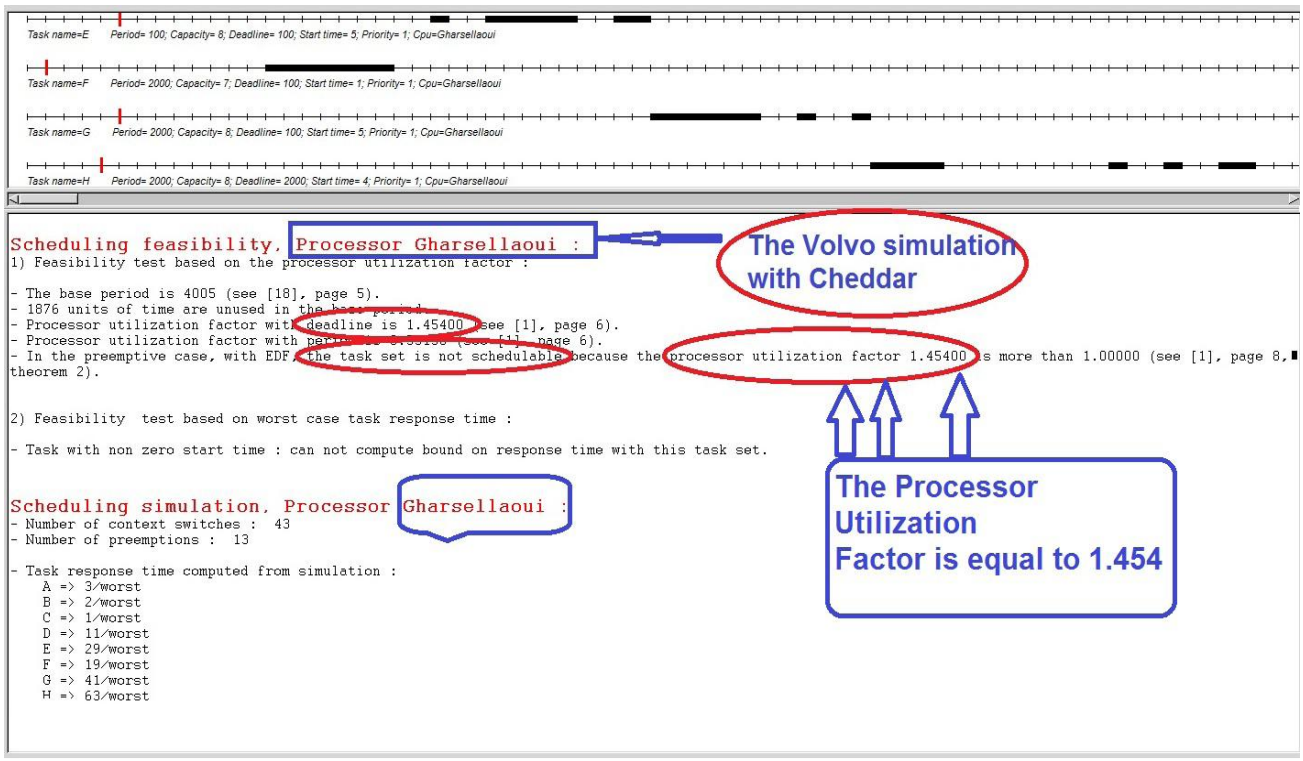


***Figure 6. Simulations of the Volvo case study with Cheddar***

**CONTRIBUTION 1: NEW THEORETICAL PRELIMINARIES**

This section aims to define a new theoretical preliminaries for a set of asynchronous real time tasks scheduling under EDF based on the concepts defined in [21, 22], which compute a feasible schedule for a set of synchronous real time tasks scheduling under EDF. These new theoretical preliminaries will be used in the following two contributions. Our main contribution is the optimal schedulability algorithm of uniprocessor periodic real-time tasks implementing reconfigurable systems. By applying a preemptive scheduling, the assumed system is characterized by periodic tasks such that each one is defined by a tuple ($S_i$;$C_i$;$D_i$;$T_i$). A system is called asynchronous, if its tasks have offsets and are not simultaneously ready. Note that in synchronous systems, all offsets are zero and all tasks are released at time t = 0. In this work, when a hardware problem occurs in the studied system, a reconfiguration scenario is automatically applied in this system which has to react by changing its implementation from a subset of tasks to a second one. A reconfiguration scenario corresponds therefore to the addition, the removal or the update of real-time tasks. The general problem of our project is to be reassured that any reconfiguration scenario changing the implementation of the system does not violate real-time constraints: i.e. the system is feasible and meets real-time constraints even if we change its implementation. In this work also, we analyze the feasibility of real-time systems with a single processor by using preemptive Earliest Deadline First (EDF) scheduling algorithm. EDF is an optimal scheduling algorithm on preemptive uniprocessors in the following sense: if a collection of independent periodic (synchronous and asynchronous) jobs can be scheduled by a particular algorithm such that all deadlines are satisfied, then EDF is able to schedule this collection of jobs (instances) [1].

We propose in this paper an agent-based architecture that checks the system's evolution and defines useful solutions for users when deadlines are violated. Therefore, for every set of OS tasks, we check its on-line feasibility by executing it when the corresponding reconfiguration scenario is automatically applied by the agent at run-time.

We apply these different solutions on the *Volvo* Benchmark (Industrial Case Study) that we consider as running example.

***Formalization:***

By considering asynchronous real-time tasks, the schedulability analysis should be done in the Hyper-Period hp = [0, 2*LCM + $\max_k(A_{k,1})$], where LCM is the well-known Least Common Multiple and ($A_{k,1}$) is the earliest start time (arrival time) of each task $\tau_k$ [11]. The reconfiguration of the system $Current_\Gamma$ (t) means the modification of its implementation that will be as follows at t time units: $Current_\Gamma(t) = \xi\_new \cup \xi\_old$ Where $\xi\_old$ is a subset of n1 old periodic tasks which are asynchronous and not affected by the reconfiguration scenario (e.g. they implement the system before the time t), and $\xi\_new$ is a subset of n2 new asynchronous tasks in the system. We assume that an updated task is considered as a new one at t time units. By considering a feasible System *Sys* before the application of the reconfiguration scenario, each task of $\xi\_old$ is feasible, e.g. the execution of each instance is finished before the corresponding deadline:

– Let n1 and n2 be the number of tasks respectively in $\xi\_old$ and $\xi\_new$ such that n1 + n2 = n (the number of a mixed workload with periodic asynchronous tasks in $Current_\Gamma$ (t)). To estimate the amount of work more priority than a certain under EDF, it is inevitably necessary to us to take into account deadlines because the more priority work is the work which has the earliest deadline. In particular, we propose one function of job arrival with deadline, one function of workload with deadline and finally, we propose the function of major job arrival with deadline for periodic asynchronous tasks.

**For example**, In the *Volvo* case study, we have the first subset $\xi\_old$ composed of the following five initial tasks $\xi\_old$ = {A; B; D; E; F} (n1 = 5), this system is feasible and U = 0.87. We suppose that a reconfiguration scenario is applied at t time units to add a second subset composed of three new tasks {C; G; H} = $\xi\_new$ (n2 = 3).

Therefore, the system $Current_\Gamma$ (t) is composed of eight tasks (n = 8 = 3 + 5) as shown in table 0.7 and it's unfeasible. Feasibility($Current_\Gamma$ (t)) ≡ False.

By applying the well-known scheduling real-time simulator Cheddar [8], the EDF scheduling result is shown in figure 6. The processor utilization factor (U) becomes equal to 1.454 after adding the 3 new tasks and the task set seems to be not schedulable.

***New function of job arrival with deadline:***

We propose new functions of job arrival which integrate the deadlines by the following levels:

***– In the instance level:***

$S_{k,n}(t1,t2,d) = C_{k,n} * \Pi_{[t1 \le Ak,n < t2]} * \Pi_{[Dk,n \le d]} = S_{k,n}(t1,t2) * \Pi_{[Dk,n \le d]}$ Where $S_{k,n}(t1,t2,d)$ is the amount of job with lower deadline or equal to d brought by the instance $\tau_{k,n}$ meanwhile of time [t1,t2[, and Π [α] = 1 if the predicat α = true.

***– In the task level we propose:***

$S_k(t1,t2,d) = \sum_{n \in \aleph} C_{k,n} * \Pi_{[t1 \le Ak,n < t2]} * \Pi_{[Dk,n \le d]}$, Where $S_k(t1,t2,d)$ is the amount of job with lower deadline or equal to d brought by all the instances of $\tau_k$ meanwhile of time [t1,t2[.

***– For a set of tasks Γ we propose:***

$S_{Current\ \Gamma(t)}(t1, t2, d) = \sum_{i \mapsto \tau_i in Current\Gamma(t)} S_i(t1, t2, d)$, Where $S_{Current\ \Gamma(t)}$ (t1, t2, d) is the amount of job with lower deadline or equal to d brought by all the instances of tasks that composed $Current_\Gamma$(t) meanwhile of time [t1, t2 [.

***New function of workload with deadline:***

In the study of the EDF policy, it is necessary to us to know at the certain moments the workload in wait of treatment of which the execution must be ended before a certain deadline. So, we propose one function of workload with deadline:

***– In the instance level:***

$W_{k,n}(t,d) = S_{k,n}(A_{k,1},t,d) - \int_{A_{k,1}}^{t} \prod_{k,n}(u,d)du$ **(a)**

Where $\prod_{k,n}(t,d) = \prod_{k,n}(t) * \Pi_{[Dk,n \le d]}$. $W_{k,n}(t,d)$ is the amount of job with lower deadline to d brought by the instance $\tau_{k,n}$ which again is to be executed at the moment t. If $A_{k,1} = 0$, we restraint to the case of synchronous tasks.

***– In the task level:***

$W_k(t,d) = S_k(A_{k,1},t,d) - \int_{A_{k,1}}^{t} \prod_{k}(u,d)du = \sum_{n \in \aleph} W_{k,n}(t,d)$

Where $W_k(t,d)$ is the amount of job with lower deadline to d brought by all the instances of $\tau_k$ which gain is to be executed at the moment t.

***– For a set of tasks Γ:***

For the Current $_\Gamma$ (t) = $\xi\_new \cup \xi\_old$, we propose: $W_{Current\ \Gamma(t)}(t,d) = \sum_{i \mapsto \tau_i in Current\Gamma(t)} W_i(t,d)$

$= S_{Current\ \Gamma(t)}(A_{k,1},t,d) - \int_{A_{k,1}}^{t} \prod_{\Gamma}(u,d)du$, Where $W_{Current\ \Gamma(t)}(t,d)$ is the amount of job with lower deadline to d brought by all the instances of tasks that composed Current $_\Gamma$ (t) which again is to be executed at the moment t.

**CONTRIBUTION 2: AGENT-BASED REAL-TIME RECONFIGURABLE MODEL**

This section aims to propose an intelligent Agent-based architecture which is able to propose technical solutions for users after any dynamic reconfiguration scenario.

