Real-Time Systems manuscript No.
(will be inserted by the editor)

Multi-Objective Design Exploration Approach for Ravenscar Real-time Systems

Rahma Bouaziz · Laurent Lemarchand · Frank Singhoff · Bechir Zalila · Mohamed Jmaiel

Received: date / Accepted: date

Abstract This article deals with the design exploration and verification of real-time critical systems. Assigning the functions to the tasks of the target real-time operating system is a part of the design process. Finding a suitable design involves many important design decisions that have a strong impact on the system quality criteria.

However, with the increasing complexity and scale of today's systems and the large number of possible design solutions, making design decisions while balancing conflicting quality criteria becomes error-prone and unmanageable for designers.

We propose an automated method using a multi-objective evolutionary algorithm guided by an architectural clustering technique. This method allows designers to search the design space for schedulable solutions with respect to multiple competing performance criteria. To assess our method, several evaluations were performed. One of them shows that we were able to produce the exact optimal solution sets for 55% of the studied problem instances.

Keywords Real-time critical systems · Design exploration · Multi-Objective Optimization (MOO) · Evolutionary Algorithms (EAs) · Scheduling Analysis · Shared Resources

Rahma Bouaziz · Bechir Zalila · Mohamed Jmaiel
ReDCAD Laboratory, University of Sfax, ENIS, B.P. 1173, 3038 Sfax, Tunisia
E-mail: {rahma.bouaziz,bechir.zalila,mohamed.jmaiel}@redcad.org

Laurent Lemarchand · Frank Singhoff
Lab-STICC UMR 6285, UBO, 20 av Le Gorgeu, CS 93837, 29238 Brest Cedex 3, France
E-mail: {laurent.lemarchand,frank.singhoff}@univ-brest.fr

Mohamed Jmaiel
Digital Research Center of Sfax, B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia

1 Introduction

This article is about the design exploration of real-time critical systems. Such systems must meet hard timing constraints implied by their environment. Failures including the violation of timing constraints could lead to severe material damages or even human life losses (Stankovic 1988).

Real-time critical systems are frequently designed according to multi-tasked architectures. Such tasks are usually interacting with each other. In a real-time context, the communication and the synchronization of tasks must be carefully handled in order to preserve the predictability of the system.

Software engineers are expected to provide the appropriate association of the functions of the system to the hardware resources that will run these functions. Particularly, they have to assign the functions to the tasks of the real-time operating system (RTOS), while ensuring the non-violation of the system constraints and its correct behavior. The verification of the timing constraints (referred to as schedulability analysis) and the communication and synchronization requirements are then performed during such design process.

1.1 Problem Statement

Despite significant advances in the development of such systems, their design remains a challenging task. When designing software architectures for real-time critical systems, designers have to make good decisions regarding to several competing performance criteria: improving one criterion may lead to the degradation of another. In this article, we investigate performance criteria stemmed from the scheduling field, such as preemptions, context switches, laxities of tasks, blocking times, etc.

Indeed, assigning the functional specifications to software architecture is a non-trivial task due to the number of functions involved and the important number of possible assignment solutions, ranging from single-task architectures to concurrent multi-tasked architectures. Given a set of functions, the functions-to-tasks assignment solution space size is defined by the *Bell number* (Rota 1964) which is exponential with regard to the number of functions. Therefore, an exhaustive and exact search among all possibilities is not practicable.

Design decisions have a strong influence on the performance of the final system. Let consider the two extreme functions-to-tasks assignment strategies. The first extreme solution is the single-task implementation solution in which the whole functional specification is executed in the context of a single task. Although this solution is well suited to highly memory-constrained applications, it leads to a less flexible design which is expensive to change when the functions of the system evolve. The other extreme is the one-to-one assignment of functions to tasks, i.e. each function is assigned to a single task. Unlike the first extreme solution, this second approach results in a more flexible design since the latter provides more overall system laxity allowing easier modifica-

tions of the current design or its extension by additional potential functions. Laxity represents a measure of the available flexibility for scheduling a task as it is defined by the amount of time between the completion of the task execution and its deadline. However, this second solution leads to a significantly higher number of tasks. This increases memory consumption (e.g. stack size for each task) in addition to the excessive scheduler overhead due to context switching or preemptions (Bartolini et al 2005).

Then, software architects should explore several architecture alternatives in order to select those that meet at best the trade-off between performance criteria. However, finding suitable architecture solutions manually is time-consuming and can be error-prone. The factors that hinder designers from handling efficiently the exploration and the decision making processes are an ever-increasing complexity and size of current systems, large number of possible design options, strict design requirements and conflicting performance criteria.

The problem we deal with in this article can be summarized as follows: given a functional specification, how to assign functions to tasks so as to produce the best design alternatives with regards to a set of conflicting performance criteria? Assigning functions to tasks is non-trivial: (i) it usually raises combinatorial complexity issues, (ii) its influence on the performance of the resulting application is difficult to predict, (iii) it has an impact on the system schedulability which requires the verification of the feasibility of the design alternatives. These problems motivate the automation of the design exploration that would help not only to make better decisions, but also to reduce the time of the design process.

1.2 Contributions of this Article

The design exploration problem described above is typically a Multi-Objective Optimization (MOO) problem. Considering the combinatorial complexity issue of this problem, we use Multi-Objective Evolutionary Algorithms (MOEAs) (Deb 2001). MOEAs are metaheuristics that allow designers to find sub-optimal (or near-optimal) solutions (called *Pareto* set) in a reasonable time when exact methods fail to handle large scale problems due to computing resource requirements.

We propose a method allowing designers to explore schedulable and sub-optimal solutions in the functions-to-tasks assignment search space according to a set of competing performance criteria. The proposed method is based on the *Pareto Archived Evolution Strategy (PAES)* (Knowles and Corne 2000) MOEA.

In this article, we consider a uniprocessor system with a preemptive scheduler. The architecture exploration method is applied on systems that consist of a set of concurrent tasks interacting through shared memories protected by semaphores. We rely on a conventional task model based on Liu and Layland model (Liu and Layland 1973). We assume periodic synchronous tasks

with implicit deadlines. We target real-time Ravenscar (Burns 1999) compliant systems, i.e. the shared resource accesses are governed by the priority ceiling protocol (PCP) (Sha et al 1990) in order to ensure the synchronization of tasks and their mutual exclusion. The computation of the performance criteria adopted in this work as well as the schedulability analysis of explored assignment solutions are performed using the Cheddar scheduling analysis tool (Singhoff et al 2004).

The method we propose in this article relies on the following contributions: (i) specify and formulate the PAES components for the addressed problem, (ii) define rules that, from a functional specification and a candidate assignment solution, determine the associated architecture in terms of tasks and shared resources (including the computation of the timing parameters). The design space composed of the possible functions-to-tasks assignment solutions is explored automatically using the PAES algorithm. Design alternatives are evaluated and compared with regards to a set of competing performance criteria in order to generate those that meet at best the criteria and fulfil timing constraints.

Several performance criteria, referred to as objectives, could be involved to drive the design exploration. The correlation between these performance criteria is not intuitive to identify. That's why we propose, in the scope of this article, an empirical study of the correlation between objectives performed on synthetically generated problem instances. The results of this study show that the correlation between pairs of objectives depends on the problem instance and cannot be decided in an absolute way.

Furthermore, two evaluations are conducted in order to assess our proposals. Results of these evaluations show that our method is able to (1) converge or produce solution sets close to the exact Pareto fronts computed using an exhaustive method, for small size problem instances, (2) produce promising Pareto fronts that are assessed thanks to a conventional quality indicator for MOEAs, for larger problem instances with different resources contention levels.

The remainder of the article is organized as follows. Section 2 introduces the system model, the scheduling analysis and the multi-objective optimization technique we are dealing with. Section 3 presents the proposed method. An overview of the implemented prototype for this work is given in Section 4. Section 5 details the experiments we made to assess our proposals and their results. Section 6 discusses the related work and section 7 concludes the article.

2 Background and System Models

This section presents the basic concepts about the system model and the scheduling analysis as well as the notations adopted throughout the article. Then, it introduces the multi-objective optimization and the PAES algorithm as the MOO technique used in our design exploration method.

2.1 Functional and Architectural Models

At the design stage of real-time critical systems, we distinguish in this article two specification levels, namely, the functional and the architectural specification.

2.1.1 Functional Specification

We define a functional specification of a real-time application by $FS = \{\Gamma, \mathcal{R}\}$ where Γ is a set of *functions* and $\mathcal{R}$ a set of software resources that can be shared between several functions (see Figure 1 left).

$\Gamma = \{F_1, F_2, ..., F_n\}$ designates n functions. Each function is an elementary sequantial program in the system specification.

In this article, we assume periodic functions. Each function F_i is characterized by three parameters $F_i = (\gamma_i, \zeta_i, \delta_i)$ where γ_i is the maximum computation time, the activation period denoted as ζ_i is a fixed delay between two release times and δ_i is the deadline defined by the time limit in which the function must complete its execution. Implicit deadlines model is adopted here, so δ_i is assumed to be equal to ζ_i.

$\mathcal{R} = \{R_1, R_2, ..., R_m\}$ represents m software resources. A resource is any software structure that can be used by a function to advance its execution or to realize asynchronous interactions with other functions. Typically, it could be I/O ports, a file, message buffers, a data such as a set of variables, a piece of program, or other shared data structures. These resources may be used by several functions. Only one function is allowed to perform any action on a resource at a given time to ensure data consistency. The k^{th} usage of a resource R_j by a function F_i is noted ω_k and it is parametrized by an earliest date $(\omega_k)_{begin}$ and a latest date $(\omega_k)_{end}$ respectively for the acquisition and release of the resource R_j by the function F_i: $\omega_k((\omega_k)_{Resource}, (\omega_k)_{Function}, (\omega_k)_{begin}, (\omega_k)_{end})$.

The parameters $(\omega_k)_{begin}$ and $(\omega_k)_{end}$ are relative to the capacity of the corresponding function.

2.1.2 Architectural Formalization

At the *architectural level*, software designers define the concrete concurrency model on which the functional abstractions must be assigned to a set of periodic tasks that will be run on the top of a RTOS (as shown in Figure 1). In this article, this design model is also the *analysis model* that we use for the schedulability analysis. We consider dependencies between tasks through software shared resources.

We assume the architectural model consists of a set of k periodic tasks denoted by $S = \{\tau_1, \tau_2, ..., \tau_k\}$ running on a uniprocessor platform. A task is defined by three parameters τ_i=(C_i, T_i, D_i): its capacity or worst case execution time C_i, its period T_i and its deadline D_i. We assume that all tasks are synchronous (i.e. all tasks are released at the same time) and have implicit deadlines.

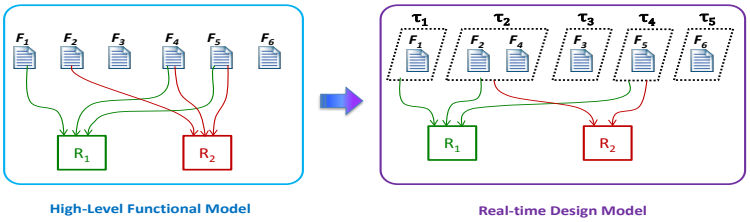


Fig. 1: Real-time Design Model

A task running a given function accesses to the resources handled by such a function. If a resource of a function may be accessed by several competing tasks, mutual exclusion has to be enforced. At the architecture level, each resource is characterized by a set of critical sections. A critical section specifies when a task τ_i locks and then unlocks a resource R_j.

We note CS_k the k^{th} critical section of a resource R_j. It is defined for the task that uses the resource on this critical section $(CS_k)_{Task}$ as well as instant of begin and end of the critical section $(CS_k)_{begin}$ and $(CS_k)_{end}$, respectively:

$$CS_k \begin{cases} (CS_k)_{Task} \\ (CS_k)_{begin} \\ (CS_k)_{end} \end{cases}$$

The parameters $(CS_k)_{begin}$ and $(CS_k)_{end}$ are relative to the capacity of the corresponding task. We should note that several critical sections can be defined for a given task on a given resource.

In order to enforce mutual exclusion on shared resources, most of modern RTOSs provide synchronization primitives such as mutexes, semaphores or monitors, under which tasks have to wait for the access to already locked resources.

At the design level, shared resources that need synchronization primitives depend on the assignment of functions to tasks. Indeed, if all functions that use a resource R_r are assigned to the same task, then the resource R_r does not need to be accessed through synchronization primitives as the functions implemented in the same task are executed sequentially. In this case, contention to the resource R_r does not need to be taken into account during schedulability analysis.

Task parameters as well as their corresponding critical sections are computed according to the parameters stemmed from the functional specification: $F_j(\gamma_j, \zeta_j, \delta_j)$ and $\omega_k((\omega_k)_{Resource}, (\omega_k)_{Function}, (\omega_k)_{begin}, (\omega_k)_{end})$. The details of how determining timing parameters in the architectural level from the functional specification are given in Section 3.3).

2.1.3 Scheduling Analysis and Synchronization Protocol

Critical real-time systems have timing constraints that must be satisfied. Ensuring that timing constraints are met at runtime requires real-time scheduling analysis at an early stage i.e. at design level.

A system is said to be schedulable if all its tasks meet their deadlines when it is executed on the target platform with a particular scheduling policy (Buttazzo 2011). The assumptions defined at the functional and the design levels to enable timing verification strongly depend on the type of the scheduling policy. Since we consider systems compliant with the Ravenscar profile (Burns 1999). The Ravenscar profile defines a set of restrictions that avoid non-deterministic behaviors in the implementation of critical real-time systems. The scheduling analysis is performed according to a fixed-task priority and preemptive policy, with the Rate Monotonic priority assignment (RM) (Klein et al 1993) or the Deadline Monotonic priority assignment (DM) (Leung and Whitehead 1982).

The presence of shared resources will make the scheduling analysis more complex. Indeed, a task may experience an additional delay called *blocking time* as a direct consequence of the mutual exclusion. The maximum blocking time B_i that a task τ_i can incur is defined by the maximum amount of time spent by this task to access a resource. Figure 2 shows an example of execution of two periodic tasks (τ_i and τ_j) sharing a resource. In this example, the task τ_i with highest priority is blocked until the release of the resource by task τ_j.

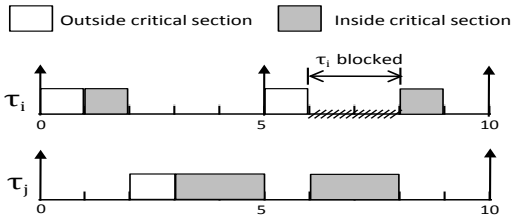


Fig. 2: Example of blocking time on a shared resource

Use of classical synchronization primitives (i.e. semaphores) can lead to priority inversions [1] or deadlocks [2]. Such phenomena introduce unbounded delays affecting both the schedulability and the predictability of the tasks.

Resource access protocols such as PCP (Sha et al 1990) are designed to handle the synchronization of tasks accessing shared resources with mutual exclusion while preventing priority inversion and deadlocks. Thanks to these protocols, the maximum blocking time of tasks can be bounded and taken into account in the schedulability analysis.

[1] a situation in which a task is blocked by a lower-priority task.

[2] a situation in which two or more tasks are waiting indefinitely for each other

According to Ravenscar, in this article, we assume that PCP is used as a synchronization protocol. This protocol assigns at each resource R_k a priority called *priority ceiling* $\Pi(R_k)$ equal to the highest priority of any task that may access the resource. Then, a task can lock an available resource only if its priority is strictly higher than all priority ceilings of the resources currently locked by other tasks.

