République Tunisienne
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Ecole Doctorale
Sciences et Technologies

Thèse de *DOCTORAT*

Ingénierie des Systèmes Informatiques

N° d'ordre: 68–17

THESE

Présentée à

L'École Nationale d'Ingénieurs de Sfax

En vue de l'obtention du

DOCTORAT

Dans la discipline Informatique
Ingénierie des Systèmes Informatiques

Par

Rahma BOUAZIZ FRIKHA

(Mastère en Nouvelles Technologies des Systèmes Informatiques Dédiés)

Multi-Objective Optimization and Design Space Exploration of Critical Real-Time Systems

Soutenu le 30 juillet 2018, devant le jury composé de:

M.	**Mohamed ABID** (Professeur)	Président
M.	**Samir BEN AHMED** (Professeur)	Rapporteur
Mme.	**Hanêne BEN-ABDALLAH** (Professeur)	Rapporteur
M.	**Adel MAHFOUDHI** (Professeur)	Examinateur
M.	**Mohamed JMAIEL** (Professeur)	Directeur de Thèse
M.	**Laurent LEMARCHAND** (Maître de conférences)	Invité
M.	**Frank SINGHOFF** (Professeur)	Invité
M.	**Bechir ZALILA** (Maître assistant)	Invité

Acknowledgments

It is with great pleasure that I reserve this page as a sign of deep gratitude to all those who have kindly provided the necessary support for the smooth running of this thesis.

I present my thanks to *Prof. Mohamed Abid* for the honor he had accorded me for agreeing to be the committee chair of my thesis. I also thank *Prof. Adel Mahfoudhi* for the valuable service to examine my thesis and to be a member of the committee. My distinguished thanks go also to *Prof. Samir Ben Ahmed* and *Prof. Hanene Ben Abdallah* for taking their time to review my dissertation and for their relevant comments.

I would like to express my deep gratitude to my supervisor *Prof. Mohamed Jmaiel* for his outstanding commitment to this thesis. I am also grateful for the support he gave me. His professionalism, friendliness and pedagogical and scientific qualities have been invaluable.

I am also indebted to my co-supervisors *Prof. Laurent Lemarchand*, *Prof. Frank Singhoff* and *Dr. Bechir Zalila* for supervising my thesis work, for their human qualities, for their patience, and especially for the time they have spent for me. They have always been the source of inspiration and motivation to me. Their knowledge, insight, scientific rigor and advices improved immensely the quality of the scientific and technical contributions of this work, but also its presentation. They have helped me gain invaluable skills as a researcher. May they find in this work the fruit of their effort and the expression of my deep gratitude.

I am thankful to all of my colleagues at both Lab-STICC Laboratory (Brest-France) and ReDCAD Laboratory (Sfax-Tunisia). Not only were our technical discussions very interesting, but it was wonderful working with them.

Last but not least I would like to express a very special gratitude to my husband, my parents and my whole family, for their support, patience, love and for making this journey as pleasant as it can be.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Introduction

Real-time systems are increasingly spread across many application fields ranging from small devices frequently used in our daily life (e.g. telecommunication systems, multimedia applications, GPS) to more sophisticated and safety-critical industrial systems (e.g. control systems for nuclear reactors, avionic systems, medical instruments, robot controllers, automotive systems).

A real-time system generally works in a continuously variable environment. To achieve a specific mission, such a system must follow the evolution of the environment and interact with it [Sta88]. Retrieving information from the environment is performed through the system sensors. Once it receives information, the system must process the input data as to produce a response by means of actuators. The system devices (i.e. sensors, actuators, etc.) are activated at particular frequencies and must accomplish their works within predetermined time limits. This distinguishes a real-time system from a classical system. Real-time systems are subject to meet different non-functional requirements (e.g. timing constraints, safety, reliability, etc.) imposed by their environment. Hence, their development needs to be carefully performed such that all non-functional constraints will be met.

They are qualified as safety-critical when failures including timing vulnerabilities (e.g. missing a deadline) lead to dramatic damages or even human life losses. A delayed response or output is considered as wrong even if it is logically correct. In other words, the correctness of results depends on both their functional accuracy (called also logical correctness) and their temporal correctness (i.e. functions meet the specified deadlines). A real-time system is referred to as *embedded* when it is run on the top of an execution platform with limited computation, storage and energy resources. In this thesis, we target systems that are both safety-critical real-time and embedded, what we refer to as real-time embedded (RTE) systems.

The design stage is of major importance in the life-cycle development of RTE systems. It allows to better master the complexity of the software development of these systems [GTT02, FMB+09]. Yet, design flaws including timing vulnerabilities and wrong decisions/choices during the design stage are often detected at late development stages (i.e. during or after the implementation) [NIS02]. This would adversely affect the development costs, time-to-market[1] and the system

[1] Time-to-market is a concept related to the duration required to develop and commercialize a product. It becomes an important criterion for the success of a product.

performance criteria.

The design stage consists mainly in building the operational architecture that will be refined until obtaining the code of the application according to a model-driven development (MDE) approach [Sch06]. The operational architecture is the result of *mapping* the functional specification of the target application onto a specific execution platform [SLS05]. The functional specification describes the system functions and their interactions. The execution platform represents the software and hardware entities (e.g. processors, tasks, shared resources, etc.) required to implement the functional specification.

The timing constraints are among the important non-functional requirements to consider during the design phase. To do so, a schedulability analysis is performed at the design stage which warrants the temporal determinism. RTE systems are frequently designed according to concurrent multi-tasking architectures. Such tasks are usually interacting with each other (e.g. sharing data). In the context of RTE systems requiring functional predictability, the communication and the synchronization of tasks must be carefully handled. In that respect, concurrency tasking models are subject to a set of standardized restrictions called Ravenscar profile [BDV04] especially tailored to enable reliable and efficient schedulability analysis. In our work, we assume Ravenscar compliant RTE systems.

The main challenge for designers is to define the most appropriate operational design that ensures correct behavior, the satisfaction of non-functional requirements and optimal performance criteria of the system.

The success key of a design stage resides in the availability of analysis and optimization methods that helps designers to establish the most suitable operational design. The work performed in the present thesis fits into this context.

Problems and motivation

In the context of uniprocessor architecture that we assume in our work, designing the operational architecture consists in assigning the system functions to a set of tasks.

The increased number of functions of today's real-time systems results in many various ways of assigning functions to tasks. The different design alternatives (i.e. all the combinations of functions to tasks assignment) form the design space. The latter grows exponentially with the number of functions involved in the functional specification which leads to a combinatorial problem.

Each design alternative has an impact on the system behavior and performance. Before assessing performance criteria of a candidate design alternative, the latter must be a feasible design solution. By feasible design, we mean a design that

guarantee all constraints regarding the schedulability, the consistency between the functional level and the design level (e.g. the periodic activation of functions), shared data consistency, etc. Furthermore, the performance criteria compete with each other: improving one aspect can have an adverse impact on other performance criteria. For instance, reducing timing overhead (e.g. narrowing the number of preemptions) may be done by limiting the number of tasks. Yet, this may leads to less flexible design that is expensive to change, since a reduced number of tasks will induce lower task laxities. The laxity of a task is a property characterising the flexibility available for scheduling the task. Considering the competing aspect between performance criteria, several design alternatives represent different trade-offs are searched for. Accordingly, the problem we deal with fits with constrained multi-objective optimization problems (MOOPs) [CLVV07].

Explorating manually the design space in order to select the most appropriate operational design is a limited approach for several reasons. On the one hand, such approach can be tedious and error prone as it relies on designers knowledge. On the other hand, given the combinatorial nature of the problem, performing an exhaustive search "by hand" (i.e. designers try all the possible combinations, evaluate the fitness of the feasible solutions with regards to a set of criteria and thereafter select the best trade-offs) is impossible. With such manual strategy, designers are able to explore only a narrow portion among all possible design alternatives. This can lead to miss some interesting design alternatives. Thus, an automatic process for the design space exploration (DSE) will reduce the time and cost of the design stage.

Solutions overview and contributions

In this thesis, we aim at automatically providing a set of feasible operational designs that exhibit meaningful trade-offs between multiple performance criteria at a reasonable computational cost. The performance criteria considered in our work are related to the scheduling field, such as preemptions/context switches, laxities of tasks, blocking times, etc. To tackle the problems mentioned above and achieve our objective, we propose our solution that includes the following contributions.

A) Automatic multi-criteria design space exploration (DSE) process

MOEA formulation of the DSE process

We propose an automatic design space exploration process. To cope with the multi-objective optimization nature of the addressed problem and its combinatorial complexity, we propose to formulate the DSE process using Multi-Objective

Evolutionary Algorithms (MOEAs) [Deb01]. 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.

The proposed DSE process allows to (1) explore the search space composed of design alternatives in terms of functions-to-tasks assignment combinations, (2) evaluate fitness (i.e. performance values that exhibit the quality of a given solution) of each feasible design alternative (3) and finally, identify the Pareto set of design alternatives (i.e. that satisfy at best the considered performance criteria). This DSE process is based on the *Pareto Archived Evolution Strategy (PAES)* [KC00a] MOEA.

Formalization of candidate architectures

Each candidate design explored during the DSE process must be analysed to check its feasibility (e.g. schedualbility analysis), then evaluated to determine its fitness towards performance criteria. These operations require the knowledge of timing parameters of each design alternative elements (i.e. tasks and shared resources). To identify these parameters, we propose a set of rules that formalizes design alternatives according to the way in which functions are assigned to tasks and the timing parameters derived from the functional specification.

Scope of application of the DSE process: assumptions and real-time scheduling context

The DSE process is applied on RTE systems with both independent tasks and tasks sharing resources. We rely on a conventional task model based on Liu and Layland model [LL73]. We assume periodic synchronous tasks with implicit deadlines running a uniprocessor platform under a preemptive and fixed-priority scheduling policy. We target Ravenscar [Bur99] compliant systems, i.e. the shared resource accesses are governed by the priority ceiling protocol (PCP) [SRL90] in order to ensure the synchronization of tasks and their mutual exclusion.

B) Improving the computational efficiency of the DSE process and the quality of produced Pareto sets

The problem we deal with involves high computation costs mainly due to the feasibility verification (including the schedulability analysis) and the fitness evaluation (i.e. objective functions computation). To enhance scalability, we introduce some improvements to the DSE process. First, we suggest to benefit from multi-processor computing platforms by parallelizing the feasibility verification and the fitness evaluation of multiple design alternatives. To do so, we adapt

the well-known *Master-Slave* parallel scheme [CLVV07]. Second, we define a new selection strategy that guides the search procedure in the DSE process. The impact of these changes is evaluated and shows significant improvement in regards to the DSE process scalability (i.e. the ability to handle large size systems) and effectiveness (in terms of produced Pareto sets quality).

C) Implementation of our prototype and evaluation of our proposals

In order to enable the use and the evaluation of the proposed DSE process, we provide a prototype embedding our contributions. This prototype is integrated in the Cheddar[2] scheduling framework [SLNM04]. A particular attention has been paid on the design of the prototype, which allows the reuse and the extension of its software artefacts.

In addition, various experiment sets are achieved in order to assess our proposals and investigate by means of empirical studies some concerns related to the addressed problem. These experiments are achieved on problem instances (called also test instances or test cases) synthetically generated thanks to a customizable generator that we propose. The performed experiments show the functional evaluation of our approach as well as its performance evaluation in terms of: (1) accuracy evaluation of Pareto sets produced by the DSE process for small size problem instances, (2) quality evaluation of Pareto sets for larger size and more complex (i.e. with different resources contention levels) problem instances and (3) scalability evaluation of the DSE process.

D) Empirical studies

As part of the experiments, two different empirical studies are carried out. The first empirical study is dedicated to investigate the correlation between different performance criteria. The second one aims at analyzing how the choice of the initial solution (required by PAES to start the search) influences the performance (in terms of convergence to the optimal Pareto set) of our DSE process.

Investigating the correlation between objectives

Several performance criteria, referred to as objectives, could be involved to drive the design exploration. The correlation (i.e. conflict relationships or support relationships) between these objectives is not obvious and could be counter intuitive. Two objectives are identified as redundant when they support each others, i.e. optimizing one of them leads to the optimization of the other. Considering

[2]Cheddar is an open-source scheduling analyzer, available for the academic and industrial researchers

two redundant objectives in the design exploration is irrelevant since this will increase the problem complexity uselessly. That's why we propose to empirically study the correlation between three pairs of objectives among those considered in our work.

Studying the impact of the initial design choice on the performance of the DSE process

The choice of the initial solution (or initial population for population based methods), from which an MOEA like PAES starts the search, is one among factors that may impact the performance of the MOEA [HGT05]. Indeed, the manner in which the initial solution is formed may bias the search in favour of a particular objective. As part of our research work on the functions-to-tasks assignment problem, we propose two different initial solutions and we study their impact on the performance of the proposed DSE process in terms of convergence towards the optimal Pareto sets.

Thesis structure

The remainder of the manuscript is structured as follows.

The first part is devoted to the research foundations and the state of the art, and it includes three chapters (Chapters 1, 2 and 3). Chapter 1 presents key concepts related to RTE systems. It also highlights the common timing properties and constraints specifying the temporal behavior of RTE systems. Furthermore, this chapter presents an overview on the scheduling analysis principles and discusses different kinds of scheduling analysis tests that could be applied to Ravenscar compliant RTE systems on which we are interested in this thesis.

Chapter 2 introduces the domain of interest for this thesis, which is essentially concerned with the design and optimization of RTE systems. It starts by exposing the whole development process, then emphasizes the specificities of the design phase and its importance throughout the development process. It also depicts fundamental aspects about MOOPs and discusses techniques used for solving them. Then, it focuses on MOEA techniques by highlighting their key components and reviews some quality indicators for assessing them.

Chapter 3 highlights the work positioning by exposing the problem statements, the work assumptions and our contributions. It also provides a literature review about existing approaches dealing with (1) mapping the functional specification

of a given RTE system towards an operational design and (2) multi-criteria design space exploration.

The contribution part starts by Chapter 4. This chapter presents elements underlying the automated multi-criteria DSE process for the mapping problem. Firstly, we detail our solution based on a MOEA technique (namely PAES) to explore a design space of functions-to-tasks assignment solutions and identify optimal or near optimal architecture alternatives. Secondly, we demonstrate how we formalize architecture alternatives, and compute parameters of their entities (in terms of tasks and shared resources). Finally, we describe how we check the feasibility of design alternatives.

In Chapter 5, we address the scalability issue of the proposed DSE process by introducing some improvements. These improvements are mainly the adaptation of a parallel processing for the DSE process combined with a new selection strategy for PAES. The experiments achieved to assess the impact of the introduced changes on the performance of the DSE process are also presented.

Chapter 6 gives an overview on the prototype implementing our solutions. This chapter describes features covered by the prototype and its software artefacts. It also introduces the Cheddar scheduling framework that embeds our prototype.

Chapter 7 is devoted to present experiments achieved to assess the proposed DSE process and drive empirical studies regarding certain concerns, namely the correlation between performance criteria and the impact of the initial design on the DSE process behavior/performance.

The last chapter concludes the thesis and outlines some directions for future work.

Part I

Research Foundations and State of the Art

1

Real-Time Embedded Systems: Terminology, Principles and Scheduling Theory

Contents

1.1 Introduction

In this chapter, we discuss about real-time embedded (RTE) systems and knowledge that form a part of the background needed in our work.

Section 1.2 defines key properties of RTE systems. The general structure of these systems is highlighted in Section 1.3. Section 1.4 points out fundamental notions about real-time tasks systems and scheduling policies, required for specifying the temporal behavior of RTE systems. Afterwards, Section 1.5 presents an overview on the scheduling analysis principles and then details most popular scheduling methods that can be applied to Ravenscar compliant RTE systems. Finally, Section 1.6 concludes the chapter.

1.2 RTE systems: definitions and classification

This section defines RTE systems by exposing their underlying features and the constraints they must meet.

1.2.1 Definitions

Real-time embedded systems are present everywhere around us. Fields of application are multiple such as medical, telecommunication, avionic, automobile, robotic, telecommunication, etc. Such a system is both an embedded system and a real-time system. Next, we define these two kinds of systems by focusing on their main characteristics.

Definition 1. ***Embedded System*** *[LLS07, Hea02]*
An embedded system consists of a set of hardware and software components that cooperate in order to control a specific range of functions. These systems are subjected to different environmental and resource constraints such as power consumption, size, cost, memories, processing resources, etc. One of the key characteristics of embedded systems is that the energy is also embedded (batteries, fuel, etc.). Such a system is often embedded within a larger device. The term embedded means that the system is not visible to the end-users since it is part of a larger equipment. Embedded systems communicate with their environments by taking decisions, processing data and reacting thereafter.

When embedded systems are subject to timing constraints, they are considered as real-time systems. In fact, embedded systems are often real-time systems since they always have limited resources, hence execution durations of their operations cannot be neglected.

Definition 2. ***Real-time System*** *[Sta88, Kop97, Che03, LLS07]*
A real-time system is a computing system that have to process inputs (data or events) within specified time limits. The behavior of such a system is deemed to be correct if and only if the result computed by the system is logically correct (functional correctness) and produced on time (timing correctness).

A real-time system is subject to timing constraints imposed by the environment with which it interacts. Real-time does not refer here to immediacy (i.e. the system reacts as fast as possible) but means that the processing of inputs must finish within a finite duration, so that results are produced before or at a specific instant called *deadline*. A logically correct result that does not meet the timing requirements is considered to be globally incorrect in the same way as a logically wrong result. The violation of timing constraints (e.g. a delay that causes the missing of one or more deadlines) is considered as a failure that could lead to severe damages.

1.2.2 Classification

Real-time systems are usually categorized based on their criticality level. A criticality level [Sta88] is specified by the system capacity to tolerate certain missed deadlines or not. Two main categories of real-time systems are distinguished and defined as follows.

- **Hard real-time** [ABD+95, BLAC05, LL73]
  Hard real-time (or critical) systems are used to control critical environments. They have to meet the required deadlines. Otherwise, an intolerable system failure may occur which could lead to serious damages and casualties. These systems are often designed under pessimistic assumptions to handle the worst-case scenarios [BLAC05]. Examples of hard real-time systems are flight control systems [DH92], automotive engine control systems [FMB+09], health care systems such as cardiac pacemaker system [MLF08], etc.