***Agent's Principal***

Let Γ be the set of all possible tasks that can implement the system, and let us denote by Current $_\Gamma$ (t) the current set of periodic asynchronous tasks implementing the system at t time units. These tasks should meet all required deadlines defined in user requirements. By considering a feasible System Γ before the application of the reconfiguration scenario, each one of the tasks of $\xi\_old$ is feasible, e.g. the execution of each instance is finished before the corresponding deadline. In this case, we note that Feasibility (Current $_\Gamma$ (t)) ≡ True.

An embedded system can be dynamically reconfigured at run-time by changing its implementation to delete old or to add new real-time tasks. We denote in this research by $\xi\_new$ a list of new asynchronous tasks to be added to Current $_\Gamma$ (t) after a particular reconfiguration scenario. In this case, the intelligent agent should check the system's feasibility that can be affected when tasks violate corresponding deadlines, and should be able to propose technical solutions for users.

Let us return now, to the equation **(a)**, we can notice that for $d < D_{k,n}$, we have $W_{k,n}(t,d) = 0$ for any value of t and the $W_{k,n}(t,D_{k,n}) > 0$ for $t > D_{k,n}$ which involves that $\tau_{k,n}$ cannot meet its deadline, (it can violates it). Consequently, the task $\tau_k$ can violates also its relative (corresponding) deadline and all the system Current $_\Gamma$ (t) will be unfeasible at t time units. In this case the following formula is satisfied: $\sum_{i=1}^{n} \frac{C_i}{\min(T_i, D_i)} > 1$,

Now, we apply at time t a dynamic reconfiguration scenario in order to adapt the system's behavior and to guarantee the system's feasibility which depends of two major goals of the reconfiguration:

– The first major goal to control the problem's complexity is to minimize response time of asynchronous periodic tasks of Current $_\Gamma$ (t) = $\xi\_new \cup \xi\_old$ , then the agent will not modify the $\xi\_old$ tasks and should provide different solutions for users by reconfigure only $\xi\_new$ which is composed by $n_2$ asynchronous periodic tasks in order to satisfy functional requirements,

– The second major goal of obtaining the system's feasibility is to meet deadlines of asynchronous periodic tasks, then, the agent should react by updating of the global system Current $_\Gamma$ (t) = $\xi\_new \cup \xi\_old$ , which is composed by $n_1$ and $n_2$ asynchronous periodic tasks in order to re-obtain the system's feasibility and provides different solutions for users.

***First Case: Minimizing the response time of periodic tasks***

In this case, the objective is to reduce the periodic response times as much as possible, still guaranteeing that all periodic tasks complete within their deadlines.

**Solution 1: Removal of Tasks (1)**

We define in this solution a perfect admission controller as a new heuristic, which is defined as an admission control scheme in which we always admit a task if and only if it can be scheduled. Such a control policy can be implemented as follows. Whenever a task arrives, the agent computes the processor utilization Ci/min(Ti,Di) of each task $\tau_i$ and generates the feasible superset $\Omega_{feasible}$ which defines the different feasible subsets of tasks in achieving good periodic responsiveness where U(t) = $\sum_{i=1}^{n} \frac{C_i}{\min(T_i, D_i)} \leq 1$, is enforced.

$\Omega_{feasible}$ = {$\tau \subseteq$ Current $_\Gamma$ /Feasibility ($\tau$) ≡ True}

Each subset $\tau$ corresponds to a possible implementation of the system such that: $\tau = \xi\_new \cup \xi\_old$

$$\sum_{\tau_i \in Asynchronous\_Tasks} \frac{C_i}{\min(T_i, D_i)} \leq 1 \quad [20]$$

In this case we remove all tasks of $\xi\_new$ , we stock them in a list and we begin by using an acceptance test, e.g., periodic tasks $\epsilon$ $\xi\_new$ that would cause U (t) to exceed this bound are not accepted for processing. In other words, when a task arrives at the system, it is tentatively added to the set of tasks in the system. The admission controller then tests whether the new task set is schedulable. The new task is admitted if the task set is schedulable, e.g., would not cause U (t) to exceed the bound (U (t) = $\sum_{i=1}^{n_1+j} \frac{C_i}{\min(T_i, D_i)} \leq 1$ is enforced where j $\in$ [0*;* $n_2$]). Otherwise, there are two possible cases:

– ***First case:*** if the arrival task is hard, then it will be accepted and we will randomly remove another soft task from the [1... n1 + j −1] previous tasks to be rejected and still guaranteeing a feasible system,

– ***Second case:*** if the arrival task is soft, it will be dropped (rejected) immediately.

The agent computes the processor utilization $C_i$/min ($T_i$,$D_i$) of each task $\tau_i$ and generates the feasible superset $\Omega_{feasible}$ which defines the different feasible subsets of tasks.

The agent suggests all possible combinations of tasks for users who have the ability to choose the best combination that satisfies functional requirements.

***Running Example:***

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our *Volvo* system, we present in Figure 7 the results described by the developed tool RT - Configuration. These

results are the possibilities of the considered tasks which can be removed from the subset. The user can choose one of these solutions to have a feasible system.
Here the agent proposes the task **C** to be removed to re-obtain the system's feasibility.

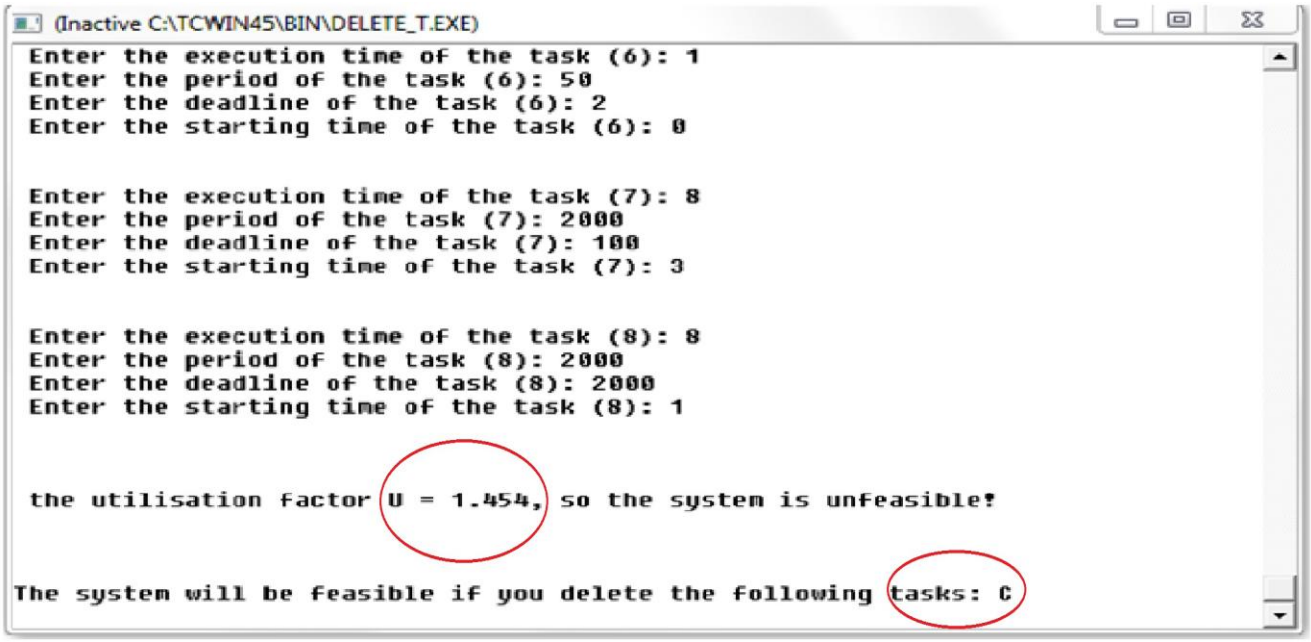


***Figure 7: the Volvo case study simulation***

By applying the well known scheduling real-time simulator Cheddar [8], the EDF scheduling result is shown in figure 8.

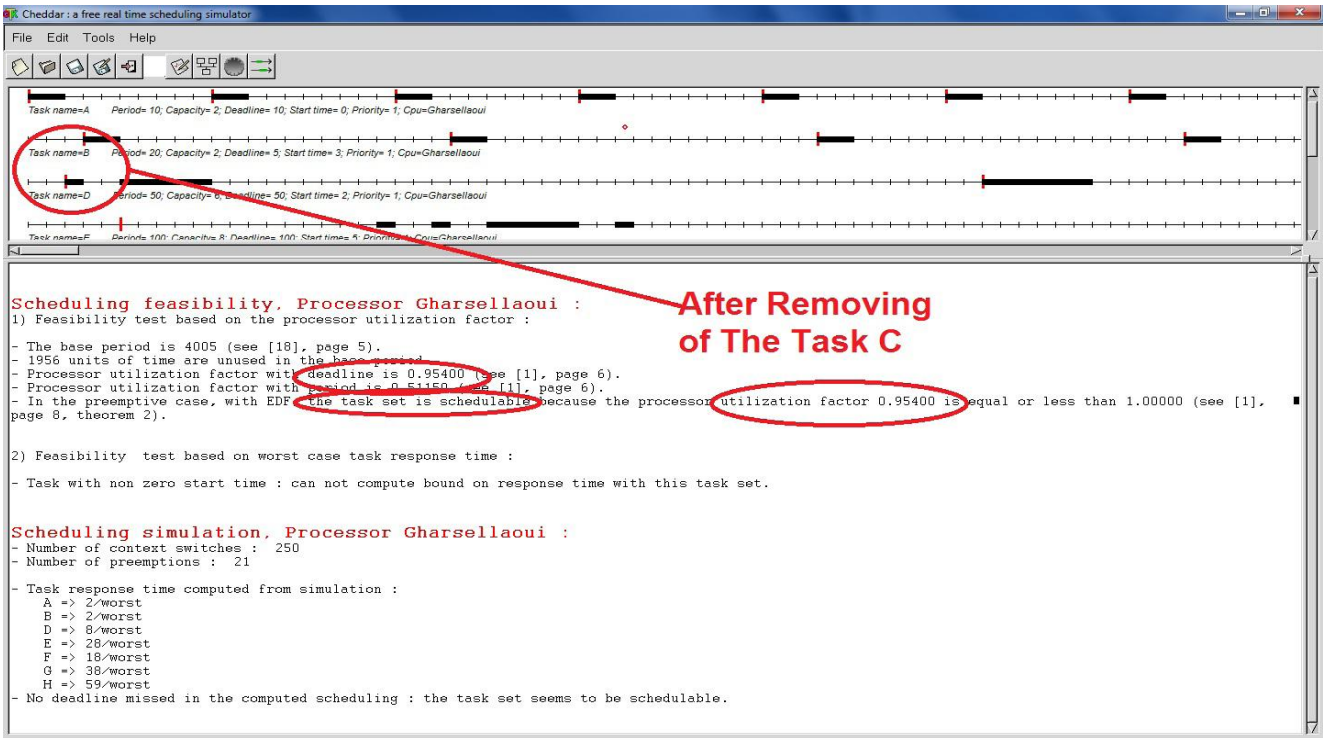


***Figure 8: the Volvo case study simulation with Cheddar***

The processor utilization factor (U) becomes equal to 0.954 after removing the task C, and the task set becomes schedulable (feasible).