We perform the scheduling analysis through a simulation-based schedulability test. A simulation test consists in producing the scheduling sequence of a task set within a finite duration in order to detect any temporal fault. This duration is called *feasibility interval*:

Definition 1 (Feasibility interval (Goossens et al 2016)) is a finite interval such that if all the deadlines of jobs released in the interval are met, then the system is schedulable

As we assume synchronous tasks systems, the feasibility interval is the *hyperperiod* (the least common multiple of task periods) according to (Goossens et al 2016). We achieve schedulability by simulation using the Cheddar scheduling simulator (Singhoff et al 2004).

2.2 Multi-Objective Optimization (MOO)

Many engineering problems involve more than one competing objectives that need to be optimized simultaneously with respect to a set of constraints. Then, the performance of candidate solutions have to be evaluated according to more than one objective (Coello Coello et al 2007). The outcome of a MOO algorithm is a single or a set of solutions, with each solution representing a trade-off between objectives.

MOO has been applied in many fields, including product and process design, economics and logistics. When an attempt to improve one objective leads to the degradation of the other, decisions need to be taken in order to define trade-offs between objectives.

A single solution that yields the best value for all objectives rarely exists. Instead, a set of alternative solutions called *non-dominated*(or *Pareto optimal*) solutions are searched for.

A solution is Pareto-optimal with respect to a set of objectives if there is no other solution in the search space that improves on all of the objectives at once. These solutions form the *Pareto set*. As depicted in Figure 3, the associated objective vectors (points) correspond to the *Pareto front* which represents the best trade-offs set for the considered objectives.

Two candidate solutions can be compared according to the *Pareto dominance* concept:

Definition 2 (Pareto dominance concept (Hruschka et al 2009)) a candidate solution c_1 ***dominates*** another candidate solution c_2 *if and only if* (i) c_1 is strictly better than c_2 for at least one of considered objective and (ii) c_1 is not worse than c_2 for any of the objectives.

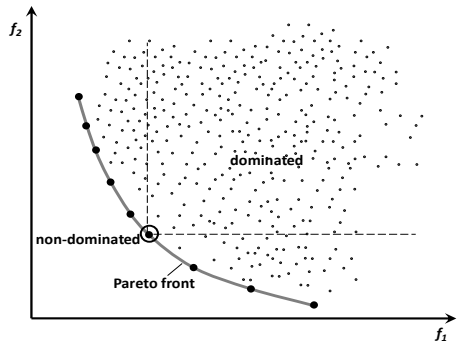


Fig. 3: Pareto front for two minimization objectives

Many metaheuristic algorithms that seek to approximate the Pareto set have been developed to solve MOO problems (Bandyopadhyay and Saha 2013). The key points for those algorithms are their accuracy (how close to the Pareto front are the values of the solutions they provide) and their diversity (are the solutions numerous and with values well spread in the objective space).

A large amount of MOO techniques is derived from Multi-Objective Evolutionary Algorithms (MOEA). MOEAs are useful in combinatorial problems that present a very large search space, when exact methods for MOO like multi-objective integer programming are not applicable because of the amount of computation resources they require. MOEA are part of evolutionary algorithms. These metaheuristic optimization algorithms are inspired from nature, e.g. particle swarm (PSO), ant colony, simulated annealing, genetic algorithms (GA). The latter use biology-inspired mechanisms like *mutation* and *crossover* in order to refine a set of candidate solutions iteratively.

2.2.1 PAES MOEA

The Pareto Archived Evolution Strategy (PAES) (Knowles and Corne 2000) is a MOEA technique using archiving. It serves to find a set of solutions properly distributed over the whole spectrum of possible trade-offs between objectives, which allows us to make the design exploration.

PAES manipulates a single solution as opposed to other methods used for MOO such as GA or PSO. This is a key point in running very time-consuming evaluation functions such as those stemmed from the scheduling analysis. Its conceptual approach is quite simple as a multi-objective local search procedure.

The sequential PAES schema is outlined in the pseudo code of Algorithm 1.

In many evolutionary algorithms, a solution is presented by a set of parameters called *chromosome* or *genotype*. The *encoding* of solutions (i.e. the data structure of a chromosome) is specific for each problem.

The PAES algorithm is based on a (1+1) evolution strategy. This means that it maintains a single current solution (*parent*), and at each iteration, generates a single new candidate (*offspring*) through a random *mutation* (line 5).

Algorithm 1: General form of PAES Algorithm

```
1  begin
2      Generate initial random solution c
3      Evaluate c and add it to the archive
4      repeat
5          Mutate c to produce a new candidate solution m;
6          Evaluate m;
7          if (c dominates m) then
8              Discard m;
9          else if (m dominates c) then
10             Replace c with m;
11             Add m to the archive;
12             Remove from the archive solutions dominated by m;
13         else if (m is dominated by any member of the archive) then
14             Discard m;
15         else
16             Apply test (c,m,archive) to determine which becomes the new current
               solution and whether to add m to the archive;
17         end if
18     until (termination condition is satisfied);
19 end
```

This algorithm is confined to a local search, i.e. it performs only a small change (mutation operator) that moves from a current solution to a nearby neighbour. The mutation procedure is also specific for each problem, and depends on the way the solution is represented into a chromosome.

The candidate solutions are evaluated (line 6) according to a set of *objective functions* and compared using the Pareto dominance concept. An objective function is used to indicate the quality of a solution according to an objective (or performance) criterion. Each objective is defined by an objective function.

The current solution is replaced at each iteration by its mutated offspring if the latter dominates or is in a less crowded region than its parent. Otherwise (i.e. dominated offspring or non-dominated offspring in a crowded region), the next iteration is realized keeping the same current solution as a basis for mutation.

PAES maintains a list of some non-dominated solutions called archive used as reference set with respect to which each new candidate is being compared (lines 11, 12).

The crowding metric depends on the location of the archive members within a self-adaptive grid over the objective space.

The algorithm iterates (lines 4-16) according to some convergence criteria of the objective function values or based onto a fixed number of iterations.

Its time complexity is in the order of $\mathbf{O}(a \cdot n)$, where a is the archive size and n is the number of iterations.

3 MOO-Based Method for Real-time Software Design Exploration

In this section, first we present an overview of our design exploration methodology. Then, we detail the PAES formulation for our problem. Afterwards, we define a set of rules allowing to compute parameters of a design alternative (in terms of tasks, resources and critical sections) from a candidate assignment solution and the functional specification. Finally, we discuss the impact of our functions-to-tasks assignment method on the system schedulability.

3.1 Methodology Description

As described in Figure 4, the entry point of our approach is the functional specification of a critical real-time system. This specification defines the functions of the system, their interactions and their real-time characteristics.

From this specification, an initial architecture is proposed so that each function is assigned to a single task. A scheduling analysis is achieved on this initial architecture using the Cheddar scheduling tool. If the initial task set is schedulable then it will be considered as the initial solution to the PAES algorithm. Otherwise, the timing parameters of the functions must be adapted.

Once the initial solution is defined, we come to the multi-objective design exploration and optimization part. The latter involves the execution of the PAES algorithm.

At each iteration, an alternative design solution (i.e. the mutated solution) is generated from the current solution by changing the assignment of a random function to a random task by the mutation procedure (① in Figure 4).

Feasibility checks ② are performed on each candidate solution in order to produce designs that fulfil the timing constraints and the functions to tasks assignment constraint (details and algorithm are given in Section 3.4).

This candidate design is then evaluated according to the considered objective functions ③.

Then, other PAES steps are performed such as the evaluation, the archiving and the selection of the current solution for the next iteration ④.

The four steps are iterated until the termination condition of PAES is reached, e.g. number of iterations. The output of our method is a set of schedulable design alternatives that approximates the Pareto set. From these solutions, software engineers would choose the suitable design.

3.2 PAES Formulation for the Functions-to-Tasks Assignment Problem

As presented in Section 2.2.1, PAES relies on the definition of several components to solve a particular MOO problem. In this section, we specify those components, i.e. the objective functions, the encoding of solutions into chromosomes, the initial current solution, and the mutation operator in order to formulate our problem.

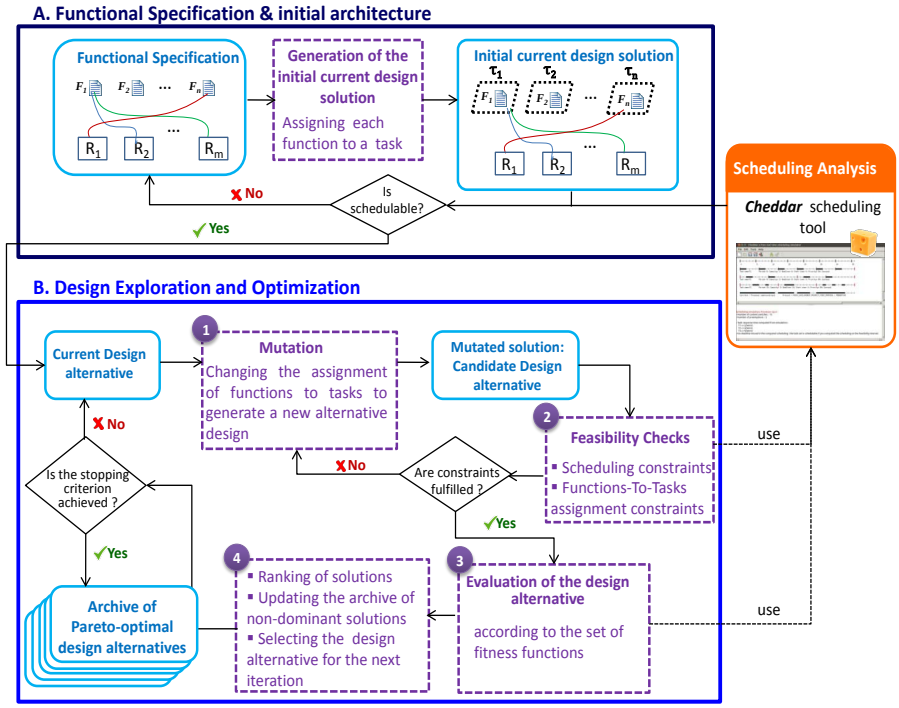


Fig. 4: Approach description

3.2.1 Objective Functions

Considering the system model that we assume and the functions-to-tasks assignment problem, several design performance criteria can be defined as objective functions to drive the multi-objective exploration and optimization process. Here, we list some possible objective functions.

1. Minimization of preemptions:
   The minimization of the number of preemptions expresses the minimization of scheduling timing overheads. A preemption occurs when a higher priority task τ_i is released during the execution of a lower priority task τ_j and when the scheduler interrupts the execution of τ_j to allow the task τ_i to run (Buttazzo 2011). A preemption causes a context switching which requires to run several processor instructions and thus an important overhead to the overall task execution time. The objective function relative to this metric counts the total number of preemptions in the scheduling simulation. We note this objective function as follows:

$$\textbf{minimize} \quad (f_1 = \#\ preemptions) \tag{1}$$

2. Minimization of context switches:

The minimization of context switches is also equivalent to the minimization of timing overheads. The context of a task consists of its memory context and its processor context, e.g. the temporary register values, the program counter value, etc. The switching between tasks consists in storing the current running task context and retrieving the context of the task selected to be executed. The objective function related to context switches counts the total number of context switches on the scheduling sequence produced through simulation. We note this objective function as follows:

$$\textbf{minimize} \ \ (f_2 = \texttt{\#}\ context\ switches) \tag{2}$$

3. Minimization of the number of tasks:
The minimization of the number of tasks may lead to the minimization of both timing and memory overhead. Indeed, a large number of tasks is one of the factors inducing high number of context switchs and requires extra memory allocations for the task execution stack. The objective function is noted as follows:
$$\textbf{minimize} \ \ (f_3 = \texttt{\#}\ tasks) \tag{3}$$

4. Maximization of tasks laxity:
The maximization of the tasks laxity may improve the design flexibility, i.e. the larger the laxities of tasks are, the more the design model could support additional tasks or possible changes of task parameters. The laxity is the maximum time a task can be delayed on its activation to complete within its deadline (Buttazzo 2011). For a task τ_i, it is defined as the difference between its deadline D_i and its worst-case response time $WCRT_i$, or $D_i - WCRT_i$. The $WCRT_i$ of a task τ_i is the maximum delay between its release time and its completion time (Audsley et al 1993).
We consider two examples of possible alternatives among many, as an objective function for the tasks laxity metric:
   (A) Maximize the overall laxity of the addressed application, that is the sum of laxities over all resulting tasks.
$$\textbf{maximize} \ \ (\sum_{i=1}^{k}(D_i - WCRT_i)) \ \ (k:\ number\ of\ tasks)$$
   (B) Maximize the minimum task laxity
$$\textbf{Maximize} \ \ (min_{i\in[1..k]}\{D_i - WCRT_i\})$$

In our experiments, we chose the alternative (A).
The PAES algorithm was designed to deal with only two kinds of problems, either maximization problems (i.e. all objectives are to be maximized) or minimization problems (i.e. all objectives are to be minimized). However, most the objectives we consider are to be minimized except the laxity of tasks objective which is for maximization. Thus, in order to harmonize the objectives, we transform this objective to a minimization objective function.

As shown in Equation 4, we have chosen the hyperperiod of the initial architecture as a constant [3] from which the original objective function will be subtracted. With such method, the minimization of the new expression would result in the maximization of the current objective:

$$\textbf{minimize} \ \ (f_4 = hyperperiod - \sum_{i=1}^{k}(D_i - WCRT_i)) \tag{4}$$

5. Minimization of WCRT of tasks:
   In the context of real-time systems, we have always an interest in minimizing the response time of tasks. As for the tasks laxity, there are two possible examples of objective function alternatives, either to minimize the overall WCRT (Equation 5.1) or to minimize the maximum WCRT (Equation 5.2).

$$\textbf{minimize} \ \ (f_5 = \sum_{i=1}^{k} WCRT_i) \tag{5.1}$$

$$\textbf{minimize} \ \ (f_5{}' = max_{i\in[1..k]}\{WCRT_i\}) \tag{5.2}$$

6. Minimization of the worst-case blocking time (WCBT) of tasks:
   As we defined in section 2.1.3, the blocking time is induced by resources mutual exclusion. It causes latencies in the tasks execution. Again, we can distinguish two possible examples of objective functions for the WCBT metric. In the experiments, we will consider the objective function of Equation 6.1.

$$\textbf{minimize} \ \ (f_6 = \sum_{i=1}^{k} B_i) \tag{6.1}$$

$$\textbf{minimize} \ \ (f_6{}' = max_{i\in[1..k]}\{B_i\}) \tag{6.2}$$

7. Minimization of the number of shared resources:
   The minimization of the number of shared resources allows the minimization of semaphores and then reduces memory cost. We note the associated objective function as follows:

$$\textbf{minimize} \ \ (f_7 = \texttt{\#}\ shared\ resources) \tag{7}$$

While formulating our problem, we desire to define each performance metric stemmed from the context of our problem as an objective, rather than as a constraint. However, involving all the aforementioned objectives simultaneously leads to a large-dimensional problem (also known as *many-objective* problem). This hinders a Pareto-based MOEA (e.g. PAES) from approximating efficiently the Pareto front (López Jaimes et al 2008). Indeed, MOEA methods are often applied to problem with at the most two or three objectives. This is due to the fact that the number of non-dominated solutions

[3] the choice of the constant is arbitrary

increases exponentially with the number of objectives (Deb and Saxena 2006) which complicates the decision making and the search processes, thereby adversely affecting the computational tractability of MOEA methods. Moreover, the visualization and the analysis of a large-dimensional Pareto front is a tedious task. In fact, it would be very hard for designers (i.e. the decision-makers in our context) to analyze a large number of solutions in order to choose the most suitable one.

