A New Approach for Automatic Development of Reconfigurable Real-Time Systems

Wafa Lakhdhar[1](✉), Rania Mzid[2,3], Mohamed Khalgui[1,5,6], and Nicolas Treves[4]

[1] LISI Lab INSAT, INSAT Centre, University of Carthage, Urbain Nord BP 676, Tunis, Tunisia
wafa.lakdhar@live.fr, khalgui.mohamed@gmail.com

[2] ISI, University Tunis-El Manar, 2 Rue Abourraihan Al Bayrouni, Ariana, Tunisia
rania.mzid@gmail.com

[3] CES Lab ENIS, University of Sfax, B.P:w.3, Sfax, Tunisia

[4] CEDRIC Lab, CNAM, 292 rue Saint-Martin, Paris, France

[5] Systems Control Lab, Xidian University, August Bebel Str 70, Halle, China
nicolas.treves@cnam.fr

[6] School of Electrical and Information Engineering, Jinan University, Zhuhai Campus, Zhuhai 519070, China

Abstract. In the industry, reconfigurable real-time systems are specified as a set of implementations and tasks with timing constraints. The reconfiguration allows to move from one implementation to another by adding/removing real-time tasks. Implementing those systems as threads generates a complex system code due to the large number of threads and the redundancy between the implementation sets. This paper shows an approach for software synthesis in reconfigurable uniprocessor real-time embedded systems. Starting from the specification to a program source code, this approach aims at minimizing the number of threads and the redundancy between the implementation sets while preserving the system feasibility. The proposed approach adopts Mixed Integer Linear Programming (MILP) techniques in the exploration phase in order to provide feasible and optimal task model. An optimal reconfigurable POSIX-based code of the system is manually generated as an output of this technique. An application to a case study and performance evaluation show the effectiveness of the proposed approach.

Keywords: Real-time system · Reconfigurable architecture · Timing constraints · Mixed Integer Linear Programming (MILP) · POSIX-based code

NOMENCLATURE

U	Processor utilization
n	Number of thread
m	Number of implementation
Sys	System implementations set
imp_i	The i^{th} implementation
F_i	The i^{th} Function
T_{f_i}	The Period of the i^{th} function
C_{f_i}	The WCET of the i^{th} function
τ_i	The i^{th} task
r_i	The release time of the i^{th} task
T_i	The period of the i^{th} task
C_i	The WCET of the i^{th} task
D_i	The deadline of the i^{th} task
P_i	The priority of the i^{th} task
Rep_i	The Response time of the i^{th} task
T_{reconf}	The reconfiguration time
T_{delete}	the spent time to delete a task
T_{creat}	the spent time to create a task
A	the number of deleted tasks
B	is the number of created tasks
Merge$_{ij}$	Merging Matrix
InitTask	Initial Task model
NewTask	New task model

1 Introduction

A real-time system is any system which has to respond to externally generated input stimuli within a finite and specified delay [1]. The development of real-time systems is not a trivial task because a failure can be critical for the safety of human beings [2]. The researchers are moving today toward proposing techniques for programming concurrent reconfigurable real-time systems. The reconfiguration refers to the architectural or behavioral modifications of a software system during its execution to meet user requirements [3]. The successful development of reconfigurable real-time systems greatly depends on low development costs and the respect of timing requirements. In fact, several approaches have been proposed to assist the designer in the synthesis of real-time systems at different levels of the development process. For real-time concerns, Cheddar tool [4] allows to model software architectures of real-time systems while ensuring the respect of real-time properties. To provide design-time guarantees on timing constraints, different scheduling methodologies can be used, such as earliest deadline first scheduling algorithm (EDF) which at each instant in time chooses for execution the currently-active job with the smallest deadline [5]. Indeed the authors in [6] propose the RT_Reconfiguration tool based on EDF scheduling to assist in designing a feasible reconfigurable real-time system using an agent-based approach. Among all priority driven policies, Rate Monotonic (RM) is a scheduling algorithm which was defined by Liu and Layland [7] where the priority of tasks is inversely proportional to their periods. The authors of [8] provide a method to drive the designer by producing a set of design solutions based on RM scheduling algorithm. In [9–11], the authors are interested in the optimization of deployment techniques from functional and platform models of real-time systems by using mixed integer linear programming (MILP). An MILP formulation is easily extensible, re-targetable to a different optimization metric and can easily accommodate additional constraints or legacy components [9]. The TASTE Toolset approach results from spin-off studies of the ASSERT project in order to propose innovative and pragmatic solutions to develop real-time systems using a language based on Simulink, SDL, ASN.1, C, and Ada [12]. There are many programming languages designed for the development of real-time systems such as POSIX (Portable Operating System Interface) [13]. The POSIX standard promotes portability of applications across different operating system platforms. The authors [14] use POSIX in the development of software for real-time and embedded systems.

The synthesis of a valid and optimal implementation model from a given specification is a crucial issue in the development of reconfigurable real-time applications. This synthesis consists in building the set of tasks implementing the applicative functions while meeting all related real-time constraints. The reconfiguration at the implementation level consists in adding/removing tasks or modifying their timing parameters to go from one implementation to another, which may require an additional time for reconfiguration. So that, the resulting implementation model should avoid redundancy between the different implementations to minimize the possible overhead.

In this paper, we present an approach toward an optimal implementation of reconfigurable uniprocessor real-time systems. The proposed approach aims to automatically produce a valid and optimal task model from a given specification. The task model consists of a set of tasks implementing the applicative functions that we assume independent and periodic. We assume also that assigning priorities to tasks is performed using rate monotonic algorithm RM [15]. The proposed approach is composed of three phases: the purpose of the first one is to produce an initial task model from the user specification. The second step aims to optimize this model by using mixed integer linear programming (MILP) techniques to generate a feasible and an optimal task model. The proposal considers timing constraints. As for metrics, we consider a multi-objective optimization which includes the minimization of the response time. Since there are many solvers handling MILP formulations, we use in this paper the CPLEX tool [16]. From this optimal model, the objective of the third phase is to produce a POSIX-based code for the considered application. The proposed approach is applied to a CCAS case study in order to show its applicability.

The paper is organized as follows. Section 2 gives an overview on related works. Section 3 provides the formalisation of approach. Section 4 explains in details the proposed approach to obtain a valid and optimal implementation model from the user specification. Section 5 illustrates the approach on the chosen case study and evaluates its efficiency. Finally, we summarize our work and discuss the future work in Sect. 6.

2 Related Works

In this section, we present the related works that deal with real-time systems and reconfigurable architectures.