**Second Case: Meeting deadlines of periodic tasks**
**– Solution 1: Modification of Periods (2)**

The agent proceeds as a second solution to change the periods of tasks of $\xi\_new$ and $\xi\_old$ . To obtain a feasible system, the following formula should be satisfied:

$$\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}+\sum_{i=n_1-j+1}^{n_1+n2}\frac{Ci}{\min(T_i,D_i)+\theta i}=1, \text{ where j } \in [0;\, n_1].$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2}\frac{Ci}{\min(T_i,D_i)+\theta i}=1-\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}\text{ , Let } \beta_j \text{ be } (\min(T_i,D_i)+\theta i)$$

$$\rightarrow \frac{1}{\beta_j}\sum_{i=n_1-j+1}^{n_1+n2}Ci=1-\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}$$

$$\rightarrow \beta_j=\left\lceil\frac{\sum_{i=n_1-j+1}^{n_1+n2}Ci}{1-\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}}\right\rceil = \text{constant, the new period of } \Gamma \text{ tasks is therefore deduced from } \beta_j .$$

***Running example:***

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our *Volvo* system, we present in Figure 9 the results described by the developed tool RT - Configuration. These results are the new temporal parameters of the considered tasks. The user can choose one of these solutions to have a feasible system. We note that: $\xi\_new$ = {C; G; H} and $\xi\_old$ = {A; B; D; E; F}.

The agent computes the constant values $\beta_j$ (j $\in$ [0; 5]) corresponding respectively as follows:

$\beta_0$ = 43, $\beta_1$ = 77, until $\beta_5$ = 42 time units where $\xi\_old$ Ø and $\xi\_new$ = {A; B; D; E; F; C; G; H}

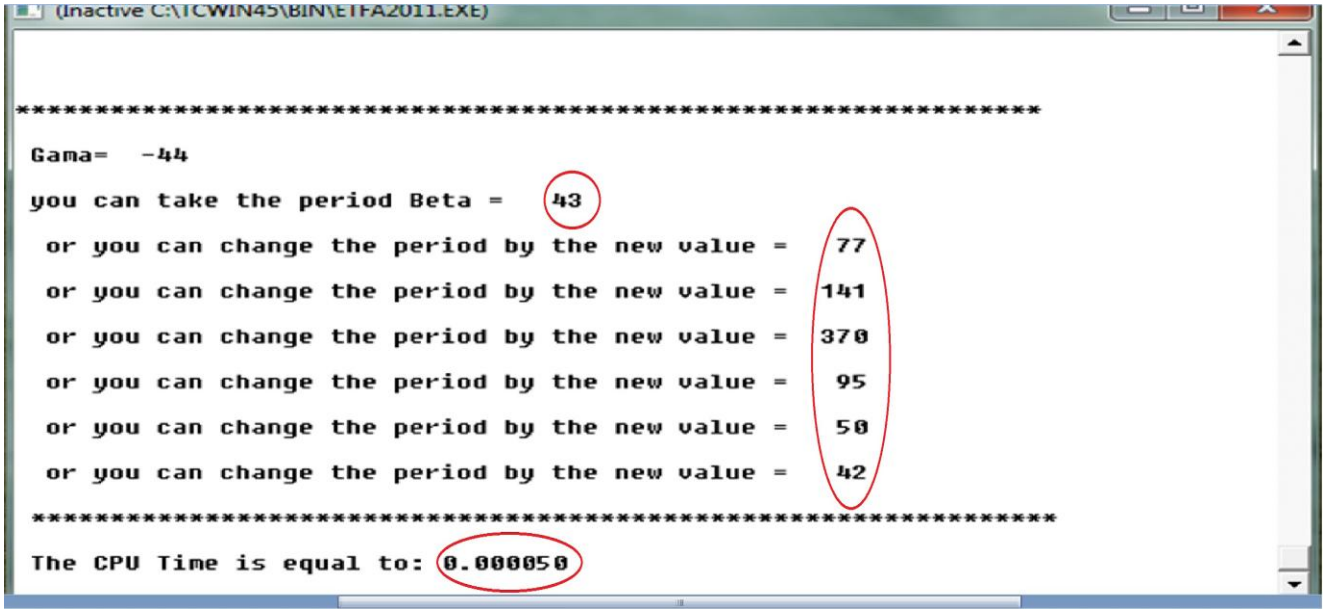


***Figure 9: The Volvo case study simulation***

By applying the well known scheduling real-time simulator Cheddar [8], the EDF scheduling result is shown in figure 10.

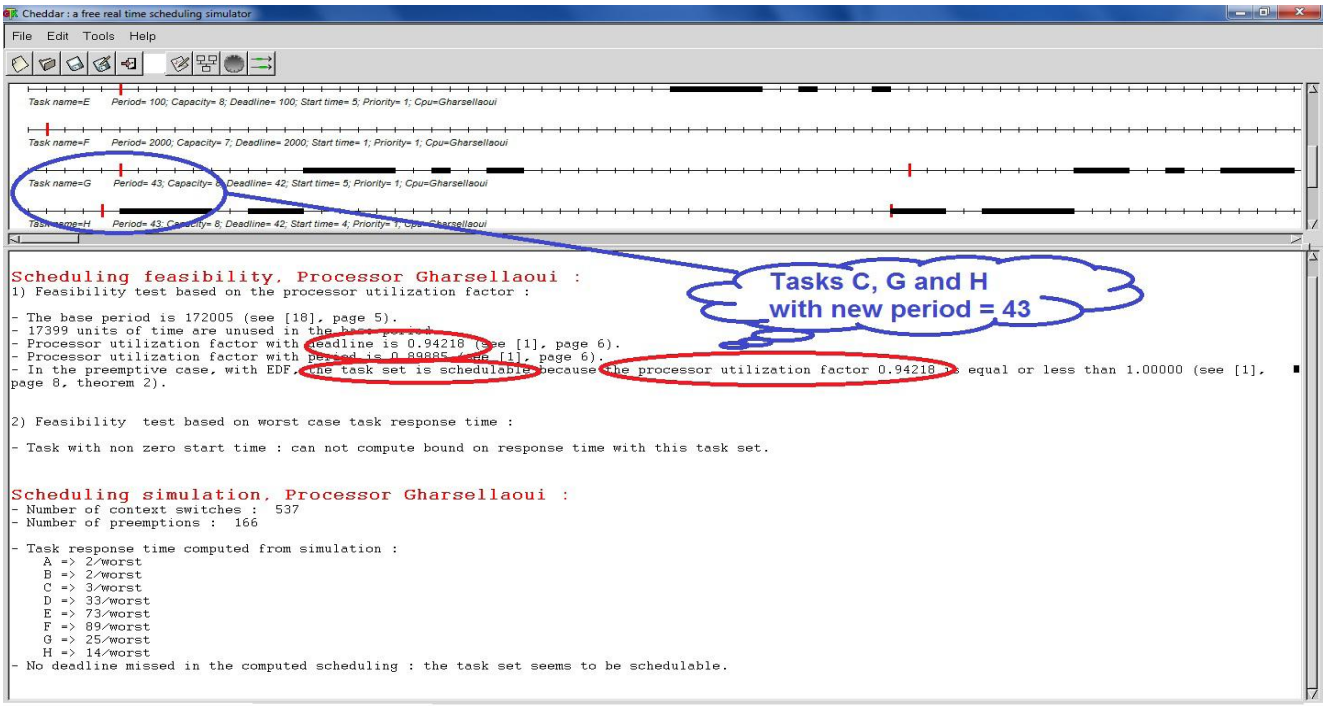


***Figure 10: the Volvo case study simulation with Cheddar***

The processor utilization factor (U) becomes equal to 0.942 after updating the tasks C, G and H by the new value of period equal to 43 and the task set becomes schedulable (feasible).

**Solution 2: Modification of Worst Case Execution Times (3)**

The agent proceeds now as a third solution to modify the Worst case Execution Times (WCET) of tasks of $\xi\_new$ and $\xi\_old$ . To obtain a feasible system, the following formula should be satisfied:

$$\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}+\sum_{i=n_1-j+1}^{n_1+n2}\frac{Ci+\alpha i}{\min(T_i,D_i)}=1$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2}\frac{Ci+\alpha i}{\min(T_i,D_i)}=1-\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2}\frac{\alpha i}{\min(T_i,D_i)}=1-\sum_{i=1}^{n_1-j}\frac{Ci}{\min(T_i,D_i)}-\sum_{i=n_1-j+1}^{n_1+n2}\frac{Ci}{\min(T_i,D_i)}$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2}\frac{\alpha i}{\min(T_i,D_i)}=1-\sum_{i=1}^{n_1+n2}\frac{Ci}{\min(T_i,D_i)}$$

Let $\gamma_j$ be the following constant: $\gamma_j = \alpha i$ = Constant,

$$\rightarrow \gamma_j = \left\lceil \frac{1 - \sum_{i=1}^{n_1+n2} \frac{Ci}{\min(T_i, D_i)}}{\sum_{i=n_1-j+1}^{n_1+n2} \frac{1}{\min(T_i, D_i)}} \right\rceil$$

The new WCET of Γ tasks is therefore deduced from $\gamma_j$ .

***Running example:***
The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our *Volvo* system, we present in Figure 11 the results described by the developed tool RT-reconfiguration. These results are the new temporal parameters of the considered tasks. The user can choose one of these solutions to have a feasible system. We note that: ξ_new = {C; G; H} and ξ_*old* = {A; B; D; E; F}

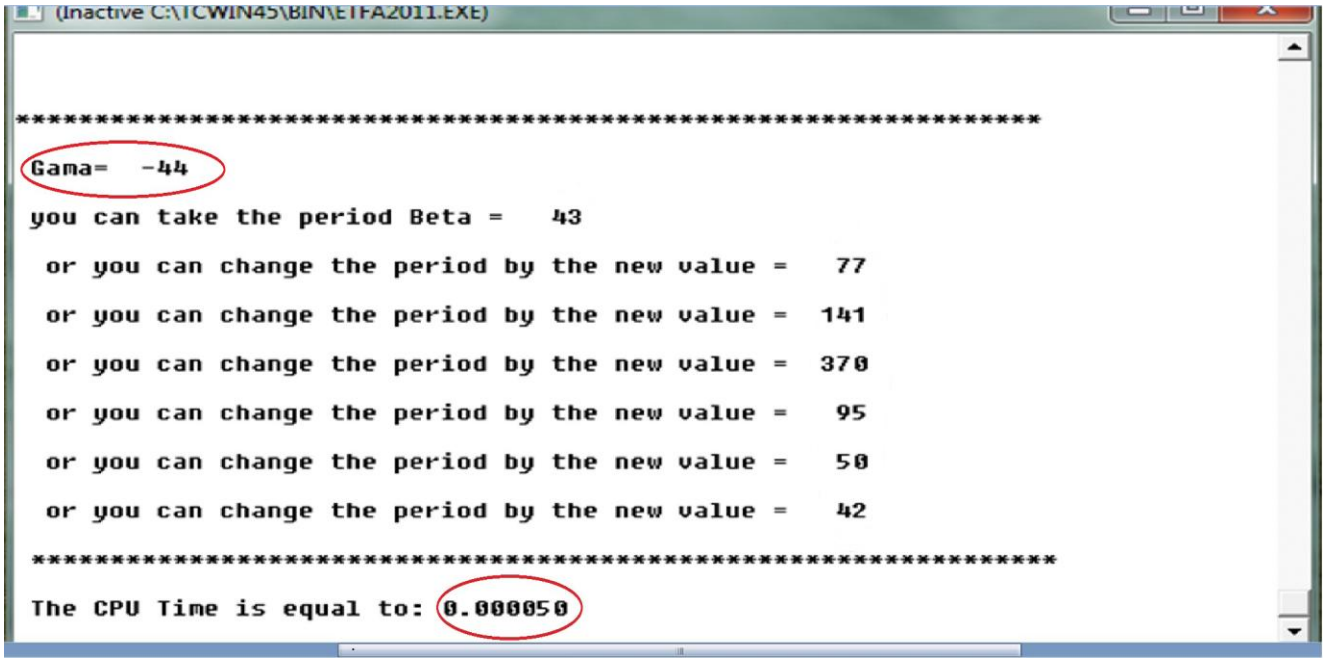


***Figure 11: The Volvo case study simulation***

The agent computes the constant values $\gamma_j$ , (j $\in$ [0; 5]) corresponding respectively to the new values of the Worst Case Execution Times (WCET). Here γ = -44, and the minimum value of WCET in the *Volvo* system is equal to 1, so γ = -44 + (Minimum WCET = 1) = -43 ≤ 0. Therefore, the agent deduces that modifications of Worst Case Execution Times (WCET) cannot solve the problem.