- **Soft real-time** [ABD+95, BLAC05, LL73]
  Soft real-time systems are systems that can tolerate whenever possible timing failures (i.e. some delays with respect to the expected deadlines), possibly because those failures do not cause damages on the environment under control. However, such timing delays may compromise the *quality of service* [MKT09]. Examples of soft real-time systems are telecommunication and multimedia systems [AMK98].

A RTE system is also characterized by the entities of its internal structure described in the next section.

1.3 Internal structure of RTE systems

Figure 1.1 illustrates a simplified structure of a real-time control system interacting with its environment. The control system gets information about the environment through sensors and acts accordingly on the process via actuators.

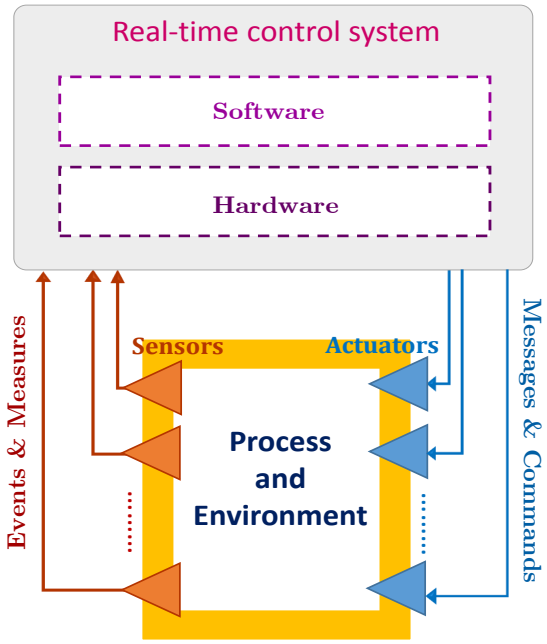


FIGURE 1.1: Real-time control system interacting with its environment

A RTE system is composed of a software layer executing on a hardware platform. In the remainder of this section, we discuss in detail about both software and hardware parts.

1.3.1 Hardware platform

The hardware layer has a crucial impact on the RTE system behavior and its analysis (e.g. schedulability). As illustrated in Figure 1.2, a hardware platform is a combination of different components such as processors/cores, networks, memories, etc. We are interested here in the computing/execution resources (i.e. processors). Various kinds of architectures can be identified according to the composition of the hardware platform in terms of the number of processors and the interaction between them. Three prominent categories of hardware architectures are widely used in the real-time scheduling analysis.

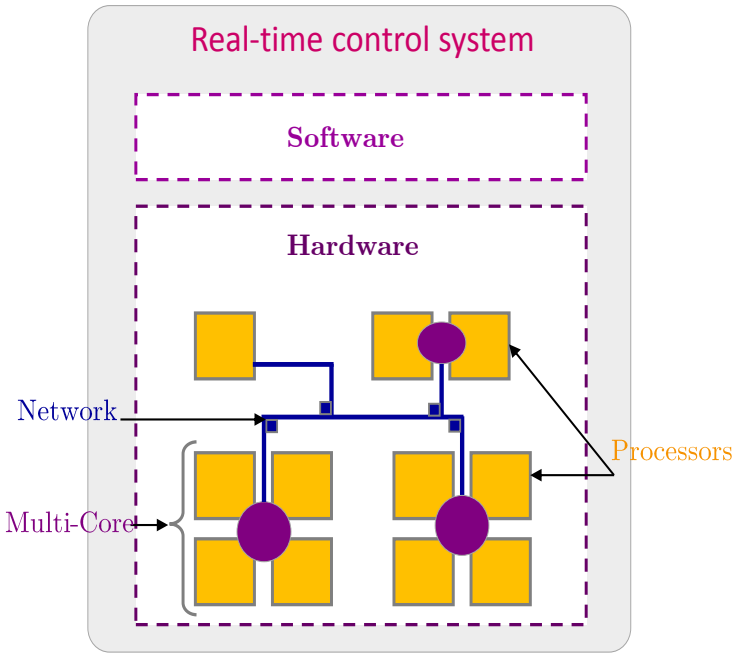


FIGURE 1.2: Hardware parts of a real-time control system

- **Uniprocessor architecture**: a unique processor or central processing unit (CPU) handles software entities that compose the software application. In other words, software entities implementing the system functions (called *tasks*) compete for acquiring the CPU in order to be executed.
- **Multiprocessor architecture**: in this kind of architecture, the hardware platform includes more than one processor sharing a common memory.
- **Distributed architecture**: the hardware structure is composed of several nodes interconnected through a network (see Figure 1.2). Unlike the multiprocessor architecture, in a distributed architecture nodes do not share a common memory. Instead, each node has its own memory. A node may be a uniprocessor or a multiprocessor architecture.

1.3.2 Software structure

The software layer performs the mission of the RTE system, using available resources from the hardware platform.

As illustrated in Figure 1.3, the software structure is composed of the software application and the real-time operating system (RTOS).

The software application corresponds to the program that implements different functions enabling the system to control its environment. The application is often

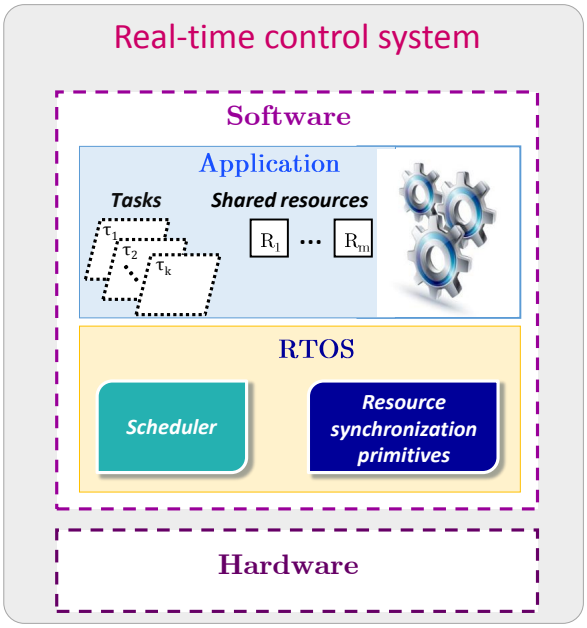


FIGURE 1.3: Software structure of a real-time control system

structured in a set of tasks that may interact to accomplish their jobs. Some programming languages such as the Ada language [MSH11] encapsulate the OS layer and provide straightforward means to manipulate tasks and concurrency features. By contrast, other languages (like C or C++) require the use of the OS utilities for tasks management.

A classical operating system (OS) plays the role of an interface that bridges the software application to the hardware resources. The RTOS is an OS with additional services designed in order to manage the *multi-tasking* with high timing accuracy and predictability. A RTOS provides a *scheduler* that decides which tasks of the program has to be executed on which processor at a given instant. It also affords various services enabling different kinds of interactions between tasks (e.g. task communications, shared resources access synchronization via primitives, etc). In addition to the general goals of a conventional OS (e.g. maintaining responsiveness, enforcing fairness, avoiding resource starvation, etc), a RTOS is intended to meet the timing requirements.

In the sequel, the focus is set on aspects related to the real-time scheduling theory. Mainstream notions for specifying the temporal behavior of RTE systems are introduced in Section 1.4. Section 1.5 explains underlying features of the real-time scheduling analysis and presents different kinds of scheduling methods that could be applied to Ravenscar compliant RTE systems.

1.4 Real-time scheduling

The design model of a RTE system must supply the information necessary for enabling analysis of the system temporal behavior. Task is the key component in software design of RTE systems using multi-tasking approach. In the following, we define the concept of task, describe its life-cycle and point out its properties.

1.4.1 Real-time tasks: definitions and temporal properties

Definition 3. ***Task*** *[ABD+95, SAA+04] is an active entity of the real-time application program. It enables the sequential execution of a set of instructions corresponding to one or many functions.*

A) Task life-cycle

A task is released to run when a specific event occurs. At each release of a task, we say that a *job* or an *instance* of that task will be executed. A task has a life-cycle typically composed of four states. The transition from a task state to another is achieved by the RTOS according to a state transition diagram illustrated in Figure 1.4.

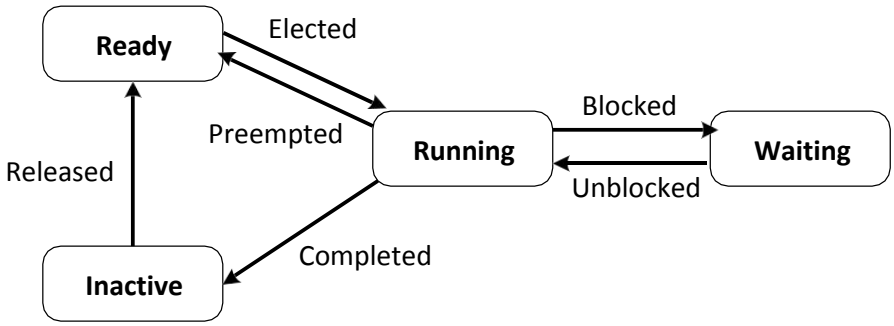


FIGURE 1.4: Task life-cycle (adapted from [LLS07])

- **Ready**: a task is in the ready state when the task is released and waits for being elected among other tasks for execution.
- **Running**: once elected, the task is active and executing on a processor. All shared resources and the processor are available to the task in the running state.
- **Waiting**: a task is waiting for one or more events or resources (e.g. interrupt, message, shared resource, etc.), except the processor, to be available. We say that the task is *blocked*.
- **Inactive**: the task is *sleeping* and waiting for a wake event to be released.

B) Task properties

A task is defined by a set of properties to identify its order of importance, its computational requirement or its timing constraints. These properties are crucial information for the scheduling analysis. The execution of a task is commonly represented by a Gantt diagram. Figure 1.5 depicts an example of a task execution that shows a part of its properties.

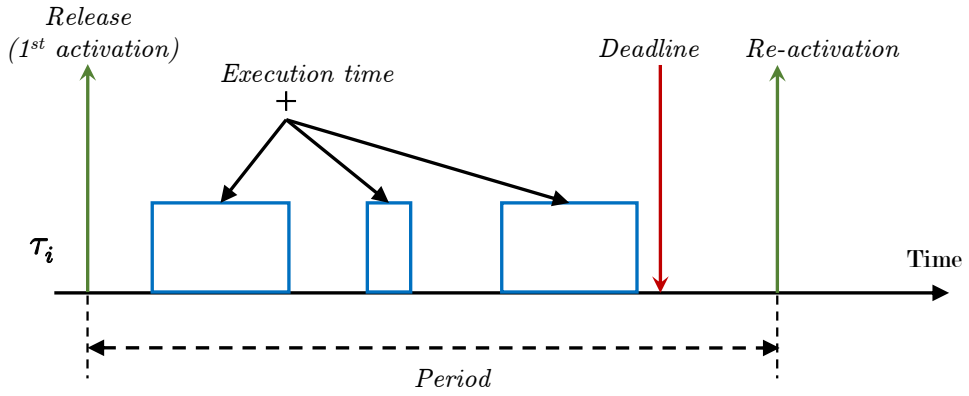


FIGURE 1.5: Main properties of a real-time task illustrated by a Gantt diagram

- **Offset** O_i: corresponds to the release instant of the first job of task τ_i.
- **Execution time (or capacity)** C_i: is the time duration needed for running a given job of the task τ_i on a particular processor. Indeed, the execution time of a task may vary from one job to another. When the task has different execution paths depending on the input data, is a typical example leading to a non-constant execution time. This parameter is usually called capacity of the task and represented by the worst-case execution time (WCET) defined as the upper bound of all possible execution times of any job. Some research work consider also the best-case (i.e. the shortest value) and the average execution time values (BCET and AET). The estimation of the WCET of a task is a challenging problem that has been tackled in many works [CP00b, RS04] and remains an open research line. The estimation of this parameter is often pessimistic, which unfortunately may affect results of the scheduling analysis that depends on that parameter.
- **Activation pattern**: a task is activated by events occurring according to a specific pattern. The activation pattern represents the execution frequency of the task jobs. Three patterns are widely used in the literature:
  - Periodic task [LL73]: a periodic task τ_i is released regularly referring to a fixed time interval T_i called period. Let $\tau_{i,j}$ be the j^{th} job of the periodic task τ_i, and $r_{i,j}$ its release time, hence $r_{i,j} = O_i + (j-1) \cdot T_i$.

An example of a periodic task is a program that retrieves data sent by a sensor every 500 *Millisecond*.

- Sporadic task [LL73]: with a sporadic task, there is a minimum inter-arrival time (MIT) between two consecutive releases. The MIT ensures a safe resource utilization by a sporadic task since it represents an upper-bound for the task execution frequency. The delivery of IP packets in a network is an example of a sporadic task. Indeed, the arrival rate may vary from one IP packet to another but it is bounded by the throughput of the network infrastructure.
- Aperiodic task [SSL89, SLS95]: An aperiodic task may arise at any instant, and there is no minimum separation duration between two consecutive jobs. Aperiodic tasks are usually released by external events. In many practical situations, some emergency events or user interactions may recur aperiodically and need to be handled during the life time of the system.

- **Deadline** D_i: is the time limit devoted to task τ_i to complete its execution [ABD+95] after each of its releases. In the literature, this parameter is referred to as *relative deadline* because it is the relative to the release time of a job [BW07]. For the j^{th} job (or occurrence) of task τ_i denoted as $\tau_{i,j}$, an *absolute deadline* $d_{i,j}$ represents a specific instant in time at or before which the job $\tau_{i,j}$ must finish its execution. The absolute deadline is computed using the relative deadline D_i and the release time $r_{i,j}$ of job $\tau_{i,j}$: $d_{i,j} = r_{i,j} + D_i$.
- **Processor utilization factor** U_i: is the fraction of a processor time required to run a task τ_i. For a periodic task, this term is defined by: $U_i = \frac{C_i}{T_i}$.
- **Priority** Π_i: tasks are in concurrence for access to the processor. Thus, there must be some mechanism to identify, at any given time, which task that deserves the access to the processor. The priority Π_i [ABD+95, SAA+04] indicates the order of importance for scheduling and execution of task τ_i with regard to the other tasks. This parameter can be fixed or dynamically assigned. The higher the value of Π_i, the higher the priority level of the task τ_i. At a given time, the task owing the highest priority level in the ready queue is selected in order to get the access to the processor.

C) Temporal properties related to the system execution

Here, we define real-time characteristics that are related to the tasks system (or task set) execution and/or derived from the properties already presented. Figure 1.6 illustrates some of these characteristics.

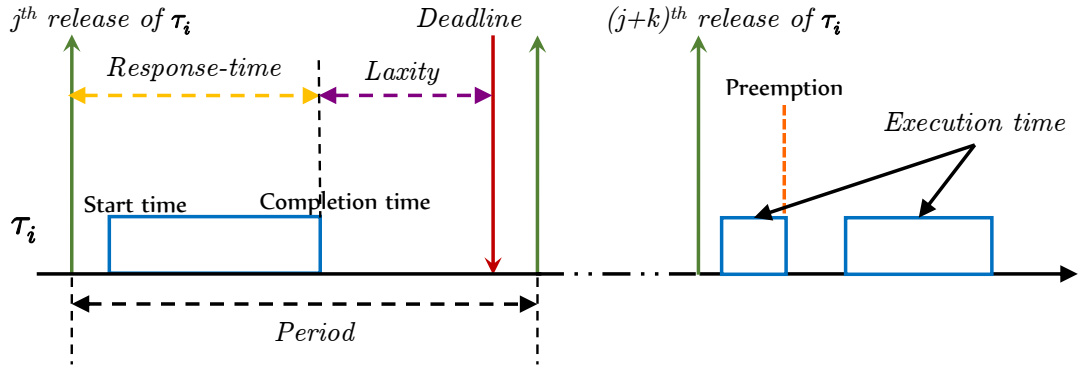


FIGURE 1.6: Task characteristics related to the system execution

Jobs related properties

- **Start time**: instant at which the job starts its execution.
- **Completion time**: instant at which the job completes its execution.
- **Preemption**: occurs when the job of the current running task is interrupted and the processor is assigned to execute a job of another task. A job can be preempted several times during its execution.
- **Response-time of a job** $RT_{i,j}$: is the time interval from the task job release to its completion [PH98]. A job may not be executed immediately when it is released or preempted during its execution. This is may due to the unavailability of a resource required by the job to continue its execution or the interference from other tasks in the system (i.e. the processor may be used by higher priority tasks), etc. Accordingly, the response time of a given job could be larger than the execution time of the corresponding task. The job meets its deadline when $RT_{i,j} \leq D_i$.

Tasks related properties

- **Worst case response-time of a task** $WCRT_i$: the worst-case response time of a task is the greatest value among its jobs response times [ABR+93]. Hence, $WCRT_i = \max_{\forall j}(RT_{i,j})$. A task is *schedulable* (the task schedulability notion will be discussed in detail in Section 1.4.3) if and only if $WCRT_i \leq D_i$.
- **Laxity of a task** L_i: is the maximum time a task τ_i can be delayed on its release to complete within its deadline [But11]. Mathematically, the task laxity attribute is computed as the difference between the task deadline and its response-time: $L_i = D_i - WCRT_i$.

1.4.2 Task set characteristics and classifications

In preparation for the review of scheduling policies, we introduce in this section key characteristics related to the task set execution and a set of common assumptions and classifications.

A task set may be classified based on several properties such as: dependency relationships between tasks, deadlines, offsets, etc.

Dependency relationships between tasks

In the real-time scheduling theory, tasks composing a task set can be either independent or dependent.

- **Independent tasks** [ABD+95]: Tasks are independent from each other, that is the execution of each of them cannot be blocked by another task. It is important to note that the contention for the processor access between tasks is not considered as a dependency.
- **Dependant tasks** [ABD+95, SAA+04]: In a RTE system, tasks can interact with each other to achieve a specific mission, so there exists dependencies between them and they are called dependent tasks. Two main ways for modelling interactions/communication between tasks are distinguished, namely: precedence relationships and shared resources.
  - *Precedence relationships* [CSB90, ABD+95]: When some tasks have to expect messages or synchronization signals coming from other tasks, we say that tasks are subject to precedence dependencies. The receiver task needs to wait for a message from the sender task.
  - *Shared resources* [SRL90, ABD+95, But11]: are another mean of interaction between tasks. In practice, tasks frequently need to share some software resources (e.g. data structures, variables, memories) or hardware resources (e.g. sensors). We talk about *mutually exclusive* resources [But11] that do not allow simultaneous accesses by competing tasks, but require their mutual exclusion. In order to enforce mutual exclusion, shared resources are protected using *critical sections* during a task execution. A critical section [SRL90, ABD+95] of a given resource used by a task, is the piece of code belonging to that task implementation and that is executed under mutual exclusion constraints. Mutual exclusion constraints are enforced by means of synchronization primitives (e.g. mutexes, semaphores or monitors) provided by most of modern RTOSs.