2.1 Real-Time Scheduling

Several works deal with the synthesis problem of real-time systems. The correctness of such systems depends both on the logical result of the computation and the time when the results are produced [17]. Thus enforcing timeliness constraints is necessary to maintain correctness of a real-time system. In order to ensure a required real-time performance, the designer should predict the behaviour of a real-time system by ensuring that all the tasks meet their deadlines. Different classes of scheduling algorithms exist where each one is developed for a particular task model or an environment in which a real-time system operates. Among all priority driven policies, Rate Monotonic (RM) is a scheduling algorithm used in real-time operating systems. In the case of n synchronous, independent and periodic tasks such that their deadlines are equal to their periods, the processor utilization factor $U \leq \sum_{i=1}^{n} n(2^{\frac{1}{n}} - 1)$ is a necessary and sufficient condition for the RM-based scheduling of real-time tasks [18]. In the literature, many approaches such as [19–22] have been carried out in the area of schedulability analysis for meeting real-time requirements. In [23], the authors focus on worst-case execution by making conservative assumptions about the system. The authors of [22] use a combined offline and online scheduling technique. A worst-case execution time (WCET) schedule, which provides the ideal operating frequency and voltage schedule assuming that tasks require their worst-case computation time, is calculated offline. The online scheduler further reduces frequency and voltage when tasks use less than their requested computing quota, but can still provide deadline guarantees by ensuring all invocations complete no later than in the WCET schedule. Pillai and Shin [24] propose an optimal algorithm for computing the minimal speed that can make a task set schedulable. Chetto et al. [19] consider the effect of precedence constraints between tasks on the dynamic priority scheduling problem. That paper proposes an algorithm to accept or reject aperiodic tasks with precedence constraints to guarantee the timing behavior of the rest of the system's tasks. Liu et al. developed an algorithm PASS for real-time tasks with different priorities and deadlines. PASS considers the hard real-time tasks and the soft real-time tasks at the same time. The authors of [8,9,25–29] explore the use of constraint programming to solve scheduling problems, and presents several optimizations to speed up the search for a valid solution.

In [25], the authors propose a technique to minimize the number of tasks in a real-time system while satisfying timing constraints. The approach in [8] aims both to reduce the number of preemptions for minimizing timing overheads and to maximize the laxity of tasks in order to improve the schedulability of the design model. In [9], the authors propose a method for an optimized synthesis of AUTOSAR (Automotive Open System Architecture) which are architectures based on Mixed Integer Linear Programming (MILP) and GA (Genetic Algorithm). It takes into account three optimization objectives which are extensibility maximization, latency and tasks number minimization. In [30], the authors present Integer Linear Programming (ILP) for scheduling problem with dependent tasks in a multiprocessor homogeneous system. Jeannenot proposed in [31] a set of algorithms under periodic real-time tasks in a processor with dynamic variable speed to determine the suitable speeds execution for each task and minimize the total energy consumption.

2.2 Reconfiguration of Real-Time System

Nowadays, many research works have been proposed to develop reconfigurable systems. The authors in [6] propose an approach that deals with reconfigurable systems to be implemented with different tasks under deadline constraints according to user requirements. For that purpose, the authors define an agent-based architecture to check after any reconfiguration scenario the system's feasibility that can be affected when the tasks violate corresponding deadlines. In this case, the agent provide new parameters for infeasible tasks in order to re-obtain the system's feasibility. In [32], the authors describe a concurrent function block model to control the run-time reconfiguration process of a real-time holonic controller. They describe a real-time java implementation to support the function block-based real-time task execution and the run-time reconfigurability.

The authors in [33] propose a complete methodology to dynamically reconfigure tasks. They present an interesting experimentation showing the dynamic change by users of tasks without disturbing the whole system. The authors in [34] use the Real-time-UML as a meta-model between design models of tasks and their implementation models to support dynamic user-based reconfigurations of control systems. In [35], the authors aim to provide an automated development process from modelling to implementation for the dynamic software part of reconfigurable systems. TimeAdapt [36] is a development process for reconfigurable system design. It allows to specify reconfiguration actions, estimate whether their execution can be carried out within a given time bound and execute them in a timely manner. In the same context, the authors of [37] present an approach which deals with reconfiguration at different levels within the development process of distributed applications. They propose a model driven approach to help specifying and configuring reconfigurable systems.

2.3 Code Generation

There are some related approaches which generate complete real-time systems. In [12], the authors deliver an approach called TASTE to enable the generation of a complete real-time distributed system. This approach involves four phases: (i)The system modeling phase using formal techniques, (ii) The transformation phase, (iii) The feasibility analysis phase, and (vi) The code generation phase where the authors propose a new language based on existing and mature technologies such as Simulink, SDL, ASN.1, C, and Ada [1]. Barreto et al. [38] propose a software synthesis method for automatic generation of executable code from the formal model is performed. This approach is an extension of their previous work [39] which uses pre-runtime approach in order to find feasible schedules satisfying timing and power constraints. The authors in [40] provide a framework that allows designers to automatically generate, from a functional specification with dependency constraints described by the Prelude language, a set of real-time tasks that can be executed on a uniprocessor architecture.

Nowadays, there are many programming languages designed for the development of real-time systems. Among the most used real-time languages, we cite real-time java (RT-java) [41] formalized in June 2000. RT-java aims to support the programming of real-time codes from different directions used by other software development platforms. POSIX (Portable Operating System Interface) is a standard written in terms of the C programming language [13]. POSIX allows to create POSIX threads (pthreads [1]) by calling the *pthread_create* API function with different thread scheduling policies and priorities to meet different application requirements. POSIX defines three scheduling policies that can be used to schedule real-time applications [42]:

– SCHED_FIFO: FIFO order among entities of the same priority.
– SCHED_RR: Round robin order among entities of the same priority.
– SCHED_SS: Sporadic server scheduling, useful for scheduling aperiodic tasks.

As we assume that assigning priorities to tasks is performed using rate monotonic algorithm RM, we can implement it using POSIX primitives: At first we assign priorities to tasks in the usual way for RM (i.e. $P_i = \frac{1}{T_i}$). Then we query the range of allowed system priorities with:

sched_get_priority_min()

sched_get_priority_max()

After that we map task set onto system priorities. Finally we start tasks using assigned priorities and SCHED_FIFO.

This standard facilitates the application portability that is why we adopt it as a target language to implement a reconfigurable real-time system in the current paper.

The main contributions of this paper are four-fold. The first part consists in ensuring the respect of timing properties before the effective implementation of the real-time system (i.e. at the design level). Second, we are interested in the reconfiguration of real-time systems where the addition and removal of tasks are applied at run-time. Third, we propose a multi-optimization metric. Indeed, the proposed approach aims to minimize the reconfiguration time by avoiding a redundancy between the different implementations from one side. From the other side, it aims to minimize the response times of the real-time tasks in order to maintain the performance of the system. Finally, this work automatically generates a complete reconfigurable real-time system from the specification level by using the programming language POSIX. None of the existing works is solving all the four problems together.

3 System Formalization

In this section, we present a formal description of a reconfigurable uniprocessor system. We present in addition real-time prerequisites required to introduce the paper's contribution.