**CONTRIBUTION 3: OPTIMIZATION OF RESPONSE TIME**
This section aims to present the principle of response time minimization. Indeed, in this paper, we are interested in an automatic reconfiguration of operating system's (OS) functional tasks. All the tasks are supposed to be independent, periodic and synchronous/asynchronous. We assume also that the whole system is scheduled by the earliest deadline first (EDF) scheduling policy. So, we shall deduct from this schedulability certain basic properties of the system, and then we shall become attached to finer characterizations with in particular the

determination of borders on response time which is a central problem in the conception of the real-time systems. For this reason, we present the function of major job arrival with deadline in the following paragraph.

**New function of major job arrival with deadline:**

In the Background, we defined the function of job arrival with deadline. Now and in order to analyze the feasibility, we shall have to quantify, the maximal amount of job of term less than or equal to one certain date was engendered on an interval of time, it is the function of major job arrival with deadline. This function applied to the task $\tau_k$, noted $\hat{S}_k(.)$, limits the function of major job arrival with deadline of the task $\tau_k$, on everything interval of time of duration $\Delta t$: $S_k(A_{k,j}, A_{k,j} + \Delta t, A_{k,j} + d) \leq \hat{S}_k(\Delta t, d)$, $\forall A_{k,j}$ (the beginning of the interval in which the function is estimated) $\geq 0$, $\forall \Delta t \geq 0$, $\forall d \geq 0$.

We assume now the case where $D_{k,n} = A_{k,n} + \bar{D}_k$, $\forall$ k,n. We consider an interval of time $[A_{k,j}, A_{k,j} + \Delta t[$, Which is the maximum amount of job which can be engendered for one periodic tasks with deadline on this interval? We know that most high time of execution of an instance is $C_k$. Let us determine the maximum number of instances in an interval of time of the type $[A_{k,j}, A_{k,j} + \Delta t[$. We note $A_{k,n0}$ the first instance of $\tau_k$, after $A_{k,j}$ and $A_{k,n1}$ the last one before $A_{k,j} + \Delta t$ with $n = n_1 - n_0 + 1$ the number of instances in this interval $[A_{k,j}, A_{k,j} + \Delta t[$.

There are two conditions so that the job of an instance $\tau_{k,i}$ is counted, it is necessary that:

1. $A_{k,i} < A_{k,j} + \Delta t$: the maximum number of instances is most big n which verifies:
$A_{k,n0} + (n-1)*T_k < A_{k,j} + \Delta t$. Where $A_{k,n0} \in [A_{k,j}, A_{k,j} + T_k[$. If $A_{k,n0} = A_{k,j}$, we have n maximum and we obtain the following expression: $n < \Delta t / T_k + 1$. **(b)**

The biggest integer n which satisfies (b) is $n = \left\lceil \frac{\Delta t}{T_k} \right\rceil$

2. $D_{k,i} < A_{k,j} + d$ : the respect for this condition involves that the deadline of $\tau_{k,n1}$ will have to verify:
$D_{k,n1} = A_{k,n1} + \bar{D}_k \leq A_{k,j} + d$.

As $A_{k,n1} \geq A_{k,n0} + (n-1)*T_k$, we have $A_{k,n0} + (n-1)*T_k + \bar{D}_k \leq A_{k,j} + d$. Where $A_{k,n0} \in [A_{k,j}, A_{k,j} + T_k[$. If $A_{k,n0} = A_{k,j}$, we have n maximum and we obtain the following expression: $n \leq (d - \bar{D}_k)/T_k + 1$. **(c)**

The biggest integer n which satisfies (c) is $n = \left\lfloor \frac{d - \bar{D}_k}{T_k} \right\rfloor + 1$ **(d).**

An implicit condition is that $n \geq 0$, notice in (d) that as $\bar{D}_k$ can be arbitrarily big, n can be negative. The biggest n which verifies three conditions (b, c and d) is finally: $n = \min\left(\left\lceil \frac{\Delta t}{T_k} \right\rceil, \left\lfloor \frac{d - \bar{D}_k}{T_k} \right\rfloor + 1\right)^+$. Where $(a)^+ =$ max (0, a). We obtain finally the function of major job arrival with following deadline for $\tau_k$:

$$\hat{S}_k(\Delta t, d) = \min\left(\left\lceil \frac{\Delta t}{T_k} \right\rceil, \left\lfloor \frac{d - \bar{D}_k}{T_k} \right\rfloor + 1\right)^+ * C_k \qquad \textbf{(e).}$$

**Interference period with deadline d under EDF**

The end of execution of an instance $\tau_{k,n1}$, with a deadline $D_{k,n}$, is the end of one period of activity of the processor in which all the executed instances have a deadline less or equal to $D_{k,n}$. So, we define the period of interference with a deadline d as:

**Definition:** A period of interference with a deadline d is an interval of time $[B_j, Ej[$ such as: $W_{1..m}(B_j,d) = W_{1..m}(E_j,d) = 0$ and $B_j < t < E_j \rightarrow W_{1..m}(t,d) > 0$. On the same trajectory, we can have several periods of interferences there with deadlines d and we shall note $B_j(d)$ and $E_j(d)$ the beginning and the end of the $i^{th}$ period of interference. So, the end of the period of interference with deadline d satisfies:

$E_j(d) = \min\{t > B_j(d) / W_{1..m}(t,d) = 0\}$. **(f)**

Inside the period of interference, the workload with deadline less than or equal to d evolves in time according to the equation **(a)**

$$W_{1..m}(t,d) = S_{1..m}(B_j(d),t,d) - \int_{B_j(d)}^{t} \prod_{1..m}(u,d)du$$

As the function $\prod_{1..m}(t,d)$ is always equal to 1 inside the period of interference, we have:

$W_{1..m}(t,d) = S_{1..m}(B_j(d),t,d) - (t - B_j(d))$. By injecting this relation in the equation **(f)** and after a change of variable $(t = B_j(d) + \Delta t)$, we obtain this characterization of $E_j(d)$ that we shall use to determine the response time borders:

$E_j(d) = \min \{\Delta t > 0 / S_{1..m}(B_j(d), B_j(d) + \Delta t, d)) = \Delta t \}$

**Significant activation dates under EDF:**

We note $E_0(d)$ the end of the first period of interference with deadline d which follows the simultaneous activation of all the tasks except of the task $\tau_k$ under study. An instance $\tau_{k,n}$ ends its execution (in later) in $E_0(d)$ if d is the deadline of $\tau_{k,n}$ ($d = D_{k,n}$), that is for $A_{k,n} = d - \bar{D}_k$. If we calculate the response time of $\tau_{k,n}$ for all the possible values of d for the first period of interference of the major activation scenario, we shall find inevitably one border at the response time of any instances of $\tau_k$ on any trajectory:

$\hat{R}_k = \mathrm{MAX}_{d \in R}(E_0(d) - (d - \bar{D}_k))$.

The biggest size of this first period of interference is noted L. So that an instance $\tau_{k,n}$ belongs to this first period of interference, it is necessary that $A_{k,n} + C_k \leq L$, where from $d \leq L + \bar{D}_k - C_k$. Furthermore, the first possible deadline date is $d = \bar{D}_k$ who corresponds to $A_{k,n} = 0$.

We have then, $\hat{R}_k = \mathrm{MAX}_{d \in R, d \geq \bar{D}_k, d \leq L + \bar{D}_k - Ck}(E_0(d) - (d - \bar{D}_k))$. As $d \in \mathbb{R}$, it is not possible to make the calculation for all possible values but we can limit ourselves to significant values of d, that is values of d which correspond to $A_{i,1}$ ($i \in [1..m]$). Let us build orderly sequence of the deadlines of all the instances of all tasks. The only values of d for which it is necessary to calculate $E_0(d)$ are values such as d is a deadline date included between $\bar{D}_k$ and $L + \bar{D}_k - C_k$. The set $D_k$ of all the values of d to examine is thus:

$D_k = \{d = n*T_i + \bar{D}_i / d \geq \bar{D}_k, d \leq L + \bar{D}_k - C_k, n \in \mathbb{N}, i \in [1..m]\}$. **(g)**

**Calculation of response time borders under EDF**

The value of the biggest possible period of interference of the system noted L is common to all the tasks. This maximal period occurs after the simultaneous provision of an instance of all the tasks:

$L = \min \{\Delta t > 0 / \hat{S}_{1..m}(0, \Delta t) = \Delta t\}$ **(h)**

With $\hat{S}_{1..m}(0,t) = \hat{S}_{1..m}(0,t,+\infty)$ is the function of major job arrival who adds the job of all the instances whatever are their deadlines. In the case of periodic tasks, as it was studied before, we have:

$$\hat{S}_{1..m}(\Delta t) = \sum_{i=1}^{m_1} \left\lceil \frac{\Delta t}{T_i} \right\rceil * C_i$$

Now, according to the previous three solutions calculated by the Intelligent Agent (Solution 1, Solution 2, and Solution 3), we define:

• $\mathbf{L_1}$ according to Solution 1, by the following expression:

$\mathbf{L_1} = \min \{\Delta t > 0 / \hat{S}_{1..m_1}(0, \Delta t) = \Delta t\}$, where $\hat{S}_{1..m_1}(0, \Delta t) = \sum_{i=1}^{m_1} \left\lceil \frac{\Delta t}{T_i} \right\rceil * C_i$

and m1 ≤ m resulting from the removal tasks generated by the first solution (Solution 1).

• $\mathbf{L_2}$ according to Solution 2, by the following expression:

$\mathbf{L_2} = \min \{\Delta t > 0 / \hat{S}_{1..m}(0, \Delta t) = \Delta t\}$, where $\hat{S}_{1..m}(\Delta t) = \sum_{i=1}^{m} \left\lceil \frac{\Delta t}{\beta_i} \right\rceil * C_i$ and $\beta_i$

resulting from the new periods generated by the second solution (Solution 2).