Deadlines

Task set deadlines may comply with one of the following patterns:

- **Implicit-deadlines** [DB11]: deadlines of the tasks are exactly equal to periods: $D_i = T_i \ \forall \ \tau_i$.
- **Constrained-deadlines** [DB11]: deadlines of the tasks are less than or equal to periods $D_i \leq T_i \ \forall \ \tau_i$.
- **Arbitrary-deadlines** [DB11]: deadlines of the tasks and periods are unrelated.

The implicit-deadlines case is a particular case of the constrained-deadlines case which in its turn is a particular case of the arbitrary-deadlines case.

Offsets

A task set may be either synchronous or asynchronous depending on offsets of tasks:

- **Synchronous task set** [DB11]: all tasks are released and ready to execute at the same point in time (i.e. the first jobs of all tasks are released simultaneously). For a synchronous task set, we have $O_i = Constant, \forall \tau_i$.
- **Asynchronous task set** [DB11]: first releases of tasks are separated by fixed offsets and there is no simultaneous arrival time. For an asynchronous task set, we have at least two task τ_i and τ_j that have different offsets, i.e. $O_i \neq O_j$.

1.4.3 Scheduler and scheduling policies

A *scheduler* is a component of the kernel of an RTOS.

Definition 4. ***Scheduler*** *[ABD+95] is a module implementing an algorithm (or a policy) that manages the execution order of tasks on the processor(s) according to some criteria.*

The *scheduling* of tasks on the processor(s) is achieved by the scheduler according to a *scheduling policy*.

Definition 5. ***Scheduling*** *[ABD+95] is a method by which tasks are assigned to available computing resources (i.e. processors) ensuring the execution of these tasks.*

Definition 6. ***Scheduling Policy (or scheduling algorithm)*** *[ABD+95] is the algorithm that describes how tasks are elected for access to the processor(s) and other shared resources.*

A scheduling policy is based on a set of criteria or rules. These criteria can be considered as characteristics of a scheduling policy, by which scheduling policies can be classified into several categories. Most common scheduling policy characteristics are outlined below.

A) Off-line and on-line scheduling

Scheduling policies can be characterized as off-line or on-line.

Definition 7. ***Off-line scheduling policy*** *[ABD+95, GGCG16] a scheduler is said off-line when all scheduling decisions are taken prior to the running of the system.*

A scheduling algorithm is used off-line when it is executed on the entire task set to generate the corresponding schedule before the system runtime. This pre-runtime computed schedule is stored in a table that lists all tasks and their activation times. Thereafter, at runtime, this scheduling table is accessed by a simplified scheduler called *dispatcher* whose role is simply to remove the next task from the table and puts it in the running state.

The key advantage of the off-line approach is that the produced schedule is correct by construction, which makes its usage extremely encouraged for safety-critical RTE systems requiring high determinism. In addition, scheduling runtime overhead is relatively low (since the role of the dispatcher is to repeat infinitely the pre-runtime computed schedule) and it does not depend on the complexity of the scheduling algorithm. This allows very sophisticated algorithms to be used to solve complex problems or find optimal scheduling sequences [But11]. On the other hand, the major drawback of this kind of scheduling policy is its inability to be adapted to the environment variations.

Definition 8. ***On-line scheduling policy*** *[ABD+95, GGCG16] a scheduler is said on-line when all scheduling decisions are taken during the run-time of the system. An on-line scheduling algorithm is a strategy executed by the scheduler, in order to allocate active jobs on available processor(s).*

With an on-line approach, scheduling decisions are based on both task characteristics and the current state of the system. On-line scheduling algorithms are useful because of their flexibility. For example, a new task can be easily added during the system design. However, when additional application constraints (like precedence constraints and/or mutual exclusion constraints) have to be considered, an

on-line scheduling method may introduce a significant amount of runtime overhead. And worse, it can result in subtle errors and renders the runtime behavior of the system very difficult to analyze and predict [XP00].

B) Priority-based scheduling policies

Many on-line scheduling algorithms are designed based on priority of tasks. The task with the highest priority among ready tasks is elected by the scheduler to be executed. Such scheduling algorithms provide a priority assignment strategy. In the literature, there are two well-known class of priority-based scheduling policies, given below.

- **Fixed priority scheduling (FPS)**: the priority of each task is fixed and computed on the basis of that task static properties like its deadline or its period [ABD+95]. Task priorities are assigned off-line (i.e. before starting the system). Most popular examples of FP assignment methods are: the Rate Monotonic (RM) [LL73] and the Deadline Monotonic (DM) [LW82].
  - *Rate-Monotonic (RM)*: is dedicated for periodic task sets with implicit deadlines. According to RM, the task with the shortest period has the highest priority.
  - *Deadline-monotonic (DM)*: task priorities are inversely proportional to their relative deadline. In other words, the task with the shortest relative deadline has the highest priority.
- **Dynamic priority scheduling (DPS)**: the priority of a task may change during execution [ABD+95]. Most prominent examples of DP assignment strategies are: the Earliest Deadline First (EDF) [Der74] and the Least Laxity First (LLF) [Mok83].
  - *Earliest Deadline First (EDF)*: the jobs of a task may have different priorities, but each job has a single static priority. At a given time, the EDF scheduling policy assigns the highest priority to the ready task with the nearest absolute deadline.
  - *Least Laxity First (LLF)*: priorities of jobs may change between their release times and their completion times. At each instant, the LLF scheduling policy assigns the highest priority to the ready job with the minimum laxity.

Priority-based scheduling algorithms are usually designated in accordance with their priority assignment strategies e.g. a scheduling algorithm that uses RM as priority assignment method is also called RM.

C) Preemptive and non-preemptive scheduling policy

Scheduling policies can also be classified according to the preemption control mode.

Definition 9. ***Preemptive scheduling policy*** *[ABD+95, But11] can arbitrarily interrupt and suspend a task execution, in order to proceed to its execution later, without affecting the logical behavior of that task. This typically occurs when a change impacts the list of ready and blocked tasks, or the priorities of the ready tasks for priority-based scheduler.*

Definition 10. ***Non-preemptive scheduling policy*** *[ABD+95] once a task is elected for execution, it occupies the processor until it is completed.*

A non-preemptive scheduling is useful in the case of systems with shared resources. In fact, with a non-preemptive policy, a task execution cannot be interrupted, hence it cannot be preempted inside a critical section. Consequently, such a policy naturally guarantees the exclusive access to shared resources [But11]. In contrast, under a preemptive scheduling policy, shared resources need to be protected by using some synchronization mechanisms. On the other hand, in [ABD+95] the authors have proved that preemptive scheduling policies enhance schedulability.

D) Shared resources access protocols

As stated above, the mutual exclusion problem is trivial in non-preemptive scheduling, that implicitly ensures the exclusive access to shared resources. However, in a preemptive scheduling context, the presence of shared resources will increase the complexity of the scheduling problem [Mok83, XP00]. Indeed, a task may experience an additional delay called *blocking time* as a direct consequence of the mutual exclusion. The worst case 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 1.7 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.

The typical synchronization mechanisms (e.g. semaphores, monitors, etc.) provided by conventional operating systems are not suited for implementing real-time applications because they may cause some uncontrolled issues like:

- *Priority inversion* [But11]: is a situation where a high-priority task is blocked by a low-priority task for an unbounded interval of time.
- Deadlock [But11]: a situation where two tasks or more are waiting indefinitely for each other.

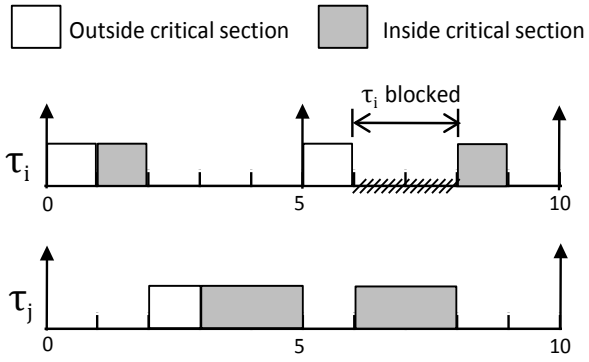


FIGURE 1.7: Example of blocking time on a shared resource

In order to prevent such phenomena and manage concurrent accesses of tasks to shared resources, several synchronization protocols have been proposed in the context of preemptive scheduling. For instance, the following protocols are widely used in uniprocessor scheduling contexts.

Priority Inheritance Protocol (PIP)

The priority inheritance protocol was proposed in [SRL90] to solve the problem of priority inversions, but not the deadlocks. In PIP, when a task τ_i gets access to a resource and blocks one or more high priority tasks, it executes its critical section at the highest priority level of all the tasks that it blocks. After its critical section, the task τ_i regains its original priority level.

Priority Ceiling Protocol (PCP)

The priority ceiling protocol [SRL90] was proposed to overcome limitations of PIP by preventing deadlocks and reducing blocking time of tasks. This protocol assigns at each resource R_k a priority called *priority ceiling* $\Pi(R_k)$ equals 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.

With these protocols, the worst case blocking time of tasks are bounded and thus they can be taken into account in the scheduling analysis.

1.5 Real-time scheduling analysis

The scheduling analysis provides evidence that the system behaves as expected and meets its timing constraints. Our purpose is not to provide an exhaustive study about scheduling analysis methods, but, we aim to emphasize fundamental

principles related to the scheduling analysis in Section 1.5.1. Afterwards, in Section 1.5.2, we consider a practical case of RTE systems namely Ravenscar compliant systems in order to show some examples of most popular schedulability tests compatible with this specific case of RTE systems.

1.5.1 Scheduling analysis principles

Schedulability analysis provides a mean to decide for a given task set under certain scheduling policy, whether all tasks deadlines will be satisfied during the life-time of the system.

In the following, first we explain some terms related to the scheduling analysis (e.g. feasibility, schedulability and sustainability). Then, main schedulability analysis characteristics and approaches are introduced.

A) Feasibility and schedulability

The ability to meet timing constraints of a task set is accessed by its feasibility and schedulability.

Definition 11. ***Feasibility*** *[ABD+95] is the assessment of the ability to satisfy all timing constraints of a task set.*

Definition 12. ***Feasible*** *[ABD+95] A task set is feasible if there exists a scheduling policy guaranteeing that all timing constraints are met.*

During the life-time of a system, a task set produces sequences of jobs called *schedule*. A schedule is called a *feasible* schedule, only if every job in the schedule can be scheduled without any deadline miss and if all job constraints are satisfied (e.g. shared resource constraints, precedence constraints, etc.).

Definition 13. ***Schedulability*** *[Bar03, ABD+95] is the assessment of the feasibility of a task set under a given scheduling policy.*

Definition 14. ***Schedulable*** *[Bar03, ABD+95] A task set is schedulable under a particular scheduling policy if none of its tasks, during execution, will ever miss their deadlines.*

The feasibility is a broader concept that includes the schedulability. For instance, given a space of task sets, the set of schedulable task sets under a particular scheduling policy will be a subset of feasible task sets.

Algorithms used to assess a system feasibility (resp. schedulability) are called feasibility (resp. schedulability) tests.

Definition 15. ***Feasibility test*** *[Bar03] verifies whether a task set is feasible or not.*

Definition 16. ***Schedulability test*** *[Bar03] verifies whether a task set is schedulable with a given scheduling policy or not.*

B) Scheduling analysis tests characteristics

Feasibility and schedulability tests are defined on the basis of different conditions. These conditions specifying feasibility/schedulability tests may either be sufficient or necessary or both:

- **Sufficient**: a schedulability (resp. feasibility) test is defined to be a sufficient condition if all of the task sets that are deemed schedulable (resp. feasible) according to that test are indeed schedulable (resp. feasible). If a sufficient schedulability (resp. feasibility) test is not satisfied for a given task set, the task set may still be schedulable (resp. feasible).
- **Necessary**: a test is referred to as a necessary condition, then we have: (1) a task set that does satisfy the test is not guaranteed to be feasible/schedulable; (2) a task set that does not satisfy the test is deemed infeasible/non-schedulable.
- **Necessary and sufficient (or exact)**: schedulability test that is both sufficient and necessary is designated as exact condition. Then it is in some sense optimal. A task set is feasible (resp. schedulable) if and only if it does satisfy the test.

A sufficient but not necessary test is pessimistic, but for many real-time scheduling contexts an exact test is computationally intractable.

Schedulability and sustainability

Feasibility and schedulability tests use some parameters of the analyzed system, that are computed based on the worst-case behavior of the system. Nonetheless, a part of these parameters can never be estimated exactly and there are always deviations in practice, which may compromise the *sustainability* of these tests. The sustainability concept has been introduced in [BB06]. It covers the deviations towards better scenarios, which means that it guarantees that the practical system indeed would be schedulable, even if at runtime the system behaves better than the worst-case behavioral task parameters considered for analysis.

Definition 17. ***Sustainability*** *[BB06, BB08] a schedulability test is referred to as sustainable, if any task set that is deemed schedulable by the schedulability test remains schedulable when the parameters of one or more task jobs are changed in*

any, some, or all of the following situations: (i) decreasing execution times, (ii) increasing periods or inter-arrival times and (iii) increasing relative deadlines.

To sum up, the sustainability requires that the schedulability is preserved in situations in which it should be easier to ensure schedulability (e.g. the opposite of the worst-case execution).

Due to the gap between theoretical analysis and practical execution, for instance, a task may execute shorter than its WCET. This change is not predictable. Thus, performing scheduling analysis test while considering WCETs is rational only when the test is sustainable with respect to the execution time parameter. Sustainability may be categorized according to the parameter whose change during run-time does not jeopardise schedulability.

- *C-sustainability*: when change is only related to a decrease in task execution times.
- *T-sustainability*: when change is only related to an increase in task periods.
- *D-sustainability*: when change is only related to an increase in task deadlines.

C) Examples of scheduling analysis methods

In the literature, there are several scheduling analysis methods that can be classified into three main approaches: model-checking approach, simulation approach and analytical approach.

- **Simulation based test**: is part of empirical analysis methods. The simulation aims at studying the system behavior in order to detect any temporal fault within a finite duration. This latter is referred to as *feasibility interval*.

  Definition 18. ***Feasibility interval*** *[LM80, GGCG16] is a finite interval such that if all the deadlines of jobs released in the interval are met, then the system is schedulable.*

  The feasibility interval depends on the system characteristics and it is based on the *simulation interval*.

  Definition 19. ***Simulation interval*** *[LM80, GGCG16] is an exact or upper bound of the time interval for the schedule to repeat in a cycle.*

  Knowing the length of the simulation interval is required for capturing the whole behavior of a system when building an offline schedule [XP00]. In the

case of periodic tasks system, the simulation interval is related to the system periodicity [CGG04, GGCG16]. Identifying feasibility and simulation intervals have been addressed in many research work [GGCG16].

A simulation test needs a complete knowledge about the system configuration (e.g. task set properties, hardware architecture, scheduling algorithm, etc.) in order to produce the corresponding schedule. It is important to note that performing exhaustive simulations for every possible system state is impossible (computationally speaking). Accordingly, simulation is carried out under some system assumptions (e.g. the worst system behavior, i.e. using the WCETs of tasks). Thus, simulation based scheduling analyses are not safe except for some conditions [Ouh13].

- **Analytical methods**: are based on algebraic methods. They define mathematical formulas under certain conditions, in order to compute various system performance attributes such as the processor utilization, tasks response-time, etc. Given the results computed by these formulas, the system behavior will thereafter specified. Analytical schedulability tests may be sufficient or exact schedulability conditions depending on the properties of the system to be analyzed. In the literature, there exists plenty of analytical schedulability tests. The most popular tests and widely used by the real-time community are given below.

  - *Processor utilization based tests* [LL73, KM91, BBB01]: evaluate the processor utilization factor to check the schedulability of a task set. The processor utilization factor represents the rate of the processor workload for a given task set and it is expressed as the sum of the processor utilization factor of each task: $U = \sum_{i=1}^{n} U_i$. In order to avoid the unschedulability of a task set, such tests verify that the utilization factor of that task set does not exceed a theoretical bound admissible by a particular scheduling policy.
  - *Response time analysis (RTA)* [JP86, ABR+93]: this method consists in calculating the worst-case response time $WCRT_i$ of each task in order to be compared against their relative deadlines D_i. Thus, a task set is schedulable if the worst-case response time of each task less than or equal to its deadlines: $\forall i, WCRT_i \leq D_i$.
  - *Processor demand criteria tests* [BMR90, LSD89]: are based on the *processor demand bound function* $DBF(t)$. The latter corresponds to the maximum execution time requirement of all jobs that have both their release times and their deadlines in a contiguous interval of length t.

1.5.2 Schedulability tests for Ravenscar RTE systems

Real-time critical systems must meet certain requirements that guarantee their safety and predictability. In order to enable the respect of the requirements, patterns such as the *Ravenscar Profile* [Bur99, BDV04] are dedicated to pilot the design and implementation of these systems. The Ravenscar Profile consists in a set of restrictions specified to enable the construction of efficient, reliable and verifiable real-time applications. Initially, it was tailored to limit the use of the Ada language [TDB+14] to only constructs ensuring a deterministic behavior of the application. Thereafter, the use of the Ravenscar Profile was generalized to be adopted in early development phases, particularly during the design phase [GSP+11, BW94]. Its use is notably preferred during the design phase because it facilitates the system schedulability analysis. Indeed, certain Ravenscar restrictions are related to the task set model and the scheduling policy, which helps designers to know suitable schedulability tests that can apply. Some of these restrictions are expressed as follows.

- Tasks must be periodic or sporadic and synchronous.
- The scheduling of tasks is performed according to a preemptive and FPS policy.
- PCP is used for synchronizing tasks to access shared resources.
- etc.

By exploring the literature, several schedulability tests could be applied in the context of Ravenscar compliant task set model running a uniprocessor platform. Most prominent schedulability tests compatible with the Ravenscar scheduling context are reviewed in the remainder of this section.