It is assumed in this work that a reconfigurable real time system Sys is defined as a set of implementations: $Sys = \{imp_1, imp_2 \dots imp_m\}$. We denote by $Sys(t)$ the implementation defining the system at a particular time t (i.e. $Sys(t) = imp_i$). An implementation imp_i is composed of n tasks that we assume *independent* and *periodic* (i.e. $imp_i = \{\tau_1, \tau_2, \tau_3 \dots \tau_n\}$). Each task τ_i executes a set of applicative functions $\tau_i = \{F_1, F_2, F_3 \dots F_k\}$. A function F_i is characterized by static parameters $F_i = (T_{f_i}, C_{f_i})$ where T_{fi} is the activation period of the function F_i and C_{fi} is an estimation of its Worst Case Execution Time WCET. Note that these parameters are considered as inputs to the proposed approach and must be specified by the user. Each task τ_i is characterized by a set of real-time parameters $(r_i, T_i, C_i, D_i, P_i)$: its release time r_i, we assume that $r_i = 0$, its activation period T_i which is deducted from the activation periods of the functions implemented by this task, its capacity or worst case execution time C_i which is equal to the sum of the WCETs of the functions executed by this task, its deadline D_i we assume that $D_i = T_i$, the priority P_i, we assume that $P_i = 1/T_i$ since we adopt the Rate Monotonic (RM) priority assignment. The Fig. 1 depicts the task parameters:

Let U be the processor utilization factor defined by: $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. For timing verification, we perform in this paper Rate-Monotonic (RM) response time analysis based on the computation of an upper bound of the response time Rep_i of the different tasks constituting the task model. This analysis aims to verify whether these tasks complete their computations within the time limit specified by the real-time application i.e. the deadline ($Rep_i \leq D_i$) [18].

The reconfiguration scenario corresponds to adding/removing tasks or modifying timing parameters. Thus, we introduce the reconfiguration time T_{reconf} which refers to the time required to jump from one implementation to another according

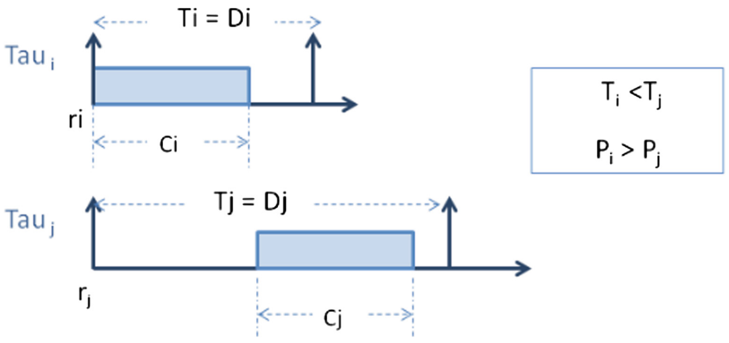


Fig. 1. Real-time task parameters.

to user requirements (i.e. reconfiguration conditions). This parameter is defined as follow:

$$T_{reconf} = A * T_{delete} + B * T_{creat}$$

where A is the number of deleted tasks, B is the number of created tasks, T_{delete} is the spent time to delete a task and T_{creat} is the spent time to create a task. We assume that the blanking time T_{delete} and creation time T_{creat} of all the tasks are equal for a considered platform (i.e. $T_{delete} = T_{creat}$). We denote by T_{cost} the spent time to create a task or to delete it (i.e. $T_{delete} = T_{creat} = T_{cost}$). Thus, the reconfiguration time is given as follow: $T_{reconf} = (A + B) * T_{cost}$.

4 Proposed Approach

In this section, we present an overview on our approach and detail the structure of different modules involved in this work.

4.1 Motivation and Definitions

We deliver an approach which automatically converts a high-level specification of a reconfigurable real-time system into an executable running on POSIX platform. The proposed approach aims to optimize the system code while meeting all related real-time constraints and avoiding any redundancy between the implementation sets. Figure 2 shows the process of the proposed approach. As entry, the designer provides the specification model which defines the reconfiguration conditions, the applicative functions that must be executed under a considered condition and the temporal parameters of each function. This model presents the input of the task generator step which aims to produce an initial task model. Then, the optimization step receives the generated model and proposes a valid and optimal task model. This model is finally converted into an executable program running under POSIX.

4.2 Task Generator

The first step consists in generating the initial task model. This stage considers the specification model as an input and aims to generate the initial task model which defines a possible implementation of the considered system. For each reconfiguration condition, this step generates an implementation and associates its appropriate functions. Then, for each generated implementation, it regroups the functions having the same period T_{fi} to be executed by one task τ_i. Since we assume that the release time $r_i = 0$ and $P_i = 1/T_i$, the task τ_i is characterized only by (T_i, C_i, D_i) where the period T_i corresponds to the period of the grouped functions, C_i is the sum of WCETs of the grouped functions and the deadline D_i of each task is equal to the corresponding period T_i.

Let us note that for the generation of this model, the optimization and real-time feasibility concerns are not considered. Algorithm 1 illustrates this

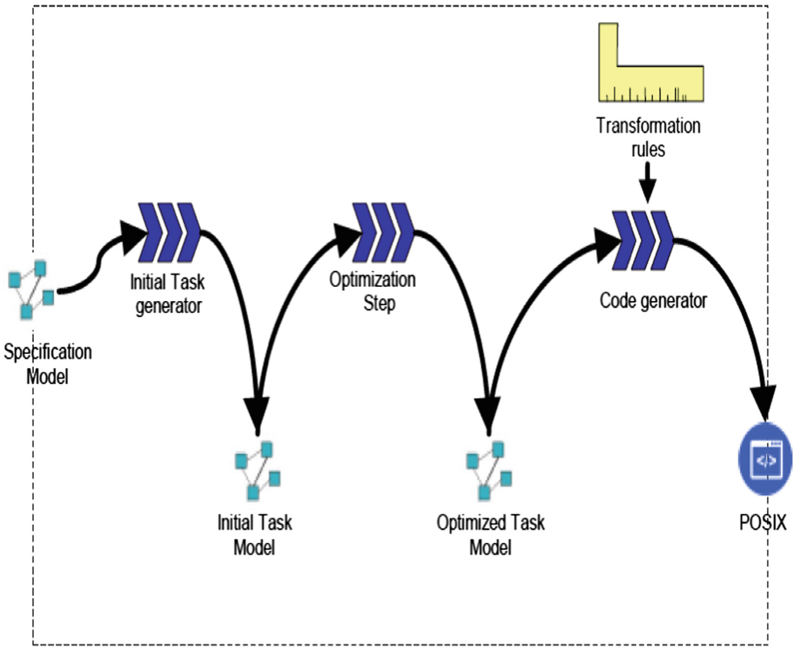


Fig. 2. Process overview.

generating step. The Initial task model can be generated with complexity $\mathcal{O}(N * M) + \mathcal{O}(P * M) = max(\mathcal{O}(N * M); \mathcal{O}(P * M))$, where N denotes the size of the conditions, M denotes the number of functions and P presents the number of implementations.

4.3 Task Model Optimization

This phase aims to produce a feasible and optimal implementation of the reconfigurable real-time system from the initial task model.

