Feasible Automatic Reconfigurations of Real-Time OS Tasks

Authors: Hamza Gharsellaoui(1), Atef Gharbi(2), Mohamed Khalgui(3), Samir Ben Ahmed(4)
(1,2,4)National Institute of Applied Sciences and Technology, Tunisia
(3)Martin Luther University, Germany
Xidian University - China

{(1)gharsellaoui.hamza@gmail.com, (2)atef.elgharbi@gmail.com, (3)khalgui.mohamed@gmail.com, (4)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, synchronous and periodic in order to meet functional and temporal properties described in user requirements. We define in the book chapter 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. 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 provides otherwise for users to remove some tasks, to change temporal parameters of tasks that violate corresponding constraints by new ones, changing the system’s execution model to be assumed as only one task that should support all required functionalities or to take a hybrid real-time scheduling approach that combines the rate monotonic scheduling algorithm RM and the Earliest Deadline First scheduling algorithm EDF. 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 on the running example system 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.

INTRODUCTION

Starting as an aid for industrially specialized embedded applications, Real time operating systems (RTOSs) are now common in a large variety of commercial products. Application areas are for example telecommunications, automotive, defense industry, medical equipment and consumer electronics. 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. 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).
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. To reduce their cost, these systems should be changed and adapted to their environment without disturbances. 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). 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. According to (Liu and Layland, 1973) we characterize each task by a period to be denoted by T equal to the deadline denoted by D and by a Worst Case Execution Time (WCET) denoted by C. 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. Five cases of suggestions are possible to be provided by the agent: remove of some tasks from the new list, modification of periods (equal to deadlines), modification of worst case execution times of tasks, changing the system's execution model to be assumed as only one task that should support all required functionalities or to take a hybrid real-time scheduling approach that combines the rate-monotonic scheduling algorithm RM and the EDF scheduling algorithm. The users should choose one of these solutions to re-obtain the system's feasibility. A tool *RT - Reconfiguration* is developed and tested in our laboratory to support the agent's services. 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 of real-time embedded systems domain.

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. After the process is dispatched, it will make a transition according to the state diagram in figure 1.

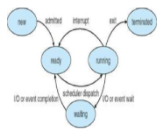


Figure 1: A process state diagram

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

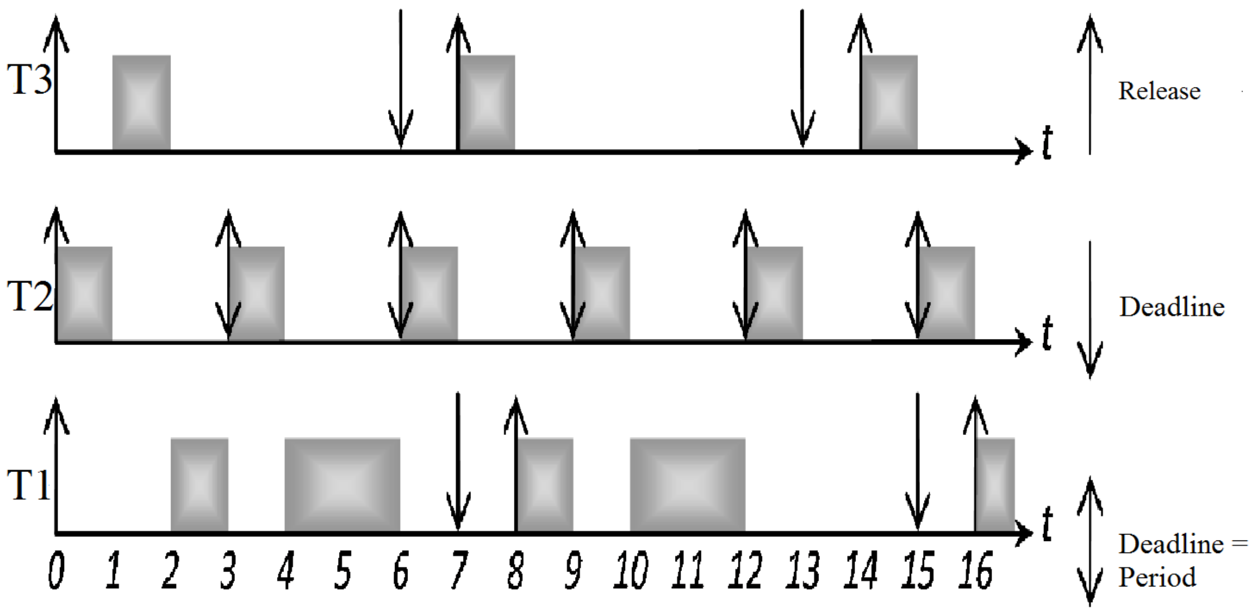


Figure 2. Scheduling of the system described in Table 0.2 by EDF

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 τ_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:

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

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:

$$\text{Wn} = n*(2^{1/n} - 1)$$ (2). (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 $U_n \le W_n$. 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 τ_1, τ_2, and τ_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 τ_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, τ_2, τ_1, and τ_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

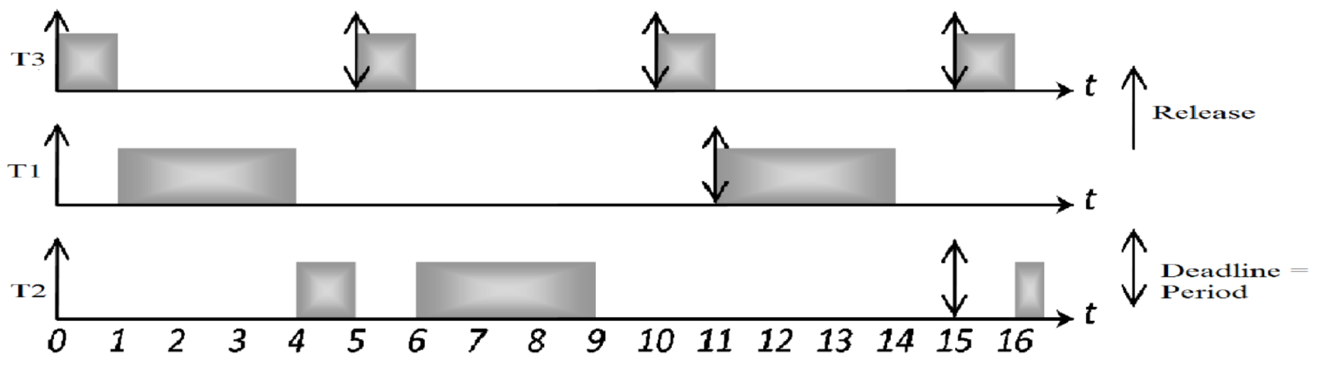


Figure 3. Scheduling of the system described in Table 0.3 by RM

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.