Furthermore, although some of the above listed objectives are *conflicting*, others may behave in a *non-conflicting* way. In the second case, the objectives support each other and are denoted as *redundant* objectives. An objective is identified as redundant when the Pareto front remains the same even if this objective is omitted from the original set of objectives (Gal and Hanne 1999).

However, it is not intuitively obvious for us to predict the relationship between pairs of the above objective function list. For that, we choose three objectives, (f_1, f_4 and f_6) and we devote Section 5.3 to investigate through experiments the relationships (conflicting or redundant) between each pair of these objectives.

Most of the objective functions (e.g. f_1, f_2, f_4, f_5 and f_6) are computed using the Cheddar scheduling tool (see Section 4).

3.2.2 Encoding

We represent a solution by means of a conventional *Integer Encoding* schema. With such encoding, a chromosome is defined as an integer vector with n positions. In our case, n represents the number of functions in the functional specification. A chromosome exhibits information about the assignment of functions to tasks. Each position in the chromosome, called *gene*, denotes a particular function, i.e. the i^{th} position corresponds to the i^{th} function. The value held by a gene represents the index of the task to which the corresponding function is assigned. Therefore, a chromosome represents a solution formed by k tasks ($k \leq n$), where each gene has a value in $\{1, 2, ..., k\}$. This means that each function is assigned to one of the tasks $\{\tau_1, \tau_2, ..., \tau_k\}$.

Figure 5 shows an example of a chromosome that corresponds to a particular solution S. The chromosome length indicates the number of functions. In this example, the functional specification consists of 8 functions. Gene F_1 holds a value equal to 2, which means that the function F_1 is assigned to the task of index 2 (τ_2). The solution modeled by this chromosome assigns F_4 to τ_1; F_1,F_3 and F_7 to τ_2; F_2 and F_6 to τ_3; F_5 and F_8 to τ_4.

This encoding is quite simple but it is redundant, i.e. the same solution can be represented by different chromosomes. For example the solution S encoded by the chromosome depicted in Figure 5 can also be represented by other chromosomes, namely [1 2 1 3 4 2 1 4], [3 1 3 2 4 1 3 4], [2 4 2 3 1 4 2 1], etc. In other words, a solution may be represented by many chromosomes that model the same assignment solution independently of the indexes of tasks.

To reduce all of the equivalent assignment solutions to the same chromosome representation, we *normalize* the chromosome representation. This

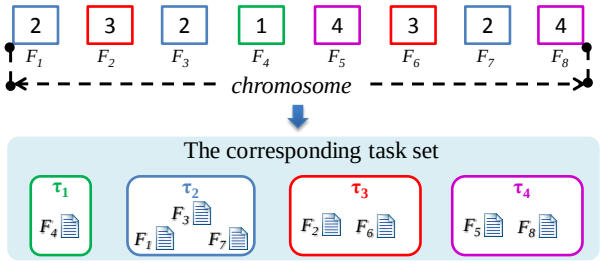


Fig. 5: Chromosome representation of a particular assignment solution S

is performed as follows: the task index of the function F_1 is always 1 (then all functions that belong to the same task as F_1 have as task index 1). The task index of F_2 is 2, except if F_2 is assigned to the same task as F_1, and so on. Figure 6 shows the normalized chromosome representation of the same assignment solution S inFigure 5.

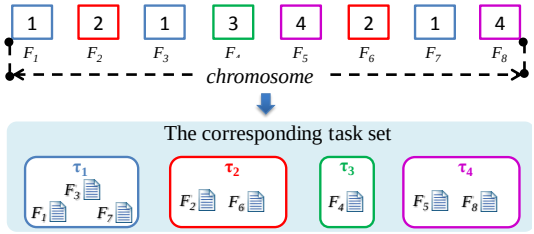


Fig. 6: The normalized chromosome representation of the assignment solution S

The normalization procedure helps to compare two solutions. It must be applied on each chromosome of candidate solutions generated by mutation.

3.2.3 Initial Current Solution

It consists in assigning each function to a single task, i.e., the number of tasks is equal to the number of functions:

$$chrom = [1\ 2\ 3\ 4\ ..\ n]\ (n:\ number\ of\ functions).$$

3.2.4 Mutation Operator

We have adapted a random mutation operator that chooses a random position in the chromosome and changes the value of the associated gene to a new random value. This means that the mutation produces a new alternative

assignment solution from the current solution by reassigning one random function to a random task. Algorithm 2 describes the mutation operator proposed for our functions-to-tasks assignment problem. This operator is implemented in order to generate only feasible solutions that satisfy constraints verified by Algorithm 4 (Section 3.4.3).

Algorithm 2: Mutation algorithm for the functions-to-tasks assignment problem

```
/* Mutation Algorithm: generate a mutated solution from the current one */
/* current solution: n functions assigned to k tasks (n ≥ k)            */
input : current solution
output: mutated solution
1  begin
2      Choose randomly a function F_i (1 ≤ i ≤ n);   /* chrom[F_i] = τ_j i.e. the
       function F_i is assigned to the task τ_j */
3      Determine the set of harmonic tasks with the chosen function F_i called
       harmonic_task_set;
4      if (harmonic_task_set = {τ_j}) then
           /* i.e. the function F_i is not harmonic with any task, only the
           task to which it is assigned                                    */
5          Restart the algorithm with another function randomly chosen;
6      else
7          Choose randomly a task τ_m over tasks in the set harmonic_task_set
           (including τ_j = chrom[F_i]);
8          if (τ_m ≠ τ_j) then
9              chrom[F_i] ← τ_m; /* F_i is moved to the task τ_m          */
10         else if (The function F_i is not alone in τ_j) then
11             Create a new task τ_{k+1};
12             chrom[F_i] ← τ_{k+1}; /* F_i is isolated in the new task τ_{k+1} */
13         else
               /* i.e. F_i is the only function assigned to τ_j           */
14             Restart the algorithm with another function chosen randomly;
15         end if
16     end if
       /* once the mutated solution is generated, we apply the assignment
       rules to generate the new composition of the task set            */
17     Apply the assignment rules on the new candidate solution to generate the
       corresponding design alternative parameters: i.e. tasks and critical sections
       parameters;
       /* Check the feasibility of the produced design alternative by
       applying Algorithm 4                                             */
18     if The design alternative is not feasible then
19         Restart the algorithm with another function randomly chosen;
20     end if
21 end
```

3.3 Functions-to-Tasks Assignment Rules

In this section, we define the rules used to assign functions to tasks. It consists in determining the parameters of tasks and resources while considering the timing parameters of the functions and how functions are assigned to tasks.

3.3.1 Initialization step

At the initialization step, each function is assigned to a task that will take the same parameters as the corresponding function.

$$\forall i \in \{1..n\} \quad F_i = (\gamma_i, \zeta_i, \delta_i)$$
$$\tau_i = (C_i, T_i, D_i) \quad \begin{cases} C_i = \gamma_i \\ T_i = \zeta_i \\ D_i = \delta_i \end{cases}$$

Since each function is assigned to exactly one task, then the shared resources to be taken into consideration in the initial design are the same that in the functional specification.

Similarly, critical sections CS_k of each resource are deduced from ω_k that specify parameters of the use of resources by functions. The only change is to set $(CS_k)_{Task}$ to the task that holds the function that uses the resource.

The mutation operator allows us to explore different software design alternatives through reassignment of functions to tasks. As described in the mutation algorithm 2, we cluster/separate functions to get a new design alternative. This would involve changes in tasks parameters as well as the resource set and critical sections. In the following, we deal with the way to compute (1) the parameters of tasks (Section 3.3.2) and (2) the parameters of resources and critical sections (Section 3.3.3) of an alternative assignment solution generated by mutation.

3.3.2 Tasks Parameters Computation

The generation of a new alternative assignment solution by mutation must preserve:

(i) The functional specification in terms of periodic activations of the functions.
(ii) The schedulability of the system (i.e. the schedulability of the resulting design).

In the sequel, we explain and motivate our assignment method and how we compute, from the function parameters, the parameters of each task, while taking into account the constraints outlined above.

Let us consider a task $\tau_k = (C_k, T_k, D_k)$ implementing a single function $F_i = (\gamma_i, \zeta_i, \delta_i)$. Let us see also a classic implementation of a periodic task with Ada (McCormick et al 2011). The Listing 1 presents such an implementation

for τ_k. The task is periodically released thanks to the DELAY UNTIL statement and then, calls the F_i sub-program to run the function implemented by the task.

Listing 1: Classical implementation of a periodic task with Ada

```
1  with Ada.Real_Time; use Ada.Real_Time;
2  ...
3  task body Tau_k is
4   -- Next_Time used for periodic suspension
5   Next_Time : Time := Clock;
6   Period    : constant Time_Span := Milliseconds(T_k);
7  begin
8   loop
9     -- calling the function run by the task  τ_k
10    F_i;
11    Next_Time := Next_Time + Period;
12    -- Time-based activation event
13    delay until Next_Time;
14  end loop;
15 end Tau_k;
```

Let us see how to set τ_k parameters.

Computation of the Task Period: We want to set a period for τ_k that releases the task as F_i and F_j should be released. First, we assume that the function F_j can be added to τ_k if and only if

$$\zeta_j \; mod \; T_k = 0 \quad or \quad T_k \; mod \; \zeta_j = 0$$

This condition means that the function F_j is *harmonic* with the task τ_k. Considering this assumption, we can set the period of τ_k as follows:

$$T_k = GCD(\zeta_i, \zeta_j) \tag{8}$$

With GCD is the Greatest Common Divisor of the periods of the functions F_i and F_j.

A task may implement several functions. It is the case of task τ_k and we then need to provide a specific implementation of such a task. It is given through Listing 2. This implementation preserves the periodic behavior of the functions assigned to the task. Indeed, the added lines (displayed in **blue boldface**) to the implementation of the task ensure an internal scheduling of the functions assigned to it according to their frequency.

Listing 2: Ada implementation of the task τ_k containing two functions

```
with Ada.Real_Time; use Ada.Real_Time;
...
task body Tau_k is
 Next_Time          :Time := Clock;
 Number_Functions   : Integer := 2;
 Period_Functions   : Integer_Array (1..Number_Functions) := (ζi, ζj);
 Index_Functions    : Integer_Array (1..Number_Functions) := (i,j);
 Period             :constant Time_Span :=
                          Milliseconds(GCD(Period_Functions));
 Counter            : Integer := 0;
 Frequency          : Integer;
begin
 loop
  for i in 1 .. Number_Functions loop
   Frequency := Period_Functions(i)/Period;
   if (counter mod Frequency = 0) then
       Call_Function_by_index(Index_Functions(i));
   end if;
  end loop;
  -- LCM : Least Common Multiple
  counter :=
        (counter + 1) mod (LCM(Period_Functions)/Period);
  Next_Time := Next_Time + Period;
  delay until Next_Time;
 end loop;
end Tau_k;
```

In order to illustrate the behavior at runtime of several functions assigned to one task (Listing 2), we consider the following example. We assume $F_1 = (2, 5, 5)$ and $F_2 = (1, 10, 10)$ assigned to a task τ_k.

According to Equation 8, the period of τ_k is equal to $GCD(10, 5) = 5$.

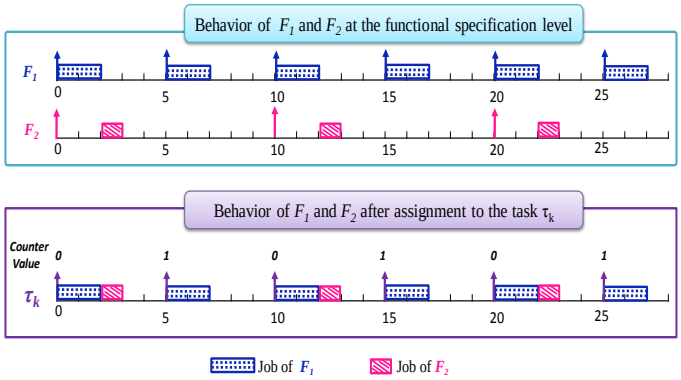


Fig. 7: Behavior of functions F_1 and F_2 before and after assignment to the task τ_k

Figure 7 shows, first, the execution sequence of F_1 and F_2 assuming that each function is assigned to a separate task. Second, it shows also the sequence of their execution after assigning both functions to the same task τ_k. This

example shows how the implementation of Listing 2 as well as Equation 8 ensure periodic executions of functions assigned to the same task.

Computation of the Task Capacity: As shown in Figure 7, the execution time of a task τ_k implementing two functions F_i and F_j differs from one period to another. However, we only consider the worst execution time of both functions run by the task. Therefore, the capacity of τ_k is set to the sum of the capacities of all functions assigned to this task:

$$C_k = \gamma_i + \gamma_j \tag{9}$$

Obviously, this method to compute capacities is pessimistic and may reduce the schedulability of the resulting design solution.

Computation of the Task Deadline: As a first approach, the deadline D_k of a task τ_k holding functions F_i and F_j is set as follows:

$$D_k = min(\delta_i, \delta_j) \tag{10}$$

This proposition may restrict the task τ_k with a smaller deadline than required by functions F_i and F_j. Again, it may reduce the schedulability of the associated design.

By considering the assignment method described above, the constraint (i) mentioned at the beginning of this section is preserved. As we aim also to ensure the constraint (ii), in Section 3.4.1, we discuss how the assignment may jeopardize schedulability of the design alternative solutions.

3.3.3 Computation of Critical Sections After Mutation

The mutation operator changes the assignment of functions to tasks which may impact the parameters of the resource set and critical sections. In this section, we address how to determine a resource set and critical sections parameters for a design alternative relative to a mutated solution.

Algorithm 3 represents the different steps and cases we deal with to set parameters of the resource set after mutation.

We illustrate Algorithm 3 with an example. The initial design of this example is given in Figure 8. It consists of 6 tasks, each one holds one function. As shown in Figure 8, there are two shared resources R_1 and R_2.