• $\mathbf{L_3}$ according to Solution 3, by the following expression:

$\mathbf{L_3} = \min \{\Delta t > 0 / \hat{S}_{1..m}(0, \Delta t) = \Delta t\}$, where $\hat{S}_{1..m}(\Delta t) = \sum_{i=1}^{m} \left\lceil \frac{\Delta t}{T_i} \right\rceil * \gamma_i$ and $\gamma_i$

resulting from the new worst case execution times generated by the third solution (Solution 3).

$\mathbf{L_1}$ is thus (respectively $\mathbf{L_2}$ and $\mathbf{L_3}$), the limit when n aims towards the infinity, of the suite:

$L_1^0 = \sum_{i=1}^{m_1} C_i$ , $L_1^n = \sum_{i=1}^{m_1} \left\lceil \frac{L_1^{n-1}}{T_i} \right\rceil * C_i$ (respectively

$L_2^0 = \sum_{i=1}^{m} C_i$ , $L_2^n = \sum_{i=1}^{m} \left\lceil \frac{L_2^{n-1}}{\beta_i} \right\rceil * C_i$ and

$L_3^0 = \sum_{i=1}^{m} \gamma_i$ , $L_3^n = \sum_{i=1}^{m} \left\lceil \frac{L_3^{n-1}}{T_i} \right\rceil * \gamma_i$ ) **(i)**

The obtaining of $\mathbf{L_1}$ (respectively $\mathbf{L_2}$ and $\mathbf{L_3}$), allows us to build the set $D_k^1$ (respectively $D_k^2$ and $D_k^3$) defined by **(g).** For every value of d $\in D_k^1$ (respectively $D_k^2$ and $D_k^3$), it is now necessary to calculate the end of the corresponding period of interference $E_{0,1}(d)$ (respectively $E_{0,2}(d)$ and $E_{0,3}(d)$).
According to **(h)** and **(e)**: $E_{0,1}(d)$ is the limit when n aims towards the infinity of the suite:

$$E_{0,1}^0(d) = \varepsilon \ , \ E_{0,1}^n(d) = \sum_{i=1}^{m_1} (\min ( \left\lceil \frac{E_{0,1}^{n-1}}{T_i} \right\rceil , \left\lfloor \frac{d - \bar{D}_i}{T_i} \right\rfloor + 1)) \ ^+ * C_i \ ,$$

$E_{0,2}(d)$ is the limit when n aims towards the infinity of the suite:

$$E_{0,2}^0(d) = \varepsilon \ , \ E_{0,2}^n(d) = \sum_{i=1}^{m} (\min ( \left\lceil \frac{E_{0,2}^{n-1}}{\beta_i} \right\rceil , \left\lfloor \frac{d - \bar{D}_i}{\beta_i} \right\rfloor + 1)) \ ^+ * C_i \ ,$$

and $E_{0,3}(d)$ is the limit when n aims towards the infinity of the suite:

$$E_{0,3}^0(d) = \varepsilon \ , \ E_{0,3}^n(d) = \sum_{i=1}^{m} (\min ( \left\lceil \frac{E_{0,3}^{n-1}}{T_i} \right\rceil , \left\lfloor \frac{d - \bar{D}_i}{T_i} \right\rfloor + 1)) \ ^+ * \gamma_i \ .$$

Where $\varepsilon$ is a positive and unimportant but necessary real value to affect the convergence. For every value of d $\in D_k^1$ (respectively $D_k^2$ and $D_k^3$), the corresponding response time is:

$R_{k,1} = (E_{0,1}(d) - (d - \bar{D}_k))$, The biggest value is the border of the response time ($R_{\{k,1\}}$max).

$R_{k,2} = (E_{0,2}(d) - (d - \bar{D}_k))$, The biggest value is the border of the response time ($R_{\{k,2\}}$max).

$R_{k,3} = (E_{0,3}(d) - (d - \bar{D}_k))$, The biggest value is the border of the response time ($R_{\{k,3\}}$max).

We define now, $R_k$ optimal noted $R_k^{opt}$ according to the previous three solutions calculated by the intelligent Agent (Solution 1, Solution 2, and Solution 3) by the following expression:

$R_k^{opt}$ **= min ($R_{k,1}$, $R_{k,2}$, $R_{k,3}$)** (the minimum of the three values) **(j).**

So, the calculation of $R_k^{opt}$ allows us to obtain and to calculate the minimizations of response times values and to get the optimum of these values.

***Final conclusion***

This research work dealing with multi-parameters reconfiguration is that we propose in the current paper in which, we give solutions to the user for all these problems presented by the tool RT-Reconfiguration. This work also, concentrates on the context of systems containing a set of tasks which is not feasible. The reconfiguration was applied in order not only to obtain the systems feasibility, but also to get the performance of the system by reducing the response time of the processes to be tolerated in interactive environment in order to obtain the optimization of the response time of the studied reconfigurable system.

***Algorithm***

Given a set of periodic, independent tasks to be scheduled by EDF on a single processor, Spuri [30] proposed an algorithm for computing an upper bound on the worst-case response time of a task. His algorithm, however, does not consider task offsets. This means that the analysis proposed by Spuri is still valid even in the case of tasks with offsets (asynchronous case), but the results may be pessimistic. A first approach to the problem of computation of worst-case relative response times would be to apply Spuri's method, considering each task to be independent from other tasks of the same set of tasks. However, this approach is extremely pessimistic. Palencia and Gonzalez [31] introduced a new method that is much less pessimistic than Spuri's one by taking into consideration the offsets among tasks of the same set of tasks.

The following algorithm is our original contribution to the problem, which is able to provide both a response time minimization and a feasibility of the studied system.

We now introduce this algorithm, our original contribution to the problem, by these different codes to be supported by the agent for a feasible reconfiguration of an embedded system.

**Begin Algorithm**

**Code1** Removal-Tasks () $\mathbf{U \leftarrow 0;}$

– **For each partition** $\beta \subseteq \tau_{\text{new}} \cup \tau_{old}$

– i = 1;

– $U+ = \frac{C_i}{\min(T_i, Di)};$

– **If** $\mathbf{U \leq 1}$

– Then display ( $\beta$ );

Save ($m_1$);

Else display i+1;

**Code2** Modify Periods_Deadlines_WCET()

– Compute ( $\beta_i$ );

– Compute ( $\gamma_i$ );

– For $\min(T_i, Di) \in \xi\_\text{new} \cup \xi\_old$

– Display parameters ();

**Code3** Generate_parameters ($m_1, \beta_i, \gamma_i$);

– Compute ($R_{k,1}$);

– Compute ($R_{k,2}$);

– Compute ($R_{k,3}$);

– Generate ( $R_k^{opt}$ );

**End Algorithm**

Intuitively, we expect that our algorithm performs better than the Spuri's, the Palencia and Gonzlez ones. We show the results of our proposed algorithm by means of experimental result's evaluation.

***Complexity***

The EDF-schedulability in the case of periodic synchronous tasks (with deadline equal to period) is decidable in polynomial time. In the case of asynchronous tasks, i.e. each task has an offset $S_i$, such that jobs are released at $k*T_i + S_i$ ($k \in \mathbb{N}$), then testing the feasibility is strongly coNP-hard [11]. This complexity was decreased in our approach to O(nlog(n)) because the proposed algorithm is recursive, and the Earliest Deadline First algorithm also, would be maintaining all tasks that are ready for execution in a queue. Any freshly arriving task would be inserted at the end of queue. Each task insertion will be achieved in O(1) or constant time, but task selection (to run next) and its deletion would require O (n) time, where n is the number of tasks in the queue. EDF simply maintaining all ready tasks in a sorted priority queue that will be used a heap data structure. When a task arrives, a record for it can be inserted into the heap in O(log (n)) time where n is the total number of tasks in the priority queue. Therefore, the time complexity of Earliest Deadline First is equal to that of a typical sorting algorithm which is O (n log (n)). So O(nlog (n)) time is required. In other hand, the busy period, which is computed for every analyzed task set and has a pseudo-polynomial complexity for $U \leq 1$ [12], is decreased also by the optimization of the response time. The most important results are presented in our work. So, we can deduce that using our proposed approach under such conditions may be advantageous.

***Theorem***

We assume a preemptive, asynchronous and periodic task system Γ to be composed of n periodic reconfigurable tasks, where each task is described by a period $T_i$, an arbitrary relative deadline $D_i$, a Worst Case Execution time (WCET) $C_i$ and a release offset $S_i$.

If Γ is unfeasible and we apply a reconfiguration scenario based on EDF algorithm using the three previous solutions described in (1), (2) and (3) then, these tasks are scheduled with minimum response time. The system Γ is feasible and more over, we obtain an optimal response time for this system in the hyper-period $hp = [0, 2*LCM+\max_k(A_{k,1})]$. **(4)**

***Proof:***

We prove the above theorem by proving the contrapositive, i.e., by showing that if Γ is not schedulable by EDF, then **(4)** is false.

Let $t_b$ be the first instant at which a job of some task $\tau_i$ misses its deadline under EDF. Since $\tau_i$ misses its deadline at $t_b$, then all the system will be unfeasible, then the hyper-period $hp = [0, 2*LCM + \max_k(A_{k,1})]$ is not bounded and it diverge.

Or, initially we supposed that the hyper-period $hp = [0, 2*LCM + \max_k(A_{k,1})]$ is bounded and converge. Thus, **(4)** is false as claimed.

We now want to prove the property of optimality addressed above, in the previous proposed theorem. That is the response time of the asynchronous periodic requests under the EDF algorithm are the best achievable. This is exactly what is stated by the following lemma.

***Lemma*** Let A be any on-line preemptive algorithm, Γ a periodic task set, and τ an asynchronous periodic task. If $R^{A}_{\Gamma\cup\tau}(\tau)$ is the response time of τ when $\Gamma \cup \tau$ is scheduled by A, then $R^{EDF}_{\Gamma\cup\tau}(\tau) \leq R^{A}_{\Gamma\cup\tau}(\tau)$

***Proof:***

According to the equation **(a),** we can notice that for $d < D_{k,n}$, we have $W_{k,n}(t,d) = 0$ for any value of t. Where $W_k(t,d)$ is the amount of job with lower deadline to d brought by all the n instances of $\tau_k$. $D_{k,n}$ is the relative deadline of the $n_{th}$ instance of the task τ . So, for each instant t' that, $t' \geq t$ and $t' = d$, we have:

$W^{EDF}_{k,n}(t,t') = 0.$ It follows that: $W^{EDF}_{\text{Current}_\Gamma}(t,t') = 0$

and $W^{EDF}_{\text{Current}_\Gamma}(t,t') \leq W^{A}_{\text{Current}_\Gamma}(t,t')$ , it follows that: $R^{EDF}_{\Gamma\cup\tau}(\tau) \leq R^{A}_{\Gamma\cup\tau}(\tau)$

That is, under the EDF algorithm, τ is never completed later than under A. By end, the optimality property was proved.