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

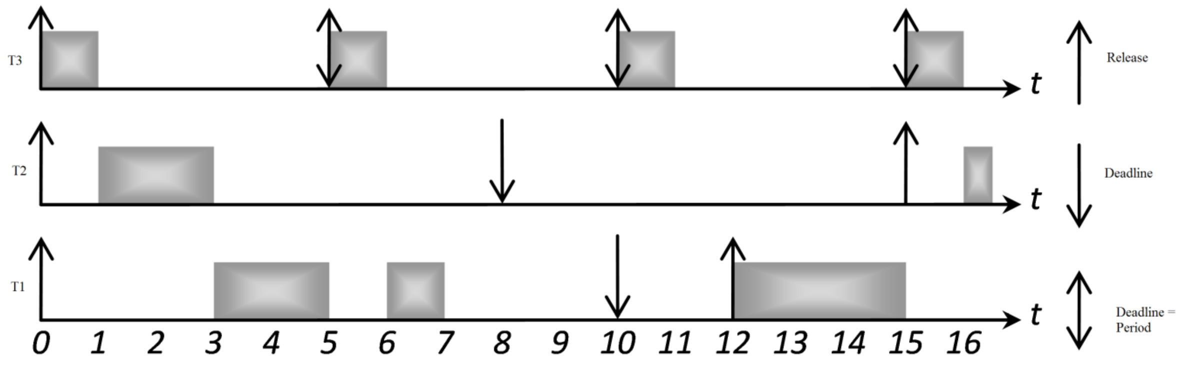


Figure 4. Scheduling of the system described in Table 0.4 by DM

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.

Example

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

Table 0.5: A task set example

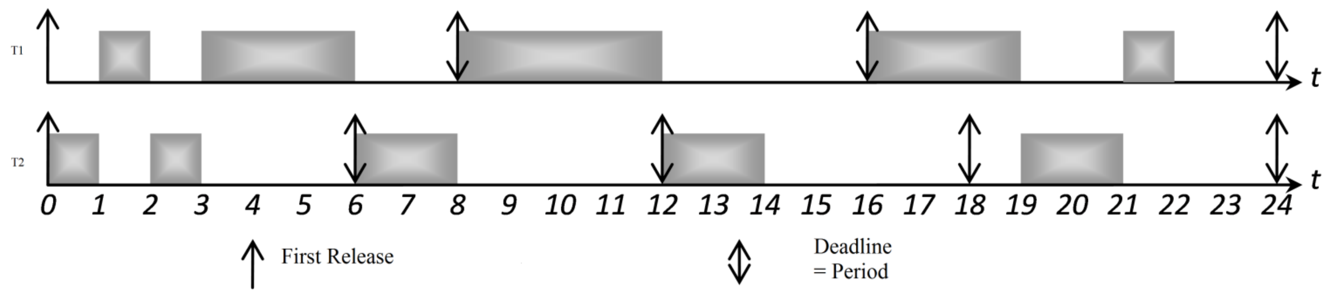


Figure 5. Scheduling of the system described in Table 0.5 by LLF

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

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

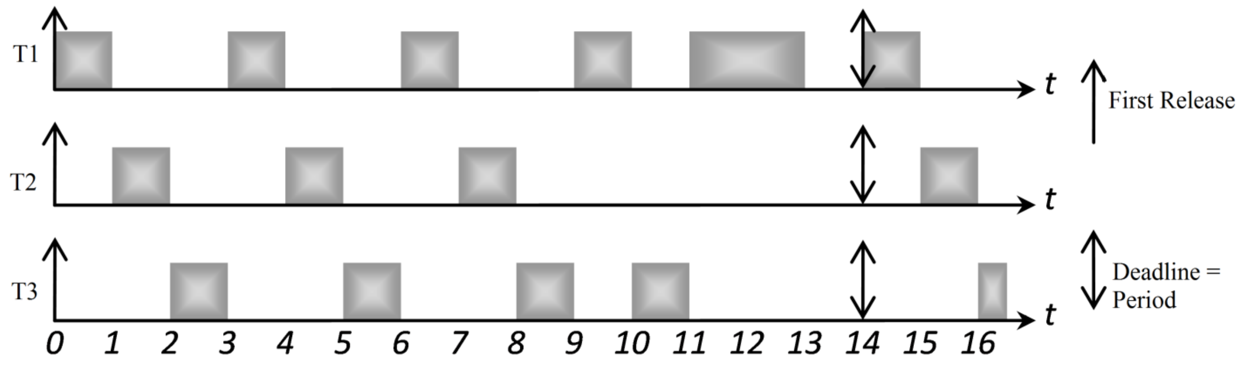


Figure 6. Scheduling of the system described in Table 0.6 by RR

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 or sporadic. 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, B and W are the best and worst 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
- Proposition of a new execution model by assuming only one feasible task that will support all the system's functionalities,
- Taking a hybrid real-time scheduling approach that combines the rate Monotonic scheduling algorithm RM and the EDF scheduling algorithm.

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 $\square_i$ 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 $\square_i$, 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 $\square_i$ 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 the case of deadlines equal to periods, in (Lehoczky, Sha, & Y. Ding, 1989), the authors introduce the critical scaling factor $\square$ on the WCETs, computed for a fixed priority scheduling such that the WCET of a task $\square_i$ is $\square\, C_i$ seen as third criteria of this work. This approach was extended in (Vestal, S., 1994), where the authors show how to introduce slack variables in the response times of the tasks. They propose the concept of scaling factor applied to one task and to all the tasks. In the case of independent deadlines and periods, $\square$ can be computed byiterations, but the computation becomes more and more costly. Indeed, when $\square$ increases, the length of the level$_i$ busy period tends towards P as the load utilization tends towards 1. In that case, increasing the task WCETs requires the recomputing the lengths of the level$_i$ busy periods which tend towards P, a potentially exponential length. 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 $\square^{C}$ *be the task set where every WCET is multiplied by a scaling factor* $\square$ *. Let* $\square^{T}$ *be the task set where every Task period is divided by* $\square$ *. The task set* $\square^{C}$ *is schedulable if and only if the task set* $\square^{T}$ *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 (equal to deadlines),
- ✓ Modification of worst case execution times of tasks,
- ✓ Proposition of a new execution model by assuming only one feasible task that will support all the system's functionalities,
- ✓ Taking a hybrid real-time scheduling approach that combines the rate Monotonic scheduling algorithm RM and the EDF scheduling algorithm.

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 a 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 $\text{Current}_{\text{Sys}}(t)$ 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 $\text{Feasibility}(\text{Current}_{\text{Sys}}(t)) \equiv \text{True}$.

Example:

Let us suppose a real-time embedded system Sys to be initially implemented by 70 characterized tasks (Figure 7). These tasks are feasible because the processor utilization factor $U = 0.95 < 1$. These tasks should meet all required deadlines defined in user requirements and we have $\text{Feasibility}(\text{Current}_{\text{Sys}}(t)) \equiv \text{True}$.