In order to avoid redundancy between the sets of implementation and reduce the number of tasks, this phase aims to merge the tasks belonging to different implementations but implementing the same functions and/or having close periods. For instance, let us consider two tasks $\tau_i \in imp_k$ and $\tau_j \in imp_l$. τ_i and τ_j are defined by a set of parameters: $\tau_i = (T_i, C_i, D_i)$ and $\tau_j = (T_j, C_j, D_j)$. These two tasks have close periods (i.e. $T_i = T_j + \delta t$) where δt is a constant defined by the user. We denote by τ_i' the task resulting from merging these two tasks which is characterized by

$$\tau_i'(T_i', C_i', D_i') = \begin{cases} T_i' = min(T_i, T_j) \\ C_i' = C_i + C_j \\ D_i' = min(D_i, D_j) \end{cases}$$

Algorithm 1. Task Generation

```
   Input:
   - F : Functions set
   - ReconfCnd : Reconfiguration condition
   Output:
   - InitTask : Initial Task Model
1  Notations:
2  - Reconf_Cnd_Func: Correlation table between the reconfiguration conditions
     and the functions.
3  - imp : Implementation set
4  $nbr\_t \leftarrow 0$
5  $k \leftarrow 0$
6  /* Generation Of Implementations */
7  for $i \leftarrow 0$ to $SizeOf(ReconfCnd)$ do
8      for $j \leftarrow 0$ to $SizeOf(F)$ do
9          if $(F[j] \in Reconf\_Cnd\_Func[i])$ then
10             $imp[i][k] = F[j]$
11         $k++;$
12 /* Generation Of Task Model */
13 for each implementation $imp_i$ do
14     for $eachfunctionF_j$ do
15         /* We create a task and we initialize its parameters with function $F_j$
             parameters */
16         $WcetOf(InitTask[nbr\_t]) = WcetOf(F_j)$
17         $PeriodOf(InitTask[nbr\_t]) = PeriodOf(F_j)$
18         $DeadlineOf(InitTask[nbr\_t]) = DeadlineOf(F_j)$
19         for $eachfunctionF_{j+1}$ do
20             /* We check if $F_j$ and $F_{j+1}$ have the same period and we evaluate if
                 the result WCET is less than the task period to ensure the system
                 feasibility */
21             if $PeriodOf(F_j) == PeriodOf(F_{j+1}))$ then
22                 if $(WcetOf(F_j) + WcetOf(F_{j+1}) <=$
                     $PeriodOf(InitTask[nbr\_t]))$ then
23                     $WcetOf(InitTask[nbr\_t]) =$
                         $WcetOf(InitTask[nbr\_t]) + WcetOf(F_{j+1})$
24         $nbr_t++;$
25 return $InitTask$
```

Where T_i' corresponds to minimum of the two periods, C_i' is equal to the sum of their WCETs and the deadline D_i' of τ_i' is equal to the corresponding period T_i'. This approach allows to merge more than two tasks by optimizing other parameters like the sum of their response times. Thus, the considered problem is a combinatorial one, and the solution depends on many parameters. In order to implement properly the problem by taking into consideration the different constraints, we propose a MILP formulation of our problem. So we should define the objective function and the required constraints for parameters and variables.

Figure 3 shows an example to illustrate the scenario of reconfiguration which correspond to the transition from imp_1 to imp_2. The reconfiguration consists in removing τ_2, τ_3,τ_4 and adding τ_5, τ_6,τ_7, thus the reconfiguration time is defined as follows:

$$T_{reconf} = 3 * T_{delete} + 3 * T_{creat} = 6T_{cost}.$$

In order is to minimize the reconfiguration time T_{reconf}, tasks having close periods (or same period T in this example) must be merged (see Fig. 3). Consequently, after merging these tasks (i.e. τ_2 with τ_6, τ_3 with τ_7) T_{reconf} becomes:

$$T_{reconf} = 2 * T_{delete} + 2 * T_{creat} = 4T_{cost}$$

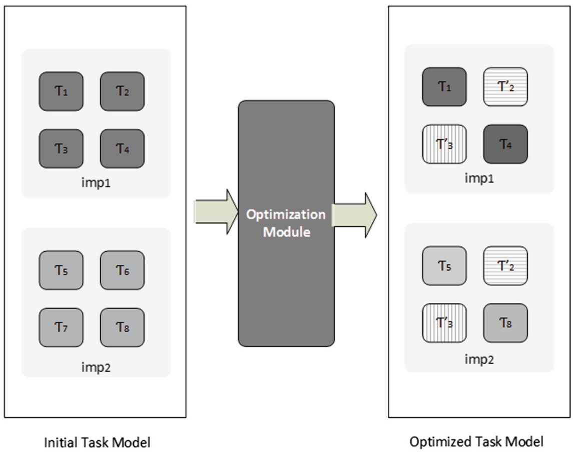


Fig. 3. Example of a reconfiguration scenario.

Definitions
Let m be the number of tasks in the initial model, let N be the number of tasks in the new task model, let s be the starting time which corresponds to effective starting time of each task. We denote by *InitTask* the initial task model which is a three column matrix where the first column presents the period T_i of task, the second one presents their WCETs C_i and the third column is their deadline D_i. *NewTask* is the resulting task model after merging the different tasks (i.e. optimized task model).

Objective Function

$$Maximize \sum_{i,j \in \{1,m\}} Merge_{ij} - \sum_{i,j \in \{1,m\}} Rep_{ij} \qquad (1)$$

This expression defines the objective function of our problem. Merge denotes a boolean variable used to mention whether two tasks τ_i and τ_j are merged. More in detail, $Merge_{ij}$ is equal to 1 if task $t_i \in imp_k$ and task $t_j \in imp_l$ are merged. The expression(1) aims to maximize the number of merged tasks and minimize the sum of response times of the different tasks constituting the task model.
In order to limit non meaningful merging situations, we define in addition the following constraints:

Merging Situation Constraints
The constraints (2) and (3) introduce the merging condition such as tow the tasks $\tau_i \in imp_k$ and $\tau_j \in imp_l$ will be merged if they have the same period.

$$\begin{gathered}\forall i,j \in \{1 \dots m\}\ et\ i \neq j,\\ if(InitTask[i,1] - InitTask[k,1]) = \delta t \quad then \quad Merge_{ij} = 1\end{gathered} \tag{2}$$

$$\begin{gathered}\forall i,j \in \{1 \dots m\}\ et\ i \neq j,\\ if(InitTask[i,1] - InitTask[k,1]) \neq \delta t \quad then \quad Merge_{ij} = 0\end{gathered} \tag{3}$$

The constraint (4) means that we have to maximize the number of merged tasks and thus minimize the number of tasks used in the task model. Indeed, this equation serves as a bound for the objective function (i.e. the number of merging operations).

$$N = m - (\sum_{i,j \in \{1 \dots m\}} Merge_{ij})/2 \tag{4}$$

Real-Time Constraints
NewTask is a three column matrix where the first column presents the periods of the new tasks computed by the constraint (5). The second column presents the WCETs of the tasks computed by the constraint (6) and the last column is the deadline presented by the constraint (7)

$$\forall k \in \{1 \dots N\}, \forall i,j \in \{1 \dots N\} : NewTask[k,1] = min(InitTask[i,1], InitTask[j,1]) \tag{5}$$

$$\begin{gathered}NewTask[i,2] = (InitTask[i,2] + InitTask[j,2])Merge[i,j]+\\ (1 - Merge[i,j])InitTask[i,2]\end{gathered} \tag{6}$$

$$\forall i \in \{1 \dots N\}, NewTask[i,3] = NewTask[i,1] \tag{7}$$

The constraint (8) verifies whether the new model meets the timing constraints.