## EXPERIMENTAL ANALYSIS AND DISCUSSION

In this section, in order to check the suggested configurations of tasks allowing the system's feasibility and the response time minimization, we simulate the agent's behavior on a Blackberry Bold 9700 presented by [32] and on a *Volvo* system presented by [16]. This simulation presents some results by virtually applying real-time reconfigurations in the operating system of Blackberry Bold 9700 and in the operating system of a Volvo system. The Blackberry Bold 9700 is assumed to be initially composed of 50 tasks and dynamically reconfigured at run-time to add 30 new ones in which a task can be a missed call, a received message, or a Skype call. According to [32], the implemented Blackberry Bold 9700 is characterized as follows:

- **Mobile type:** 3G (WCDMA), GSM, and WCDMA,
- **Support band:** GSM 850/900/1800/1900,
- **Data transmission:** GPRS, EDGE, and HSDPA,
- **Producer:** RIM, Canada,
- **Commerce available:** November, 2009,
- **OS:** Blackberry OS 5.0,
- **Battery:** Lithium battery, 1500 mAh,
- **CPU:** Marvell PXA930, 624MHz,
- **Storage:** Micro-SD 32GB, and
- **Memory:** 256MB ROM and 256MB RAM.

The *Volvo* system as shown in table 1 is assumed to be initially composed of 5 tasks and dynamically reconfigured at run-time to add 3 new ones. In this paper, any real-time reconfiguration and response time minimization is based on the real-time embedded control system reconfiguration. Moreover, in order to meet all real-time constraints, both initial WCETs $C_i$, the relative deadline $D_i$ and also periods $T_i$ of each task are reconfigured by the intelligent agent RT-Reconfiguration.

In this case, we are interested in the reconfiguration of these task's parameters, however we just present $S_i$, $C_i$ and $T_i$ of each task, and we assume that periods are equal to deadlines. The goal is to minimize the response time of the whole system and to meet their relative deadlines.

By applying the well known scheduling real-time simulator Cheddar [8], the EDF scheduling result is shown in figure 12.

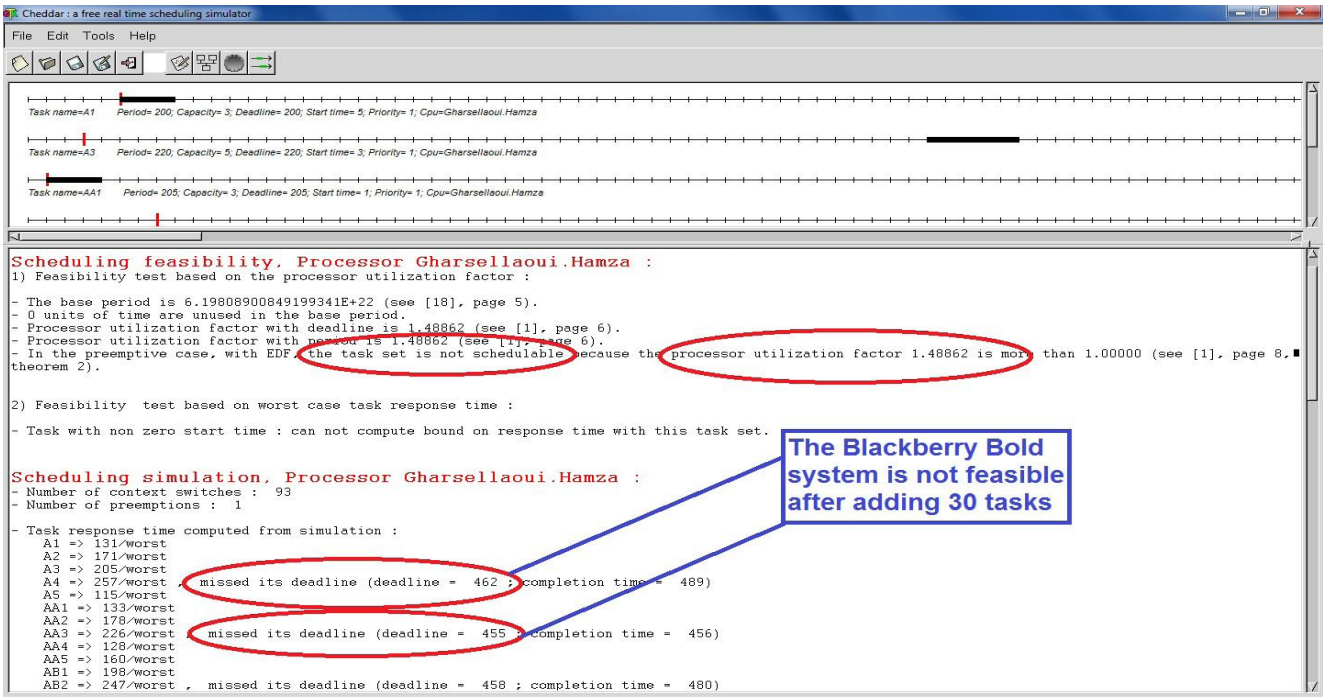


***Figure 12: the Blackberry Bold 9700 simulation with Cheddar***

The developed intelligent agent RT-Reconfiguration can configure all the parameters and evaluate the processor utilization factor $U = \sum_{i=1}^{n} \frac{Ci}{\min(T_i, D_i)}$.

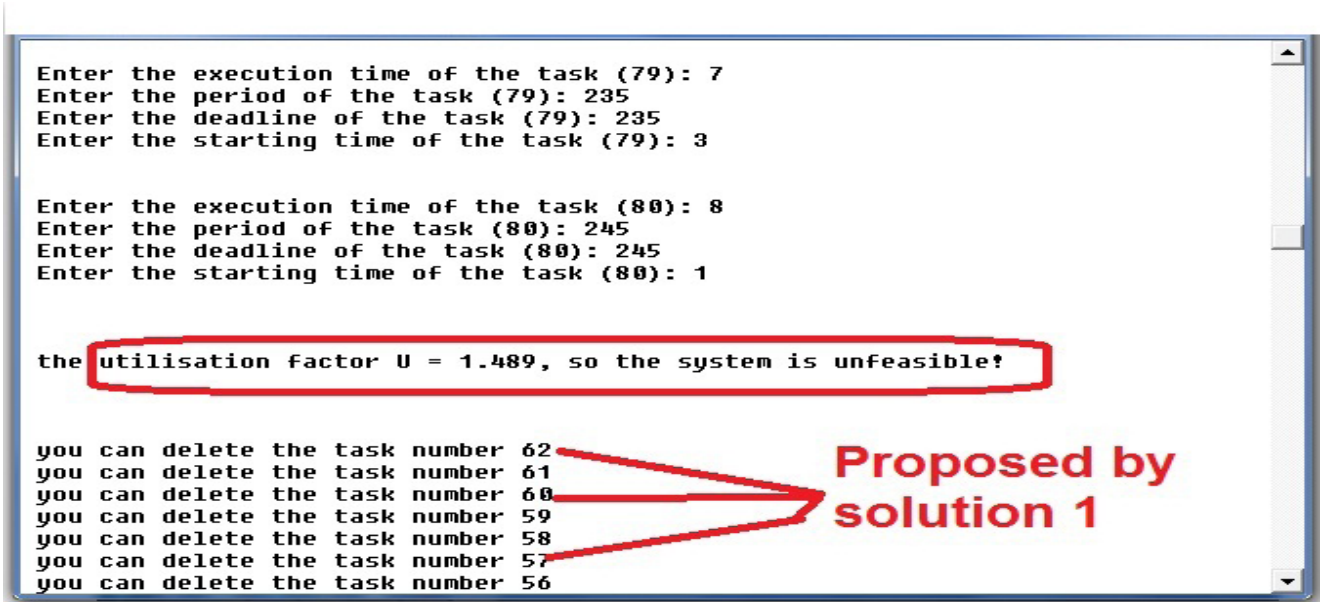


***Figure 13: The Blackberry Bold 9700 simulation with RT-Reconfiguration***

The previous tests were performed on an Intel(R) Core(TM) 2 Duo CPU (2.00 GHz, 3GHz RAM) on MS Windows 7. By considering asynchronous real-time tasks, the schedulability analysis should be done in the Hyper-Period HP = [0, 2*LCM + $\max_k(A_{k,1})$] ,where LCM is the well-known Least Common Multiple. Let n be the number of tasks in Current $_\Gamma$ (t). The reconfiguration of the system Γ means the modification of its implementation that will be as follows at t time units:

$\text{Current}_\Gamma(t) = \xi\_new \cup \xi\_old$ , Where $\xi\_old$ is a subset of old tasks which are not affected by the reconfiguration scenario (e.g. they implement the system before the time t), and $\xi\_new$ a subset of new tasks in the system. We assume that an updated task is considered as a new one at t time units. By considering a feasible System Γ before the application of the reconfiguration scenario, each one of the tasks of $\xi\_old$ is feasible, e.g. the execution of each instance is finished before the corresponding deadline.

***Analysis of Results***

In order to evaluate the proposed approach and to determine their advantages we consider the systems Blackberry Bold 9700 and *Volvo* defined in the running examples.

As shown in figures 14 and 15, the X axis (abscissa axis) represents the number of removal tasks. If the removal rate is equal to 5, implying that we remove 5 tasks at each reconfiguration scenario. Then, more than 50 tasks, we can't remove another ones because the studied system will be disastrous.

As shown in figure 16, the X axis (abscissa axis) represents the number of reconfigured tasks. If the reconfiguration rate is equal to 10, implying that we modify 10 task's parameters (Deadlines/Periods or WCETs) at each reconfiguration scenario. The running time for the Blackberry Bold 9700 system ranges from 0 to 550 microseconds using the first solution. The utilization factor (U) decreases from 1.489 to 0.389 microseconds using the first solution, from 1.489 to 0.709 microseconds using the second solution and it decreases from 1.489 to 0.478 microseconds using the third solution.