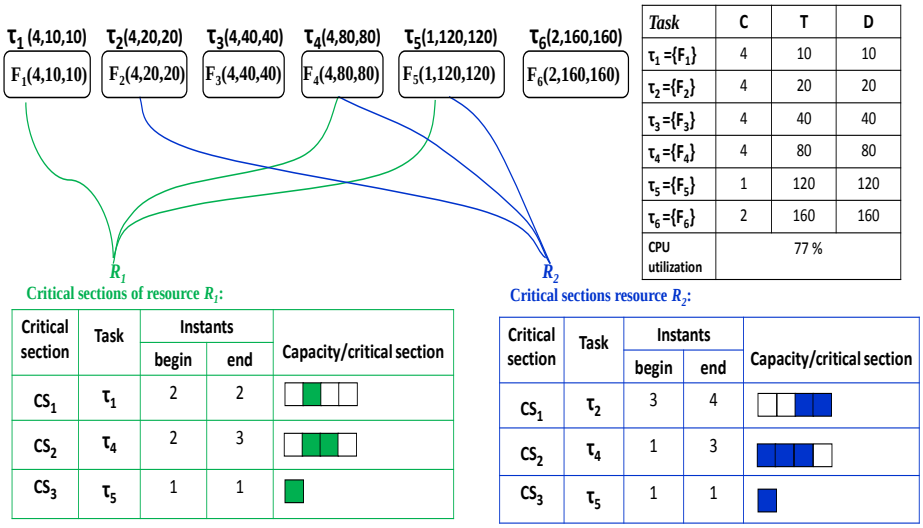
From this initial design, a possible mutation is to assign function F_4 with the function F_2 to the task τ_2 as F_4 is harmonic with τ_2 ($Period(F_4)\ mod\ Period(\tau_2) = 80\ mod\ 20 = 0$). We apply Algorithm 3 on this mutated solution and we get the resource set and critical sections as shown in Figure 9. We can notice that the change of parameters of CS_2 of R_1 is stemmed from rules defined in Algorithm 3: the task index changes from τ_4 to τ_2 as F_4 is reassigned to τ_2 (according to line 6) and the begin/end instants are changed taking into account the capacity of F_2 (according to lines 8,9). We can also notice that the task accessing R_1 on CS_3 is set to τ_4 (instead of τ_5 in the initial design)

Algorithm 3: Computation of resource set and critical sections of a mutated solution

```
   input : mutated solution, initial design
1  - mutated solution: the solution generated by mutation
2  - initial design: each function is assigned to a task
   output: resource set and critical sections relative to the mutated solution
3  begin
4      foreach resource R_i in the resource set of the initial design do
5          foreach critical section CS_j of the resource R_i in the initial design do
               /* Let consider F_h the function that uses the resource R_i on
               the critical section CS_j in the initial design                */
6              Set the task that holds R_i on CS_j with the task to which F_h is assigned
               according to the mutated solution;
7              if in the mutated solution, the function F_h is grouped with other
               functions in the same task then
                   /* The begin and end instants of CS_j must be updated
                   according to capacities of functions with lower indexes
                   than F_h which are grouped with it in the same task       */
                   /* Let consider (F_l)_{l≤h} the set of functions with lower
                   indexes than F_h and which are grouped with it in the same
                   task                                                      */
8                  (CS_j)_begin ← (CS_j)_begin + Σ γ_l;
9                  (CS_j)_end ← (CS_j)_end + Σ γ_l;
10             end if
               /* Check consistency of resource set and critical sections. We
               have to deal with two cases:                                  */
               /* Case 1: Check if in the mutated solution the resource R_i is
               accessed by a single task, i.e. all functions that use R_i are
               assigned to the same task                                     */
11             if in the mutated solution, the resource R_i is no longer shared then
12                 Delete R_i from the resource set of the associated design alternative;
13             end if
               /* Case 2: Check if in the mutated solution there are two or
               more consecutive critical sections for the same task          */
14             if in the mutated solution, there are two or more consecutive critical
               sections for the same task then
15                 Merge these critical sections in one;
16             end if
17         end foreach
18     end foreach
19 end
```

because in the mutated solution, the function F_5, that uses R_1 on CS_3, is assigned to the task τ_4 (according to line 6).

Let us consider the resource R_2. In the functional specification the resource R_2 is used by functions F_2, F_4 and F_5. Then, in the initial design, as we can observe in Figure 8, the critical sections CS_1 and CS_2 of R_2 are held by τ_2 and τ_4 respectively. A possible assignment solution that can be produced by mutation on the initial design is the solution shown in Figure 9: the function F_4 is assigned with F_2 to the task τ_2. The application of lines 6-9 of Algorithm 3 on this candidate solution gives the following parameters of critical sections



Task	C	T	D
$\tau_1=\{F_1\}$	4	10	10
$\tau_2=\{F_2\}$	4	20	20
$\tau_3=\{F_3\}$	4	40	40
$\tau_4=\{F_4\}$	4	80	80
$\tau_5=\{F_5\}$	1	120	120
$\tau_6=\{F_6\}$	2	160	160
CPU utilization	77 %		

Critical sections of resource R_1:

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_1	2	2	
CS_2	τ_4	2	3	
CS_3	τ_5	1	1	

Critical sections resource R_2:

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_2	3	4	
CS_2	τ_4	1	3	
CS_3	τ_5	1	1	

Fig. 8: Initial design model example: each function is assigned to one task

CS_1 and CS_2 of resource R_2:

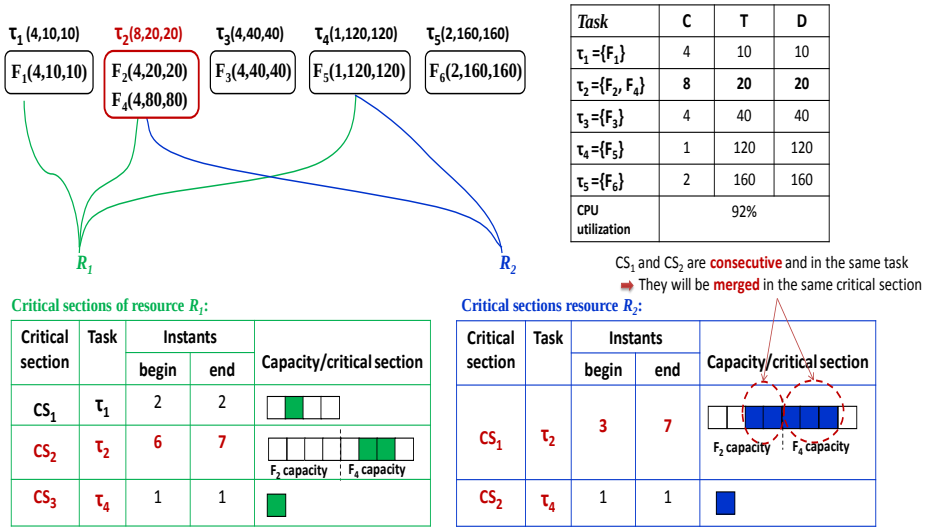
$$CS_1 \begin{cases} (CS_1)_{Task} = \tau_2 \\ (CS_1)_{begin} = 3 \\ (CS_1)_{end} = 4 \end{cases} \qquad CS_2 \begin{cases} (CS_2)_{Task} = \tau_2 \\ (CS_2)_{begin} = 5 \\ (CS_2)_{end} = 7 \end{cases}$$

We can notice that CS_1 and CS_2 are held by the same task τ_2 and are consecutive i.e. CS_1 finishes at the 4^{th} unit of time of the capacity of τ_2 and CS_2 begins at the 5^{th} unit of time of the capacity of τ_2. Then, the rule defined at lines 14-16 of Algorithm 3 requires the merge of CS_1 and CS_2 in one critical section as illustrated in Figure 9.

From the solution given in Figure 9, a possible mutation is to assign function F_5 with the functions F_2 and F_4 to the task τ_2 as F_5 is harmonic with τ_2. Again, we apply Algorithm 3 on this mutated solution and we obtain the resource set and critical sections as given in Figure 10. We can observe that the resource R_2 is no longer shared. It is thus eliminated from the design model relative to this mutated solution (according to lines 11-13).

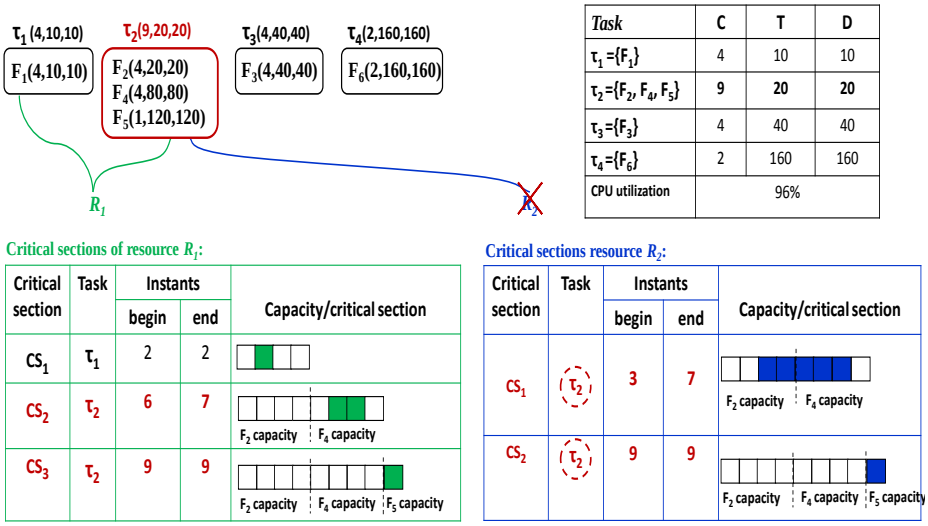
3.4 Feasibility Checks on Candidate Solutions

Feasibility checks are used to avoid the generation of non-feasible candidate solutions after mutation. We distinguish two kinds of feasibility checks: timing/schedulability requirements checks and functions-to-tasks assignment related checks. In the remainder of this section, first we discuss the impact of the mutation and the assignment method on the schedulability of a design alternative. Second, we show how they may lead to unnecessary activations



Task	C	T	D
$\tau_1=\{F_1\}$	4	10	10
$\tau_2=\{F_2, F_4\}$	**8**	**20**	**20**
$\tau_3=\{F_3\}$	4	40	40
$\tau_4=\{F_5\}$	1	120	120
$\tau_5=\{F_6\}$	2	160	160
CPU utilization	92%		

Fig. 9: A possible assignment solution: the resulting resource set and critical sections of the mutated solution



Task	C	T	D
$\tau_1=\{F_1\}$	4	10	10
$\tau_2=\{F_2, F_4, F_5\}$	**9**	**20**	**20**
$\tau_3=\{F_3\}$	4	40	40
$\tau_4=\{F_6\}$	2	160	160
CPU utilization	96%		

Fig. 10: Another possible assignment solution: the resulting resource set and critical sections of the mutated solution

of some tasks. Finally, we give the algorithm that checks all the constraints, allowing thereby to decide about the feasibility of a given candidate solution generated by mutation.

3.4.1 Impact of the Mutation and the Assignment Method on the Schedulability of Candidate Solutions

In Section 3.3, we proposed rules to manage the assignment of functions to tasks. We noted that the assumptions made in the assignment method may produce unschedulable designs. In order to illustrate that, we consider a simple example of a functional model $\Gamma = \{F_1(1,5,5), F_2(3,10,10), F_3(3,20,20)\}$ without shared resources. The initial assignment solution is given in Table 1. This task set is schedulable since its processor utilization is equal to 65% ($< 69\%$) and we assume RM for the priority assignment.

An assignment alternative is to add F_3 to the task τ_1 as F_3 is harmonic with τ_1. The resulting task set will be as depicted in Table 2 and it is not schedulable (its processor utilization is equal to 110%).

Table 1: Initial assignment solution

Task	C	T	D
$\tau_1 = \{F_1\}$	1	5	5
$\tau_2 = \{F_2\}$	3	10	10
$\tau_3 = \{F_3\}$	3	20	20
CPU utilization	65%		

Table 2: A possible assignment solution

Task	C	T	D
$\tau_1 = \{F_1, F_3\}$	4	5	5
$\tau_2 = \{F_2\}$	3	10	10
CPU utilization	110%		

Since the mutation may generate non-schedulable candidate solutions, a schedulability analysis must be conducted on each design alternative explored by mutation during the search process. However, there is two cases where a candidate solution must be directly discarded without performing the scheduling analysis, thereby saving runtime (as the scheduling simulation is time-consuming). The first case is when the candidate solution exhibits one task (or more) whose capacity exceeds its deadline. The second case is when the overall processor utilization of the candidate solution exceeds 100 %.

3.4.2 Functions-To-Tasks Assignment Constraint

The mutation and the assignment method may lead to unnecessary activations of some tasks of the candidate solution. We illustrate such situation through the following example. We consider a system of 6 functions. As shown in Figure 11, a current solution consists of 4 tasks. A possible mutation on this current solution consists in assigning the function F_1 to the task τ_4 as F_1 is harmonic with τ_4 ($Period(\tau_4)\ mod\ Period(F_1) = 60\ mod\ 20 = 0$).

The resulting candidate solution is given in the same Figure. Parameters of the associated task set are computed through the application of the assignment rules on the candidate solution (i.e. Section 3.3). By examining the task set of the candidate solution, we note that the period of the task τ_4 is different from all periods of the functions to which they are assigned (i.e. F_4 and F_5). According to the task implementation given in Listing 2, there are

some activations of the task τ_4 (e.g. at instants 20, 100, 140, etc) at which the functions F_4 and F_5 should not be activated. In order to avoid such situation, any candidate solution that have (at least) a task whose period is different to the minimum period of its functions, is considered as non-feasible solution. This constraint is referred to as functions-to-tasks assignment constraint.

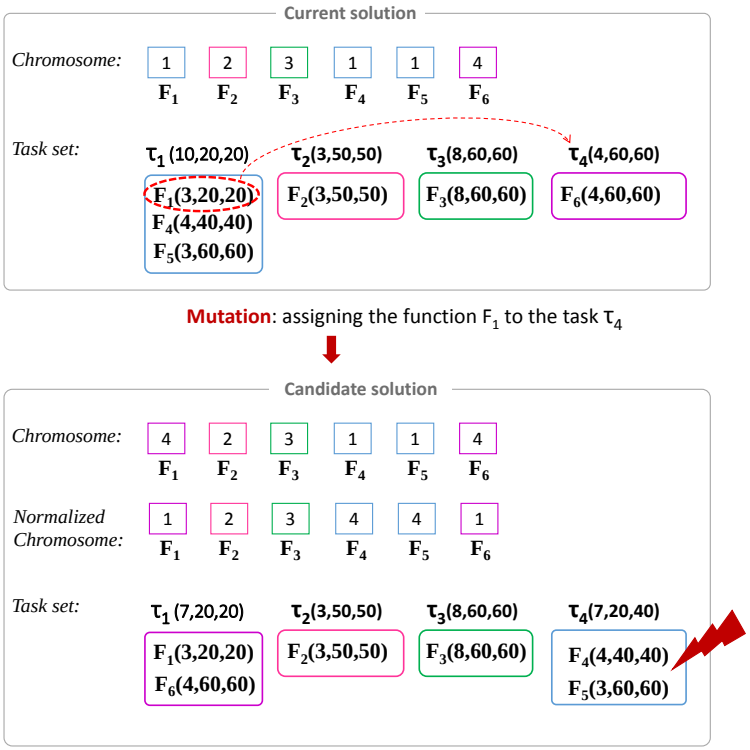


Fig. 11: Example of non-feasible candidate solution towards the functions-to-tasks assignment constraint

3.4.3 Feasibility Checks Algorithm

In Sections 3.4.1 and 3.4.2, we showed situations for which a mutation action may produce non-feasible solutions. In order to ensure that mutation generates only feasible solutions, a set of feasibility checks is performed on each explored design alternative. These feasibility checks are structured in Algorithm 4.

In the current section, we described elements of our approach. In the next section, we give an overview of the prototype that implements our solution for the problem we deal with.