RTA test:

As a reminder, the RTA test consists in verifying if the WCRT of each task is less than or equal to its deadline. The $WCRT_i$ of a task τ_i is computed by iterating the following recurrent relation until reaching a fixed point [JP86, SRL90].

$$\begin{cases} WCRT_i^{(0)} = C_i + B_i \\ WCRT_i^{(k)} = C_i + B_i + \sum_{\tau_j \in hp_i} \left\lceil \frac{WCRT_i^{(k-1)}}{T_j} \right\rceil \cdot C_j \end{cases} \tag{1.1}$$

Where hp_i represents the set of tasks with higher or equal priority to task τ_i and $WCRT_i^{(k)}$ is the k-th estimate of $WCRT_i$. As shown in these formulas 1.1, the computation of the WCRT of tasks requires the knowledge of the blocking time term B_i. The usual way of computing the blocking time term is an upper

bound (not the actual blocking that occurs at runtime), since the exact computation of each blocking time term requires a combinatorial search [But11]. Hence, in the presence of shared resources, the RTA test is only a sufficient schedulability condition. However, when tasks are independent (absence of shared resource), the blocking term is omitted from the above formulas and the RTA test will be an exact schedulability condition. In [BB08], authors proved that the RTA test is sustainable with respect to execution times, periods and deadlines (i.e. C-sustainable, T-sustainable and D-sustainable). This test has a *pseudo-polynomial* computational complexity.

Processor utilization based test:

In [LL73], Liu and Layland have formulated two seminal schedulability tests for independent periodic task systems under RM and DM scheduling algorithms.

Theorem 1 ([LL73]). *Given a system S of n independent periodic tasks with implicit deadlines (i.e. $\forall i$, $D_i = T_i$), if $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$, which converges to $\ln 2 (\approx 0.69)$ when $n \to \infty$, then the system S is RM-schedulable.*

Theorem 2 ([LL73]). *Given a system S of n independent periodic tasks with constrained deadlines (i.e. $\forall i$, $D_i \leq T_i$), if $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right)$, then S is schedulable under DM algorithm.*

These two tests have a linear complexity $O(n)$. In the context of synchronous tasks, they represent exact schedulability conditions [LL73]. According to authors of [BB08], these schedulability tests are also sustainable with respect to execution times, deadlines and periods.

Later, authors of [SRL90] have extended the processor utilization test for RM to support the presence of shared resources under PCP control by including the blocking time term B_i (by convention, tasks are ordered by increasing periods):

$$\forall i \in [1..n], \frac{B_i}{T_i} + \sum_{j=1}^{i} \frac{C_j}{T_j} \leq i\left(2^{1/i} - 1\right) \tag{1.2}$$

With the presence of shared resources, the processor utilization test became only a sufficient schedulability condition [SRL90] (because blocking time terms are upper bound values). In addition, its computational complexity became $O(n^2)$. But, it persists sustainable with respect to execution times, deadlines and periods [BB08].

Simulation based test:

According to authors of [CGG04], when tasks are synchronous, the periodicity of the entire task set is defined by the *hyperperiod* H [LW82] that is equal to the LCM of all the task periods, i.e. $H = LCM_{\forall i}(T_i)$. Thus, $[0..H]$ represents the simulation interval required to verify the schedulability. The computational complexity of the simulation based test is exponential since the simulation interval is equal to H that may grow exponentially with the number of tasks.

In the presence of shared resources, the simulation test is not sustainable with respect to execution time when considering an implementation on an on-line scheduler [BB08, GGCG16]. In fact, the simulation explores one execution trace assuming that task execution times are exactly equal to their WCETs and critical sections durations are set to the maximum durations. Unfortunately, worst behavior of individual jobs does not often simulate the worst behavior of the entire system. For instance, a small decrease in one execution time value or a critical section duration can cause a change in execution order which may induce an increase in some blocking time [BB08]. Nevertheless, according to authors of [GGCG16, XP00], an off-line dispatcher that executes infinitely a feasible schedule computed in advance, can be effective (i.e. preventing scheduling anomalies) in the context of tasks sharing resources. On the other hand, according to authors of [GGCG16], when tasks are independent (no shared resources), the simulation test is sustainable with respect to execution times (i.e. C-sustainable). Moreover, they claimed that for C-sustainable contexts, while considering the worst system behavior in the scheduling simulation (i.e. using the WCETs of tasks as their execution times), the simulation based test is exact.

Synthesis

Table 1.1 summarizes schedulability tests described above by highlighting their main features in terms of the schedulability condition kind (i.e. sufficient, necessary or exact), the sustainability and the computational complexity.

TABLE 1.1: Characteristics of most common schedulability tests compatible with Ravenscar compliant task set running uniprocessor platform

Features / Schedulability test	*schedulability condition kind*	*sustainability*	*complexity*
RTA [SRL90]	sufficient	sustainable w.r.t all parameters	pseudo-polynomial
Processor utilization [SRL90]	sufficient	sustainable w.r.t all parameters	$O(n^2)$
Simulation on $[0..H]$ [LM80]	-	not sustainable	exponential

Schedulability tests characteristics provided in Table 1.1 are determined while assuming the presence of shared resources for the task set to be processed by these schedulability tests. Yet, as shown in Table 1.2, some characteristics may change when these schedulability tests are applied in the context of independent tasks.

TABLE 1.2: Characteristics of most common schedulability tests for Ravenscar compliant task set composed of independent tasks running uniprocessor platform

Features / Schedulability test	*schedulability condition kind*	*sustainability*	*complexity*
RTA [JP86]	exact	sustainable w.r.t all parameters	pseudo-polynomial
Processor utilization [LL73]	exact	sustainable w.r.t all parameters	linear
Simulation on $[0..H]$ [LM80]	exact	sustainable w.r.t all parameters	exponential

To sum up, all the outlined schedulability tests are only sufficient in the presence of shared resources (i.e. including the blocking terms in schedulability tests). This is due to the fact that blocking conditions are established in worst-case scenarios that differ for each task and could never occur simultaneously [But11]. Furthermore, when tasks are independent, the schedulability problem is relatively "easier". That's why all the presented schedulability tests are exact and sustainable in the context of independent tasks.

1.6 Conclusion

In this chapter, key concepts related to RTE systems were introduced. A major focus was laid on the real-time scheduling domain since the analysis and the validation of the temporal behavior of RTE systems are crucial steps in the design process of these systems.

In order to master the development complexity of RTE systems and to enhance/optimize their performance, a great interest has been given, in the last decades, towards the design process and particularly the design space exploration (DSE) and optimization. The major challenge is how to guide the DSE towards correct (logical and temporal correctness) and optimal (regarding several performance criteria) design alternatives. In the literature, the DSE of RTE systems have been tackled as a multi-objective optimization problem (MOOP). Thus, solution methods proposed for solving MOOPs are typically exploited to cope with the DSE issue.

The next chapter introduces the domain of interest for this thesis, which is mainly concerned with RTE systems design and more precisely the DSE process. It starts by presenting the development and design of RTE systems. Then, it exposes key elements of MOOPs and solution methods for solving them.

2

RTE Systems Design and Optimization: Background and Basic Concepts

Contents

2.1 Introduction

In the previous chapter underlying concepts related to RTE systems were detailed, especially concerning tasks and scheduling analysis allowing the verification of timing constraints imposed on such systems. In order to obtain a RTE system structured into a set of tasks satisfying the system specifications (mainly the timing constraints), a special attention has to be drawn to the development of

these systems. Indeed, any incoherent development may induce a non-compliance and a violation of timing constraints, and may sharply increase the development time and costs.

The main intent of this chapter is to introduce the domain of interest for this thesis, which is essentially concerned with the design and optimization of RTE systems. The design of RTE systems plays a central role in the development flow of these systems, since the foundation for both the fulfillment of the system requirements and the optimization of its performance criteria (often conflicting) has to be set during the design phase. Indeed, RTE systems designers are always faced with several design decisions/choices and they struggled to make the most suitable decisions by exploring a large space of design alternatives. Accordingly, the design of RTE systems falls into *multi-criteria decision-making* category of problems that are frequently solved using the so-called multi-objective optimization (MOO) techniques.

To understand the importance of the design during the development of a RTE system, Section 2.2 starts by introducing the whole development process, then emphasizes the specificities of the design phase. Section 2.3 provides an overview about MOO area by highlighting its underlying concepts and techniques. Finally, Section 2.4 concludes the chapter.

2.2 Development and design of RTE systems

The development of RTE systems needs a careful attention, since they have to control physical and critical environments where the slightest failure is not tolerable. In order to meet the increasingly complex requirements of these systems, many characteristics should be taken into consideration during their development, such as [TKK+98]: (a) anticipation of the risks and the problems at early development stages, (b) reducing the development time and costs, (c) mastering the complexity, (d) easing the cooperation between multiple actors (or teams) working at different phases of the development process, just to name a few.

RTE systems are considered as heterogeneous systems since they are composed of hardware (HW) and software (SW) components that always execute concurrently. This heterogeneous nature, with the ever-increasing evolution in HW and SW technologies and the requirements mentioned above, become dominant factors behind the development complexity of these systems. To deal with this inherent complexity, dedicated and sophisticated development methodologies have been proposed. Thus, as a particular kind of computer systems, the development of RTE systems may follow different approaches.

In the sequel, first, Section 2.2.1 highlights the software development (called also software engineering) process as part of the whole RTE system development

elements. Afterwards, Section 2.2.2 focuses on the boundary between HW and SW parts by introducing the HW/SW *co-design* methodology that has proven its effectiveness to cope with development challenges of those heterogeneous systems. Subsequently, Section 2.2.3 highlights key concepts underlying the design phase being the backbone of the development of systems under consideration.

2.2.1 Software development process

Because of the critical and complex nature of a RTE system, and the different expertise required for the establishment of its different entities, the software development (or engineering) of such a system is achieved by following a set of phases described by a *development cycle* (called also *software life-cycle*).

Definition 20. ***Development cycle*** *[Bla04] is an assisted treatment process that decomposes the development of a product into a set of steps structured according to a certain philosophical approach.*

The decomposition of concerns allows to master development time and costs. There exists various conventional models of development cycles [Bla04], such as the *"V" model*, the *"spiral" model*, the *"waterfall" model*, the *"Y" model*, etc. Despite differences between the different development cycle models, all of them comprise the same principal phases that are staggered over time:

- requirements specifications,
- design,
- implementation,
- integration,
- validation,
- exploitation and utilization.

Each phase produces a set of outputs that must be validated by the responsible actors (i.e. designers for design phase, developers for implementation phase, end-users for exploitation phase, etc.). The outputs of a given phase are used as the inputs of the next phase according to the structure defined by the development process. Hereafter, the different phases are explained and illustrated within a "V" model development cycle.

Illustration with "V" model development cycle

As shown in Figure 2.1, the "V" model consists of two distinct slopes: a *downward* slope and an *upward* slope (hence the "V" shape). The downward slope

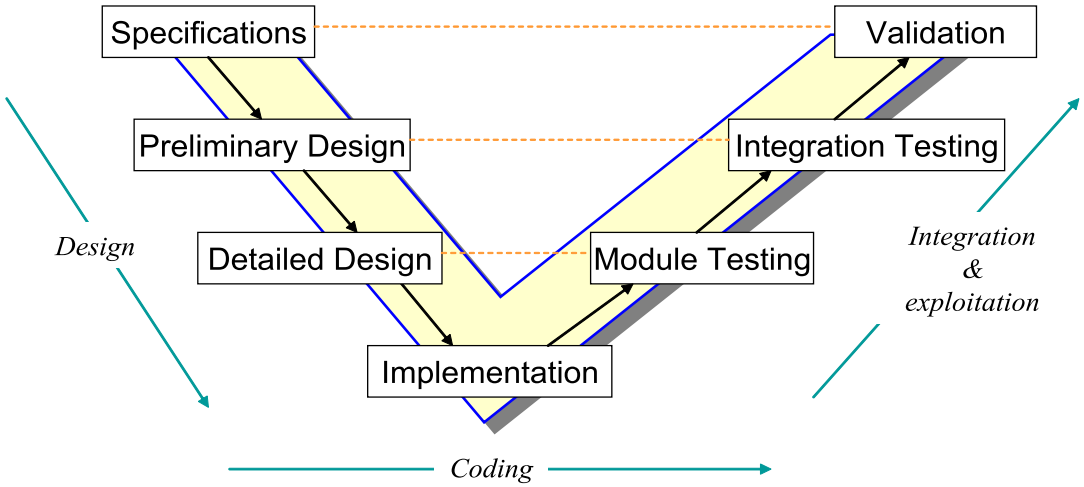


FIGURE 2.1: "V" model development cycle (taken from [Ouh13])

describes the progressive phases from specification to implementation. Subsequently, the upward slope focuses on the verification (e.g. testing) and validation of the outputs of each phase from the downward slope. That's why, each phase from the downward slope is linked with another phase from the upward slope. Each phase in the "V" model is detailed below.

- **Specification**: The specification phase is the first phase in the system life-cycle and it is devoted to properly delineate the system requirements. It lies in the identification of the services to be provided by the system according to the end-users' needs as well as the constraints imposed on the system. Thus, two kinds of requirements are distinguished: functional and non-functional. Functional requirements describe the system behavior, i.e. what the system is intended to do. While non-functional requirements define constraints and properties related to the system performance (or quality-of-service). For instance, referring to the context of RTE systems, the timing constraints represent a sub-set of non-functional requirements. The specification of requirements can be expressed in a formal or semi-formal language or even informally by means of a natural language.
- **Design**: The design phase defines the internal structure of the system and the interactions between its different components referring to the requirements previously identified (i.e. in the specification phase). Within the "V" model approach, the design phase is divided into two stages, namely, the preliminary design and the detailed design. The preliminary design stage delivers a *high-level* system architecture describing the required components, their interfaces and the functions they must implement. The detailed design stage refines the system preliminary design by detailing the description of each component

and specifying data structures, communication protocols, etc. Many design formalisms could be used in order to express the system design. For example, several standard design languages have been proposed to drive the design of RTE systems, such as MARTE [OMG08] or AADL [FG12]. These design languages provides support for specification, design, and validation stages.

- **Implementation**: This stage is used to implement the application by encoding all components proposed through design phase in a programming language. For example, the implementation of real-time applications can be performed using a concurrent language that provides specific constructs for writing concurrent programs (e.g. Ada) or a sequential language that calls some primitives of the host OS to manage concurrency.
- **Module testing**: The integration and exploitation slope starts with module testing phase. The latter consists in testing each component separately in order to verify its correct behavior and the consistency with its specification.
- **Integration testing**: Once each component of the software is validated, they are assembled and subsequently integration tests are applied in order to check and validate components relationships and interactions, communication and synchronization mechanisms, etc.
- **Validation**: In this phase, the entire system is tested in order to check that the developed software meets all the requirements identified in the specification phase. Moreover, end-users can verify if the software product fulfils their needs and expectations.

Each software development approach has its benefits and its limitations. In the context of real-time software development, a common shortcoming of traditional approaches is that the verification and validation are performed late in the software life-cycle (i.e. during or after the implementation phase). Thus, if an error occurs in one of the early phases, it would spread to the other subsequent phases, and necessary corrections may not be achieved until the end of the last phase.

2.2.2 Hardware/software interface co-design

A RTE system involves a software layer as well as a hardware part on which the software will run. As a matter of fact, some of the system requirements are stemmed from the hardware platform. Although we have focused on the software development of RTE systems, considering the hardware part during the life-cycle development is crucial in order to ensure the respect of all the requirements by the final product.

Indeed, key problems in RTE systems development are due to the close conjunction between the software and the hardware parts that makes the boundary

between the two parts increasingly complex. To cope with the inherent complexity of such heterogeneous systems, more sophisticated and relevant development methodologies, have been established called *hardware/software (HW/SW) co-design* [DMG97, LHG13, Tei12]. The HW/SW co-design approach emphasizes the concurrent development of HW and SW parts, that is, enabling mutual influence of both parts at an early stage in the development process. Such an approach is intended to alleviate the strict time-to-market pressure, satisfy requirements and optimize the quality of the final product by exploiting the synergy of HW and SW.

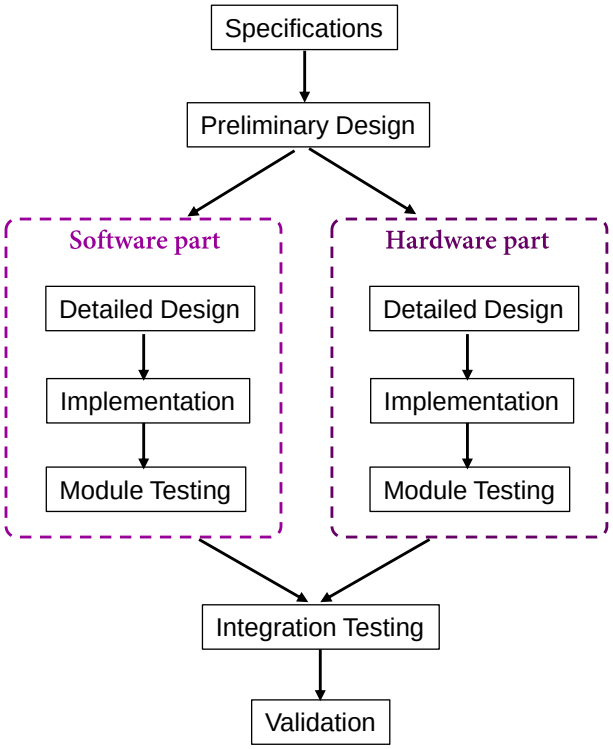


FIGURE 2.2: Classic hardware/software concurrent development process

Figure 2.2 shows a typical concurrent development process for a combined HW/SW system. Front-end phases such as specification and preliminary design simultaneously consider HW and SW aspects. Likewise, back-end integration testing and validation are applied on the entire system. Whereas at the middle of the process, the development of each part (i.e. HW and SW) is proceeded relatively independently of the other.

Conventionally, HW/SW design and verification take place after some essential decisions have been already taken. This is what makes the HW/SW integration problem increasingly hard. In fact, experiences on real-world applications show that wrong design decisions (made in the early stages of development but often

detected after implementation) strongly affects development costs and performance of the final implementation. The study provided in [NIS02] has reported that a major part of development costs (about 80%) are due to the cost of the backtracking in the development process.

In order to better master the decision making process, RTE systems industry has stressed the need of dealing with major HW/SW integration challenges and decisions at **higher levels of abstraction** and early in the development process i.e. **at design phase**. That is, almost all development efforts are concentrated in the design phase rather than implementation phase. Subsequently, the system implementation is automatically (as much as possible) generated from the final system design so that to ensure *"correct by construction"* implementations.