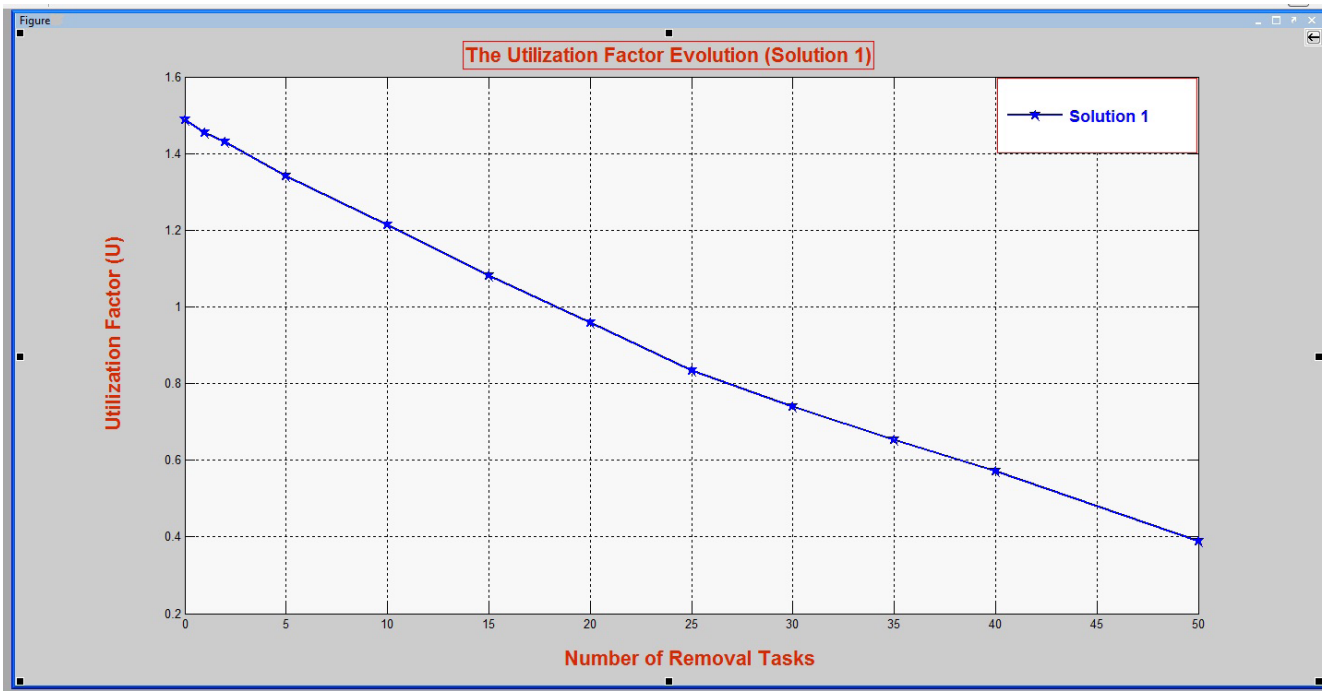


***Figure 14: The Utilization Factor Evolution of the Blackberry Bold System (Solution 1)***

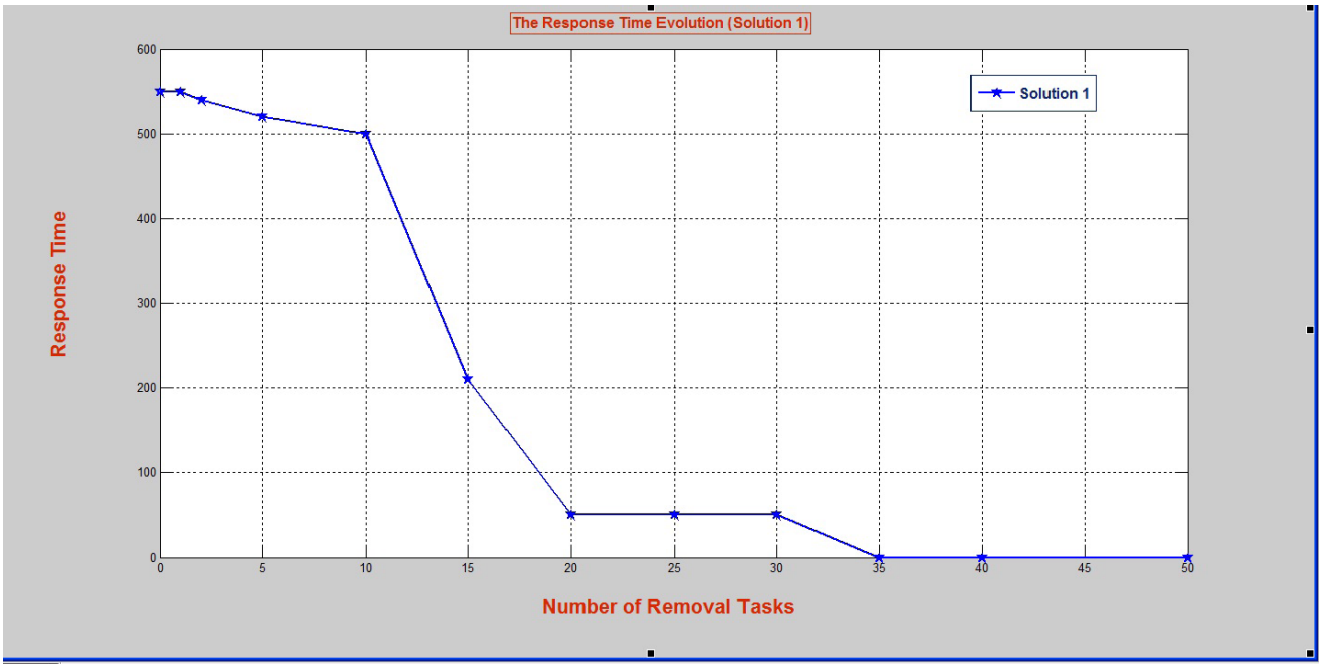


*Figure 15: The Response Time Evolution of the Blackberry Bold System (Solution1)*

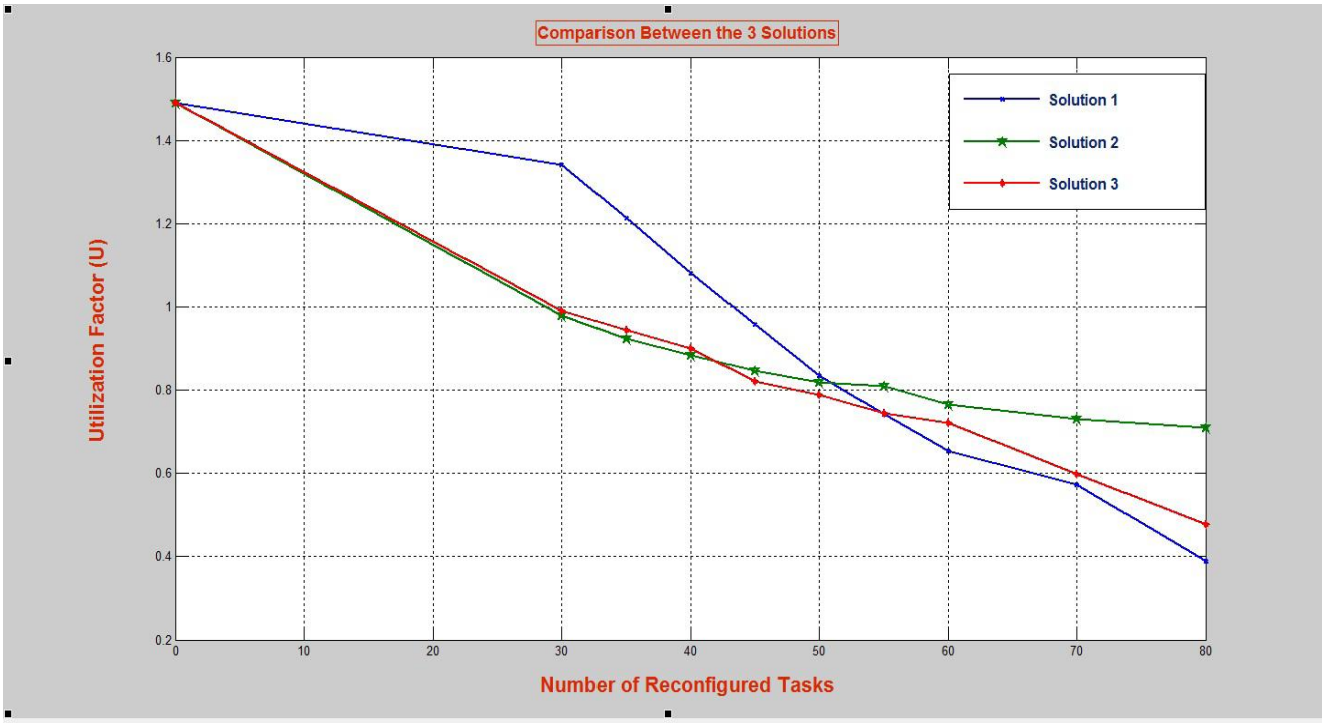


*Figure 16: The Comparison between the 3 Solutions (The Blackberry Bold System)*

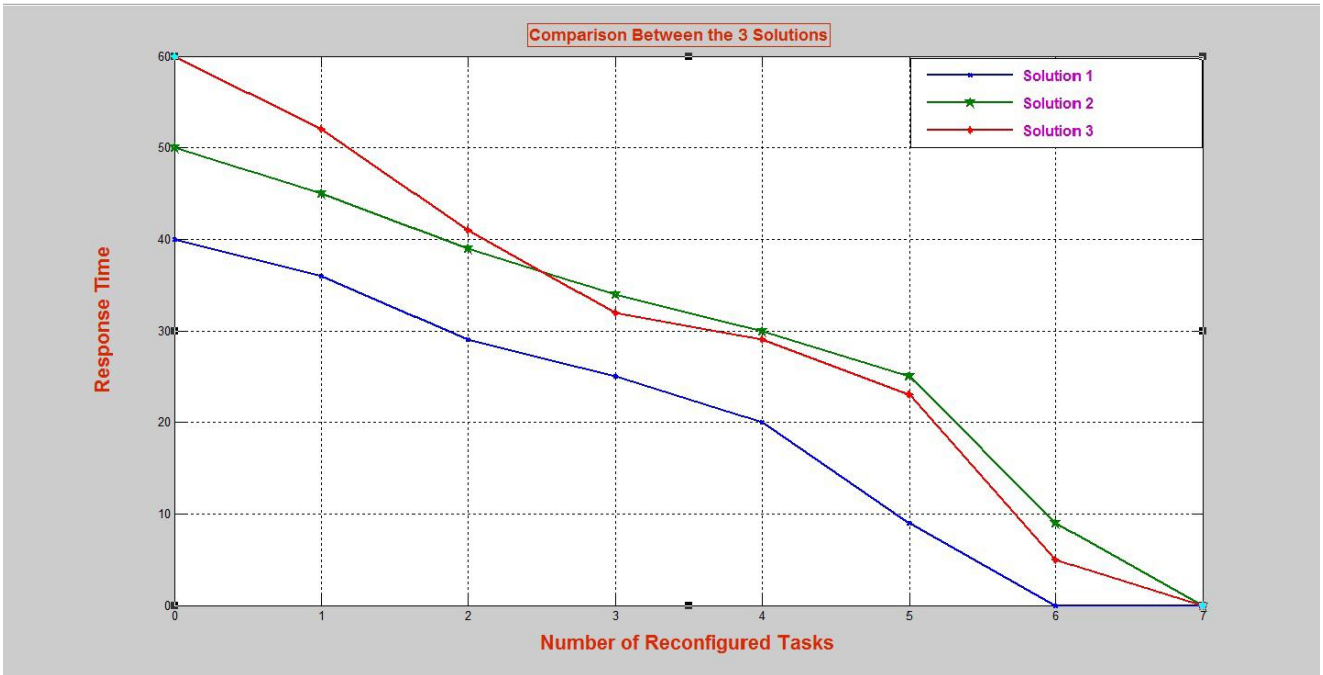


***Figure 17: The Comparison between the 3 Solutions (The Volvo System)***

As the results show (figure 16), we can observe, especially from the second half of the curve, and demonstrate the importance and efficiency of the solution 1 against the solution 2 and the solution 3 and the efficiency of the solution 3 against the second solution in term of utilization factor (U) decreasing.
Based on these observed results, we can compare our work to the corresponding papers in the state of the art which cannot reach these results and why, we can confirm that this method is very advantageous given the fast response time and the performance of the schedulability of such studied system.