Agent-based architecture for Reconfigurable Embedded Control Systems

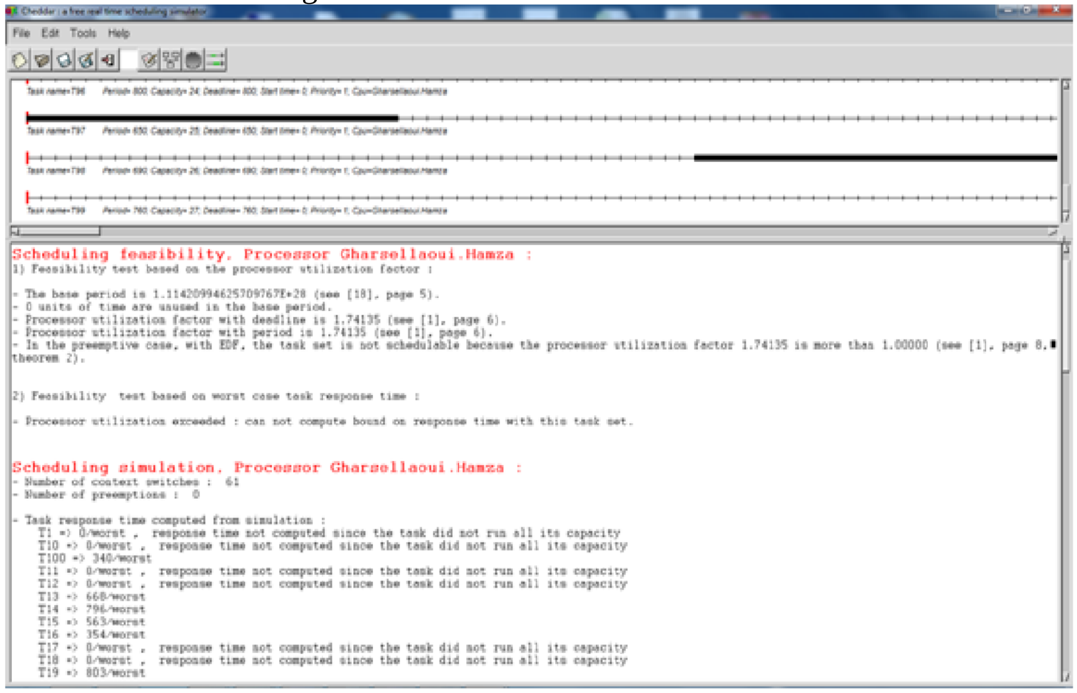
We define in this section an agent-based architecture for reconfigurable real-time embedded systems that should classically meet different deadlines defined in user requirements. The agent controls all the system's evolution and provides useful solutions for users when deadlines are violated after any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

Running Example:

In our real-time embedded system Sys to be initially implemented by 70 characterized tasks which are feasible because the processor utilization factor $U = 0.95 < 1$. We suppose that a reconfiguration scenario is applied at t1 time units to add 30 new tasks T71; T72… T100 (Figure 7). The new processor utilization becomes $U = 1.74 > 1$ time units. Therefore the system is unfeasible.

Feasibility$(Current_{Sys}(t)) \equiv$ *False.*

To check the whole system behavior of this system *sys* we present simulations given by the real-time simulator Cheddar in Figure 2.


Cheddar : a free real time scheduling simulator
File Edit Tools Help
Task name=T96 Period= 800; Capacity= 24; Deadline= 800; Start time= 0; Priority= 1; Cpu=Gharsellaoui.Hamza
Task name=T97 Period= 650; Capacity= 25; Deadline= 650; Start time= 0; Priority= 1; Cpu=Gharsellaoui.Hamza
Task name=T98 Period= 690; Capacity= 26; Deadline= 690; Start time= 0; Priority= 1; Cpu=Gharsellaoui.Hamza
Task name=T99 Period= 760; Capacity= 27; Deadline= 760; Start time= 0; Priority= 1; Cpu=Gharsellaoui.Hamza

```
Scheduling feasibility, Processor Gharsellaoui.Hamza :
1) Feasibility test based on the processor utilization factor :

- The base period is 1.11420994625709767E+28 (see [18], page 5).
- 0 units of time are unused in the base period.
- Processor utilization factor with deadline is 1.74135 (see [1], page 6).
- Processor utilization factor with period is 1.74135 (see [1], page 6).
- In the preemptive case, with EDF, the task set is not schedulable because the processor utilization factor 1.74135 is more than 1.00000 (see [1], page 8,
theorem 2).


2) Feasibility  test based on worst case task response time :

- Processor utilization exceeded : can not compute bound on response time with this task set.


Scheduling simulation, Processor Gharsellaoui.Hamza :
- Number of context switches :  61
- Number of preemptions :  0

- Task response time computed from simulation :
    T1 => 0/worst ,  response time not computed since the task did not run all its capacity
    T10 => 0/worst ,  response time not computed since the task did not run all its capacity
    T100 => 340/worst
    T11 => 0/worst ,  response time not computed since the task did not run all its capacity
    T12 => 0/worst ,  response time not computed since the task did not run all its capacity
    T13 => 668/worst
    T14 => 796/worst
    T15 => 563/worst
    T16 => 354/worst
    T17 => 0/worst ,  response time not computed since the task did not run all its capacity
    T18 => 0/worst ,  response time not computed since the task did not run all its capacity
    T19 => 803/worst
```

Figure 8. Simulations of the system sys by Cheddar

Agent's Principal

According to different conditions, the embedded system *Sys* 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 ξ_new a list of new tasks to be added to *Sys* 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 it should be able to propose the following technical solutions for users:

First Solution: it provides a list of tasks to be removed from ξ_new in order to re-obtain the system's feasibility,

Second Solution: it proposes new periods and deadlines for particular new and old tasks in order to minimize the processor utilization,

Third Solution: it proposes new Worst Case Execution Times (e.g. WCET) for tasks to minimize also such utilization,

Fourth Solution: it proposes a new execution model by assuming only one feasible task that will support all the system's functionalities,

Fifth Solution: the agent proposes a useful hybrid real-time scheduling that combines EDF and RM for the feasible execution of new and old tasks.

Formalization

By considering synchronous real-time tasks, the schedulability analysis should be done in the Hyper-Period
hp = [0, lcm], where lcm is the well known Least Common Multiple (S.Baruah and J. Goossens, 2004). Let n be the number of tasks in $Current_{Sys}(t)$. The reconfiguration of the system *Sys* means the modification of its implementation that will be as follows at t time units:
$Current_{Sys}(t) = \xi\_new \cup \xi\_old$.

Where ξ_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 ξ_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 *Sys* before the application of the reconfiguration scenario, each one of the tasks of ξ_old is feasible, e.g. the execution of each instance is finished before the corresponding deadline:
$\forall$ i $\in \xi\_old$, $\forall$ j $\in$ IN (e.g. instance j of task i) such that $j * Ti \leq t$ and $E_{i,j} \leq D_{i,j}$, Where Ei,j is the time when the execution of the instance j is finished and $D_{i,j}$ is the deadline of the instance j of task i. When the reconfiguration scenario is applied at time t, two cases exist:

- If tasks of CurrentSys(t) = $\xi\_new \cup \xi\_old$ are feasible, then no reaction should be done by the agent,
- Otherwise, the agent should provide different solutions for users in order to re-obtain the system's feasibility.

Let n1 and n2 be the number of tasks respectively in ξ_old and ξ_new . By assuming unfeasible system at *t* time units, the following formula is satisfied:

$$\sum_{i=1}^{n} \frac{Ci}{Ti} > 1.$$

We propose to add the tasks of ξ_old to a linked list L_{old} that we sort in the decreased order of the processor times. Let *j* be the first *j* tasks of L_{old}.