Algorithm 4: Feasibility checks algorithm

input : design alternative: task set associated to a candidate solution generated by mutation
output: a boolean that designates the feasibility of the candidate solution
- false: non-feasible solution
- true: feasible solution

begin
  `/* 1) Check for each task that` $C_i < D_i$ `*/`
  if *there is at least a task* τ_i *for which* $C_i \geq D_i$ **then**
    return *false*;
  `/* 2) Check that the total processor utilization` $\leqslant 1$ `*/`
  else if $U > 1$ **then**
    return *false*;
  `/* 3) Check that all tasks fulfill the functions-to-tasks assignment constraint */`
  else if *there is at least one task* τ_i *that violates the functions-to-tasks assignment constraint* **then**
    return *false*;
  `/* 4) Check the schedulability through scheduling simulation */`
  else if *the design alternative is not schedulable* **then**
    return *false*;
  else
    return *true*;
  end if
end

4 Prototype

In this section, we present the prototype developed as part of our work. This prototype has been implemented in the Cheddar [4] framework (Singhoff et al 2004).

Cheddar is an open-source real-time scheduling analysis tool. It is designed for verifying temporal constraints of real-time critical systems. The developed software is built in Ada (Taft et al 2014).

The prototype covers the following aspects:

- The implementation of the PAES-based architecture exploration through functions-to-tasks assignment (method proposed in Section 3). In (Bouaziz et al 2016), we proposed a parallel implementation and a new selection strategy for the PAES algorithm. Then, our prototype provides both a sequential and a parallel implementations and several selection strategies including the original PAES selection strategy.
- Function sets generator: we provide generators of function sets with shared resources (methods detailed in Section 5.1).
- The implementation of an exhaustive search method defined in Section 5.4, mostly for evaluation purposes.

Figure 12 illustrates an overview of the software architecture of our prototype and a subset of the Cheddar libraries.

[4] All sources available at http://beru.univ-brest.fr/svn/CHEDDAR/trunk

With Cheddar, designers should specify the architecture of the application to analyze with AADL (Feiler and Gluch 2012), or the Cheddar specific ADL called *Cheddar-ADL*. Cheddar-ADL is an ADL devoted to real-time scheduling analysis. Cheddar-ADL allows users to define, for a given real-time system, the software components (e.g. tasks, shared resources, messages, etc), the hardware components (e.g. processors, cores, caches, etc) and the interactions between them. Designers can use the Cheddar-ADL associated graphical-editor in order to easily build their real-time application models.

Two different scheduling analysis methods could be applied with Cheddar: feasibility tests [5] and scheduling simulations.

Cheddar supports most of classical real-time scheduling algorithms and feasibility tests for different kinds of real-time applications (uniprocessor/multiprocessor, independent tasks/tasks with precedence constraints/tasks sharing resources, etc). More details about the Cheddar utilities and the Cheddar-ADL can be found in (Singhoff et al 2015).

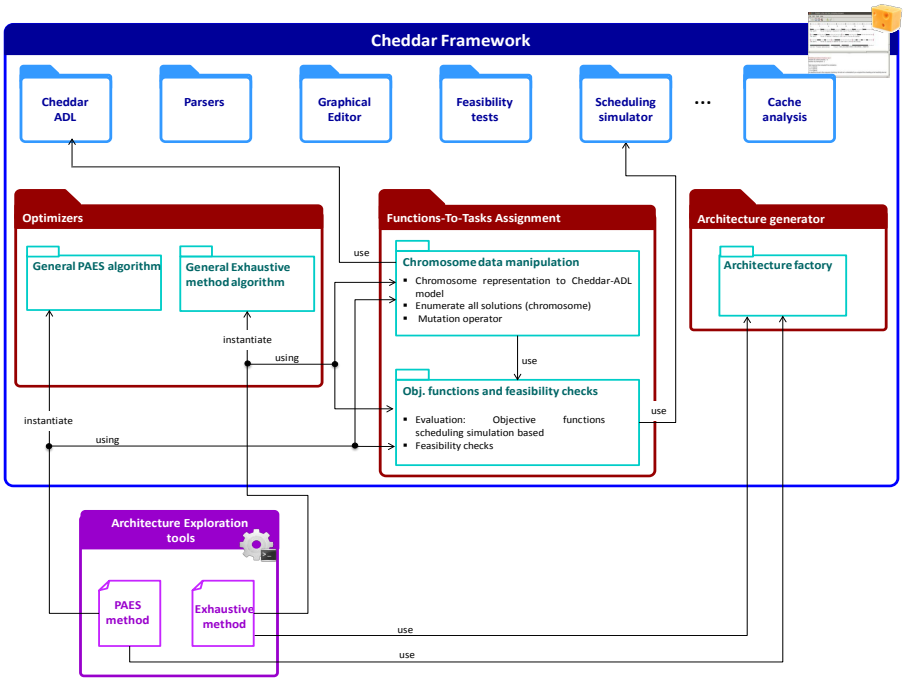


Fig. 12: Prototype overview

As shown by Figure 12, in order to implement our prototype, we extend the Cheddar framework as follows. We add two libraries called `Optimizers`

[5] A feasibility test is an analytical method which usually allows users to compute performance criteria in order to assess if task deadlines will be met.

and `Functions-To-Tasks Assignment` and we extend an existing one called `Architecture generator`.

The optimizers library consists of two Ada packages that define the basic and common sub-programs that could be reused to formulate any problem with either the PAES algorithm or the exhaustive method algorithm respectively. The former requires a mutation operator, the latter some way of enumerating solutions. Both are also parametrized with some objective functions.

The `functions-to-tasks assignment` library provides those components, which are specific to our optimization problem domain. As shown in Figure 12, it contains two packages. The first package (`chromosome data manipulation`) provides functions for manipulating the chromosome representation customized to the functions-to-tasks assignment. For example, it allows users to transform a chromosome representation of a given candidate solution to a Cheddar-ADL model using the `Cheddar ADL` library. It includes the functions-to-tasks assignment rules that determine and compute parameters of tasks, resources and critical sections of a given candidate solution (see Section 3.3). The mutation operator proposed for our problem (details in Section 3.2.4) is also implemented in this package. This package also defines a method that, from an initial solution (with chromosome representation), enumerates all the functions-to-tasks possible assignments (again with chromosome representations). This method is used to implement the exhaustive method.

The second package (`objective functions and feasibility checks`) is dedicated to the evaluation of solutions by way of objective functions (Section 3.2.1), and to the achievement of the feasibility checks on design alternatives (Section 3.4.3). This package uses the scheduling simulator library in order to perform the scheduling analysis and compute the objective functions from the resulting simulation (e.g. the number of preemptions, the number of context switches, WCRT of tasks and WCBTs, etc).

The implementation of our function sets generator (details are given in Section 5.1) extends the `Architecture generator` library of Cheddar. This library includes an Ada package that provides different methods to generate/build Cheddar-ADL models for different kinds of real-time architectures, thereby allowing to perform empirical studies related to scheduling analysis.

As we can see in this figure, our prototype provides two new tools allowing the execution of the PAES method and the exhaustive method for the architecture exploration problem that we address. Each one of these programs instantiates the general form of the corresponding algorithm from the `optimizers` library with sub-programs defined in the `functions-to-tasks assignment library`. These programs provide two ways of use. First, the user can define and give to the method to run the Cheddar-ADL model of the initial solution as argument to the program. Second, the user can configure the function sets generator through a set of parameters that he provides to the program in order to execute the desired method on a synthetically generated function sets.

5 Experimental Studies

In this section, we present three experiments carried out in order to investigate some aspects related to our problem as well as to evaluate our proposals.

First, we explore the correlation between three objective functions selected from the list aforementioned in Section 3.2.1.

Second, we evaluate the accuracy (in terms of convergence and coverage ability) of our PAES-based design exploration method. This evaluation is performed on small-size problem instances through a comparison with an exhaustive search method.

Last, we assess the effectiveness of our method for larger systems by investigating the quality of produced solution sets for different resources contention levels.

In order to perform experiments, we propose and implement a function set generator that produces customizable test instances of our design exploration problem. In those experiments, we use the parallel implementation with the global selection strategy of the PAES-based method that we proposed in (Bouaziz et al 2016). Experiments are conducted on a SMP machine with 48 processors at 2.2 GHz frequency and 125-GBytes of RAM, running Linux CentOS.

In the following, we describe our test instance generator in section 5.1. Afterwards, we give an overview of the performance metrics used for measuring the results and to assess our proposals (Section 5.2). In Sections 5.3, 5.4 and 5.5, we present the experiments protocol, the results and their analysis for the three experiments.

5.1 Test Instance Generator for the Design Exploration Problem

In order to perform the different experiments with a broad range of configurations, we propose to generate synthetically functional input models, i.e. sets of n functions interacting through shared resources.

5.1.1 Functions Parameters

Each function period of the n functions is uniformly distributed between a set of a predefined number n_k of a maximum of different periods per function set ($n_k < n$), following the method of Goossens and Macq (2001). This ensures that the scheduling simulations have to be run on limited feasibility intervals. Given a processor utilization U of a function set, the utilization factor u_i of each function F_i is generated using the UUnifast algorithm (Bini and Buttazzo 2005). We note that all random variables are generated with a uniform distribution. We assume implicit deadlines, i.e. deadlines of the functions are set to be equal to the periods. Finally, the capacities are computed based on the generated periods and processor utilization factors.

5.1.2 Resources and Critical Sections Parameters

Each function set handles a set of shared resources between them. Each resource R_j is accessed by a random number of functions in the range $[2, rsf * n]$ randomly selected from the set of functions. The *rsf* parameter (resource sharing factor) is used in order to ensure that there will be a sufficient number of resource conflicts (i.e. resource contention between the tasks) and to vary the resources contention according to the experiments.

We assume that each function F_i accessing a resource R_j, issues only a single request of R_j per job. The parameters of ω_k associated to the usage of R_j by F_i are adjusted as follows. The length of ω_k (i.e. the duration of usage of R_j by F_i) is defined as a percentage (called *critical section ratio*: *csr*) of the computation time of F_i: $Length(\omega_k) = csr * \gamma_i$. In addition to the resource sharing factor, the critical section ratio allows us to customize easily the resources contention in our experiments. The date of begin of ω_k is randomly selected within the function execution time.

5.2 Performance Metrics

In this section, we present the hypervolume and the inverted generational distance metrics that we need in the assessment of results.

Hypervolume Indicator (HV): The set of solutions provided by our PAES-based method are to be compared qualitatively. However, the evaluation of the optimization results of MOO techniques is not intuitive since one solution set (i.e. Pareto front) is not decidedly better than another. To this end, several *unary* metrics exist (Coello Coello et al 2007), that map a Pareto front to a single value thereby allowing an easy evaluation of the quality of the approximate Pareto fronts PF_{approx}. These metrics take into account both the closeness of the obtained solutions PF_{approx} to the optimal set PF_{true} (accuracy) and their spread across objective space (diversity). One of these metrics is the *hypervolume* indicator (Zitzler and Thiele 1998).

This latter allows to measure how well algorithms perform in identifying non-dominated solutions along the full extent of the objective space. Given a solution set (e.g. generated by a particular algorithm) and associated objective values (points in objective space), it computes the area of the objective space dominated by all the solutions bounded by a reference point as shown in Figure 13. For a given problem, the reference point is selected so that it is dominated by every solution in the fronts to be investigated. In our experiments, the reference point is formed using the worst value for each objective (known as the *anti-ideal* point) observed over all the produced PF_{approx} for a problem instance. The better the Pareto front, the larger the hypervolume will be.

Moreover, in order to allow the objectives to contribute approximately equally, objective values are normalized according to a linear normalization technique defined by Fonseca et al (2005): $f_i^{normalized} = \frac{f_i - f_i^{(min)}}{f_i^{(max)} - f_i^{(min)}}$

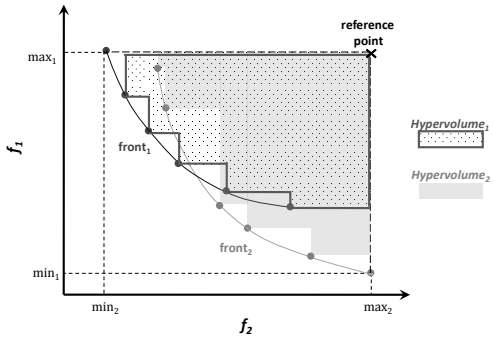


Fig. 13: Hypervolume performance indicator for two objectives problem

where $f_i^{(min)}$ and $f_i^{(max)}$ are the minimum and maximum values respectively, that the i^{th} objective is taking within the considered fronts. The reference point is also normalized. In our experiments, the hypervolume indicator is computed by the means of a dedicated tool provided by Fonseca et al (2006). With normalized values, hypervolume values are in the range [0..1].

Inverted Generational Distance Metric (IGD): The IGD metric (Van Veldhuizen and Lamont 1998; Zhang et al 2008) is usually adopted when the true Pareto front (PF_{true}) is known. It is defined by the following expression:

$$IDG(PF_{true}, PF_{approx}) = \frac{\sum_{\nu \in PF_{true}} d(\nu - PF_{approx})}{|PF_{true}|}$$

where $d(\nu - PF_{approx})$ is the euclidean distance between each solution ν of PF_{true} and the nearest member of PF_{approx}.

The IGD indicator measures not only the convergence of a generated Pareto front (PF_{approx}) to the true Pareto front (PF_{true}) but also the coverage ability of the algorithm (i.e. how well PF_{true} is covered by PF_{approx}). Hence, a PF_{approx} whose solutions are located on a limited area of the extent of the PF_{true}, will be penalized in its IGD value even if its solutions belong (or are too near) to PF_{true}. For the IGD metric, the lower are values, better are corresponding fronts.

In this work, in order to be able to compare the IGD values among different problem instances, we compute the IGD on normalized fronts. Both PF_{true} and PF_{approx} are normalized using the same technique applied in the computation of the hypervolume indicator. With normalized values, IGD values are in the range $[0..\sqrt{2}]$.

5.3 1^{st} Experiment: Empirical Study of the Correlation between Objectives

In a previous work (Bouaziz et al 2015), we have defined our architecture exploration method on limited functional models (i.e. independent functions)

while considering the preemptions and task laxities (defined by f_1 and f_4 respectively in Section 3.2.1) as optimization objectives. Although in (Bouaziz et al 2015) we did not investigate the conflict between these objectives, results of the performed experiments showed that they were effectively conflicting.

In the present work, we extend the functional input model to our methodology to consider interactions between functions by means of shared resources. Therefore we would like to involve at least one performance metric directly related to resource sharing as an optimization objective. We have already mentioned in Section 3.2.1 that, on the one hand, MOEA methods are often applied to problems with limited number of objectives. On the other hand, it is not obvious or intuitive to identify if two given objectives are conflicting or not. That is, we restrict the problem to three objectives: preemptions (f_1), tasks laxities (f_4) and blocking time (f_6). We perform a set of experiments to investigate the conflict relationship between each pair of these objectives.

The purpose of this experiment is to verify, when passing from independent function models to models with shared resources, whether it is necessary to consider three objectives or to remain with two objectives.

In order to be able to consider only two objectives, we need to show through experiments that the third objective (f_6) behaves in a non-conflicting way with either one or both of the other objectives (f_1 and f_4). Otherwise, even if experiments are realized onto a restricted set of test instances that will show that the three objectives are necessary, we must consider all of them.

In the remainder of this section, we describe first the applied method for measuring the correlation between objectives. Then, we give the experiments protocol and the parameters settings. Finally, we present results of the experiments.

5.3.1 Pearson Correlation Coefficient Analysis for Identifying Correlation between Objectives

Interactions arising between objectives are either a conflict or a support relation. In the case of a conflict relation, a change of a solution which yields to an improvement of one objective is seen to cause deterioration of a second objective. However, in the case of a support relation, a solution change causes at the same time either improvement or deterioration to both objectives (Purshouse and Fleming 2007).