As shown in figure 17, the X axis (abscissa axis) represents the number of reconfigured tasks. If the reconfiguration rate is equal to 1, implying that we modify 1 task's parameters (Deadlines/Periods or WCETs) at each reconfiguration scenario. The running time for the *Volvo* system ranges from 0 to 40 microseconds using the first solution, it ranges from 0 to 50 microseconds using the second one and it ranges from 0 to 60 microseconds using the third solution.

As the results show (figure 17), we can observe, especially from the second half of the curve, and demonstrate the importance and efficiency of the solution 1 against the solution 2 and the solution 3, and the efficiency of the solution 3 against the second solution in term of response time speed.
The second important observation was obtained by the comparison of our proposed approach against the others from the literature about the current values. We tested the feasibility of the same task sets Blackberry Bold 9700, and *Volvo* by other algorithms, so that we can compare the results directly. We carried out several test runs and examined them under different aspects. The total utilization of the static schedule is 75%, the classic one is 145.4 % and the other proposed by our method is 63.1%.

**Discussion and Evaluation**

The test greatly reduces the processor utilization factor $U = \sum_{i=1}^{n} \frac{Ci}{\min(T_i, D_i)}$ in comparison to the original processor utilization factor, so the combination of both three solutions in order to obtain the optimization of the response time by calculating $L_{opt}$ leads to an improved algorithm for the analysis of asynchronous systems.
So, we can therefore confirm that this method is nowadays very advantageous given the fast response time and the performance of the RT-Reconfiguration tool.
By applying the three solutions of this tool RT-Reconfiguration, we conclude that our approach can allow more reactive and also more efficient feasible systems. This advantage can be important in many cases where critical control tasks should be intensively executed in small periods of time. This work also, concentrates on the context of systems containing a set of tasks which is not feasible; the reconfiguration was applied in order not only to obtain the system's feasibility but also to get the performance of the system by reducing the response time of the processes to be tolerated in interactive environment and by avoiding unnecessarily frequent context switch leading to more overheads resulting in less throughput. This advantage was increased and proved clearly with the Blackberry Bold 9700 system proposed by [32] and by the *Volvo* case study proposed by [16].
Both, the figures 10, 14, 15 and 16 illustrate this advantage. Moreover, with the revolution of semiconductors technology and the development of efficient reconfiguration tools, the use of our method and the RT-Reconfiguration tool will becomes increasingly important, and very advantageous for rapid and efficient response time of the periodic reconfigurable OS tasks, especially when the user has no other choice than to choose the previous proposed solutions and to decide the proper values of each reconfigured task's parameters in order to obtain the system's feasibility and to minimize the response time of the studied systems.

## FUTURE RESEARCH DIRECTIONS

We plan in future works to resolve several problems for the Reconfigurable real-time embedded systems. Another problem that has to be resolved in the future deals with the study of each reconfiguration scenario of sporadic and aperiodic tasks to be released in different times and the minimization of their response time. We plan also to study the reconfigurations of dependent and distributed real-time tasks. Finally, our important future work is the generalization of our contributions for the Reconfigurable real-time embedded systems.

## CONCLUSION

The book chapter deals with reconfigurable systems to be implemented by different tasks that should meet real time constraints. In this paper, we propose a new theory for the minimization of the response time of periodic asynchronous constrained deadline real-time tasks with EDF algorithm that can be applied to uniprocessor systems and proved it correct. We showed that this theory was capable to reconfigure the whole system by calculating worst case response times for a simple example using EDF scheduler. Previous work in this area has been described, and several different solution techniques have been suggested. These solutions techniques are primarily intended to reduce the processor demand and the response time by adapting the scheduling parameters (WCET, Period or Deadline) in a uniprocessor system by removing some tasks, changing the periods/deadlines or by reducing the worst case execution time of each task set independent of the number of tasks.
A tool is developed and tested to support all these services. This approach is applied to a Blackberry Bold 9700 and to a *volvo* system. To satisfy user requirements, we apply the Real-Time Simulator, cheddar to check the whole system behavior.

**REFERENCES**

[1] M. L. Dertouzos. Control robotics: The procedural control of physical processes. Information Processing, 1974.

[2] S. Tucker Taft, Robert A. Duff, Randall L.Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.), Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1, LNCS 4348, Springer, 2006.

[3] RTSJ (Real-Time Specification for Java) home page, http://www.rtsj.org/

[4] S.Ha.R.K. (Soft Hard Real-Time Kernel) home page, http://shark.sssup.it/

[5] P. Pedreiras, and L. Almeida, EDF Message Scheduling on Controller Area Network, Computing & Control Engineering Journal, 13(4), pp. 163-170, 2002.

[6] J.-T. Leung and M. Merril. A note on preemptive scheduling of periodic real-time tasks. Information Processing Letters, 3(11), 1980.

[7] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. The Journal of Real-Time Systems, 2, 1990.

[8] L. N. L. M. F. Singhoff, J. Legrand, ”Cheddar: a Flexible Real Time Scheduling Framework”, in ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, ISSN: 1094-
36-41, 2004.

[9] C. Liu and J. Layland, Scheduling algorithms for multi-programming in a hard-real-time environment, in Journal of the ACM, 20(1):46-61, 1973.

[10] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, Institut National de Recherche en Informatique et en Automatique, 1996.

[11] J. Y.-T. Leung and M. L. Merrill, A note on preemptive scheduling of periodic, real-time tasks, Information Processing Letters, 11 (1980), pp: 115-118.

[12] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report RR-2772, 1996.

[13] A. Burns and G. Baxter, Time bands in systems structure, in Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective. Springer London, 2006, pp. 74-88.
[14] J. W. Liu, Real-Time Systems. Prentice Hall, 2000.

[15] R. I. Davis, A. Zabos, and A. Burns, Efficient exact schedulability tests for fixed priority real-time systems, IEEE Trans. Comput., vol. 57, no. 9, pp. 1261-1276, 2008.

[16] K. Hnninen and T. Riutta. Optimal Design. Masters thesis, Mlardalens Hgskola, Dept of Computer Science and Engineering, 2003.

[17] Volvo Construction Equipment. http://www.volvoce.-com.

[18] Arcticus Systems Web-Page. http://www.arcticus.se.

[19] J. Mki-Turja and M. Nolin. Tighter Response-Times for Tasks with Offsets. In Proc.of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA04), August 2004.

[20] STANKOVIC J., SPURI M., RAMAMRITHAM K., BUTTAZZO C., Deadline Scheduling for Real-Time Systems, Kluwer Academic Publishers, Norwell, USA, 1998.

[21] www.loria.fr/ nnavet/cours/DEA2004-2005/slide1.pdf

[22] www.loria.fr/ nnavet/cours/DEA2004-2005/slide1.pdf

[23] Mohamed Khalgui, NCES-based modeling and CTL-based verification of reconfigurable embedded control systems. Computers in Industry journal, vol. 61, no. 3, pp. 198-212, 2010.

[24] Mohamed Khalgui, A deployment methodology of real-time industrial control applications in distributed controllers. Computers in Industry journal, vol. 59, no. 5, pp. 450-462, 2008.

[25] C. Angelov, K. Sierszecki, and N. Marian, Design models for reusable and reconfigurable state machines, in: L.T. Yang et al., Eds., Proc. of Embedded Ubiquitous Comput., pp. 152-163, 2005.

[26] M. N. Rooker, C. Sunder, T. Strasser, A.Zoitl, O. Hummer, and G.Ebenhofer, Zero downtime Reconfiguration of distributed automation systems: The"CEDAC approach, in: Proc. 3rd Int. Conf. Indust. Appl.Holonic Multi-Agent Syst., Regensburg, Sept. 2007, pp. 326-337.

[27] M. Khalgui, O. Mosbahi, Z. W. Li, and H.-M.Hanisch, Reconfigurable multi-agent embedded control systems: From modeling to implementation, IEEE Trans. Comput., vol. 60, no. 4, pp. 538-551, Apr. 2011.

[28] Y. Al-Safi and V. Vyatkin, An ontology based reconfiguration agent for intelligent mechatronic systems, in: Proc. 4th Int. Conf. Hol. Multi- Agent Syst. Manuf., Regensburg, Germany, 2007, vol. 4659, pp. 114-126.

[29] S. Baruah and J. Goossens, Scheduling real-time tasks: Algorithms and complexity, in: Joseph Y-T Leung (ed)., Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 2004.

[30] M. Spuri, Analysis of deadline scheduled real-time systems, Tech. Rep. RR-2772, INRIA, France, January 1996.

[31] J. Palencia, M.G. Harbour, Offset-based response time analysis of distributed systems scheduled under EDF, in: Proceedings of the 15th Euromicro Conference on Real-Time Systems, Porto, ortugal, 2003.

[32] X. Wang, M. Khalgui, and Z. W. Li, Dynamic low power reconfigurations of real-time embedded systems, in: Proc. 1st Pervas. Embedded Comput. Commu. Syst. Mar. 2011, Algarve, Portugal.

[33] L. GEORGE, P. COURBIN,"Reconfiguration of Uniprocessor Sporadic Real-Time Systems: The Sensitivity Approach", book chapter in IGI-Global Knowledge on Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility. Ed: Khalgui & Hanisch. ISBN 978-1-60960-086-0, 2011. Published by Information Science, USA.

[34] Hamza Gharsellaoui, Atef Gharbi, Mohamed Khalgui, Samir Ben Ahmed: "Feasible Automatic Reconfigurations of Real-Time OS Tasks", "Handbook of Research on Industrial Informatics and Manufacturing Intelligence: Innovations and Solutions", editor(s): Mohammad Ayoub Khan, Abdul Quaiyum Ansari, India.

**KEY TERMS**

**Embedded systems:** An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today

**Dynamic Reconfiguration: :** The dynamic reconfiguration means qualitative changes on-line (at run time) in the structures, functionalities and algorithms of the "Control System" as a response to qualitative changes of control goals, of the controlled "physical system", or of the environment the control system behaves within.

**Real-Time OS Tasks:** Smallest identifiable and essential piece of a job that serves as a unit of work, and as a means of differentiating between the various components of a project. Often used as an alternative term for activity.

**Real-time constraints:** Execution of real-time tasks must satisfy three types of constraints. Timing constraints enforce each task instance to complete its execution before D after the date the task is released (D is a relative deadline); precedence constraints force partially task instance order and the read of current data values; resource constraints represent the exclusive access to shared resources.

**Processor utilization factor:** Given a set of n periodic tasks, processor utilization factor U is the fraction of processor time spent in the execution of the task set.

**Intelligent Agent:** On the Internet, an intelligent agent (or simply an *agent*) is a program that gathers information or performs some other service without your immediate presence and on some regular schedule. Typically, an agent program, using parameters you have provided, searches all or some part of the Internet, gathers information you're interested in, and presents it to you on a daily or other periodic basis. An agent is sometimes called a bot (short for robot).

**Run-time Environment:** Stands for "Runtime Environment." As soon as a software program is executed, it is in a runtime state. In this state, the program can send instructions to the computer's processor and access the computer's memory (RAM) and other system resources.
When software developers write programs, they need to test them in the runtime environment.

**Response time of an instance:** is the time (measured from the release time) at which the instance is terminated.