- ***Approach For each*** $j \in [0; n1]$,

Add the first *j* tasks of L_{old} to ξ_new .
Comments. If $j = 0$, Then no tasks to be add to ξ_new .
Five different solutions exist for the feasibility of the system:

Experimentation

Solution 1: *Remove of Tasks*

The agent computes the processor utilization $\frac{Ci}{Ti}$ of each task Ti and generates the feasible superset Ω_{feasible} which defines the different feasible subsets of tasks.

$\Omega_{\text{feasible}} = \{ \tau \subseteq$ CurrentSys/Feasibility(τ) = True}

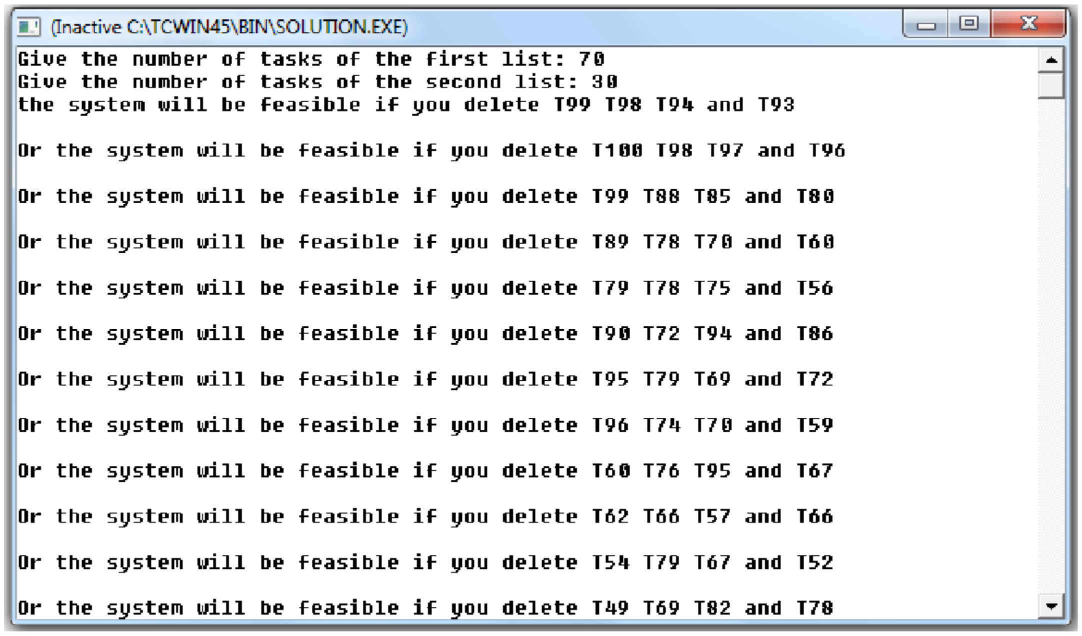
Each subset τ corresponds to a possible implementation of the system such that:

$$\tau = \xi\_\text{new} \cup \xi\_old \text{ , } \sum_{\text{Ti} \in \tau} \frac{Ci}{Ti} \leq 1$$

The agent suggests all possible combinations of tasks which can be removed from the subset τ for users who have the ability to choose the best combination that satisfies functional requirements. 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 in the case when they present the lowest importance. The importance is characterized by a value of the processor utilization $\frac{Ci}{Ti}$ of each task Ti from the subset τ that we sort in the decreased order of the processor times, e.g. Task with the highest processor utilization presents the lowest importance.

Running Example:

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our sys system, we present in Figure 9 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 $\tau = \xi\_\text{new} \cup \xi\_old$ *. The user can choose one of these solutions to have a feasible system.*



```
(Inactive C:\TCWIN45\BIN\SOLUTION.EXE)
Give the number of tasks of the first list: 70
Give the number of tasks of the second list: 30
the system will be feasible if you delete T99 T98 T94 and T93

Or the system will be feasible if you delete T100 T98 T97 and T96

Or the system will be feasible if you delete T99 T88 T85 and T80

Or the system will be feasible if you delete T89 T78 T70 and T60

Or the system will be feasible if you delete T79 T78 T75 and T56

Or the system will be feasible if you delete T90 T72 T94 and T86

Or the system will be feasible if you delete T95 T79 T69 and T72

Or the system will be feasible if you delete T96 T74 T70 and T59

Or the system will be feasible if you delete T60 T76 T95 and T67

Or the system will be feasible if you delete T62 T66 T57 and T66

Or the system will be feasible if you delete T54 T79 T67 and T52

Or the system will be feasible if you delete T49 T69 T82 and T78
```

Figure 9. Presentation of Solution 1 results described by the developed tool RT − Configuration

Solution 2: *Modification of Periods*

The agent proceeds as a second solution to change the periods of tasks of ξ_new. To obtain a feasible system, the following formula should be satisfied;

$$\sum_{i=1}^{n_1-j} \frac{Ci}{Ti} + \sum_{i=n_1-j+1}^{n_1+n2} \frac{Ci}{Ti+\theta i} = 1$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2} \frac{Ci}{Ti+\theta i} = 1 - \sum_{i=1}^{n_1-j} \frac{Ci}{Ti}$$

Let β_j be $Ti+\theta i$

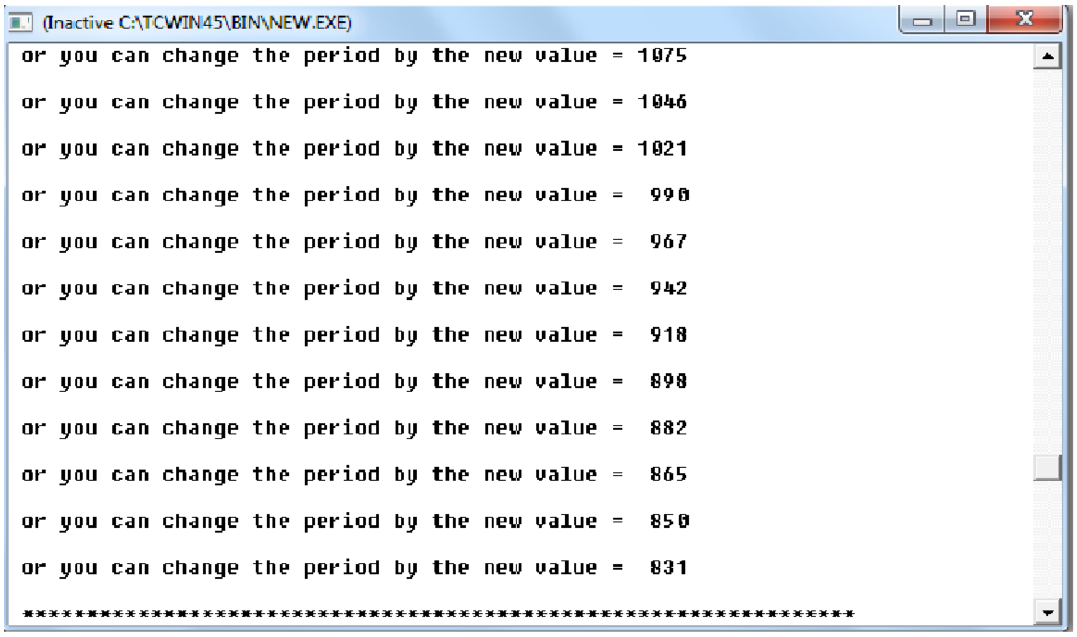
$$\rightarrow \frac{1}{\beta_j} \sum_{i=n_1-j+1}^{n_1+n2} Ci = 1 - \sum_{i=1}^{n_1-j} \frac{Ci}{Ti}$$

$$\rightarrow \beta_j = \left[\frac{\sum_{i=n_1-j+1}^{n_1+n2} Ci}{1 - \sum_{i=1}^{n_1-j} \frac{Ci}{Ti}} \right] \text{ = Constante}$$

The new period of tasks of ξ_new is therefore deduced from β_j.