$$U = \sum_{i=1}^{N} \frac{NewTask[i,2]}{NewTask[i,1]} \leq \sum_{i=1}^{N} N(2^{\frac{1}{N}} - 1) \tag{8}$$

Constraint (9) ensures that the response times Rep_i of the different tasks in the optimized model are lower or equal than their deadlines:

$$\forall i \in \{1 \dots N\} Rep_i \leq NewTask[i,3] \tag{9}$$

Constraint (10) gives the computation formula of the response time Rep_i of task τ_i:

$$Rep_i = s[i] + NewTask[i, 2] \tag{10}$$

The response time Rep_i of a task τ_i is defined as the sum of its start time and its execution time.

$$\forall i \in \{1 \ldots N\} s[i] - s[j] >= NewTask[j, 2] \tag{11}$$

$$\forall i \in \{1 \ldots N\} s[j] - s[i] >= NewTask[i, 2] \tag{12}$$

To ensure a single executed task at any time, we should have either $s[i] - s[j] - NewTask[j, 2] >= 0$ or $s[j] - s[i] - NewTask[i, 2] >= 0$, for every pair of tasks t_i and t_j.

$$\forall i \in \{1 \ldots N\} s[i] <= r[i] \tag{13}$$

By respecting these constraints, the objective function will seek for the best way to merge tasks, so as to reduce the reconfiguration time while ensuring the respect of timing properties. The task model generated by the linear program will be interpreted by the code generator in order to generate a running program in POSIX.

4.4 Code Generator

The last step of our approach consists in building the executable application from the optimized task model. We generate a POSIX code on the basis of transformation rules. For each task in the optimized task model, the code generator implements a POSIX thread by using pthread. In addition, this step produces the controller code of the reconfigurable real-time system, which allow moving from implementation to another, following well-defined conditions (i.e. user requirements).

5 Case Study

In this section, we illustrate the proposed approach through a case study. The considered case study consists in a Car Collision Avoidance System (CCAS) [43]. Firstly, we present the CCAS specification. Then we apply the proposed approach using the suite of tools associated to an automatic construction of a feasible and optimal implementation of a reconfigurable real-time system.

5.1 CCAS Presentation

The Car Collision Avoidance System (CCAS) detects obstacles in front of the vehicle to which it is mounted and, if an imminent collision is detected, applies the brakes to slow the vehicle. To show the applicability of our approach, we consider in this paper a simplified version of this system. For clarity, several features of the system (CCAS) were omitted. Therefore, we only define two modes of operation:

(i) *Default mode:* represents a traditional use of CCAS,
(ii) *Economic mode:* represents a restrictive use of CCAS with safety requirements.

In the case where the economic mode must be enabled, the system jumps from the default mode to the secure one.

Default Mode: This mode is defined by five functions:

(i) F_1 (*ReadImage*): reads images from the input to the system from a radar,
(ii) F_2 (*Discrete Cosine Transformation: DCT*) moves the representation of the image from the spatial domain into the frequency domain
(iii) F_3 (*Quantization*): data in the frequency domain is selectively discarded to compress the image
(iv) F_4 (*InverseDCT*): moves the image back into the spatial domain
(v) F_5 (*Display*): displays the images for monitoring

The Figs. 4 and 5 present the overview of the CCAS system.

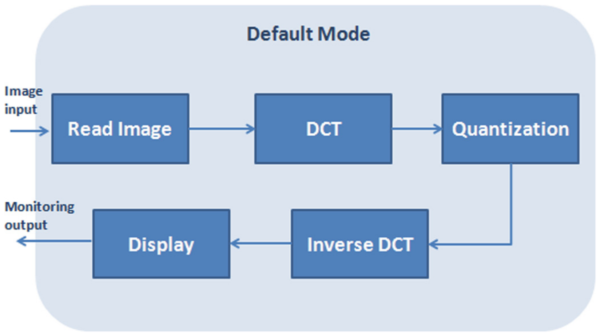


Fig. 4. CCAS overview in default mode.

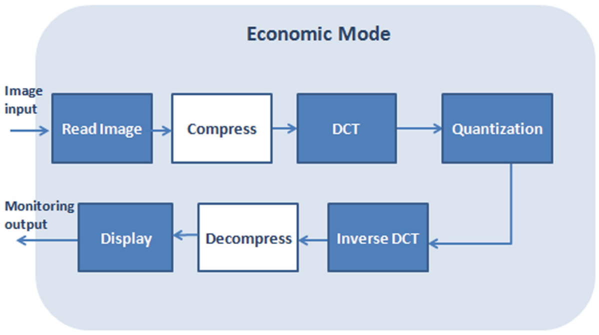


Fig. 5. CCAS overview in economic mode.

Table 1. CCAS specification in economic mode.

F_i	T_f	C_f
F_1	5 ms	1 ms
F_1'	5 ms	1 ms
F_2	5 ms	1 ms
F_3	15 ms	0.5 ms
F_4	15 ms	0.25 ms
F_4'	15 ms	0.25 ms
F_5	20 ms	2 ms

Table 2. CCAS specification in default mode.

F_i	T_f	C_f
F_1	5 ms	1 ms
F_2	5 ms	1 ms
F_3	15 ms	0.5 ms
F_4	15 ms	0.5 ms
F_5	20 ms	2 ms

Economic Mode: The economic Mode is defined by seven functions. Compared with the default mode, we have added two function F_1' to compress the received image and F_4' to decompress it. Tables 1 and 2 give a tabular presentation of the specification model describing the different functions of the CCAS system.

5.2 CCAS Initial Task Model

The second step consists in generating the implementations and their tasks from the specification model by applying Algorithm 1. Tables 3 and 4 give a tabular description of the initial task model describing the CCAS. This model shows two possible implementations of the CCAS which refer respectively to the two execution modes already specified. Thus, we denote by *CCASsys* the reconfigurable real-time system of the Car Collision Avoidance which defines two implementations:

$$CCASsys = \{DefaultMode, EconomicMode\}$$

The first implementation executes three tasks $DefaultMode = \{\tau_1, \tau_2, \tau_3\}$ and the second executes also three tasks $EconomicMode = \{\tau_4, \tau_5, \tau_6\}$. Each task is defined by the specific real-time parameters and implements the set of applicative functions having the same period.

Table 3. Tabular description of the initial task model of the CCAS in default mode.

Task	T_i (ms)	C_i (ms)	D_i (ms)	F_i
τ_1	5	3	5	$\{F_1, F_2\}$
τ_2	15	1	15	$\{F_3,F_4\}$
τ_3	20	2	20	$\{F_5\}$

Table 4. Tabular description of the initial task model of the CCAS in economic mode.