Formally speaking, as defined by Carlsson and Fullér (1995), two objectives are in conflict means that one objective increases while the other decreases and vice versa. But, if they support each other then both objectives increase or decrease simultaneously. In the former case the objectives are said to be *negatively correlated*, and in the latter case they are *positively correlated*.

Measuring the conflict or the correlation among objectives is a baseline element of objective reduction approaches (Deb and Saxena 2006; López Jaimes et al 2008, 2009, 2014; Saxena et al 2013; Wang and Yao 2016) that deal with many-objective problems.

In order to estimate the correlation between the objectives, we suggest to use a simple method similar to those applied in (Deb and Saxena 2006; López Jaimes et al 2008, 2009, 2014). Thus, we rely on the *Pearson correlation coefficient*. This correlation coefficient r_{xy} measures the linear relationship between two objectives x and y through their observed data sets. By definition the correlation coefficient values are in the range $[-1, 1]$. When $r_{xy} > 0$, it is said that x and y are positively correlated, and when $r_{xy} < 0$, they are said to be negatively correlated. If $r_{xy} = 0$ they are not correlated.

The correlation between two objectives is computed using the approximation set of the Pareto front PF_{approx}, produced by our PAES-based method, as the data set. Each solution in PF_{approx} is an observation. The obtained correlation coefficient is associated with a $p - value$ (Westfall and Young 1993), representing the statistical risk of error on the correlation result significance. In our experiments, we use the standard threshold of 0.05: two objectives are considered as correlated when the corresponding $p - value$ is less than 0.05, and the nature of the correlation (positive or negative) is determined by the sign of their correlation coefficient r_{xy}.

5.3.2 1^{st} *Experiment : Protocol and Parameters*

A number of experiments were run in order to investigate the correlation between the objectives following the method detailed above by the mean of synthetically generated function sets (test instances).

The parameters settings of the function sets generator are given in Table 3. Experiments are performed for two function sets sizes: 20 functions and 30 functions. For each size, 50 different test instances are generated. In these experiments, some parameters are set for both function sets sizes: (i) the overall processor utilization factor is fixed at 0.8, (ii) each function period of a given function set is uniformly distributed between a set of $n_k = 5$ different periods randomly generated (iii) and the critical section ratio (csr) is selected randomly from $\{0.1, 0.3, 0.5\}$. In addition to the function set size, we vary the number of resources and the resource sharing factor (rsf) as shown in Table 3. Hence, there will arise a sufficient number of resource conflicts.

Each generated test instance is processed by the PAES-based method for 5 independent runs because of the random nature of PAES. The number of PAES iterations for each run is fixed at 2000. For a given test instance, the correlation coefficients of objectives pairs are computed over all the PF_{approx} generated by all the runs.

5.3.3 1^{st} *Experiment: Results*

Figure 14 reports for each pair of objectives (f_1, f_4), (f_1, f_6) and (f_4, f_6) the rates of each correlation kind (i.e. negative, positive or insignificant) captured over the total number of generated test instances (50 test instances with 20 functions and 50 test instances with 30 functions).

Table 3: 1^{st} experiment : synthetically generated function sets parameters

Common parameters		
Parameter	Value	
# generated test instances per function set size	50	
overall processor utilization	0.8	
# different periods per function set, n_k	5	
critical section ratio, csr	0.1, 0.3, 0.5	
Per function set size parameters		
	20 fcts	30 fcts
# resources	6	10
resource sharing factor, rsf	0.2	0.25

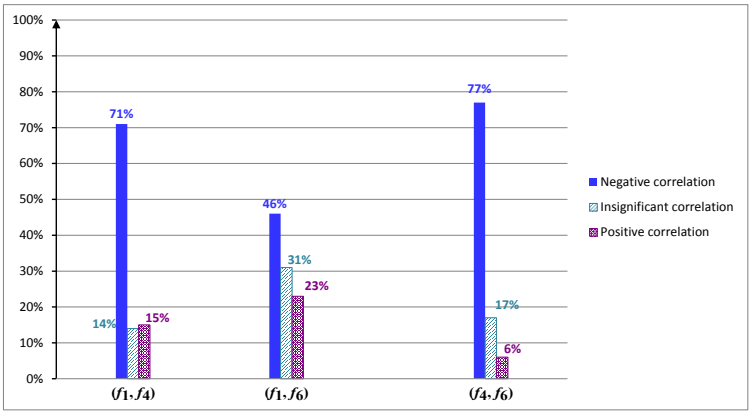


Fig. 14: Negative, positive and insignificant correlation rates between the objectives pairs over all the generated test instances

We can observe that the presence of the conflict relationship among pairs of objectives is more important for some pairs than others. As shown in this figure, both pairs of objectives (f_1, f_4) and (f_4, f_6) present a conflict relationship for most of the studied test instances (more than 70% of the test instances), whereas for (f_1, f_6), only 46% of the test instances present a conflict relationship. We can notice also that the original objectives f_1 and f_4 are sometimes (for 15% of test instances) supporting each other. This is explained by the fact that for these test instances the added objective f_6 is in conflict with each one of the former objectives.

However, none of the objective pairs presents a support relationship for a major part of the processed test instances, since the best supporting ratio is 23% for the couple (f_1, f_6). Thus, the hypothesis of remaining with only two objectives is not relevant and we must consider the three objectives. Therefore, in our problem the conflict among the objectives depends on the

addressed problem instance and cannot be decided in an absolute way (i.e. for any instance of our problem).

5.4 2^{nd} Experiment: Accuracy Evaluation of the PAES-based Architecture Exploration Method for Small-Sizes Problem Instances

The second experiment aims at assessing the convergence and coverage of our PAES-based architecture exploration method. The IGD indicator (defined in Section 5.2) is applied for this evaluation.

The computation of the IGD requires the knowledge of the true Pareto front. Thus, we propose a method that determines the true Pareto front through an exhaustive search among all functions-to-tasks assignment solutions. The exhaustive method works as follows:

(1) Enumerate all possible assignment solutions for a given function set. The number of solutions is equal to the *Bell* number (Rota 1964) of the processed function set size.
(2) Determine feasible solutions among all the enumerated solutions (i.e, applying Algorithm 4).
(3) Evaluate solutions and identify schedulable ones
(4) Finally, compute the Pareto front from schedulable solutions according to the dominance concept.

However, as it was already mentioned in the problem statement (Section 1), we deal with a combinatorial problem. An exhaustive search as the above proposed method can be applied only for small-size instances of our problem since the search space size increases exponentially with the problem instance size (i.e. the number of functions).

With the exhaustive method, we can handle, in a reasonable amount of time, problem instances up to 10 functions. For a problem instance of 10 functions, the number of all possible assignment solutions to proceed with the exhaustive method is equal to the 10^{th} *Bell number*, $B_{10} = 115975$. This is already an important number of solutions that will take a considerable time to be proceeded by the exhaustive method. However, if we want to increase the problem instances to 11 functions, the number of possible solutions is equal to $B_{11} = 678570$ which is unmanageable by the exhaustive method.

5.4.1 2^{nd} Experiment: Protocol and Parameters

We perform several experiments. We take into account problem instances with 9 and 10 functions. For each size, 30 different test instances are generated using our function sets generator. The parameters of the latter for the performed experiments of both sizes, are detailed in Table 4.

The small size of the test instances and the constraints that they must fulfil may lead to get PF_{true} with a single solution. These cases are ignored

Table 4: 2^{nd} Experiment: synthetically generated function sets parameters settings

Parameter	Value
# generated test instances per function set size	30
overall processor utilization	0.5
# different periods per function set, n_k	2
# resources	7
critical section ratio, csr	0.5
resource sharing factor, rsf	0.5

and we consider only test instances with PF_{true} containing at least two non-dominated solutions.

According to the first experiment (Section 5.3), we consider in the current evaluation, the three objectives f_1, f_4 and f_6. At a first stage, each generated test instance is processed by the exhaustive method in order to produce its PF_{true}. Then, we apply our PAES-based method for 30 independent runs. Each IGD value shown in this experiment is the average over the 30 runs. The number of PAES iterations for each run is fixed at 3000.

5.4.2 2^{nd} Experiment: Results

Table 5 and Table 6 show the results for test instances of 9 functions and 10 functions respectively, with respect to the average IGD, the size of the PF_{true} (i.e. the number of non-dominated solutions) and the size of the PF_{approx}. For the latter, we present the average, the minimum and the maximum over all the runs.

Table 5: The inverted generational distance (IGD) relative to 30 synthetically generated test instances with 9 functions

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$	
			Avg.	Min - Max
1	**0.0**	3	3.0	3 - 3
2	**0.0**	2	2.0	2 - 2
3	**0.0**	3	3.0	3 - 3
4	**0.0**	2	2.0	2 - 2
5	**0.0**	12	12.0	12 - 12
6	**0.0**	2	2.0	2 - 2
7	**0.0**	7	7.0	7 - 7
8	**0.0**	2	2.0	2 - 2
9	**0.0**	12	12.0	12 - 12
10	**0.0**	3	3.0	3 - 3
11	**0.0**	4	4.0	4 - 4
12	**0.0**	4	4.0	4 - 4
13	**0.0**	2	2.0	2 - 2
14	**0.0**	5	5.0	5 - 5
15	**0.0**	4	4.0	4 - 4

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$	
			Avg.	Min - Max
16	**0.0**	5	5.0	5 - 5
17	**0.0**	2	2.0	2 - 2
18	0.0002	12	12.0	12 - 12
19	0.0022	18	17.86	17 - 18
20	0.0084	14	13.16	12 - 14
21	0.0131	6	6.0	6 - 6
22	0.0327	8	7.23	7 - 8
23	0.1237	9	7.0	7 - 7
24	0.2589	4	3.06	3 - 4
25	0.2654	15	6.0	6 - 6
26	0.3106	7	9.0	9 - 9
27	0.3334	3	2.0	2 - 2
18	0.3535	4	2.0	2 - 2
29	0.7043	6	2.0	2 - 2
30	0.7071	2	1.0	1 - 1

Table 6: The inverted generational distance (IGD) relative to 30 synthetically generated test instances with 10 functions

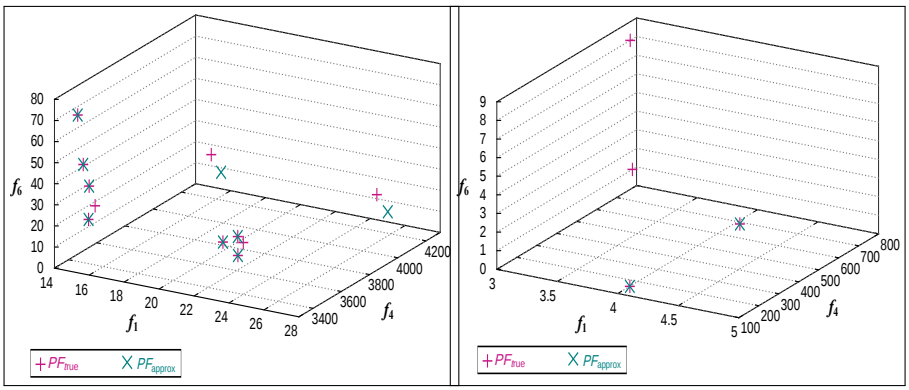
Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$	
			Avg.	Min - Max
1	**0.0**	2	2.0	2 - 2
2	**0.0**	3	3.0	3 - 3
3	**0.0**	3	3.0	3 - 3
4	**0.0**	5	5.0	5 - 5
5	**0.0**	12	12.0	12 - 12
6	**0.0**	3	3.0	3 - 3
7	**0.0**	2	2.0	2 - 2
8	**0.0**	3	3.0	3 - 3
9	**0.0**	2	2.0	2 - 2
10	**0.0**	4	4.0	4 - 4
11	**0.0**	2	2.0	2 - 2
12	**0.0**	9	9.0	9 - 9
13	**0.0**	9	9.0	9 - 9
14	**0.0**	6	6.0	6 - 6
15	**0.0**	5	5.0	5 - 5

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$	
			Avg.	Min - Max
16	**0.0**	3	3.0	3 - 3
17	0.0005	26	25.93	25 - 26
18	0.0454	10	9.03	9 - 10
19	0.0431	18	16.3	15 - 18
20	0.1209	6	5.03	5 - 6
21	0.1742	12	9.40	6 - 12
22	0.1659	7	4.16	4 - 5
23	0.1670	14	8.33	8 - 14
24	0.1786	11	7.33	7 - 9
25	0.2214	4	3.16	3 - 4
26	0.3479	9	4.1	4 - 7
27	0.4886	7	3.0	3 - 3
28	0.5909	5	3.03	3 - 4
29	0.6224	4	2.0	2 - 2
30	0.7071	2	1.0	1 - 1

In order to ease the readability and the analysis of the results, in both tables, we sort the test instances in ascending order of their IGD and we divide the results into 3 classes according to IGD ranges:

- The first class encloses test instances holding a *zero* IGD value, which means that the PAES-based method succeeds in converging to the exact PF_{true} of these test instances in all the runs. This class represents 55% of all the presented test instances (17 test instances in Table 5 and 16 test instances in Table 6). Moreover, as we can see in both tables, the number of solutions in PF_{true} for about 30% of test instances of this class is greater than or equal to 5 solutions. This shows the ability of our PAES-based method to handle instances with diversified Pareto front.
- The second class contains test instances with *low* IGD values that belong to the range $[0.0002, 0.05]$ and it includes about 13% over the 60 studied test instances. Low IGD values may be explained by the fact that for test instances of this second class, PAES is able to find solutions that belong to PF_{true} by an average rate about: 94.52% for test instances of the second class in Table 5 and 77.89% for test instances of the second class in Table 6.
- The third class is defined for test instances with relatively *medium* and *high* IGD values that belong to the range $[0.1, 0.7]$. We find in this class around 32% of the studied test instances. For this class the average rates of solutions found by PAES and that belong to PF_{true} are about: 56.44% for test instances of the third class in Table 5 and 53.86% for test instances of the third class in Table 6. The comparison of these rates with those obtained for the second class justify the increasing of the IGD values in the third class with regard to the second one.
  Furthermore, the range of IGD values of this class which is between 0.1 and 0.7 (i.e. a not tight range) may be due to the variation of the coverage ability of PAES among test instances. For example, let consider from

Table 6, test instance 24 having an IGD of 0.1786 and test instance 29 with an IGD of 0.6224. As we can see for these two test instances PAES misses 2 solutions: for the former it can find at the best runs 9 solutions among 11 in the PF_{true} and for the latter it achieves at best 2 solutions among 4 in the PF_{true}. Even though the number of missed solutions by PAES is the same for the two test instances, their IGD are different. As shown in Figure 15, in test instance 24 (Subfigure 15a) PAES covers all the Pareto area, whereas in test instance 29 (Subfigure 15b) PAES misses a non-negligible area of the Pareto space. Hence, the test instance 29 is penalized in its IGD value (0.6224).



(a) PF_{true} and PF_{approx} of test instance 24 of Table 6

(b) PF_{true} and PF_{approx} of test instance 29 of Table 6

Fig. 15: Comparison of the PAES coverage ability between two test instances

The results obtained with PAES for class one and two that represent 68% of the studied test instances, can be considered as good, since their IGD values are less than 0.04 and we know that the upper bound value of normalized IGD is $\sqrt{2}$ (≈ 1.4142).