Running Example:

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our sys system, we present in Figure 10 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:
ξ_new = *{T_{71}; T_{72}... T_{100}}*
L_{old} = ξ_old = {T_1; T_2... T_{70}}
The agent computes the constant values β_j (j ∈ [0; 30]) corresponding respectively as follows:
$\beta_0 = ...$, $\beta_1 = ...$, until β_{30} = 831 time units where $L_{old} = \varnothing$ and ξ_new = {T_1; T_2... T_{70}; T71; ...; T100}.


(Inactive C:\TCWIN45\BIN\NEW.EXE)

```
or you can change the period by the new value = 1075

or you can change the period by the new value = 1046

or you can change the period by the new value = 1021

or you can change the period by the new value =  990

or you can change the period by the new value =  967

or you can change the period by the new value =  942

or you can change the period by the new value =  918

or you can change the period by the new value =  898

or you can change the period by the new value =  882

or you can change the period by the new value =  865

or you can change the period by the new value =  850

or you can change the period by the new value =  831

*****************************************************************
```

Figure 10. Presentation of Solution 2 results described by the developed tool RT − Configuration

Solution 3: *Modification of Worst Case Execution Times*
The agent proceeds now as a third solution to modify the Worst case Execution Times of tasks of ξ_new. To obtain a feasible system, the following formula should be satisfied:

$$\sum_{i=1}^{n_1-j} \frac{Ci}{Ti} + \sum_{i=n_1-j+1}^{n_1+n2} \frac{Ci+\alpha i}{Ti} = 1$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2} \frac{Ci+\alpha i}{Ti} = 1 - \sum_{i=1}^{n_1-j} \frac{Ci}{Ti}$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2} \frac{\alpha i}{Ti} = 1 - \sum_{i=1}^{n_1-j} \frac{Ci}{Ti} - \sum_{i=n_1-j+1}^{n_1+n2} \frac{Ci}{Ti}$$

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2} \frac{\alpha i}{Ti} = 1 - \sum_{i=1}^{n_1+n2} \frac{Ci}{Ti}$$

Let γ_j be the following constant: γ_j = αi = Constante,

$$\rightarrow \sum_{i=n_1-j+1}^{n_1+n2} \frac{\alpha i}{Ti} = 1 - \sum_{i=1}^{n_1+n2} \frac{Ci}{Ti}$$

$$\rightarrow \gamma_j = \left\lceil \frac{1 - \sum_{i=1}^{n_1+n2} \frac{Ci}{Ti}}{\sum_{i=n_1-j+1}^{n_1+n2} \frac{1}{Ti}} \right\rceil$$

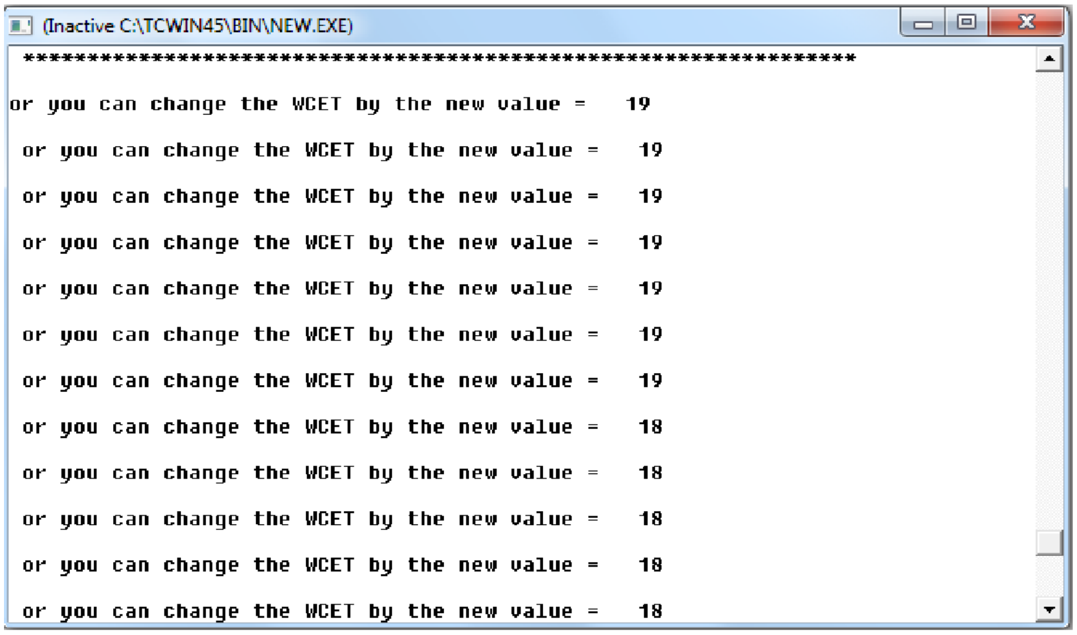
The new WCET of tasks of ξ_new is therefore deduced from γ_j .

Running Example:

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our sys system, we present in Figure 11 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:

ξ_new = $\{T_{71}; T_{72} \dots T_{100}\}$
L_{old} = ξ_*old* = $\{T_1; T_2 \dots T_{70}\}$
The agent computes the constant values γ_j *(j ∈ [0; 30]) corresponding respectively to the new values of the Worst Case Execution Times (WCET) as follows:* $\mathrm{WCET}_0 = \dots$, $\mathrm{WCET}_1 = \dots$, *until* $\mathrm{WCET}_{30} = 18$ *time units where* $L_{old} = \varnothing$ *and* ξ_new = $\{T_1; T_2 \dots T_{70}; T_{71}; \dots; T100\}$.


(Inactive C:\TCWIN45\BIN\NEW.EXE)

```
 ********************************************************************

or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   19

 or you can change the WCET by the new value =   18

 or you can change the WCET by the new value =   18

 or you can change the WCET by the new value =   18

 or you can change the WCET by the new value =   18

 or you can change the WCET by the new value =   18
```

Figure 11. Presentation of Solution 3 results described by the developed tool RT − Configuration

Solution 4: *New Execution Model for the System*

The agent proceeds as a fourth solution to change the system's execution model to be assumed as only one task Γ that should support all required functionalities. Let n_i be the number of instances of the task Ti $\in \xi\_\mathrm{new} \cup \xi\_old$ in the hyper-period.

ni = lcm/Pi

The global execution time of the system in the whole hyper-period is as follows:

$$\text{WCETlcm} = \sum_{Ti \in \xi\_new \cup \xi\_old} ni * C_i$$

Let pi be the shortest period of tasks in the system. If WCETlcm/lcm ≤ 1, then the agent suggests the following parameters for Γ :

– R (Γ) = 0,

– P (Γ) = WCETlcm/pi = D (Γ),

– WCET (Γ) = $\sum_{Ti \in \xi\_new \cup \xi\_old}$ (ni * C_i)/Ti.