Task	T_i (ms)	C_i (ms)	D_i (ms)	F_i
τ_4	5	3	5	$\{F_1, F_1', F_2\}$
τ_5	15	1	15	$\{F_3,F_4, F_4'\}$
τ_6	20	2	20	$\{F_5\}$

5.3 CCAS Optimized Task Model

The third step corresponds to the generation of the optimized task model from the initial one. The objective of this step is to optimize the initial task model by minimizing the redundancy, the number of tasks and the response times of the different tasks. The merging matrix given by the task model optimization phase is given as follow:

$$Merge = \begin{pmatrix} 0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \end{pmatrix}$$

This matrix shows that the solution considered by the solver is the merge of τ_2 and τ_5 and the merge of τ_3 and τ_6. We note that tasks τ_1 and τ_4 are not merged by the solver even they have the same period because due to feasibility concerns (i.e. if the solver decide to merge τ_1 and τ_4, the resulting task will not meet its deadline). Tabular descriptions of task models generated by this phase are given in Tables 5 and 6.

Table 5. Tabular description of the optimized task model of the CCAS in default mode.

Task	T_i (ms)	C_i (ms)	D_i (ms)	Rep_i (ms)	Function
τ_1	5	3	5	7	F_1,F_2
τ_2'	15	2	15	10.7	F_3,F_4,F_4'
τ_3'	20	4	20	5.2	F_5

Table 6. Tabular description of the optimized task model of the CCAS in economic mode.

Task	T_i (ms)	C_i (ms)	D_i (ms)	Rep_i (ms)	Function
τ'_1	5	3	5	8	F_1,F'_1,F_2
τ'_2	15	2	15	10.8	F_3,F_4,F'_4
τ'_3	20	4	20	5.3	F_5

After optimisation, the CCAS system consists also in two implementations but implementing different functions: *DefaultMode* $= \{\tau_1, \tau'_2, \tau'_3\}$ and *Economic-Mode* $= \{\tau'_1, \tau'_2, \tau'_3\}$. We note that the number of tasks implementing the CCAS after optimization is only 4 tasks compared to 6 tasks in the initial task model. In addition, as we can see from Tables 5 and 6, the response times Rep_i of the different tasks are lower than their deadlines D_i and thus the timing constraints of the CCAS system are met. Figures 6 and 7 present the execution graphs of the tasks in the default mode and the economic mode of the CCAS system given by cheddar simulator [4]. These figures confirm that the system is feasible since all the tasks meet the related deadlines.

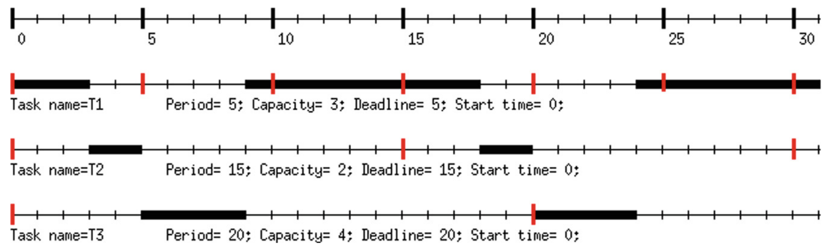


Fig. 6. Execution graph of CCAS tasks in default mode.

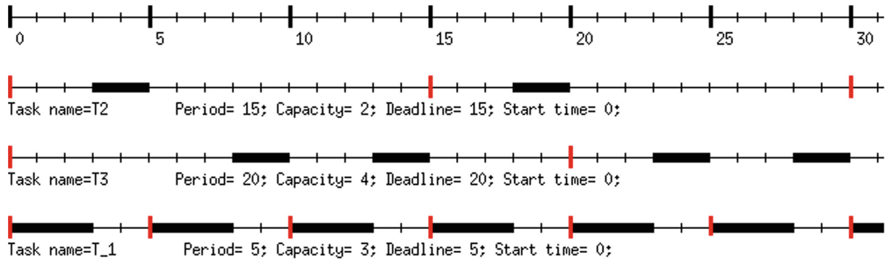
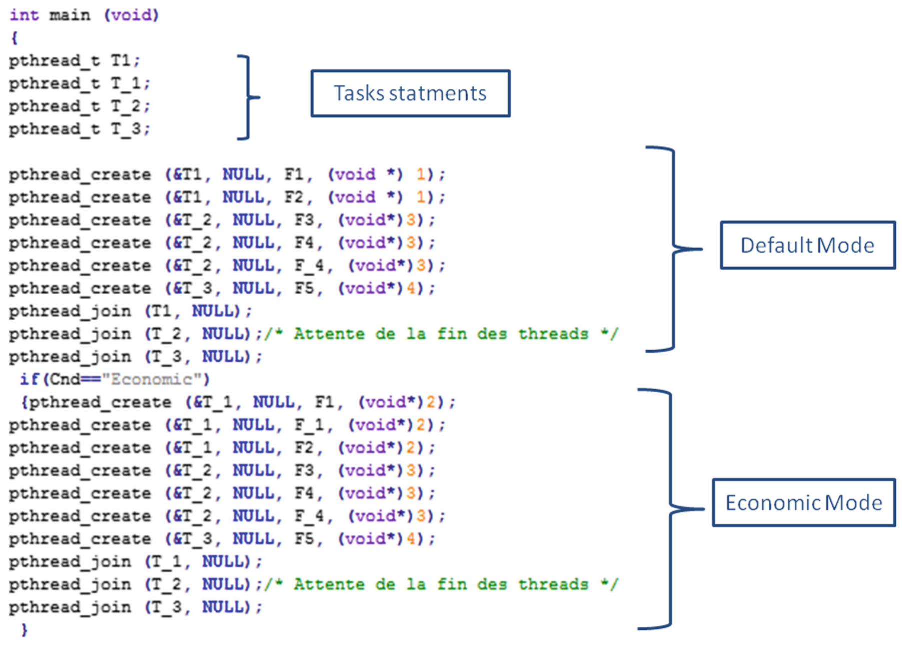


Fig. 7. Execution graph of CCAS tasks in economic mode.



```
int main (void)
{
pthread_t T1;
pthread_t T_1;
pthread_t T_2;
pthread_t T_3;

pthread_create (&T1, NULL, F1, (void *) 1);
pthread_create (&T1, NULL, F2, (void *) 1);
pthread_create (&T_2, NULL, F3, (void*)3);
pthread_create (&T_2, NULL, F4, (void*)3);
pthread_create (&T_2, NULL, F_4, (void*)3);
pthread_create (&T_3, NULL, F5, (void*)4);
pthread_join (T1, NULL);
pthread_join (T_2, NULL);/* Attente de la fin des threads */
pthread_join (T_3, NULL);
 if(Cnd=="Economic")
 {pthread_create (&T_1, NULL, F1, (void*)2);
pthread_create (&T_1, NULL, F_1, (void*)2);
pthread_create (&T_1, NULL, F2, (void*)2);
pthread_create (&T_2, NULL, F3, (void*)3);
pthread_create (&T_2, NULL, F4, (void*)3);
pthread_create (&T_2, NULL, F_4, (void*)3);
pthread_create (&T_3, NULL, F5, (void*)4);
pthread_join (T_1, NULL);
pthread_join (T_2, NULL);/* Attente de la fin des threads */
pthread_join (T_3, NULL);
 }
```

Fig. 8. Excerpt of the controller implementation.