2.2.3 Design phase: Y-chart design paradigm

Designing RTE systems from scratch was no longer practical strategy as the complexity of such systems and the time-to-market pressure grow relentlessly. Instead, *design reuse* [FSV99, KNRSV00] between multiple products was the key to being effective (i.e. raising productivity and reducing overall costs), ensure the required quality and hit the market on time [Bal97]. Indeed, the quest for design reuse has driven the RTE systems industry towards designs that could be "quickly" built from *pre-designed* and *pre-characterized* components rather than full custom design strategies [SV02]. More precisely, HW/SW resources (e.g. CPUs, cores, memories, buses, RTOSs, etc.) are usually pre-designed and standard parts that will be selected by designers to establish the execution platform on which the software application will run [KNRSV00, Wol03]. At this stage, designers face several challenging questions, such as:

- Which execution platform is most appropriate for realizing the required system functions (i.e. the application)? i.e. how to select HW/SW resources (type, number, etc.) such that all the system functional and non-functional requirements will met?
- How to assign the application entities onto the execution platform components?
- Is the final system will respect its requirements and answer at best to performance properties?
- etc.

In that respect, the *Y-chart* [Bal97, KDVVDW97, KDWV02] has emerged as a mainstream paradigm/scheme for the design of heterogeneous systems which assists designers to cope with the above questions.

As Figure 2.3 shows, in the Y-chart an obvious distinction is made between the application (what the system is supposed to do) and the execution platform, which are subsequently joined/coupled through an explicit *mapping* step. The

philosophy of separating application part from execution platform part, known as the *separation of concerns* [SVDN07], fosters design reuse and extendibility of each part.

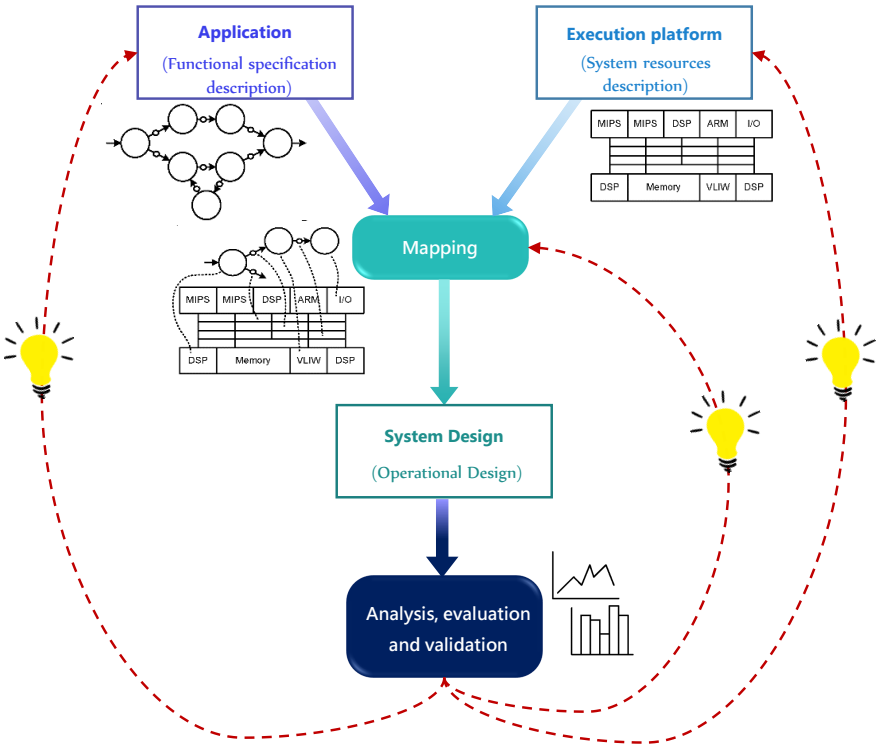


FIGURE 2.3: Conceptual view of the Y-chart design paradigm

Formally speaking, key tenets underlying the Y-chart paradigm are highlighted in the following five-step approach:

(1) **Creation of the functional specification**: The *functional specification* (called also functional architecture) is a description of the application (i.e. a formal specification of the system structure and behavior) in terms of functions and their interactions. Moreover, it includes the different services required by the application (like computation services, storage services, etc.) as well as all the requirements gathered during the specification phase. This functional specification must be created at a high level of abstraction, which means regardless of any specific platform or implementation technology.

(2) **Creation of the execution platform specification**: The *execution platform* (sometimes called just *platform* or *architecture* [SVDN07]) specification captures, on the one hand, the structure and prominent characteristics of the HW resource set (e.g. computing, storage and communication resources) that can support the application. On the other hand, it specifies

SW resources properties (such as the scheduling algorithm used by the scheduler, etc.) involved in the management of HW resources.

The specification of the execution platform must be captured at the highest level of abstraction by getting rid of lower-level details. But, at the same time, it has to provide enough information on parameters needed to allow analysis (e.g. scheduling analysis) and evaluation of performance with a fairly accurate prediction of the final implementation properties.

(3) **Mapping**: For a given application, the explicit mapping defines how the functional specification (i.e. functions and their interactions) will be assigned to resources of a candidate execution platform (i.e. processors, tasks, shared resources, buses, messages, etc.). Moreover, parameters required for establishing services related to the execution of the application (for example, task priority assignment needed for the scheduling of the application) are set during the mapping step. The latter results in an **operational design** representing an abstraction of the system implementation. In other words, the operational design determines all entities necessary for the implementation of the system in terms of concurrent tasks communicating by sharing a set of common resources.

(4) **Analysis, evaluation and validation**: By defining a mapping, it is possible to ask questions about the feasibility and performance of the overall design solution. There are several criteria (e.g. reliability, energy consumption, cost, resources utilization, timing, etc.) which can be predicted/computed and then analyzed to assess the resulting performance.

(5) **Design space exploration**: The evaluation of performance may show unsatisfactory results. These results may inspire designers to improve certain design choices at the different levels:

- At functional specification level: e.g. restructure the application;
- At execution platform level: e.g. modify resources dimensioning and structure, memory size, scheduling policy, etc;
- At mapping level: e.g. alter the assignment of functions to tasks.

Y-chart paradigm envisions to iteratively apply such improvements until finding an operational design solution that satisfies all requirements and responds at best to the end performance. The set of possible design alternatives forms the *design space*. The process of exploring several design alternatives is referred to as *design space exploration* (DSE).

By exploring the literature, we have noticed that the implementation of the Y-chart paradigm by RTE systems design methodologies [LVDWVD01, Pin04, BHM+05, SVDN07, EAP17] has been and still remains a very active research area. In fact, the adoption of the Y-chart scheme raises several challenges for the

implementation of its different steps. Y-chart design methodologies have aimed to support designers by providing: (a) a set of formalisms allowing to easily define models/abstractions for the functional specification, the execution platform and the design solutions i.e. these formalisms assist designers in the *modeling* task, (b) techniques for analyzing design solutions, and (c) tool based methods for searching the design space.

Traditionally, decisions-making and the DSE process are based on designers knowledge and experience. This manual DSE process is an iterative *trial-and-error* procedure [ZG11]. Unfortunately, due to aspects characterizing an RTE application (e.g. timing constraints, concurrency, restrained resources, resources sharing, etc.) coupled with the conflicting nature between performance criteria, a trial-and-error design strategy increases the time and cost of the overall design process. Furthermore, given the huge size of the design space, such a design strategy is not able to efficiently explore that design space since it allows to explore only a small portion of the overall design space. Thus, at the end of the DSE process, the correctness of the design is ensured, but its optimality is unknown.

Accordingly, the automation of the DSE process has been crucial for satisfying all design requirements. The underlying idea is to provide a method that allows to systematically browse a large space of design solutions, evaluate and compare solutions, to finally end up with the most suitable design solution(s). By most suitable design solutions, we mean those that can best handle the target application while ensuring the respect of all non-functional requirements and answering at best to a set of performance criteria. This DSE problem is typically identified as a **multi-objective combinatorial optimization problem** [LLP+09, MWTP+13].

To sum up, given a functional specification and an execution platform, the crux of the design is concerned with an automated DSE method that identifies the most suitable final design solution(s) representing "optimal" mapping of the functional specification into the target platform.

2.3 Multi-objective optimization (MOO)

Multi-objective optimization problems (MOOPs) arise in various areas [LA15] such as: industrial applications (e.g. manufacturing, design, management, etc), scientific applications (e.g. bioinformatic and chemistry fields, protein design, drug design, water resource systems design, etc). Such problems are characterized by the presence of multiple mutually *conflicting* objectives (or criteria) that need to be optimized simultaneously, and they are associated to several constraints. By mutually conflicting objectives we mean that an attempt to improve one objective leads to the degradation of others. Thus, no single solution can be considered

optimum with respect to all objectives, and then decisions need to be taken in order to define compromises (or trade-offs) between objectives.

Multi-objective optimization techniques provide disciplined search strategies and guidelines to resolve different MOOPs. In fact, MOO techniques are used to browse a large search space, compare solutions in the search space with respect to multiple objectives, and finally produce optimal solutions.

Multi-objective evolutionary algorithms (MOEAs) have been increasingly popular, mainly because of their suitability for MOOPs with difficult search landscapes (e.g. large solution spaces, constraints, non-linear and non-differentiable objective functions, etc.) [AM10]. Measuring and assessing MOEAs performance is a challenging task that has been extensively investigated by the MOO community. In this regard, several performance indicators (or metrics) have been developed.

In the following, Section 2.3.1 introduces fundamental aspects related to MOOPs and MOO. Section 2.3.2 discusses about techniques used for solving MOOPs. Section 2.3.3 presents an overview on MOEAs by highlighting their basic concepts. Section 2.3.4 points out some of the most prominent MOEAs performance indicators.

2.3.1 Fundamental concepts and terminology in MOO

Authors of [Osy78] defines MOO as a problem that has to solve "a vector of decision variables" meeting constraints and optimizing a vector function where each element is an objective function. Mathematically, a MOOP can be formulated as follows:

$$\begin{cases} Optimize \quad F(X) = [f_1(X), f_2(X), ..., f_k(X)] \\ Subject\ to \quad X \in \mathcal{D} \end{cases} \tag{2.1}$$

where $X = (x_1, x_2, ..., x_n)$ is the vector of decision variables, k is the number of objectives ($k \geqslant 2$), $\mathcal{D}$ is the set of feasible solutions, i.e. those satisfying all the constraints imposed on decision variables, and finally, each objective function $f_i(X)$ is to be optimized (i.e. minimized or maximized).

Given the conflicting nature between objectives, the *Pareto dominance relation* is usually adopted in MOO to compare candidate solutions.

Definition 21 (Pareto dominance relation [HCFDC09, CDJ10]). *A candidate solution X^1 is said to **dominate** another solution X^2 in the objective space, if and only if: (i) X^1 is strictly better than X^2 for at least one of the objective and (ii) X^1 is not worse than X^2 for any of the objectives.*

Solving a MOOP is concerned with finding all *non-dominated*, called also *Pareto optimal*, solutions.

Definition 22 (Pareto optimal/non-dominated solution [CLVV07]). *A solution is said to be* ***Pareto optimal*** *or* ***non-dominated*** *with respect to a set of objectives if there does not exist another feasible solution in the objective space that improves on all of the objectives at once.*

These non-dominated solutions form the *Pareto set*.

Definition 23 (Pareto set [JCAT14]). *The Pareto set $\mathcal{P}$, is composed by all the non-dominated solutions of $\mathcal{D}$.*

Definition 24 (Pareto front [JCAT14]). *The image of the Pareto set $\mathcal{P}$ in the objective space constitutes the Pareto front $PF = \{F(X)|X \in \mathcal{P}\}$.*

An example multi-objective optimization problem domain is depicted in Figure 2.4. The left-side illustration shows a decision-space where decision vectors

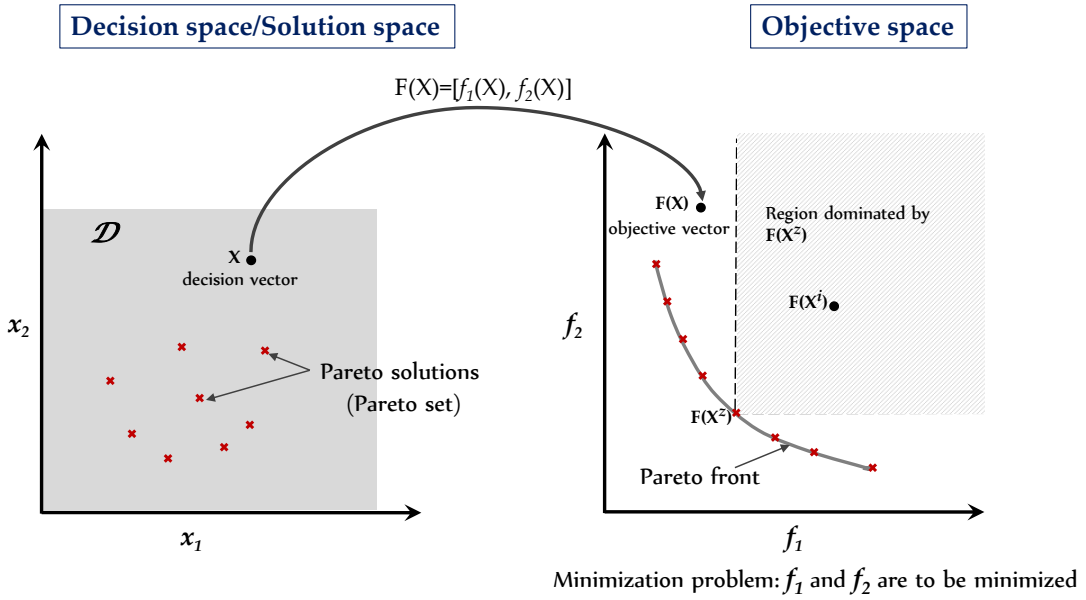


FIGURE 2.4: Illustration of the multi-objective optimization problem domain: mapping from decision space to objective space (assuming two decision variables and two objective minimization functions).

(or solutions) are composed of two variables. The problem constraints imposed on solutions lead to a feasible region $\mathcal{D}$ (highlighted in grey). Each decision vector in the decision space corresponds to a unique objective vector in objective space as shown in the right-side illustration. It is worth noting that the inverse mapping may be non-unique, i.e. for an objective vector may corresponds different solutions. Figure 2.4 also shows the Pareto set and the associated Pareto front that represents the best trade-off set for the considered objectives. Note that this example shows two decision variables and two objectives, however, the decision variables number and the objectives number are independent and each of them may be any positive integer.

2.3.2 Techniques for solving MOOPs

With significantly increased attention on MOOPs, an overwhelming number of techniques for solving these problems have been developed. In fact, MOO techniques are a well established and wide research area [CDJ10]. In the literature, many comprehensive surveys on MOOPs and solution methods [UT94, EG00, EG04, EGP16, CWC+17] have been proposed since the nineties until now.

Traditionally, MOOPs are solved by transforming the problem into a single objective optimization problem through the so-called *scalar approaches* [CWC+17]. For instance, one well-known technique is the *weighted-sum* method [EG00]. This method combines multiple objectives using designer-specified *weights*, where each weight represents the importance of the associated objective function [CWC+17]. The selection of appropriate weight coefficients is of paramount importance and often raises a challenge, as it will sharply impact results. Scalar methods are quite simple and easy to use in the sense that they rely on established single-objective optimization techniques. However, their major drawback is that they produce a single solution for a given MOOP, rather than the Pareto solution set (i.e. any information about the trade-offs among different objectives) [CDJ10].

Since 2000, there have been considerable efforts in establishing methods to simultaneously identify many Pareto optimal solutions. These methods are either *exact* or *approximation* (*heuristic*) methods. Exact methods have the advantage of accuracy, as they are intended to enumerate all the Pareto solutions of a MOOP [EGP16]. But, these methods suffer from their inability to cope with large-scale problems, which require too much time and resources. Since almost all MOOPs are typically known to be $\mathcal{NP}$-*hard* [CDJ10, CWC+17], achieving an exact resolution of an arbitrary MOOP is quite difficult, and accordingly impractical. Hence, approximation methods have become the most common alternative to deal with large instance MOOPs [CDJ10]. The underlying idea of such approximation methods is to provide reasonably "good" approximations of the Pareto front and Pareto optimal solutions (called *near-optimal* or *sub-optimal* solutions), within limited computational time and cost. A key challenge for heuristic approaches is to balance between the quality of the approximate Pareto solution set, and the time and computation resources requirements.

When the approximation method refers to a *metaheuristic* one talks about multi-objective metaheuristic.

Definition 25 (Metaheuristic [BR03]). *A metaheuristic is a high-level algorithmic strategy for exploring the search space of a problem and identifying optimal and near-optimal solutions. It is used to guide other heuristics or algorithms.*

In the literature, there exists many multi-objective metaheuristic techniques such as simulated annealing [LT99], tabu search [GF00], ant colony optimization [GMCH07], particle swarm optimization [RSC06], evolutionary algorithms

[CLVV07], etc. Two fundamental challenges for those metaheuristic methods are the *accuracy* (how close to the Pareto set are the approximation solutions they provide) and the *diversity* (are the solutions numerous and with objective values well spread in the objective space). Typically, to cope with these challenges, metaheuristics rely on mechanisms that foster both the *exploration* (i.e. to explore undiscovered regions of the search space) and the *exploitation* (i.e. to exploit promising solutions previously found). The equilibrium between the exploration and the exploitation aspects is one of the key issues in any metaheuristic, since it significantly influences performance. As part of multi-objective metaheuristics, a central question concerns the assessment of their performance.

In the following, we focus on multi-objective evolutionary algorithms (MOEAs) by highlighting their underlying components. Afterwards, we expose mainstream performance indicators that have been proposed for assessing multi-objective metaheuristic methods.

2.3.3 Multi-objective evolutionary algorithms (MOEAs)

A large amount of MOO techniques is derived from MOEAs. These techniques have been applied to solve a wide variety of complex MOOPs, e.g., those that present intractably large search spaces, non-linear objective functions, etc. MOEAs are powerful *random-based* search strategies that mimic the biological evolutionary process.

They are founded on the basis of the three evolution principles: *reproduction*, *random variations*, and *natural selection* according to the *Darwinian* concept of *"Survival of the Fittest"* [CLVV07, AM10]. Just as in nature, the reproduction creates new *individuals* from previous ones. Random variations occur during the reproduction. When individuals compete for survival, then the fittest individuals will be selected as survivors while the weakest ones demise.