The agents suggests that each instance will be composed of (ni * C_i)/Ti of Ti.

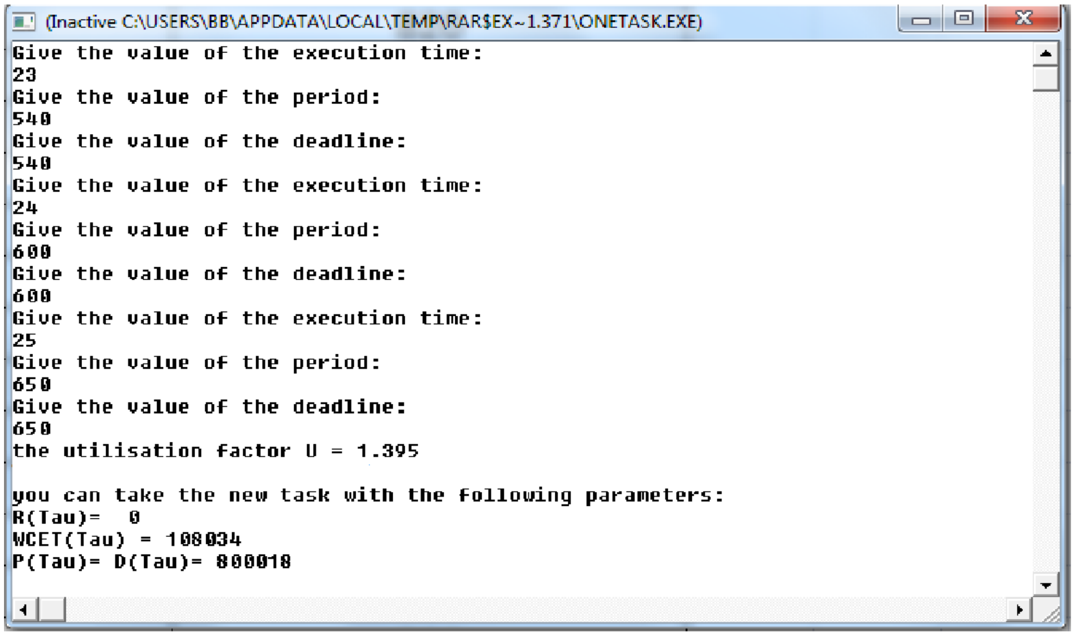
Running Example:

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our sys system, we present in Figure 12 the results described by the developed tool RT – Configuration. These results are the following parameters for Γ *suggested by the Agent as follows:*

– R (Γ *) = 0,*

– P (Γ *) = D (* Γ *) = 800018,*

– WCET (Γ *) =* $\sum_{Ti \in \xi\_new \cup \xi\_old}$ *(ni * C_i)/Ti = 108034 .*



```
(Inactive C:\USERS\BB\APPDATA\LOCAL\TEMP\RAR$EX~1.371\ONETASK.EXE)
Give the value of the execution time:
23
Give the value of the period:
540
Give the value of the deadline:
540
Give the value of the execution time:
24
Give the value of the period:
600
Give the value of the deadline:
600
Give the value of the execution time:
25
Give the value of the period:
650
Give the value of the deadline:
650
the utilisation factor U = 1.395

you can take the new task with the following parameters:
R(Tau)=  0
WCET(Tau) = 108034
P(Tau)= D(Tau)= 800018
```

Figure 12. Presentation of Solution 4 results described by the developed tool RT – Configuration

Solution 5: *Hybrid Real-Time Scheduling*

The Agent proposes in the fifth solution to combine EDF with RM for the scheduling of the system's tasks. Let τ_{EDF} and τ_{RM} be the set of tasks to be scheduled by the policies EDF and RM as follows:

$\tau_{\mathrm{EDF}} \subseteq S_{Sys}$ And $\tau_{\mathrm{RM}} \subseteq S_{Sys} \setminus \tau_{\mathrm{EDF}}$ such that,

$$- \sum_{\mathrm{Ti} \in \tau_{\mathrm{EDF}}} \frac{Ci}{Ti} \leq 1$$

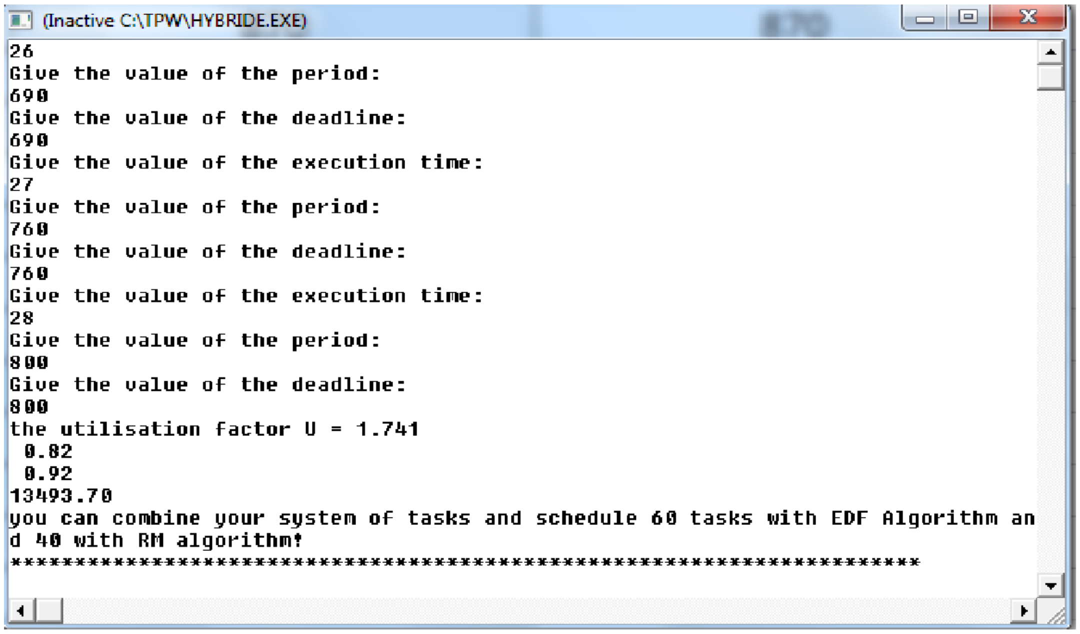
$$- \sum_{\mathrm{Ti} \in \tau_{\mathrm{RM}}} \frac{Ci}{Ti} \leq (n_1 + n_2)*(2^{1/(n_1+n_2)} - 1) - \sum_{\mathrm{Ti} \in \tau_{\mathrm{EDF}}} \frac{Ci}{Ti}$$

The agent should define the different solutions of the sets τ_{EDF} and τ_{RM}. In this case, the users can choose the best solution that satisfies functional requirements. Indeed, many tasks of the system should meet deadlines, whereas other can be executed without satisfaction of deadlines.