5.4 Code Generation

The last step in our approach is the code generator. We generate a POSIX code from the optimized task model describing the CCAS. Figure 8 presents an excerpt of the controller POSIX code. The controller's role is to switch from one implementation to another under a considered condition. As shown in Fig. 8 if the variable "CND" is equal to "Economic" then the control executes the *EconomicMode* implementation else it executes the *DefaultMode*.

5.5 Performance Evaluation

The experiments are carried-out on Intel Core i5-4200U processor running at 1.6 GHz with 6 GB of cache memory. CPLEX is used as a MILP solver for the whole set of experiments. We evaluate the proposed approach by considering the CCAS system $CCASsys$ previously defined.

We denote by $T_{reconf_{initial}}$ the reconfiguration time in the initial task model of the CCAS and $T_{reconf_{Current}}$ the reconfiguration time in the optimized task one. These parameters are given as follow:

$$T_{reconf_{initial}} = 5 * T_{delete} + 6 * T_{creat} = 11 * T_{cost}$$

$$T_{reconf_{Current}} = 1 * T_{delete} + 2 * T_{creat} = 3 * T_{cost}$$

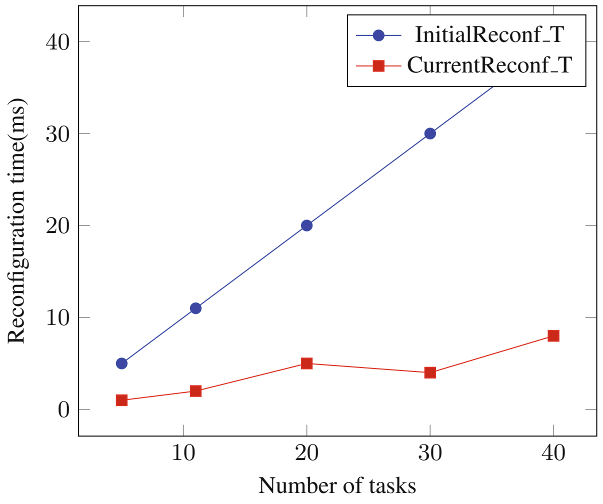


Fig. 9. Comparison between current reconfiguration time and initial one.

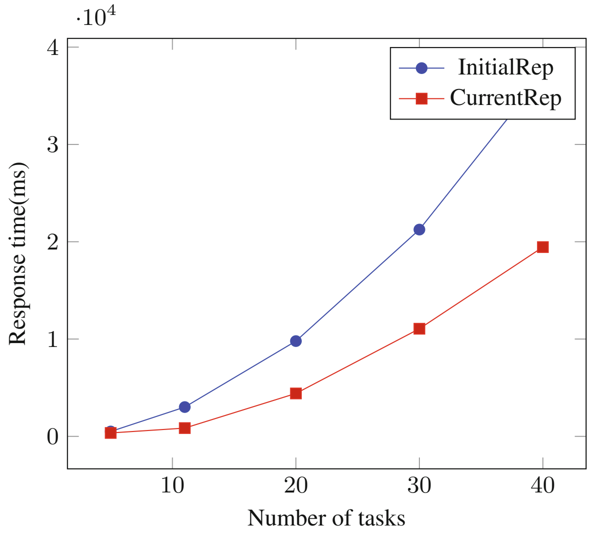


Fig. 10. Comparison between current response time and initial one.

We note that the proposed approach allows to reduce the reconfiguration time and thus improves the overall performance of the reconfigurable real-time system (CCAS). It minimizes also the sum of the response times of the considered tasks such as $Rep_{initial} = 3010$ ms and after optimization it becomes $Rep_{optimized} =$ 1920 ms.

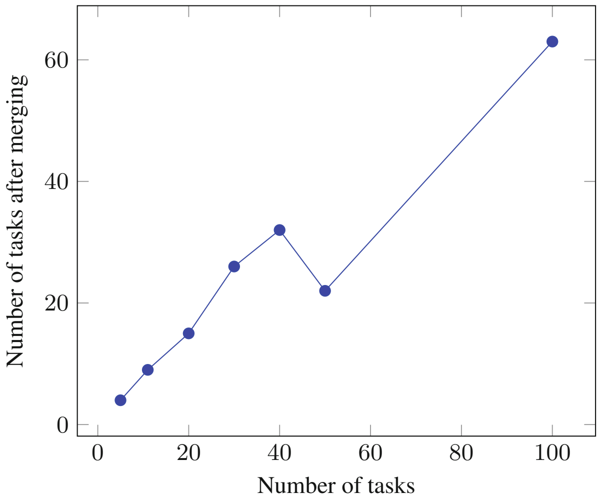


Fig. 11. Results of task merging.

In addition, we have randomly generated instances with 5 to 40 tasks. We compare the reconfiguration time and the sum of response times after optimization with the initial corresponding parameters. The numerical results are depicted in Figs. 9 and 10.

These figures show that the optimization of the reconfiguration and response times are clearer and more important for large scale reconfigurable real-time systems. In fact, when the number of tasks is more important, the optimisation phase will seek for solutions to merge tasks having the same periods while minimizing the response times of the different tasks. Such optimisation, reduce the reconfiguration time and guarantee the system feasibility. Figure 11 shows the number of tasks obtained after merging compared to the initial number of task. We compute average results by executing several times the linear program on randomly generated task sets. We observe from this figure that we are able to merge many tasks. Thus, we minimize the additional time overhead.

6 Conclusion

The contribution presented in this paper consists in a methodology that supports the development of reconfigurable real-time systems. By defining the specification such as reconfigurable conditions, functions and temporal constraints the approach generates as first step an initial task model. Then, this model will be optimized using MILP techniques to produce an optimized task model. Finally, our approach generates a POSIX-based code which describes the reconfigurable real-time system. We have evaluated the performance of the three-step approach. The numerical results show that the integer programming model allows to minimize the reconfiguration time and response times. As a future work, we aim to extend our approach by considering multiprocessor systems and other optimization metrics. So that we expect to evaluate scalability of the proposed method with an industrial example.