Moreover, we have measured the runtime of both methods for the 60 studied test instances. For the PAES method we consider the average runtime over all the runs. Then, for each test instance we computed the runtime reduction rate of the PAES method against the exhaustive method: $(1-\frac{T_{PAES}}{T_{Exhaustive}})*100$. The average runtime reduction rate over all the test instances is about 74% which shows that the PAES method outperformed the exhaustive method with respect to the runtime. Furthermore, we notice that the runtime comparison results between the two methods are better for instances with 10 functions than 9 functions. This is due the fact that on one side, the exhaustive method spends more time for test instances with 10 functions than 9 functions, since the search space is larger ($B_{10} = 115975$ *vs* $B_9 = 21147$). On the other side, we run the PAES method for a fixed number of iterations for both test instances sizes.

5.5 3^{rd} Experiment: Quality Evaluation of Solution Sets for Different Resources Contention Levels

In previous works (Bouaziz et al 2015, 2016), we have already evaluated our architecture exploration method for independent functions systems by assessing the quality of produced solution sets when varying the systems size up to 100 functions. We have showed that our method not only scales well but also makes effectively the architecture exploration and provides a set of trade-offs for the designer (e.g. the average hypervolume is about 0.78 for test instances with 100 functions).

In this experiment, we are interested in assessing the effectiveness of our method by investigating the quality of the generated solution sets for different resources contention levels. The solution sets obtained in the current experiments are to be evaluated quantitatively (by way of number of non-dominated solutions found) and qualitatively (using the hypervolume performance indicator defined in Section 5.2).

5.5.1 3^{rd} Experiment: Protocol and Parameters

For the experiments, we set the parameters of the function sets generator as detailed in Table 7. Experiments are performed for three resources contention levels. For each level, 20 different test instances are generated. Some parameters are set commonly for the three resources contention levels: (i) the number of functions per function set is fixed at 30, (ii) the overall processor utilization factor is set at 0.8, (iii) each function period of a given function set is uniformly distributed between a set of $n_k = 5$ different periods randomly generated. The resources contention variation is performed by varying the number of resources, the resource sharing factor and the critical section ratio. As shown in Table 7, the function sets associated to $level_1$ (respectively $level_2$, $level_3$) will have a low (respectively medium, high) resources contention level.

Table 7: 3^{rd} Experiment: synthetically generated function sets parameters settings

Common parameters settings			
Parameter	Value		
# functions per test instance	30		
# generated test instances per resources contention level	20		
overall processor utilization	0.8		
# different periods per function set, n_k	5		
Per resources contention level parameters settings			
	$level_1$	$level_2$	$level_3$
# resources	5	10	15
resource sharing factor, rsf	0.1	0.2	0.3
critical section ratio, csr	0.1	0.3	0.5

As in the second experiment (Section 5.4), we take into account the three objectives f_1, f_4 and f_6. The test instances of each level are processed by the PAES-based method for 10 independent runs. The number of PAES iterations for each run is set at 3000.

5.5.2 3^{rd} Experiment: Results

Table 8, Table 9 and Table 10 present the results for the processing of 20 test instances relative to each of the three considered resources contention levels, with respect to the hypervolume (average and standard deviation values over all runs) and the size of the PF_{approx} (average, minimum and maximum values over all the runs). In order to ease the readability and the analysis of the results, we sort the test instances in descending order of their hypervolume values, in the three tables.

Let consider Table 8, where the resources contention level of the studied test instances is low. We notice that hypervolume values relative to 50% of test instances are in the range of $[0.53, 0.84]$, whereas the second half of test instances have zero hypervolume value. Zero hypervolume value is due to the reduced resources contention that results in zero values for the blocking time of tasks, thereby leading to zero values of the third objective f_6 of all solutions in fronts associated to these test instances. For these test instances, only objectives f_1 and f_4 present a conflict relationship, which induces the reduction of the search space dimension (2 instead of 3). For test instances with non-zero hypervolume values (i.e. the first part of Table 8), we can see that the hypervolume values are greater than 0.5 which means that the obtained solution sets are well spread in the part of the objective space that has been explored, since the hypervolume value upper bound is 1.0.

The average hypervolume over test instances displayed in Table 9 and Table 10 are about 0.6915 and 0.7250 respectively which represent relatively large hypervolume values. This is explained by the large size of the search space associated to these test instances induced by the magnitude of the resources contention. Hence, the number of feasible solutions is so important, thereby resulting in large fronts. This is reinforced by the number of solutions in the obtained fronts that is quite high: up to 147 in Table 9 and 196 in Table 10.

Besides, when we inspect the size of PF_{approx} in the three tables, we notice that the number of solutions is so different between runs associated to each test instance. This variation is more important in Table 9 and Table 10. Again, this is due to the increase of the search space size with the resources contention rise. This will cause a variation in the PAES behavior from one run to another for the same test instance. For a given test instance, despite the variation in the number of solutions between runs, the standard deviation of the hypervolume values over the runs is relatively low (8.75% in the worst case), showing that the quality of the produced fronts is still good and roughly stable over the different runs.

We have computed the runtime of the 20 test instances of each resources contention level. Table 11 shows the average, the minimum and the maximum

Table 8: Hypervolume values and Pareto front sizes relative to 20 generated test instances with the resources contention $level_1$

Test	Hypervolume		$\|PF_{approx}\|$		Test	Hypervolume		$\|PF_{approx}\|$	
instances	Avg.	Std.	Avg.	Min - Max	instances	Avg.	Std.	Avg.	Min - Max
1	**0.8487**	0.0127	18.80	12 - 24	11	0.0	0.0	4.00	4 - 4
2	**0.8117**	0.0247	7.00	6 - 9	12	0.0	0.0	3.00	3 - 3
3	**0.8015**	0.0113	15.30	13 - 18	13	0.0	0.0	8.00	7 - 9
4	**0.7595**	0.0663	25.40	21 - 32	14	0.0	0.0	3.80	3 - 4
5	**0.7357**	0.1316	5.00	4 - 8	15	0.0	0.0	2.40	2 - 5
6	**0.6746**	0.0406	10.40	6 - 17	16	0.0	0.0	7.00	7 - 7
7	**0.6551**	0.0290	13.00	11 - 16	17	0.0	0.0	8.10	7 - 9
8	**0.6221**	0.0555	8.50	7 - 10	18	0.0	0.0	3.00	3 - 3
9	**0.5868**	0.0252	17.90	13 - 21	19	0.0	0.0	5.80	5 - 6
10	**0.5339**	0.0000	8.70	8 - 9	20	0.0	0.0	8.30	7 - 10

Table 9: Hypervolume values and Pareto front sizes relative to 20 generated test instances with the resources contention $level_2$

Test	Hypervolume		$\|PF_{approx}\|$		Test	Hypervolume		$\|PF_{approx}\|$	
instances	Avg.	Std.	Avg.	Min - Max	instances	Avg.	Std.	Avg.	Min - Max
1	**0.8337**	0.0154	25.80	19 - 36	11	**0.6976**	0.0396	41.80	24 - 56
2	**0.8336**	0.0332	**79.20**	**59 - 101**	12	**0.6559**	0.0276	25.40	22 - 28
3	**0.8142**	0.0173	**90.60**	**74 - 119**	13	**0.6527**	0.0498	39.40	22 - 46
4	**0.7896**	0.0208	**60.00**	**51 - 72**	14	**0.6474**	0.0335	**64.30**	**50 - 99**
5	**0.7835**	0.0375	15.00	11 - 22	15	**0.6164**	0.0400	**115.20**	**68 - 147**
6	**0.7723**	0.0220	35.50	33 - 40	16	**0.6089**	0.0313	19.70	16 - 23
7	**0.7699**	0.0589	**103.60**	**68 - 132**	17	**0.5970**	0.0505	34.40	28 - 44
8	**0.7420**	0.0324	34.20	26 - 42	18	**0.5801**	0.0257	**82.90**	**62 - 98**
9	**0.7360**	0.0315	19.30	14 - 28	19	0.4886	0.0015	21.00	21 - 21
10	**0.7275**	0.0096	5.50	5 - 6	20	0.4838	0.0339	21.50	16 - 28

Table 10: Hypervolume values and Pareto front sizes relative to 20 generated test instances with the resources contention $level_3$

Test	Hypervolume		$\|PF_{approx}\|$		Test	Hypervolume		$\|PF_{approx}\|$	
instances	Avg.	Std.	Avg.	Min - Max	instances	Avg.	Std.	Avg.	Min - Max
1	**0.8802**	0.0041	**120.70**	**97 - 150**	11	**0.7298**	0.0381	**113.70**	**42 - 150**
2	**0.8551**	0.0165	**145.20**	**117 - 158**	12	**0.7195**	0.0558	46.40	30 - 60
3	**0.8507**	0.0109	**163.20**	**148 - 186**	13	**0.6866**	0.0875	**89.50**	**57 - 109**
4	**0.8311**	0.0155	30.80	23 - 35	14	**0.6832**	0.0337	36.70	28 - 43
5	**0.7814**	0.0252	**111.10**	**75 - 138**	15	**0.6700**	0.0617	**173.00**	**146 - 196**
6	**0.7725**	0.0539	**96.50**	**50 - 148**	16	**0.6654**	0.0298	**102.00**	**81 - 127**
7	**0.7667**	0.0129	**102.90**	**67 - 155**	17	**0.6538**	0.0515	47.90	41 - 54
8	**0.7546**	0.0197	**75.90**	**63 - 88**	18	**0.6311**	0.0164	**146.80**	**123 - 186**
9	**0.7407**	0.0453	39.60	28 - 57	19	**0.5723**	0.0361	**144.80**	**131 - 165**
10	**0.7328**	0.0488	**141.50**	**128 - 160**	20	**0.5222**	0.0626	**64.80**	**53 - 77**

runtime over the test instances of each resources contention level. It gives also the simulation period (SP) of the test instances associated to the Min and the Max runtime values of each level. We can see in this table that for each level, the difference between the minimum and the maximum runtime values is about $2h\ 26min$, $2h\ 44min$ and $3h\ 45min$ for $level_1$, $level_2$ and $level_3$ respectively. This runtime variation is caused by the variation of the simulation period between the generated test instances as shown in Table 11. Furthermore, we

can observe that the runtime increases when the resources contention level raises. Let consider the maximum runtime values of the three levels. We note that even though the simulation periods of $level_1$ ($SP = 27720$) and $level_2$ ($SP = 27720$) are greater than $level_3$ ($SP = 13860$), the maximum runtime relative to $level_3$ is more important. This can be explained by the fact that the rise of the resources contention will increase the number of events to take into consideration in the simulation, thereby spending more time to compute the schedule of each candidate design by simulation. Besides, the shown runtime values for the three levels indicate that our method is able to handle problem instances with reasonable system size (i.e. 30 functions for the studied test instances) and high resources contention.

Table 11: Runtime computed over all test instances generated for each resource contention level

Resources contention level	Avg.	Min	Max
$Level_1$	41min 6s	6min (SP=4620)	2h 32min 8s (SP=27720)
$Level_2$	1h 27min 54s	29min 13s (SP=13860)	3h 13min 30s (SP=27720)
$Level_3$	3h 13min 14s	1h 17min 25s (SP= 2520)	5h 3min 12s (SP=13860)

To conclude, although this experiment does not address explicitly the accuracy of the generated fronts (contrary to the second experiment, where true Pareto fronts can be computed), the results in terms of hypervolume and number of solutions (e.g. for the studied system instances with high resource contention, $HV \in [0.52, 0.88]$ and $\#solutions \in [30, 173]$) show that our method makes effectively the architecture exploration for systems with shared resources by generating a set of promising design trade-offs.

6 Related Work

In this section, we present the related work aiming at driving the design of real-time critical systems, followed by works that deal with the design exploration using multi-objective optimization techniques.

Then, we introduce some software architecture design approaches based on clustering techniques.

In the literature, many approaches have been proposed to drive the design of architectural models that need meet timing requirements, by mapping a functional specification to a multi-tasking architecture.

In (Bartolini et al 2005), authors developed heuristic algorithms that generate the architectural model from a dataflow functional model with timing properties. The main objective of this work is to automate the functional to architectural mapping.

Authors of (Pagetti et al 2011) provide a framework for the integration and the development of real-time embedded systems. With this framework, designers write a functional specification with dependency constraints using the *Prelude* language. From this model, the proposed framework allows to generate a set of real-time tasks that can be executed on a uniprocessor architecture. Authors proved that the generated implementation enforces the system behavior as well as timing constraints described in the functional specification.

Furthermore, a MARTE-based methodology was proposed in (Mraidha et al 2011), enabling scheduling analysis at early stages of the software life cycle. It allows designers to generate, from a functional specification expressed with UML MARTE, a design model compliant with the functional specification timing requirements.

Unlike these works, we propose to explore various architecture alternatives using a clustering technique for assigning functions to tasks. We perform scheduling analysis on each architecture alternative and those that meet the timing constraints are evaluated with regard to a set of competing performance criteria (e.g. #preemptions, tasks laxities, blocking time of tasks, etc.), to finally select good trade-offs.

In distributed AUTOSAR (AUTomotive Open System ARchitecture) systems (Fürst et al 2009), functions-to-tasks assignment is known by *runnables-to-tasks mapping* and is identified as a primordial step in the design process of such systems (Scheickl and Rudorfer 2008; Fürst et al 2009). Existing methods in this context (Monot et al 2012; Mehiaoui et al 2013; Wozniak et al 2013) aim at automating the integration of functional specification defined by the runnables and the data signals exchanged between them. The AUTOSAR platform consists of a network of distributed execution nodes called ECUs (Electronic Control Unit) connected through buses and OS tasks/messages as well.

Investigating runnables-to-tasks assignments raises two issues. The first one consists in assigning runnables to cores taking into account inter-runnable dependencies and locality constraints while optimizing the core load. The second one is to build the sequencing of the runnable entities of each core using one task per core.

In (Monot et al 2012) multicores ECUs architectures are considered and two heuristics are proposed to solve those two issues.

Authors of (Mehiaoui et al 2013; Wozniak et al 2013) presented also a two steps approach aiming at optimizing the runnables-to-tasks mapping with respect to a set of optimization metrics such as end-to-end response time, memory consumption, bus throughput, etc. In the first step runnables/data signals are partitioned on ECUs/buses afterwards runnables/data signals of each ECU/bus are assigned to tasks/messages.

The two issues are abstracted and resolved using different theories and optimization techniques, namely with a mixed integer linear programming (MILP) and a genetic algorithm (GA) as well.

These approaches define multi-criteria design exploration methods (as they involve several performance metrics), but they don't use MOO techniques dedicated for such kind of problems.

In (Mehiaoui et al 2013; Wozniak et al 2013), criteria driving the search are combined into a weighted objective function, and thereby the result of the exploration is a single design solution instead of a set of design alternatives that exhibits trade-offs between criteria. In order to perform an effective design exploration, authors of these works proposed to iterate the exploration while varying objectives weights. However, such exploration method could in some cases miss interesting solutions that do not fit with any weights combination (solutions known as *unsupported* solutions (Coello Coello et al 2007)). Contrary to these approaches, we propose a multi-criteria design exploration method based on a dedicated MOO technique.

Some research contributions have been developed in the scope of design exploration using multi-objective optimization techniques. Most of them are based on MOEAs.

In (Rahmoun et al 2015a,b), the authors proposed a method that explores architecture alternatives for real-time embedded systems, in order to produce architectures that fulfil at best a set of conflicting non-functional properties stemmed from the requirements of the addressed systems. This method is based on model transformations composition and MOEA by means of the NSGA-II multi-objective optimization strategy (Deb et al 2002).