Running Example:

The agent should react to propose useful solutions for users in order to re-obtain the system's feasibility. In our sys system, we present in Figure 13 the best result described by the developed tool

RT – Configuration. This result is the best solution of the sets τ_{EDF} and τ_{RM} . In this case, the user can schedule 60 tasks with EDF algorithm (τ_{EDF} is composed of 60 tasks) and 40 tasks with RM algorithm (τ_{RM} is composed of 40 tasks) in order to have a feasible system.



```
26
Give the value of the period:
690
Give the value of the deadline:
690
Give the value of the execution time:
27
Give the value of the period:
760
Give the value of the deadline:
760
Give the value of the execution time:
28
Give the value of the period:
800
Give the value of the deadline:
800
the utilisation factor U = 1.741
 0.82
 0.92
13493.70
you can combine your system of tasks and schedule 60 tasks with EDF Algorithm an
d 40 with RM algorithm!
***********************************************************************
```

Figure 13. Presentation of Solution 5 results described by the developed tool RT – Configuration

Algorithm:

We present the different codes to be supported by the agent for a feasible reconfiguration of an embedded system.

```
**Code1** Removal_Tasks () $U \leftarrow 0;$
– **For each** partition $\beta \subseteq \tau_{new} \cup \tau_{old}$
– **For each** $Ti \in \beta$
        $- U + = \frac{Ci}{Ti};$
– **If** $U \leq 1$
    – **Then** display ( $\beta$ );
```

```
**Code2** Modify Periods_Deadlines_WCET()
– **Compute** ( $\beta_i$ );
– Compute ( $\gamma_i$ );
– For Ti $\in \xi\_new \cup \xi\_old$
        – Display parameters ();
```

Code3 Modify_New_Execution_Model ()
– generate parameters (P (Γ), WCET (Γ));

Code4 Modify New Execution Model ()
– Generate (τ_{EDF} , τ_{RM});

We note that the complexity of the proposed algorithm is *O*(*n*2). We implemented a prototype *RT-Reconfiguration* to give helps and indications for users when automatic reconfigurations of real-time embedded systems are applied at run-time. The simulator Cheddar (L. N. L. M. F. Singhoff, J. Legrand, 2004) is used for the schedulability analysis of the considered tasks.

Finally, we believe in the limitation of all related works belong to the automatic reconfiguration of real-time embedded systems 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.*

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.

FUTURE RESEARCH DIRECTIONS

We plan in future works to resolve several problems for the Reconfigurable real-time embedded systems.
The minimization criterion of the energy consumption has to be studied in order to control the production cost: is it useful to automatically change the OS tasks in order to reduce the energy consumption? Another problem that has to be resolved in the future deals with the study of each reconfiguration scenario of 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. We assume in this work independent, periodic and synchronous tasks. We define a reconfiguration scenario as a dynamic operation allowing automatic reconfigurations of the system like additions, removes or updates of temporal parameters of tasks at run-time. When a reconfiguration scenario is automatically applied, the system can become unfeasible because some tasks violate corresponding deadlines. We define an agent-based architecture where an agent is proposed to provide new parameters of unfeasible tasks in order to re-obtain the system's feasibility. To satisfy user requirements, we apply the Real-Time Simulator, cheddar to check the whole system behavior. Therefore, these contributions are now the subject of negotiations with industrial partners for the next generation of real-time embedded systems in manufacturing industry.

REFERENCES

A.-L. Gehin and M. Staroswiecki, Reconfiguration Anal ysis Using Generic Component Models, in IEEE Transactions on Systems, Machine and Cybernetics, Vol.38, N.3, 2008.

Balbastre, P. & Ripoll, I. (2002). Schedulability analysis of window-constrained execution time tasks for real-time control. *Proc. Of the* 14*th Euromicro Conference on Real- Time Systems (ECRTS'02)*, 2002.

Balbastre, P. & Crespo, A. (2006). Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, Dresden, Germany July 5-7, 2006.

Baruah and J. Goossens, Scheduling Real-time Tasks: Algorithms and Complexity, in In Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Joseph YT Leung (ed). Chapman Hall/ CRC Press, 2004.

Baruah, S., Buttazo, G., Gorinsky, S. & Lipari, G. (1999). Scheduling periodic task systems to minimize output jitter. *In* 6*th Conference on Real-Time Computing Systems and Applications*, pp. 62-69, 1999.

Bernat, G., Burns, A. & A., L. (2001). Weakly Hard Real-Time Systems. *IEEE Transactions on Computers*, vol. 50(4), pp. 308-321, 2001.

Bini, E. & Buttazzo, G. (2003). Rate monotonic analysis : the hyperbolic bound. *IEEE Transactions On Computers, Vol. 52, No. 7*, Jul.2003.
Bini, E. & Di Natale, M. (2005). Optimal Task Rateselection in Fixed Priority Systems. *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005.

Buttazzo, G., Lipari, G. & Abeni, L. (1998). Elastic task model for adaptive rate control. *In proc. of the 19th IEEE Real- Time System Symposium*, Dec.1998.

Buttazzo, G. (2006). Achieving scalability in real-time systems. *IEEE Computer*, 2006.

Hamdaoui, M. & Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computer*, Vol 44, pp. 1443-1451, December 1995.

Buttazzo, G., Lipari, G., Caccamo, M. & Abeni, L. (2002). Elastic scheduling for flexible workload management. *IEEE Transactions on computers*, Vol. 51, Num 3., March 2002.

Buttazzo, G. & Stankovic, J. (1993). RED: A Robust Earliest Deadline Scheduling. *3rd international Workshop on responsive Computing*, Sept. 1993.

C. Angelov, K. Sierszecki, and N. Marian, Design models for reusable and reconfigurable state machines, in L.T. Yang and All (Eds): EUC 2005, LNCS 3824, pp:152- 163. International Federation for Information Processing, 2005.

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

Cervin, A., Lincoln, B., Eker, J., Arzen, K. & G., B. (2004). The jitter margin and its application in the design of real-time control systems. *In proceedings of the IEEE Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.

Koren, G. & Shasha, D. (1992). D-over: An Optimal On-line Scheduling Algorithm for over loaded real-time system. *Research report Num. 138*, Feb. 1992.

K. Thramboulidis, G. Doukas, and A. Frantzis, towards an implementation model for FB-based reconfigurable distributed control applications, in Proceedings of 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 193-200, 2004.

Lehoczky, J. (1989), L. Sha, & Y. Ding. The Rate-Monotonic scheduling algorithm: exact characterization and average case behavior. *Proceedings of 10th IEEE Real-Time Systems Symposium*, pp 166-172, 1989.

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.

Lock, C. (1986). Best effort decision making for real-time scheduling. PhD thesis, Computer science department, Carnegie-Mellon University, 1986.

M. N. Rooker, C. Sunder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer, Zero Downtime Reconfiguration of Distributed Automation Systems: The ”CEDAC Ap- proach, in Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007. Springer Verlag.

M. Khalgui, NCES-based modelling and CTL-based verification of reconfigurable embedded control systems, in Computers in Industry. 61(3), 2010.

Marinca, D., Minet, P. & George, L. (2004). Analysis of deadline assignment methods in distributed real-time systems. *Computer Communications*, Elsevier, 2004.

Nelis, V., Goossens, J. & Andersson, B. (2009), Two Protocols for Scheduling Multi-mode Real-Time Systems upon Identical Multiprocessor Platforms, *21st Euromicro Conference on Real-Time Systems (ECRTS'09), pp. 151-160, 2009.*

R. W. Brennan, M. Fletcher, and D. H. Norrie, A holonic approach to reconfiguring realtime distributed control systems, in Book: Multi-Agent Systems and Applications: MASA01, 2001. Springer-Verlag.