Basic terms and components of MOEAs are pictured in Figure 2.5. An individual represents an encoded solution to some problem. Typically, a solution structure is defined by means of a vector (or string) referred to as *chromosome* (or *genotype*). Each position in the chromosome contains information corresponding to what is called a *gene*. Each solution (or individual) is assigned with a *fitness* allowing to identify which individuals survive into the next *generation*. The fitness of a solution is measured by *fitness functions*. The objective functions (that are features of the problem domain), plays the role of the fitness functions (that are part of the algorithm domain) [CLVV07]. A set of individual solutions forms the so-called *population*.

Evolutionary (or *genetic*) *operators* manipulate a population aiming at improving the population by generating new candidate solutions (called *offspring* or *chil-*

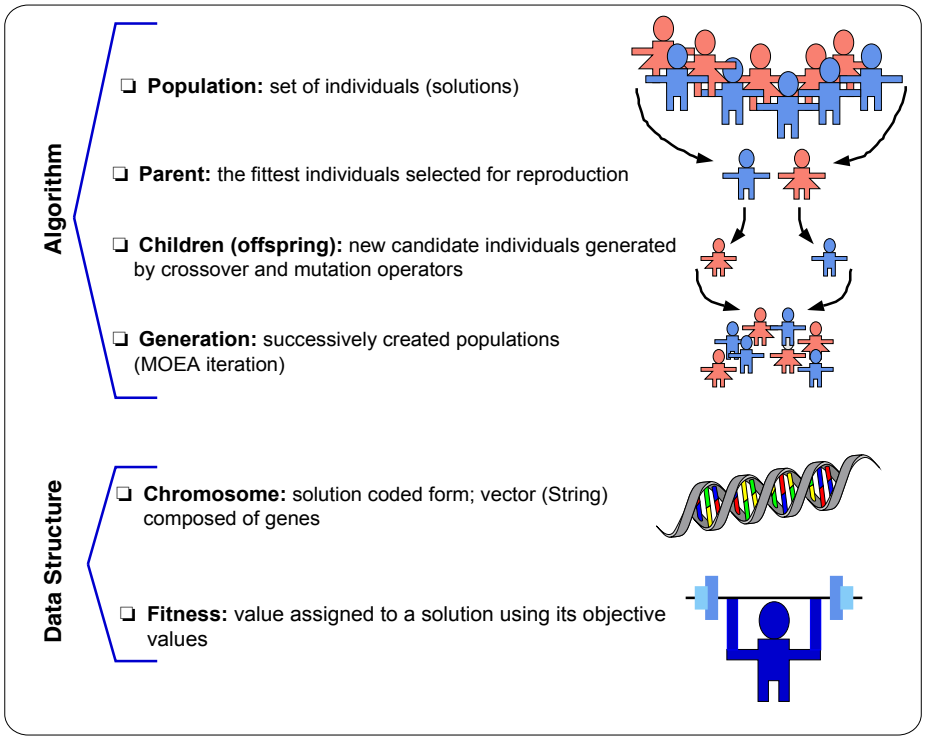
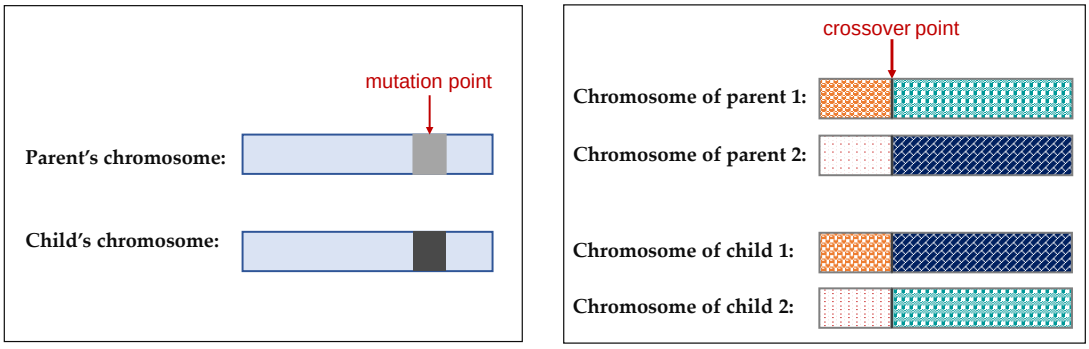


FIGURE 2.5: MOEAs basic terms and components (adapted from [CLVV07])

dren) that compete (based on their fitness) with the old ones for a place in the next generation.

There are three prominent genetic operators, namely: *mutation*, *crossover* (or *recombination*), and *selection*. The mutation and crossover operators (referred to as *variation* operators) are used during the reproduction phase. The underlying idea of these variation operators is to produce offspring by varying or combining the chromosomes of the individuals selected for reproduction (called parents). Illustrating this, Figure 2.6a shows an example of mutation that operates on one candidate parent to produce one new child by changing a gene of the parent's chromosome. Figure 2.6b depicts a form of crossover called *single-point crossover*, which operates on two parents to generate two children, where each parent's chromosome is cut and recombined with a piece of the other. The selection is the step to select the best or fittest individuals in order to guide the population towards optimal and near-optimal Pareto solutions.

A generic structure of a MOEA is outlined in Algorithm 1. A MOEA operates as follows. It starts its search with an initial population that could be created at random or user-defined (i.e. if some "good" solutions are known in advance, it is wise to exploit them in creating the initial population). Solutions of the initial population are evaluated according to the objective functions. Thereafter, the



(A) Mutation example (B) Crossover example

FIGURE 2.6: Examples of variation operators

Algorithm 1: MOEA generic structure

```
1  begin
2      Initialize population with random candidate solutions;
3      Evaluate fitness of each solution in the population;
4      repeat
5          Select parents;
6          Apply recombination and/or mutation operators on parents;
7          Evaluate fitness of new candidate solutions;
8          Select individuals for the next generation;
9      until (termination criteria are met);
10 end
```

MOEA procedure enters into an iterative process (lines $4-9$). In this process, evolutionary operators are involved to create new candidate solutions, update the current population and produce the next generation. The process stops when one or more pre-specified termination criteria are satisfied, such as reaching a computational limit (e.g. a fixed number of iterations), or converging towards a stable Pareto set, etc.

Each MOOP intended to be solved with a MOEA requires the definition of an encoding of solutions (i.e. the data structure of a chromosome), crossover and/or mutation operators, and objective functions. The encoding mostly differs from one MOOP to another. Some standard crossover or mutation operators can be reused for different MOOPs. However, in many cases *customized* variation operators may help the algorithm in achieving faster and more effective exploration of the search space [LEEC11].

It is worth noting that some MOEAs (e.g. PAES, details of this MOEA are given in Section 4.2.1) don't work with a population of solutions, instead they manipulate a single solution per iteration. Indeed, the disadvantage of using a

population of solutions is the computational cost and memory associated to the execution of each iteration [Deb08].

2.3.4 MOEAs performance metrics

The outcome of MOEA techniques is an approximation of the optimal Pareto front. We define the PF_{true} and the PF_{approx} as follows.

Definition 26 (PF_{true}). *is the theoretical optimal Pareto front which is not explicitly known a priori for problems of any difficulty.*

Definition 27 (PF_{approx}). *is an approximation set of the optimal Pareto front.*

Two crucial issues arise about: (1) how to evaluate the quality of an approximation set produced by a particular MOEA, and also (2) how to compare approximation sets obtained by different MOEAs. To tackle these issues, several quality indicators have been proposed [CLVV07, RVLB15, ZTL+03]. These indicators measure the quality of an approximation set regarding to one or more of the following aspects:

- **Accuracy/convergence** [ZTL+03, RVLB15]: this aspect is associated to the closeness of PF_{approx} to PF_{true} (i.e. how distant is PF_{approx} from PF_{true}).
- **Diversity** [ZTL+03, RVLB15]: this aspect is related to the *distribution* as well as the *spread*. The distribution refers to the relative distance among elements in PF_{approx} while the spread refers to the extent of PF_{approx} across the objective space (i.e. the range of values covered by elements in PF_{approx}).
- **Cardinality** [OJS03, RVLB15]: this aspect concerns the number of solutions associated to elements in PF_{approx}.

The overwhelming majority of existing quality indicators are *unary* [RVLB15]. The indicator is said to be unary if it takes as parameter only one approximation set PF_{approx} (obtained by a particular MOEA) to be assessed. These indicators transform a given PF_{approx} into a single value that is meant to reflect one or more of the quality aspects given above.

One category of unary indicators assume that theoretical optimal Pareto front PF_{true} is known and their computations rely on it, such as the *inverted generational distance* (IGD) [VVL98b], the *optimal Pareto front coverage ratio* (CR) [UTFT99], etc. However, for most real-world complex MOOPs (i.e. combinatorial, large-scale, with hard constraints, etc.) the attainment of theoretical optimal Pareto front is very difficult and even impractical. Thus, another category of unary indicators are defined on the basis of PF_{approx}, an attainment set by definition. For this category, the study presented in [RVLB15] has shown that

the *hypervolume* (HV) [ZT98] is the most popular indicator and commonly used by the MOEA community.

In the remainder of this section, we present the mentioned indicators, i.e. HV, IGD and CR.

Hypervolume indicator (HV)

The hypervolume indicator (HV) includes all three quality aspects (i.e. accuracy, diversity and cardinality), being the only unary indicator with this capability [RVLB15]. As shown in Figure 2.7, this metric computes the area of the objective space covered by the considered PF_{approx}, bounded by some reference point. The reference point is selected so that it is dominated by all elements in the approximation set to be evaluated. For instance, the reference point is frequently formed using the worst value for each objective, known as the *anti-ideal* or *Nadir* point. The greater the HV value, the better the approximation set.

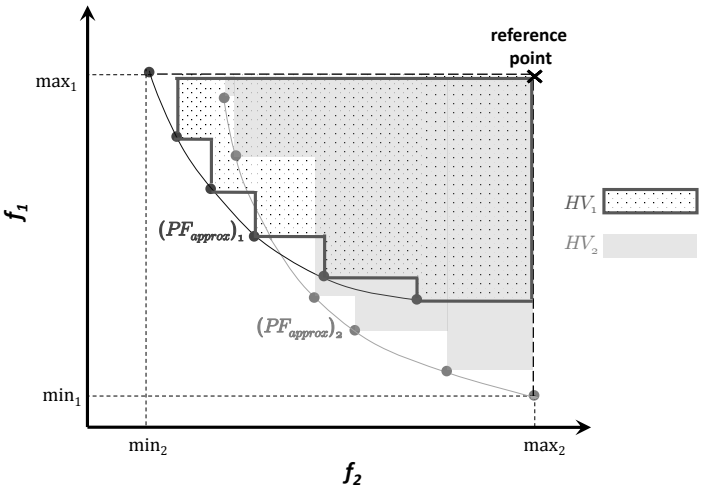


FIGURE 2.7: Hypervolume performance indicator for two sets, for a two minimization objectives problem

In order to allow the objectives to contribute approximately equally to the HV indicator values, objective values of elements in the approximation set must be normalized. A standard, linear normalization technique [FKTZ05] consists in applying the following transformation to each objective dimension values:

$$f_i^{normalized} = \frac{f_i - f_i^{(min)}}{f_i^{(max)} - f_i^{(min)}}$$

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 approximation set. The reference

point is also normalized. With this normalization technique, HV values are in the range [0..1].

Inverted Generational Distance (IGD) metric

The IGD metric [VVL98b, ZZZ+08] can be adopted when PF_{true} is known. It is defined by the following expression.

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

where $d(\nu - PF_{approx})$ is the euclidean distance between each element ν in PF_{true} and the nearest member of PF_{approx}, and $|PF_{true}|$ is the number of elements in PF_{true}.

The IGD indicator could measure both the accuracy (or convergence) and the diversity [RVLB15]. The diversity is inferred by the coverage capacity that means how well PF_{true} is covered by PF_{approx}. Indeed, a PF_{approx} whose elements are located on a limited area of the extent of the PF_{true}, will be penalized in its IGD value even if its elements belong (or are too near) to PF_{true}. For the IGD metric, the lower are values, better are corresponding approximation sets.

As the hypervolume metric, the IGD indicator also needs to be normalized in order to generate comparative values (i.e. in an absolute sense). Objective values of elements in both PF_{true} and PF_{approx} are normalized using the same technique described above. Normalized IGD values are in the range $[0..\sqrt{2}]$.

Optimal Pareto front coverage ratio (CR)

The optimal Pareto front coverage ratio (CR) metric is among cardinality-based indicators [OJS03]. It measures the proportion of optimal Pareto elements reached by the approximation set PF_{approx}:

$$CR = \frac{|PF_{approx} \cap PF_{true}|}{|PF_{true}|}$$

Mathematically, this metric is computed using the following expression:

$$CR = \frac{\sum_{i=1}^{|PF_{approx}|} c_i}{|PF_{true}|}$$

$$where \quad c_i = \begin{cases} 1 & \text{if } i^{th} \text{ element of } PF_{approx} \text{ is a member of } PF_{true}, \\ 0 & \text{otherwise.} \end{cases}$$

Ideally, CR is equal to 1, which indicates that PF_{approx} is identical to PF_{true}. In other words, the algorithm converges to the optimal Pareto front. The worst situation is when $CR = 0$, it points out that none of elements in PF_{true} are attained in PF_{approx}. In other situations, higher values of CR are better. The example shown in Figure 2.8 has $CR = \frac{1}{4} = 0.25$.

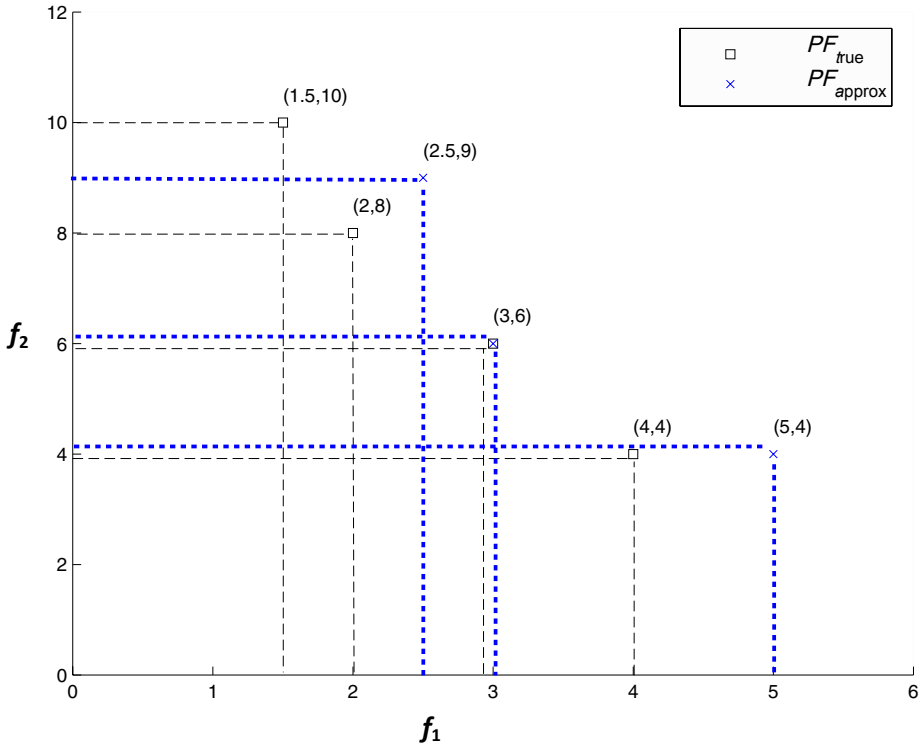


FIGURE 2.8: PF_{approx} and PF_{true} example to show the CR metric (taken from [CLVV07])

2.4 Conclusion

The first part of this chapter has focused on the development and design of RTE systems by exposing their key phases. This study led us to highlight major challenges faced by designers during the mapping stage. The mapping stage consists in merging the functional specification of the system within the targeted execution platform while ensuring the respect of non-functional constraints (mainly

timing constraints) as well as the optimization of the system performance criteria. The outcome of this mapping step is called the operational design, which represents an abstraction of the system implementation. The mapping step requires the exploration of different design alternatives (i.e. different combinations of the functional specification with the execution platform) in order to select those that meet the timing constraints and respond at best to the performance criteria. This is referred to as design space exploration and it is typically identified as a multi-objective optimization problem. Thus, the second part of this chapter was devoted to present underlying concepts on MOOPs and MOO techniques for solving them. Especially, key principles and general approach underlying MOEA techniques were introduced. Afterwards, an overview about performance metrics used for assessing MOEAs was provided.

The next chapter highlights the work orientation by exposing problem statements and contributions of this thesis. It also presents and discusses the related work.

3

Work Orientation and Related Work

Contents

3.1 Introduction

Having introduced research foundations and background in the previous chapters, we now point out the main axis of our work and discuss related work. Thus, in

Section 3.2, we enumerate the problem statements addressed in this thesis. Then, Section 3.3 provides the assumptions of this work delineating the application scope of our contributions. This section introduces also the basic concepts about the system model and the notations adopted throughout this thesis. Afterwards, Section 3.4 exposes our goals and contributions. Finally, Section 3.5 discusses the related work and Section 3.6 concludes the chapter.

3.2 Problem statement

By exploring the literature in the field of development, design and optimization of RTE systems, we have identified the following issues.

Despite significant advances in the development of RTE systems, their design remains a challenging task. A key design step is to assign the application software components (e.g. computation and communication resources) onto an execution platform. The complexity of making design decisions is still rising since designers always have numerous options at hand, involving different ways of assigning the functions of the application to the entities (e.g. tasks) of the software architecture. Furthermore, RTE systems are subject to stringent requirements (e.g. timing constraints, predictability, etc.) which makes the design of such systems more difficult. These requirements must be considered at an earlier stage in the development, i.e. the design phase, in order to ensure the predictability of the system behavior.

We devote this section to sum up the key challenges raised during the development and design of RTE systems, being the focus of this thesis.

3.2.1 RTE systems development challenges

Nowadays, reducing development time and cost, mastering complexity, and providing the required quality raise as major challenges for RTE systems manufacturers. Various factors and characteristics of the systems under consideration combine to create challenges in their design and development:

- **Real-time constraints**: functions of such systems are usually subject to time constraints. These constraints must be met because their violation not only deteriorates the behavior of the system but, even more critically, it could endanger human lives.
- **Size**: the number of features implemented by nowadays RTE systems is substantial. For example, in the aerospace domain, a flight control system includes up to several hundreds functions to specify the system behavior [BHP+08, BFO14].