Koziolek et al. (Koziolek et al 2011) developed a framework called Peropteryx. This framework assists software architects during the design stage to approximate the Pareto set architectures. The exploration and selection processes are driven by architectural strategies and NSGA-II.

AQOSA (Li et al 2011) is another generic framework that provides an automated design exploration process based on a set of MOEAs namely NSGA-II, SPEA2, and SMS-EMOA. Unlike these generic frameworks, we address a specific architecture exploration problem from a scheduling point of view, for critical Ravenscar compliant real-time systems. In our work, the exploration process is guided by competing performance attributes stemmed from the scheduling context.

Our approach is also related to clustering techniques that we apply in order to explore and evaluate several design solutions. Clustering techniques are used in different contexts and for different purposes.

In (Bertout et al 2014), authors aim at minimizing overheads (timing and memory) of architecture composed of a large number of tasks by reducing the number of tasks through clustering. A heuristic is proposed to automatically reduce the number of tasks while preserving timing requirements. Contrasting to our clustering method, the clustering among tasks is restricted to tasks with identical periods.

Clustering tasks is also investigated in (Santinelli et al 2014), in order to tackle the scheduling of a large set of tasks with precedence constraints in a uniprocessor system. The authors propose to group tasks together according to their functional properties. Furthermore, in order to ensure the refinement of

the architectural model of a real-time application to a specific RTOS, authors of (Mzid et al 2013) aim at reducing the number of priority levels used by the application in order to comply with the number of priority levels provided by the targeted RTOS. They proposed a task clustering method based on a MILP formulation.

Finally, in the context of distributed systems, several approaches (Ramamritham 1995; Guodong et al 2003; Ahmadinia et al 2003; Palis et al 1996) seek to reduce communication costs through clustering techniques. In this context, clustering is applied as part of partitioning heuristics. They usually expect to minimize the overall execution time. A cluster can there be a set of tasks allocated to the same processor. Tasks are assigned to clusters in order to ensure schedulable task sets with minimum communication costs. In our article, we do not focus on distributed systems as we assume uniprocessor architectures.

To summarize, our method takes advantage of some of the strengths of approaches cited above: it combines the two fields of (1) clustering techniques and (2) multi-objective optimization techniques.

7 Conclusion and Future Work

In this article we proposed a design exploration method based on multi-objective optimization for real-time Ravenscar compliant systems.

This method allows to explore the design space by investigating numerous functions-to-tasks assignment alternatives in order to select those that meet at best a set of competing performance criteria.

The final selected designs enforce the system behavior and the timing requirements described in the functional specification.

A clustering technique is used in order to explore the design search space. To tackle the combinatorial issue of the addressed problem, our method relies on a multi-objective optimization evolutionary algorithm, namely the Pareto Archived Evolution Strategy.

Our method was implemented and integrated in Cheddar, an existing scheduling framework. The main benefit of this implementation is its reusability and the extensibility of its software artifacts. For example, the optimizers library could be reused in different problems or extended with other optimization methods. Similarly, the engine dedicated to the specification of the functions-to-tasks assignment is reusable in the instantiation of our problem with other MOEAs, etc.

In this article, we also presented results of experimental studies performed to investigate some features stemmed from our problem and assess the effectiveness of our approach. The experiments are made on synthetically generated problem instances.

We performed an empirical study aiming at investigating the correlation between pairs of objectives (i.e. performance criteria). Our experiments show that the correlation depends on the addressed problem instance.

Besides, in order to assess the accuracy of the proposed method, the solutions sets produced by our method were compared against an exhaustive method results. The experiments for this evaluation were realized on small size problem instances.

Results of these experiments showed that the proposed method can produce the optimal solution sets for 55% of the studied instances, and for other instances (about 13%) results were very close to optimal solutions set (i.e. instances with low IGD values, see Section 5.4.2).

A runtime comparison study showed that our method outperforms the exhaustive method by about 74% of runtime reduction.

For larger systems with different resources contention levels, a second evaluation was achieved in order to assess the quality of produced solutions sets. Results of experiments approved that our method made effectively the design exploration and generated meaningful trade-offs. For example, with high resources contention level, the average hypervolume and the average number of solutions over 20 test instances were about 0.725 and 100 solutions respectively.

Currently, we used a simple method to investigate the correlation between objectives based on the Pearson coefficient (linear correlation) and associated p-value. As part of our future work, we want to employ more sophisticated tools dealing with both linear and non-linear correlations as those defined in (Saxena et al 2013; Wang and Yao 2016).

In the same context, in view of the objectives correlation study results, we plan to apply an objective reduction method that uses the conflict information during the search process like in (López Jaimes et al 2009, 2014; Saxena et al 2013; Wang and Yao 2016). Such methods are defined to address many-objective problems. Hence with an objective reduction method, it will be possible to consider all the objectives listed in Section 3.2.1 or even other user defined objectives and then non-redundant objectives (i.e. most conflicting) will be determined during the algorithm progress.

Although the proposed method achieved promising results, it would be interesting to formulate the problem with metaheuristics other than PAES.

In addition to that we also aim to extend our approach in order to take into account more complex systems as multi-processor real-time systems.

Acknowledgements Cheddar is supported by Brest Métrople, Ellidiss Technologies, CR de Bretagne, CG du Finistère, and Campus France.

References

Ahmadinia A, Bobda C, Teich J (2003) Temporal task clustering for online placement on reconfigurable hardware. In: Proceedings of the IEEE International Conference on Field-Programmable Technology, IEEE, pp 359–362

Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ (1993) Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal 8(5):284–292

Bandyopadhyay S, Saha S (2013) Some single- and multiobjective optimization techniques. In: Unsupervised Classification, Springer Berlin Heidelberg, pp 17–58

Bartolini C, Lipari G, Di Natale M (2005) From functional blocks to the synthesis of the architectural model in embedded real-time applications. In: Proceedings of the 11^{th} IEEE Real Time and Embedded Technology and Applications Symposium, pp 458–467

Bertout A, Forget J, Olejnik R (2014) Minimizing a real-time task set through task clustering. In: Proceedings of the 22^{nd} International Conference on Real-Time Networks and Systems, ACM, pp 23–31

Bini E, Buttazzo GC (2005) Measuring the performance of schedulability tests. Real-Time Systems 30(1-2):129–154, DOI 10.1007/s11241-005-0507-9

Bouaziz R, Lemarchand L, Singhoff F, Zalila B, Jmaiel M (2015) Architecture exploration of real-time systems based on multi-objective optimization. In: Proceedings of the 20^{th} International Conference on Engineering of Complex Computer Systems, pp 1–10

Bouaziz R, Lemarchand L, Singhoff F, Zalila B, Jmaiel M (2016) Efficient parallel multi-objective optimization for real-time systems software design exploration. In: Proceedings of the 27^{th} International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, ACM, New York, NY, USA, pp 58–64

Burns A (1999) The ravenscar profile. ACM SIGAda Ada Letters 19(4):49–52

Buttazzo G (2011) Hard real-time computing systems: predictable scheduling algorithms and applications, vol 24. Springer Science & Business Media

Carlsson C, Fullér R (1995) Multiple criteria decision making: The case for interdependence. Computers & Operations Research 22(3):251–260

Coello Coello CA, Lamont GB, Veldhuizen DAV (2007) Evolutionary algorithms for solving multi-objective problems. Springer Science & Business Media

Deb K (2001) Multi-objective optimization using evolutionary algorithms, vol 16. John Wiley & Sons

Deb K, Saxena D (2006) Searching for pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems. In: Proceedings of the World Congress on Computational Intelligence, pp 3352–3360

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. Evolutionary Computation, IEEE Transactions on 6(2):182–197

Feiler PH, Gluch DP (2012) Model-based engineering with AADL: An Introduction to the SAE architecture analysis & design language. Addison-Wesley

Fonseca CM, Knowles JD, Thiele L, Zitzler E (2005) A tutorial on the performance assessment of stochastic multiobjective optimizers. In: Proceedings of the 3^{rd} International Conference on Evolutionary Multi-Criterion Optimization, vol 216, p 240

Fonseca CM, Paquete L, López-Ibánez M (2006) An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE Congress on Evolutionary Computation, IEEE, pp 1157–1163

Fürst S, Mössinger J, Bunzel S, Weber T, Kirschke-Biller F, Heitkämper P, Kinkelin G, Nishikawa K, Lange K (2009) Autosar–a worldwide standard is on the road. In: the 14^{th} International VDI Congress Electronic Systems for Vehicles, Baden-Baden, vol 62

Gal T, Hanne T (1999) Consequences of dropping nonessential objectives for the application of mcdm methods. European Journal of Operational Research 119(2):373–378

Goossens J, Macq C (2001) Limitation of the hyper-period in real-time periodic task set generation. In: Proceedings of the Real-Time Embedded Systems

Goossens J, Grolleau E, Cucu-Grosjean L (2016) Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. Journal of Real-Time Systems

Guodong L, Daoxu C, Daming W, Defu Z (2003) Task clustering and scheduling to multiprocessors with duplication. In: Proceedings of the International Parallel and Distributed Processing Symposium, IEEE

Hruschka ER, Campello RJGB, Freitas AA, De Carvalho ACPLF (2009) A survey of evolutionary algorithms for clustering. Trans Sys Man Cyber Part C 39(2):133–155, DOI 10.1109/TSMCC.2008.2007252

Klein MH, Ralya T, Pollak B, Obenza R, Harbour MG (1993) A practitioners handbook for real-time analysis: guide to rate monotonic analysis for real-time systems. Springer US

Knowles JD, Corne DW (2000) Approximating the nondominated front using the pareto archived evolution strategy. Evolutionary Computation 8(2):149–172, DOI 10.1162/106365600568167

Koziolek A, Koziolek H, Reussner R (2011) Peropteryx: automated application of tactics in multi-objective software architecture optimization. In: Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS, ACM, pp 33–42

Leung JYT, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation 2(4):237–250

Li R, Etemaadi R, Emmerich MT, Chaudron MR (2011) An evolutionary multiobjective optimization approach to component-based software architecture design. In: IEEE Congress on Evolutionary Computation, IEEE, pp 432–439

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61, DOI 10.1145/321738.321743

López Jaimes A, Coello Coello CA, Chakraborty D (2008) Objective reduction using a feature selection technique. In: Proceedings of the 10^{th} Annual Conference on Genetic and Evolutionary Computation, ACM, pp 673–680

López Jaimes A, Coello Coello CA, Urías Barrientos JE (2009) Online Objective Reduction to Deal with Many-Objective Problems, Springer Berlin Heidelberg, pp 423–437

López Jaimes A, Coello Coello CA, Aguirre H, Tanaka K (2014) Objective space partitioning using conflict information for solving many-objective problems. Information Sciences 268:305–327

McCormick JW, Singhoff F, Hugues J (2011) Building parallel, embedded, and real-time applications with Ada, vol 1. Cambridge University Press

Mehiaoui A, Wozniak E, Tucci-Piergiovanni S, Mraidha C, Di Natale M, Zeng H, Babau JP, Lemarchand L, Gerard S (2013) A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. ACM SIGPLAN Notices 48(5):121–132

Monot A, Navet N, Bavoux B, Simonot-Lion F (2012) Multisource software on multicore automotive ecuscombining runnable sequencing with task scheduling. IEEE Transactions on Industrial Electronics 59(10):3934–3942

Mraidha C, Tucci-Piergiovanni S, Gerard S (2011) Optimum: a marte-based methodology for schedulability analysis at early design stages. ACM SIGSOFT Software Engineering Notes 36(1):1–8

Mzid R, Mraidha C, Mehiaoui A, Tucci-Piergiovanni S, Babau JP, Abid M (2013) Dpmp: a software pattern for real-time tasks merge. In: Modelling Foundations and Applications, vol 7949, pp 101–117

Pagetti C, Forget J, Boniol F, Cordovilla M, Lesens D (2011) Multi-task implementation of multi-periodic synchronous programs. Discrete Event Dynamic Systems 21(3):307–338

Palis MA, Liou JC, Wei DSL (1996) Task clustering and scheduling for distributed memory parallel architectures. Transactions on Parallel and Distributed Systems 7(1):46–55

Purshouse RC, Fleming PJ (2007) On the evolutionary optimization of many conflicting objectives. IEEE Transactions on Evolutionary Computation 11(6):770–784

Rahmoun S, Borde E, Pautet L (2015a) Automatic selection and composition of model transformations alternatives using evolutionary algorithms. In: Proceedings of the 9^{th} European Conference on Software Architecture Workshops, ACM, p 25

Rahmoun S, Borde E, Pautet L (2015b) Multi-objectives refinement of aadl models for the synthesis embedded systems (mu-ramses). In: Proceedings of the 20^{th} International Conference on Engineering of Complex Computer Systems, pp 21–30

Ramamritham K (1995) Allocation and scheduling of precedence-related periodic tasks. IEEE Trans Parallel Distrib Syst 6(4):412–420

Rota GC (1964) The number of partitions of a set. The American Mathematical Monthly 71(5):498–504

Santinelli L, Puffitsch W, Dumerat A, Boniol F, Pagetti C, Victor J (2014) A grouping approach to task scheduling with functional and non-functional requirements. Proceedings of the 2^{nd} Embedded Real-time Software and Systems

Saxena DK, Duro JA, Tiwari A, Deb K, Zhang Q (2013) Objective reduction in many-objective optimization: Linear and nonlinear algorithms. IEEE Transactions on Evolutionary Computation 17(1):77–99

Scheickl O, Rudorfer M (2008) Automotive real-time development using a timing-augmented autosar specification. Proceedings of the 4^{th} International Congress on Embedded Real-Time Systems

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

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

Singhoff F, Plantec A, Rubini S, Tran H, Gaudel V, Boukhobza J, Lemarchand L, Li S, Borde E, Pautet L, et al (2015) Teaching real-time scheduling analysis with cheddar. In: 9ème édition de l'Ecole d'Eté Temps Réel

Stankovic JA (1988) Misconceptions about real-time computing: A serious problem for next-generation systems. Computer 21(10):10–19

Taft ST, Duff RA, Brukardt RL, Ploedereder E, Leroy P, Schonberg E (2014) Ada 2012 reference manual. Language and standard libraries: International standard ISO/IEC 8652/2012 (E), vol 8339. Springer

Van Veldhuizen DA, Lamont GB (1998) Multiobjective evolutionary algorithm research: A history and analysis. Tech. Rep. TR-98-03, Department of Electrical and Computer Engineering, Air France Institute of Technology, Ohio

Wang H, Yao X (2016) Objective reduction based on nonlinear correlation information entropy. Soft Computing 20(6):2393–2407, DOI 10.1007/s00500-015-1648-y

Westfall PH, Young SS (1993) Resampling-based multiple testing: Examples and methods for p-value adjustment, vol 279. John Wiley & Sons

Wozniak E, Mehiaoui A, Mraidha C, Tucci-Piergiovanni S, Gerard S (2013) An optimization approach for the synthesis of autosar architectures. In: The 18^{th} IEEE Conference on Emerging Technologies & Factory Automation, IEEE, pp 1–10

Zhang Q, Zhou A, Zhao S, Suganthan PN, Liu W, Tiwari S (2008) Multiobjective optimization test instances for the cec 2009 special session and competition. University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report 264

Zitzler E, Thiele L (1998) Multiobjective optimization using evolutionary algorithms-a comparative case study. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 292–301