Vestal, S. (1994). Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering*, VOL. 20, NO. 4,, April 1994.

Y. Al-Safi and V. Vyatkin, An ontology based reconfiguration agent for intelligent mechatronicsystems, in Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007. Springer-Verlag.

KEY TERMS & DEFINITIONS

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

Reconfiguration: The reconfiguration means qualitative changes 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).

Hybrid Algorithm: Any algorithm composed of simpler algorithms.

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.

ADDITIONAL READING SECTION

A.-L. Gehin and M. Staroswiecki, Reconfiguration Anal ysis Using Generic Component Models, in IEEE Transactions on Systems, Machine and Cybernetics, Vol.38, N.3, 2008.

A. Cheng, “Real-time systems: scheduling, analysis and verification”, ISBN: 0-471-18406-3.

Balbastre, P. & Ripoll, I. (2002). Schedulability analysis of window-constrained execution time tasks for real-time control. *Proc. Of the* 14*th Euromicro Conference on Real- Time Systems (ECRTS’02)*, 2002.

Balbastre, P. & Crespo, A. (2006). Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS’06)*, Dresden, Germany July 5-7, 2006.

Baruah and J. Goossens, Scheduling Real-time Tasks: Algorithms and Complexity, in In Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Joseph YT Leung (ed). Chapman Hall/ CRC Press, 2004.

Baruah, S., Buttazo, G., Gorinsky, S. & Lipari, G. (1999). Scheduling periodic task systems to minimize output jitter. *In* 6*th Conference on Real-Time Computing Systems and Applications*, pp. 62-69, 1999.

Bernat, G., Burns, A. & A., L. (2001). Weakly Hard Real-Time Systems. *IEEE Transactions on Computers*, vol. 50(4), pp. 308-321, 2001.

Bini, E. & Buttazzo, G. (2003). Rate monotonic analysis : the hyperbolic bound. *IEEE Transactions On Computers, Vol. 52, No. 7*, Jul.2003.
Bini, E. & Di Natale, M. (2005). Optimal Task Rateselection in Fixed Priority Systems. *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS’05)*, 2005.

Buttazzo, G., Lipari, G. & Abeni, L. (1998). Elastic task model for adaptive rate control. *In proc. of the 19th IEEE Real- Time System Symposium*, Dec.1998.

Buttazzo, G. (2006). Achieving scalability in real-time systems. *IEEE Computer*, 2006.

Hamdaoui, M. & Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computer*, Vol 44, pp. 1443-1451, December 1995.

Buttazzo, G., Lipari, G., Caccamo, M. & Abeni, L. (2002). Elastic scheduling for flexible workload management. *IEEE Transactions on computers*, Vol. 51, Num 3., March 2002.

Buttazzo, G. & Stankovic, J. (1993). RED: A Robust Earliest Deadline Scheduling. *3rd international Workshop on responsive Computing*, Sept. 1993.

B. Gaujal, N. Navet, "Dynamic Voltage Scaling under EDF Revisited", Real-Time Systems, Springer Verlag, vol. 37, n°1, pp77-97, 2007. Some results are available as research report INRIA RR-5125.
A.-L. Gehin and M. Staroswiecki, Reconfiguration Anal ysis Using Generic Component Models, in IEEE Transactions on Systems, Machine and Cybernetics, Vol.38, N.3, 2008.

C. Angelov, K. Sierszecki, and N. Marian, Design models for reusable and reconfigurable state machines, in L.T. Yang and All (Eds): EUC 2005, LNCS 3824, pp:152- 163. International Federation for Information Processing, 2005.

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

Cervin, A., Lincoln, B., Eker, J., Arzen, K. & G., B. (2004). The jitter margin and its application in the design of real-time control systems. *In proceedings of the IEEE Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.

F. Yao, A. Demers, S. Shenker, “A scheduling model for reduced CPU energy”, IEEE Annual Foundations of Computer Science, 1995,

G. Quan, X. Hu, “Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors”, Design, Automation and Test in Europe Conference and Exhibition, 2002,

H-S. Yun, J. Kim, On energy optimal voltage scheduling for fixed-priority hard real-time systems”, ACM Transactions on Embedded Computing Systems, 2003,

I. Pyle, P. Hruschka, M. Lissandre and K. Jackson, “Real-time Systems: Investigating Industrial Practice”, Jhon Wiley & Sons ISBN: 0-471-93553-0.

J.A. Stankovic, M. Spuri, K. Ramamritham, G.C. Butazzo, “Deadline scheduling for real-time systems”, ISBN: 978-0-7923-8269-0.

Koren, G. & Shasha, D. (1992). D-over: An Optimal On-line Scheduling Algorithm for over loaded real-time system. *Research report Num. 138*, Feb. 1992.

K. Thramboulidis, G. Doukas, and A. Frantzis, towards an implementation model for FB-based reconfigurable distributed control applications, in Proceedings of 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 193-200, 2004.

Lehoczky, J. (1989), L. Sha, & Y. Ding. The Rate-Monotonic scheduling algorithm: exact characterization and average case behavior. *Proceedings of 10th IEEE Real-Time Systems Symposium*, pp 166-172, 1989.

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.

L. Briand, D. Roy, "Meeting deadlines in hard real-time systems: Rate Monotonic approach" ISBN 0-8186-7406-7**.**

Lock, C. (1986). Best effort decision making for real-time scheduling. PhD thesis, Computer science department, Carnegie-Mellon University, 1986.

M. N. Rooker, C. Sunder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer, Zero Downtime Reconfiguration of Distributed Automation Systems: The "CEDAC Ap- proach, in Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007. Springer Verlag.

M. Khalgui, NCES-based modelling and CTL-based verification of reconfigurable embedded control systems, in Computers in Industry. 61(3), 2010.

Marinca, D., Minet, P. & George, L. (2004). Analysis of deadline assignment methods in distributed real-time systems. *Computer Communications*, Elsevier, 2004.

Nelis, V., Goossens, J. & Andersson, B. (2009), Two Protocols for Scheduling Multi-mode Real-Time Systems upon Identical Multiprocessor Platforms, *21st Euromicro Conference on Real-Time Systems (ECRTS'09), pp. 151-160, 2009.*

P. Laplante, "Real-time systems: Design and Analysis" ISBN: 0-7803-0402-0.

R. W. Brennan, M. Fletcher, and D. H. Norrie, A holonic approach to reconfiguring realtime distributed control systems, in Book: Multi-Agent Systems and Applications: MASA01, 2001. Springer-Verlag.

Vestal, S. (1994). Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering*, VOL. 20, NO. 4,, April 1994.

Y. Al-Safi and V. Vyatkin, An ontology based reconfiguration agent for intelligent mechatronicsystems, in Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007. Springer-Verlag.

Y. Shin, K, Choi, Power Conscious Fixed Priority Scheduling for Hard Real-time Systems, Design Automation Conference, 1999,

Y. Al-Safi and V. Vyatkin, An ontology based reconfiguration agent for intelligent mechatronicsystems, in Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, 2007. Springer-Verlag.