- **Concurrency**: RTE systems are frequently designed according to concurrent multi-tasking architectures. This means that several tasks implementing the system functions share the same computation resources, i.e. the processor. Managing the concurrency between tasks to access the processor is required and must ensure the timing requirements of the application.

- **Interaction through sharing critical resources**: in real-time multi-tasking systems, the application consists of a set of tasks that cooperate to achieve interaction with the environment. This cooperation could be realized by means of shared resources that may be hardware (e.g. memory or peripherals) or software (e.g. variables or files). In order to guarantee consistency of shared resources, tasks must be synchronized to access these resources. Yet, in the case of preemptive scheduling, classical synchronization mechanisms (e.g. semaphores, monitors, etc.) may arise issues affecting the schedulability, which gives rise to the necessity to carefully handle resources access synchronization during the scheduling process.

- **Sensitivity to changes**: slight changes of certain non-functional properties on some functions may require a radical modification of the system design. This implies that RTE systems need to be fairly flexible to accommodate changes, when the system functions evolve, at a lower cost.

This complexity that was and is still growing imposes a big challenge when designing RTE systems.

3.2.2 Difficulty and importance of the design phase

The design phase is central to the development of RTE systems. The functional specification of the application (e.g. structure and behavior of the application functions, several requirements, etc.) and the description of the target platform technical features (e.g. computation, storage and communication resources characteristics; the scheduling policy provided by the RTOS; etc.) are initially separated. The mapping step consists in assigning the application components defined at the functional specification (in terms of functions and their interactions) to the execution and communication entities (i.e. processors, tasks, buses, messages, etc.). The outcome of the mapping step is the operational design (in this dissertation, it is also simply called design). Furthermore, during the mapping step, different configurations and analysis are performed, such as the setup of the timing parameters associated to the operational design entities, the schedulability analysis, etc. The produced operational design represents an abstraction of the system implementation. Indeed, it defines all the entities required for the

implementation, in terms of tasks sharing common resources and assuming execution on top of a specific execution platform. Thus, the quality and maintenance of the final system implementation depend heavily on the resulting design.

The transformation of the functional specification components into a set of tasks running on the top of the target RTOS is a difficult challenge. In fact, designers must decide how to assign functions to tasks while ensuring the fulfilment of non-functional constraints and balancing between multiple conflicting performance criteria.

Several factors hinder designers to make efficiently design decisions and elaborate the most suitable operational design. These factors are detailed below.

Multiple design alternatives: combinatorial problem

The important number of functions involved in the functional specification results in a large number of possible assignment of these functions to the RTOS tasks, ranging from single-task design to concurrent multi-tasked design alternatives. All possible design alternatives, with respect to functions-to-tasks assignment, constitute the *design space*. Given a set of n functions, the number of all possible combinations of functions-to-tasks assignment is defined by the n^{th} *Bell number* [Rot64], denoted as B_n. The latter counts the number of different ways to partition a set that has exactly n elements. The Bell number is exponential with respect to the size of the set. To give a better idea about the design space size, notice that for a set of 15 functions, the number of all possible functions-to-tasks assignment alternatives is equal to $B_{15} = 1\ 382\ 958\ 545$. Accordingly, the design space size grows exponentially with the system size (i.e. #functions). Considering the extensive size of RTE systems, the space of design alternatives to be explored and evaluated is extremely large. This makes an exhaustive search strategy impractical.

Conflicting performance criteria and influence of design decisions on these performance criteria

Design decisions are very important because of their strong influence on the system performance. When establishing operational designs for RTE systems, designers often face problems associated with making good decisions regarding to several performance criteria. In our work, we take into account performance criteria related to timing properties, such as preemptions, laxities of tasks, blocking time of tasks due to shared resources, etc. Let us consider the two extreme functions-to-tasks assignment strategies.

The first extreme solution is the single-task design solution in which the whole functional specification is assigned to only one task. Although this solution is quite simple and well suited to high memory-constrained applications, it leads to

an inflexible design that is expensive to change or maintain when the functions of the system evolve. Indeed, in the context of a single task design, a small change may break the system timing requirements.

The other extreme solution is the *1-1 assignment* of functions to tasks, i.e. each function is assigned to exactly one task. Unlike the first extreme solution, this alternative results in a more flexible design since the latter provides more overall system laxity. This allows easier modification of the current design or its extension by additional potential functions. Remind that the laxity of a task is the margin available for scheduling a task (see Section 1.4.1). That's why the laxity is related to the flexibility of the design. Nevertheless, this second design solution may be inefficient [BLDN05] because it leads to a significantly higher number of tasks. This involves a consequential memory consumption (e.g. stack size for each task) in addition to the excessive scheduler overhead due to context switching or preemptions.

This way each design alternative may have a positive impact on some system performance criteria to the detriment of others. This is due to the fact that performance criteria are often conflicting, which means improving one criterion can have a negative impact on other criteria. For example, let consider the impact of the way of assigning functions to tasks on the trade-off between the number of preemptions and the overall laxity of tasks. A design is considered as "good" when it implies a reduced number of preemptions (low timing overhead) and high laxities of tasks (flexible design). However, the number of preemptions is reduced when more functions are assigned to one task, whereas, the tasks laxities are increased with fine-grained assignment (maximum laxity values are obtained with 1-1 assignment alternative).

Limits of a manual design strategy

As already mentioned, there are several alternatives for integrating a functional specification into a target execution platform and each alternative has an impact on both the non-functional requirements and the performance of the final system. Exploring the design space "by hand" in order to find the suitable design, is a laborious task for designers. Ideally, they have to try all design alternatives by investigating their impact on the system requirements and the trade-offs between performance criteria and then select the most suitable design. Unfortunately, this exhaustive-search based approach is unmanageable for a human due to the ever-increasing complexity and size of current systems on one side, and the large number of design alternatives on the other side.

Traditionally, designers perform the mapping step according to a "trial-and-error" strategy [ZG11] that relies on their intuition and experience. However, such a design strategy allows to investigate only a small subset of solutions among all

design alternatives. This may lead to produce applications that are not completely optimized according to the systems requirements. Therefore, exploring the design space manually is a tedious task that could be time-consuming, costly and error-prone.

The problem we deal with in this thesis can be summarized as follows: How to integrate the functional specification of a particular RTE application into a specific execution platform while taking into account the non-functional requirements and trade-offs between multiple conflicting performance criteria? Assigning functions to RTOS tasks is non-trivial: (i) it usually raises combinatorial complexity issues, (ii) its influence on the performance of the resulting application is difficult to anticipate, (iii) it has an impact on the schedulability which implies the verification of the feasibility of the design alternatives.

In the same regard, the addressed problem is typically one of the main steps in the design of AUTOSAR[1] compliant automotive systems [FMB+09]. In accordance with the standard methodology of AUTOSAR, the design of such systems consists in the allocation of the application software components (*SW-Cs*) onto hardware resources called Electronic Control Units (*ECUs*). The AUTOSAR standard defines each *SW-C* as a set of interacting *runnable* entities. As shown in Figure 3.1,

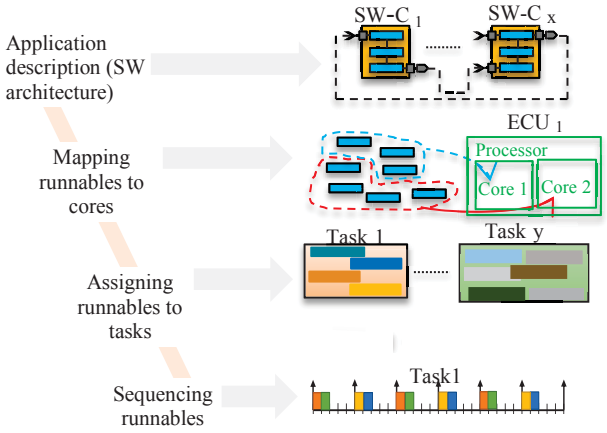


FIGURE 3.1: Runnables to tasks mapping in AUTOSAR methodology (adapted from [SCCM15])

a key design step in the AUTOSAR methodology is to map runnables entities within each ECU (running a RTOS) into the RTOS tasks while guaranteeing the system non-functional requirements (i.e. schedulability of task sets on all ECUs, consistency of data shared between tasks, etc.) and optimizing its performance

[1]AUTOSAR (AUTomotive Open System ARchitecture): an open standard for automotive software development. It provides a standard architecture, application interfaces and a methodology that are adopted by the automotive industry.

criteria. Thus, without loss of generality and according to the AUTOSAR terminology, the so-called runnables entities correspond to what we call the system functions.

3.3 Context of work

In this section, first, we outline assumptions taken for this work in order to delineate the scope of our solutions for the above described problems. Afterwards, we define the system models considered in our work as well as the adopted notations.

3.3.1 Assumptions of the work

During the design phase, a set of specific characteristics of the target execution platform and RTOS must be identified. They represent all assumptions related to:

- the temporal characteristics of tasks (periodic/sporadic/aperiodic, implicit/-constrained deadlines, etc.);
- the relationships between tasks (e.g. precedence relationships, existence of shared resources);
- the hardware architecture of the execution platform (e.g. uniprocessor/multiprocessor/distributed architecture);
- the timing behavior (e.g. synchronous/asynchronous task set);
- the synchronization and communication protocols;
- etc.

The assumptions made in our work are the following:

▷ The architecture of the execution platform is uniprocessor single core;

▷ Produced systems must be Ravenscar compliant;

▷ Tasks are:

- periodic
- synchronous
- either independent or dependent through shared resources
- with implicit deadlines
- with fixed priorities

- ▷ The capacity (or worst-case execution time) of each task is less or equal than its period value;
- ▷ The scheduling policy is preemptive fixed priority scheduling with priority assignment such as RM or DM.
- ▷ The priority ceiling protocol (PCP) is used as a resource access synchronization protocol.

The definition of assumptions is essential to choose and conduct the most suitable schedulability analysis test that matches the characteristics of the designed system. Accordingly, we refer to the current context in order to choose the schedulability analysis test used for the verification of design alternatives (in Section 4.5.2).

3.3.2 System models and notations

The formalization of the functional specification and that of the operational design, considered in this thesis, are described below.

Functional specification formalization

At functional level, the functional specification depicts the structure and the behavior of the system functions. Formally speaking, the functional specification model is defined by $FS = \{\Gamma, \mathcal{R}\}$, where Γ is a set of *functions* and $\mathcal{R}$ represents a set of software resources that can be shared between several functions (see Figure 3.2 (a)).

$\Gamma = \{F_1, F_2, ..., F_n\}$ designates n functions. These functions are elementary non-decomposable sequential programs specifying the processing to be achieved by the system.

In our work, we assume periodic and synchronous 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. Yet, 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 is denoted ω_k. It is parametrized by the function using the resource, 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:

$$\omega_k \begin{cases} (\omega_k)_{Function} \\ (\omega_k)_{begin} \\ (\omega_k)_{end} \end{cases}$$

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

Operational design formalization

As we consider a uniprocessor execution platform, the operational design is then composed of a set of k tasks (i.e. the RTOS tasks), denoted by $S = \{\tau_1, \tau_2, ..., \tau_k\}$, to which the system functions are assigned (as shown in Figure 3.2 (b)).

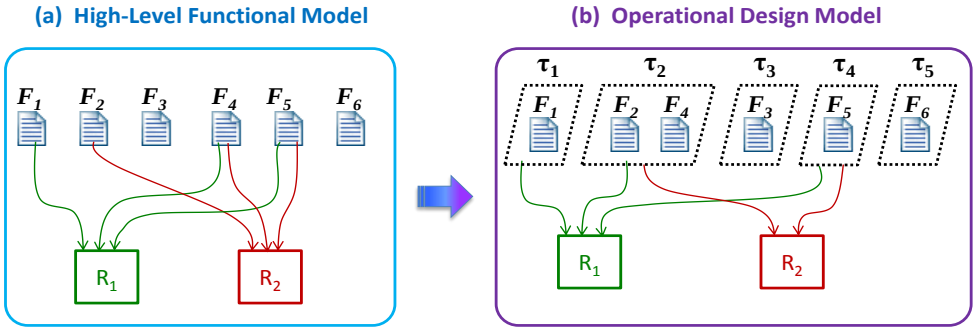


FIGURE 3.2: Real-time design model

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. A task implementing a given function accesses to the resources used by such a function. Since one resource may be accessed by several tasks, mutual exclusion has to be enforced to warrant data consistency. For such a purpose, a shared resource is accessed by a task in a critical section during the task execution. A resource is characterized by a set of critical sections. We denote 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 task that access the corresponding critical section.

At design level, shared resources that need protection (via 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 to be 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 using parameters stemmed from the functional specification (details related to the computation of these parameters are presented in Section 4.4).

3.4 Contributions outline

The main purpose of this thesis is to assist RTE systems designers during the mapping stage. Particularly, we focus on finding the most appropriate operational design through the exploration of the design alternatives search space.

The problems raised in Section 3.2 motivate the automation of the design space exploration (DSE) process. Thus, we propose an automatic multi-criteria DSE process. This process browse the space of design alternatives, evaluate them regarding a set of predefined conflicting criteria to end with design alternatives answering at best to the trade-offs among these criteria. Furthermore, in order to remedy the combinatorial complexity of the addressed DSE problem, we pay particular attention to the scalability of the proposed method.

The solutions proposed in this thesis include the following contributions.

3.4.1 Automatic multi-criteria DSE process

The design exploration problem addressed in this thesis is identified as a combinatorial MOOP (see definition in Section 2.3.1). Considering the combinatorial nature of this problem, enumerating all alternatives of the design space leads to unacceptable processing time. Rather than using an exhaustive search method, one way to tackle this problem is to exploit a *heuristic-based* problem solver. This kind of methods, including a part of randomness, aim at finding "good" solutions without exploring the whole search space, thereby producing results at a reasonable amount of time. Although a heuristic-based method does not guarantee the optimality, it is of great interest when exact methods fail to handle large search spaces and can not find optimal solutions.

MOEA formulation of the DSE process

We propose an automated DSE process using a metaheuristic in the class of MOEA techniques, namely the *Pareto Archived Evolution Strategy* (PAES) [KC00a]. This DSE process allows to search the design space in terms of functions-to-tasks assignment alternatives with respect to multiple conflicting performance criteria. It identifies optimal and near-optimal design alternatives that meet the application non-functional constraints (e.g. schedulability) and answer at best to the trade-offs among the considered performance criteria.

Formalization of design alternatives

A candidate design represents a possible configuration of assignment of functions to tasks. Each candidate design explored during the DSE process must be analysed to check its feasibility (e.g. schedulability analysis), then evaluated to determine its fitness towards performance criteria. These operations require the knowledge of timing parameters of entities (i.e. tasks and shared resources) composing a candidate design. Thus, design alternatives are formalized according to a set of rules that we propose. These rules allow to determine parameters of the different entities associated to a candidate design, based on the way in which functions are assigned to tasks as well as the timing parameters derived from the functional specification. Furthermore, the rules we propose ensure consistency between the functional level and the design level.

3.4.2 Mastering scalability and effectiveness of the DSE process

A MOEA based DSE method has two main challenges: the effectiveness and the scalability. The effectiveness is paramount for any heuristic-based approach and it could be assessed through performance metrics (see Section 2.3.4). The scalability is related to the computing resource requirements and the ability of the method to handle complex problem instances with increased effectiveness.

Several factors combine to accentuate the complexity of our DSE problem and thereby hindering a MOEA-based method to overcome the above challenges. Hence, one of our contributions is about dealing with the scalability and the effectiveness of our DSE process. First, to cope with the scalability issue, we propose to adapt a parallel asynchronous schema namely the well-known "Master-Slave" parallel paradigm to our DSE process. With this coarse-grained approach, multiple candidate design solutions are processed in parallel for checking constraints and evaluating objective functions. Second, we define the *global selection*, a new selection strategy that aims at improving the search procedure of our DSE process, thereby achieving better overall effectiveness, as compared to the default local selection embedded in PAES.

Experiments conducted to assess our proposals show that the asynchronous parallel implementation of the DSE process allow to handle larger scale and more complex problem instances. Moreover, experiments also exhibit that the parallel approach combined with the global selection strategy make the DSE process more effective than its sequential counterpart.

3.4.3 Prototype implementation

As part of our work, we designed and implemented our solution for the functions-to-tasks assignment DSE problem. This tool covers the following aspects.

First, the prototype allows to execute our DSE process on a given functional specification. This could be achieved by means of either the sequential or the parallel versions of the DSE process. Furthermore, users could choose one of the provided selection strategy for the search procedure.

Second, our tool provides users with the opportunity of running small size problem instances (i.e. presenting small enumerable search spaces) with an exhaustive search based DSE method. The latter is defined mostly for evaluation purposes.

Moreover, in order to perform different experiments with a broad range of configurations, we propose to generate synthetically functional specification input models. Therefore, our tool provides a customizable problem instance generator (in terms of function sets interacting or not via shared resources) according to the experiment requirements.

Our prototype is integrated in the Cheddar scheduling framework [SLNM04]. The software architecture of the Cheddar framework was well defined and structured which facilitate the exploitation of its software artefacts and its extension with new features. Thanks to its architecture description language Cheddar-ADL, we can easily define and manipulate design alternatives explored during the DSE process. Additionally, we take advantage of the schedulability analysis utilities provided by Cheddar for the verification and evaluation of the design alternatives.

Finally, we have paid a particular attention on the software design of our tool by following the design requirements of the Cheddar framework, so as to promote the reuse and the extension of the produced software.

3.4.4 Empirical studies

As part of the experiments, two empirical studies are performed. The first empirical study is devoted to explore the correlation between different performance criteria. The second one investigates the impact of the initial design alternative from which the DSE process starts the search.

Investigation of the relationship between objectives

Several performance criteria stemmed from the context of the systems under consideration, are influenced by the manner of assignment of functions to tasks. While formulating the DSE problem, we aim to define each performance criteria as an objective to be optimized during the exploration rather than a constraint. Yet, involving many objectives simultaneously leads to a large-dimensional problem (also known as *many-objective* problem). This hinders a MOEA based method from approximating efficiently the Pareto front [LJCCC08]. Indeed, MOEA methods are designed to solve MOOPs with low-dimension (typically with two or three objectives). This is due to the fact that the number of non-dominated solutions grows rapidly and exponentially with the number of objectives [DS06]. This complicates the search procedure and the decision making, 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 to analyze a large number of solutions in order to choose the most suitable one.