References

1. Burns, A., Wellings, A.: Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX, 4th edn. Addison-Wesley Educational Publishers Inc., USA (2009)
2. Cottet, F., Grolleau, E.: Systmes Temps Réel de Contrôle-Commande. Dunod, Paris (2005)
3. Polakovic, J., Mazare, S., Stefani, J.-B., David, P.-C.: Experience with safe dynamic reconfigurations in component-based embedded systems. In: Schmidt, H.W., Crnkovic, I., Heineman, G.T., Stafford, J.A. (eds.) CBSE 2007. LNCS, vol. 4608, pp. 242–257. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73551-9_17
4. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. In: Proceedings of the ACM SIGADA International Conference, Atlanta. ACM (2004)
5. Baruah, S., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. Handb. Sched.: Algorithms Models Perform. Anal. **3** (2004)
6. Gharsellaoui, H., Gharbi, A., Khalgui, M., Ahmed, S.: Feasible automatic reconfigurations of real-time OS tasks. In: Handbook of Research on Industrial Informatics and Manufacturing Intelligence: Innovations and Solutions: Innovations and Solutions (2012)
7. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM (JACM) **20**, 46–61 (1973)
8. Bouaziz, R., Lemarchand, L., Singhoff, F., Zalila, B., Jmaiel, M.: Architecture exploration of real-time systems based on multi-objective optimization. In: Proceedings of the 20th International Conference on Engineering of Complex Computer Systems (ICECCS), Gold Coast, QLD, pp. 1–10. IEEE (2015)
9. Mehiaoui, A., Wozniak, E., Tucci-Piergiovanni, S., Mraidha, C., Natale, M.D., Zeng, H., Babau, J., Lemarchand, L., Gerard, S.: A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. ACM SIGPLAN Not. **48**, 121–132 (2013)
10. Woźniak, E.: Model-based synthesis of distributed real-time automotive architectures. Ph.D. thesis, Université Paris Sud-Paris XI
11. Marinca, D., Minet, P., George, L.: Analysis of deadline assignment methods in distributed real-time systems. Comput. Commun. **27**, 1412–1423 (2004)
12. Pillai, P., Shin, K.: Taste-an open-source tool-chain for embedded system and software development. In: Proceedings of the Embedded Real Time Software and Systems Conference (ERTS), Toulouse, France (2012)
13. Lewine, D.: POSIX programmers guide. O'Reilly Media Inc., USA (1991)
14. Obenland, K.M.: The Use of Posix in Real-time Systems, Assessing its Effectiveness and Performance. The MITRE Corporation, McLean (2000)
15. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the Real Time Systems Symposium, pp. 166–171. IEEE (1989)
16. Ltkebohle, I.: IBM CPLEX Optimizer - United States (2016). http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. Accessed 10 Apr 2016

17. Stankovic, J.: Misconceptions about real-time computing: a serious problem for next-generation systems. Computer **21**, 10–19 (1988)
18. Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G.: Analyzing complex systems. In: Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G. (eds.) Proceedings of Real-Time Analysis, pp. 535–578. Springer, US (1993)
19. Chetto, H., Silly, M., Bouchentouf, T.: Dynamic scheduling of real-time tasks under precedence constraints. Real-Time Syst. **2**, 181–194 (1991)
20. Swaminathan, V., Chakrabarty, K.: Real-time task scheduling for energy-aware embedded systems. J. Franklin Inst. **338**, 729–750 (2001)
21. Gruian, F.: Hard real-time scheduling for low-energy using stochastic data and DVS processors. In: Proceedings of the 2001 international symposium on Low power electronics and design, pp. 46–51. ACM (2001)
22. Krishna, C.M., Lee, Y.H.: Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. IEEE Trans. Comput. **52**, 1586–1593 (2003)
23. Bini, E., Buttazzo, G.: A hyperbolic bound for the rate monotonic algorithm. In: Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft, pp. 59–66. IEEE (2001)
24. Pillai, P., Shin, K.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of the 13th Euromicro Conference on Real-Time Systems, USA, pp. 59–66. ACM (2001)
25. Bertout, A., Forget, J., Olejnik, R.: Minimizing a real-time task set through task clustering. In: Proceedings of the 22nd International Conference on Real-Time Networks and Systems, p. 23, Versailles, France. ACM (2014)
26. Racu, R., Jersak, M., Ernst, R.: Applying sensitivity analysis in real-time distributed systems. In: 11th IEEE Real Time and Embedded Technology and Applications Symposium, pp. 160–169. IEEE (2005)
27. Pop, T., Eles, P., Peng, Z.: Design optimization of mixed time/event-triggered distributed embedded systems. In: Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 83–89. ACM (2003)
28. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: a systematic literature review. IEEE Trans. Software Eng. **39**, 658–683 (2013)
29. Mraidha, C., Tucci-Piergiovanni, S., Gerard, S.: Optimum: a marte-based methodology for schedulability analysis at early design stages. ACM SIGSOFT Softw. Eng. Notes **36**, 1–8 (2011)
30. Hladik, P.-E., Cambazard, H., Déplanche, A.M., Jussien, N.: Solving a real-time allocation problem with constraint programming. Comput. Ind. Eng. **81**, 132–149 (2008)
31. Harbour, M.G.: Ordonnancement temps reel avec profilsvariables de consommation d'energie. In: Embedded Systems (2004)
32. Xu, Y., Brennan, R.W., Zhang, X., Norrie, H.: A reconfigurable concurrent function block model and its implementation in real-time java. Discret. Event Dynamic Syst. **9**, 263–279 (2002)
33. Rooker, M.N., Sünder, C., Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G.: Zero downtime reconfiguration of distributed automation systems: the ϵCEDAC approach. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS, vol. 4659, pp. 326–337. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74481-8_31

34. Thramboulidis, K., Doukas, G., Frantzis, A.: Towards an implementation model for FB-based reconfigurable distributed control applications. In: IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 193–200. IEEE (2007)
35. Krichen, F., Hamid, B., Zalila, B., Coulette, B.: Designing dynamic reconfiguration for distributed real time embedded systems. In: Proceedings of 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE), Tozeur, Tunisia, pp. 249–254. IEEE (2010)
36. Guo, Y., Sierszecki, K., Angelov, C.A.: A reconfiguration mechanism for resource-constrained embedded systems, pp. 1315–1320. IEEE Computer Society, Washington, DC, USA (2008)
37. B. Hamid, A. Lanusse, A.R., Gérard, S.: Designing reconfigurable component systems with a model based approach. In: ARTIST Workshop on Adaptive and Reconfigurable Embedded Systems, Saint Louis, MO, USA, pp. 69–73 (2008)
38. Barreto, R., Neves, M., Oliveira Jr., M., Maciel, P., Tavares, E., Lima, R.: A formal software synthesis approach for embedded hard real-time systems. In: Proceedings of the 17th symposium on Integrated circuits and system design, pp. 163–168. ACM (2004)
39. Tavares, E., Barreto, R., Junior, M.O., Maciel, P., Neves, M., Lima, R.: An approach for pre-runtime scheduling in embedded hard real-time systems with power constraints. In: 16th Symposium on Computer Architecture and High Performance Computing, 2004, SBAC-PAD 2004, pp. 188–195. IEEE (2004)
40. Pagetti, C., Forget, J., Boniol, F., Cordovilla, M., Lesens, D.: Multi-task implementation of multi-periodic synchronous programs. Discret. Event Dyn. Syst. **21**, 307–338 (2011)
41. Binder, W., Hulaas, J.: Using bytecode instruction counting as portable cpu consumption metric. Electron. Notes Theoret. Comput. Sci. **153**, 57–77 (2006)
42. Harbour, M.G.: Real-time posix: an overview. In: VVConex 93 International Conference, Moscu. Citeseer (1993)
43. Brosse, E.: Marte-designer example-ccas - marte user manual (english) - modelio community forge (2011). https://forge.modelio.org/projects/marte-user-manual-english/wiki/marte-designer_example-CCAS. Accessed 1 Nov 2016