Interactions arising between objectives (performance criteria) are either a conflict or a support relation. Two objectives that support each other are called *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 [GH99]. When solving a MOO problem, it would be pointless to consider redundant objectives as this will raise the problem dimension (i.e. increasing the complexity) without any impact on the search results.

In our problem, it is not intuitively obvious to predict the relationship between the potential set of objectives. Thus, we propose to empirically study of the relationship between three pairs of objectives among those considered in our work.

Study of the impact of the initial design choice on the performance of the DSE process

This empirical study aims at analyzing how the choice of the initial solution to start the search procedure influences the DSE process performance. Indeed, the manner in which the initial solution is formed may bias the search in favor of a particular objective.

We propose two initial solution alternatives. The first alternative is called *1-1 assignment solution* as it assigns each function to one task. This solution is quite simple to make and it is biased towards the laxity objective.

The second initial solution alternative is built by a method that assigns functions to tasks while taking into account dependencies between functions. This method

allows to produce a more compacted (i.e. with less tasks) design solution with regards to the 1-1 assignment solution. The second alternative (called *preprocessed initial solution*) favors the blocking-time objective. Unlike the first alternative, the construction of the second initial solution alternative requires an extra computational effort. Experiments are performed in order to compare Pareto fronts obtained by the DSE process with each of these two initial solutions.

Experiments results reveal that the performance of the DSE process (in terms of accuracy/convergence) is improved with the preprocessed initial solution as compared to the 1-1 assignment initial solution. However, they also show that the efficiency of the method proposed to build the preprocessed initial solution decreases when resources contention and system size increase.

3.5 Related work

In this section, we discuss related work around the mapping step and the architectural exploration during the design of RTE systems, which represent the key concerns of this thesis.

In the following, Section 3.5.1 presents approaches addressing the mapping from the functional level to the design level by assigning the system functions to the RTOS tasks. Next, Section 3.5.2 reviews work that deal with multi-criteria design space exploration.

3.5.1 Functions to tasks mapping approaches

In the literature, many approaches have been proposed to drive the mapping of a functional specification into a specific platform, as part of RTE systems design.

The approach proposed in [SKW00] dealt with the automatic synthesis of a valid multi-tasking design from the application model describing the system functional aspects along with the end-to-end timing requirements. This approach has targeted a uniprocessor architecture for the execution platform. The functional model includes a set of actions interacting through synchronous and asynchronous signals. Since functional models were expressed through an object-oriented modeling formalism, actions are defined via objects where each object encapsulates a set of actions. The proposed approach allows to automatically explore all valid possible assignments of actions to tasks using a single-objective and exact optimization method, namely a *branch and bound* based search algorithm. Two objectives are considered: minimization of task number and inter-task communication. This approach also checks the schedulability of design alternatives. In order to simplify the implementation of the proposed method and enhance its applicability, authors have suggested a number of restrictions on the assignment,

such as assigning all actions of the same object to the same task. In this work, tasks are assumed to be executed in a non-preemptive manner. However, this implies a potentially significant blocking-time.

In [BLDN05], authors developed heuristic algorithms that generate the architectural model from a dataflow functional model with timing properties. In this work, the functional model is composed of a set of functions interacting by the exchange of asynchronous events as well as shared resources. The main objective of this work is to automate the mapping while finding a trade-off between the two extreme mapping solutions, namely: the 1-1 assignment solution and the single-task solution. A set of rules was proposed to guide the heuristic to assign functions onto tasks, e.g. an entire data-flow is assigned to exactly one task, etc. Authors of this work assumed a uniprocessor execution platform and a dynamic priority assignment strategy according to EDF algorithm. Nevertheless, the EDF algorithm is difficult to implement and it is not compatible with requirements of hard real-time systems like Ravenscar compliant systems. In addition, the scheduling analysis is achieved independently of the mapping step, i.e. after the generation of the task set. Yet, this limits the proposed approach from exploring other assignment alternatives when the produced task set is not schedulable. Besides, this approach investigates a unique design alternative, which means that design space exploration is not taken into consideration.

Authors of [WS06] have presented a *task-construction* approach. This approach automates the construction of a schedulable task set from a model implementing the functions of the system with end-to-end timing constraints. The model considered in this work consists of a set of communicating components, where each component executes a set of functions in a non-preemptive manner. Components are iteratively merged into tasks according to their period. Once components are merged, a branch and bound method is adopted in order to determine the component sequence within each task. The proposed approach intends to satisfy the precedence relationships between components and their timing constraints while minimizing the number of tasks (which, in turn, minimizes runtime overheads). This approach manipulates coarse-grained components rather than functions, which restricts functions within one component to be assigned onto the same task. Furthermore, again, a unique assignment alternative is investigated.

In [LW08], authors have considered the same model adopted in [WS06]. Their goal is also to automatically generate a task set from a component model so as to meet the precedence relationships between components and the timing constraints. An algorithm was proposed to create the task set. It is based on a set of rules, such as each component is assigned to a exactly one task and if two components have a precedence relationship, then they are assigned to the same task. This work is very close to [BLDN05] and also considers EDF as priority assignment strategy. As in [BLDN05], the scheduling analysis is performed only after the generation of the task set.

Authors of [PFB+11] provided a framework for the integration and the development of RTE 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. Nevertheless, in this work, functions are assigned to tasks according to a 1-1 assignment strategy. This assignment strategy induces a large number of tasks that in turn (a) can cause excessive scheduler overhead due to context switching (b) and may exceed the maximum number of tasks supported by the target RTOS. Again, this work does not investigate design space exploration.

Furthermore, a MARTE-based methodology was proposed in [MTPG11], 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. The exploration of the design space towards the optimization of performance criteria is out of scope of this work also.

In the context of distributed AUTOSAR systems, authors of [MNBSL12] have proposed two heuristics for assigning runnables to a multi-core ECU architecture. The first heuristic deals with the mapping of runnables to cores taking into account inter-runnable dependencies and locality constraints while optimizing the core load. For systems where most of the runnables are dependent, the strategy used to assign runnables to cores may result in assigning most of them to one core. The second heuristic allows to build the sequencing of the runnable entities of each core using one task per core. Again, a unique feasible design is evaluated. Accordingly, both the exploration of several feasible alternatives and the optimization of multiple performance criteria are not taken into account.

Mapping AUTOSAR runnables on tasks was also addressed in [LLP+09]. This work aims at finding the mapping solution that reduces both context switching overhead induced by a high number of tasks and communication cost. To do so, the authors have provided a set of rules for guiding the mapping of runnables to tasks. When applied on a given application, these rules result in many mapping alternatives. In order to select the most suitable mapping solution, the authors have proposed an equation for evaluating the performance of a given mapping solution. However, this equation does not consider timing properties. Moreover, the context switching overhead is simply expressed by the number of tasks. In addition, the verification of timing constraints of mapping solutions is not considered in this work. Finally, this approach does not provide any process for exploring the space of possible mapping solutions.

Discussion

Table 3.1 highlights the different approaches described in this section according to the following criteria:

- **Assumptions**: in order to identify the scope of application of the mapping approach, we give here assumptions regarding the dependencies between tasks, the scheduling policy and the execution platform architecture.
- **Scheduling analysis**: it is important for a mapping approach to generate a schedulable solution. In this case, the mapping step is accompanied by a scheduling analysis procedure (this case is marked by "✓"). Otherwise, the scheduling analysis is either performed independently of the mapping (this case is marked by "∼"), or not considered in the approach (this case is marked by "✗").
- **Optimization**: this criterion points out whether the mapping approach entails optimization objective(s), and if so, what are these objectives and which optimization technique was adopted. The cross ("✗") means that no optimization was considered by the approach.
- **Design space exploration**: this criterion identifies whether the mapping approach involves a design space exploration process by which several mapping solutions are evaluated. The cross ("✗") means that no DSE process was provided by the approach and only one mapping alternative is investigated.
- **Mapping rules**: used to drive the mapping of functions to tasks.

Based on the previous study on existing mapping approaches, we note that although these approaches present some positive points, however, some of them have certain limitations. The major open limitations are as follows:

- For some approaches (e.g. [LW08, BLDN05]), the scheduling analysis is performed independently of the mapping;
- The exploration of several feasible alternatives is not addressed in the majority of these approaches (except [SKW00]);
- The consideration of multiple performance criteria to be optimized during the mapping is also not addressed;
- Some approaches impose restrictive functions-to-tasks assignment rules, like: one function is assigned to one task (e.g [PFB+11]), or one dataflow is assigned to one task (e.g [BLDN05, MTPG11]), or functions with same period are assigned to the same task (e.g [WS06]), etc;
- The scheduling policy assumed in some work (e.g. [SKW00, LW08, BLDN05]) is not compliant with requirements of critical RTE systems, such as the dynamic priority assignment, or the non-preemptive execution mode.

– Some approaches (e.g. [SKW00, WS06]) manipulate coarse-grained components rather than functions, which restricts the number of possible mappings.

Unlike the studied approaches, in this thesis, we propose to automatically explore various design alternatives in terms of functions to tasks assignment solutions. We perform scheduling analysis on each design alternative and those that meet the timing constraints are evaluated with regard to a set of conflicting performance criteria (e.g. #preemptions, task laxities, blocking-time of tasks, etc.), to finally select the best trade-offs. Furthermore, we explore design alternatives through a clustering technique that allows the assignment of more than one function to the same task according to a set of rules, which helps to limit the number of tasks. Thanks to the multi-objective aspect provided by our approach, the number of tasks in produced designs (i.e. the design Pareto set) is balanced through the two conflicting objectives, namely (i) the minimization of preemptions that is enhanced when more functions are assigned to one task and (ii) the maximization of task laxities that could be improved with fine-grained assignment. Moreover, in our approach, the assignment of functions to tasks is based on harmonic periods (rather than similar periods).

TABLE 3.1: Functions to tasks mapping approaches

	Assumptions			Scheduling analysis	Optimization		Design space exploration	Mapping rules
	task dependencies	scheduling policy	platform		technique	objective(s)		
Monot et al. [MNBSL12]	independent tasks	-preemptive -fixed priority	multiprocessor	✓	heuristic algorithms	Min (cores load)	×	-locality constraint -cores load
Pagetti et al. [PFB+11]	precedence constraints	-preemptive -fixed or dynamic priority	uniprocessor	✓	×	×	×	a function = a task
Mraidha et al. [MTPG11]	shared resources	-preemptive -fixed or dynamic priority	multiprocessor	✓	×	×	×	an entire dataflow = a task
Long et al. [LLP+09]	shared data	not mentioned	uniprocessor	×	×	Min (#tasks)	×	set of rules
Li et al. [LW08]	precedence constraints	-preemptive -fixed or dynamic priority	uniprocessor	∼	×	×	×	set of rules
Wang et al. [WS06]	precedence constraints	no assumptions	multiprocessor	✓	branch and bound	Min (#tasks)	×	simular period
Bartolini et al. [BLDN05]	-precedence constraints -shared resources	-preemptive -dynamic priority	uniprocessor	∼	heuristic algorithms	Min (#tasks)	×	set of rules
Saksena et al. [SKW00]	shared resources	-non-preemptive -fixed priority	uniprocessor	✓	branch and bound	-Min(#tasks) -Min (communication costs)	✓	set of rules

3.5.2 Multi-criteria design space exploration approaches

In the literature, several research work were interested in exploring a design space composed of architecture alternatives regarding multiple criteria by means of optimization techniques. These work provide different DSE approaches that allow to formalize different design alternatives, search the space of design alternatives, analyse these alternatives regarding non-functional constraints, evaluate them with respect to multiple performance criteria, and extract optimal (or near optimal) architecture alternatives.

One category of these DSE approaches focus on the mapping problem, obviously taking into account the scheduling analysis. In [AT15], authors proposed a holistic framework for Distributed Integrated Modular Avionics (DIMA)[2] architecture design and optimization. Different ways are used to explore the design space ranging from function assignment, signal assignment and network definition to complete architecture generation. The problem is decomposed into eight optimization routines that depend on each other. All optimization routines are combinatorial MOOPs. They are formulated as binary linear programs and solved using a *mixed integer linear programming* (MILP) tool. The authors identified a set of constraints related to DIMA architectures requirements, to be verified during the mapping. The considered optimization objectives are mass, ship-set-cost, development cost, and operational interruption cost. The proposed framework allows users to customize the verification and evaluation by adding constraints and/or optimization objectives. The proposed approach has been extended in order to enable the generation of Pareto optimal solutions rather than a single optimum. To this end, authors have adopted a multi-objective integer programming (MOIP) method, namely the Pareto front sampling technique [OBM14] (that is an exact MOO method). However, such a method suffers from a time-consuming process, and in the case of large Pareto front, it becomes impractical since it could suffer from the combinatorial explosion.

In the context of AUTOSAR systems, several multi-criteria DSE approaches [MWTP+13, WMM+13, SCCM15, MPAI16] have been proposed to deal with the mapping problem (i.e. from the functional specification to the operational design). Authors of [MWTP+13, WMM+13] presented 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 mapping problem is abstracted and resolved using two different optimization strategies and techniques, namely (a) an exact strategy using a MILP method and (b) a metaheuristic approach through a genetic algorithm (GA) to cope with the scalability issue of the MILP-based exact method.

[2]Integrated Modular Avionics (IMA) are a standardization of avionics components.

In [SCCM15], the authors proposed a linear formulation of the mapping problem through a MILP optimization technique. Minimizing inter-core communication and balancing the core load are objectives considered for optimization. Authors of [MPAI16] developed a simulated annealing approach for the mapping of runnables with different criticality levels. This approach allows to determine a mapping solution that minimizes the overall communication bandwidth and the variance of the core utilization while the schedulability and the safety constraints are met.

With the exception of [AT15], the previous described approaches [MWTP+13, WMM+13, SCCM15, MPAI16] define multi-criteria design exploration methods. To tackle the multi-objective optimization aspect of the mapping problem, these approaches used single objective optimization techniques together with a unique objective function defined as the sum of two or more weighted objectives. According to a specific objective weights configuration, such design exploration method will result in a single design solution instead of a set of design alternatives that exhibits different trade-offs between criteria. In order to perform an effective design exploration, some of these work 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 [CLVV07]). Contrary to these approaches, we propose a multi-criteria design exploration method based on a dedicated MOO metaheuristic technique (namely MOEA method) that produces a set of Pareto solutions (representing the best trade-off among the considered objectives) instead of a unique one. The optimization approach adopted in [AT15] theoretically provides the optimal Pareto set, but suffers scalability limitations.

The second category of DSE approaches intend to provide generic degrees of freedom in exploration and optimization. Generic degrees of freedom means that the exploration of the design space could be achieved according to different ways and not limited to the mapping of the functional specification into the execution platform. These approaches are based on MOO techniques, particularly MOEAs. Authors of [KKR11] developed a framework called Peropteryx, using the Palladio Component Model (PCM) [BKR09] to describe design alternatives. This framework assists designers to approximate the Pareto set of design solutions representing the best trade-offs regarding some conflicting criteria defined on the basis of four main quality dimensions: cost, maintainability, reliability and performance. The authors proposed to use a MOEA method (namely NSGA-II technique [DPAM02]) combined with the so-called performance tactics. These performance tactics are domain-specific rules that encode the expertise of designers with well established design patterns. They are used to guide the exploration procedure in order to enhance the optimization.

AQOSA [LEEC11] is another DSE framework that provides an automated DSE process based on a set of MOEAs namely NSGA-II, SPEA2 [Zit01], and SMS-

EMOA [BNE07]. Design alternatives are expressed in AADL [FG12]. AQOSA supports multiple degrees of freedom, such as the assignment of software components onto processors, architecture topology, hardware components replacement) for automatically generating design alternatives. This framework also integrates a set of performance analysis tools that allow to evaluate different objective functions of a given design alternative. Objective functions to be optimized can include communication overhead, resource (i.e. CPU and bus) utilization, response time, etc. The optimization module is subject to certain constraints like timing constraints and deployment constraints.

In [RBP15b, RBP15a], the authors proposed a method that explores architecture alternatives for real-time embedded systems such that produced architectures meet at best a set of conflicting non-functional properties (NFPs). In this work, design alternatives are described in AADL and formalized using model transformations. The latter are used to formalize in reusable artefacts the implementation of design decisions (e.g. assignment of software components on hardware components), design patterns (e.g. safety or security design patterns), or model refinements (e.g. transformation steps towards the generation of the system implementation). The DSE process is based on a MOEA method, namely NSGA-II, which allows to explore the space of model transformation alternatives and identify the best ones with respect to NFPs. The authors illustrated their approach through a case study from the railway domain by addressing the mapping problem. In this application example, they considered two optimization objectives (the minimization of response time and the maximization of reliability) while ensuring the respect of timing constraints.

All the above described generic DSE frameworks [KKR11, LEEC11, RBP15b], operate on different models such as PCM, AQOSA-IR, AADL, etc. These frameworks rely on standard variation operators (i.e. mutation and crossover operators). However, such standard operators may adversely affect the search procedure performance, and subsequently can hinder the DSE method to converge towards the Pareto optimal front. In contrary, we define a customized mutation operator in accordance with characteristics of the addressed functions-to-tasks assignment DSE problem. In addition, all these frameworks adopted population-based MOEAs that are time and memory consuming especially for MOOPs involving computationally expensive objective functions evaluation or constraints verification like the problem addressed in this thesis. For that, we have chosen PAES as MOEA since it manages a single solution at each iteration.

Synthesis

In this section, we described and studied different multi-criteria DSE approaches. We noticed that these approaches vary in scope and concerns.

The first category of approaches deal with the mapping of the functional specification into the execution platform. Some of these approaches [SCCM15, MWTP+13] rely on single-objective exact optimization methods such as MILP or ILP solvers. However, these solvers are known to be non-scalable as they can solve at reasonable computational costs only small DSE problems. Furthermore, linear programming methods (e.g. MILP and ILP) are not usable to solve problems with non-linear objective functions [CWC+17] like the objective functions considered in this thesis. Other approaches [MPAI16, WMM+13] have adopted single-objective metaheuristic optimization techniques such GA or simulated annealing.

With the exception of [AT15], almost all the mentioned DSE approaches addressing the mapping of the functional specification into the execution platform used single-objective optimization techniques (either exact or metaheuristic) combined with the weighted sum technique in order to enable the optimization of multiple objectives. The application of the weighted sum method requires to set manually weights on all the objectives and combine them into a single objective function. It is worth mentioning that the outcome of such a method is a single solution that strongly depends on the selected weights. Thus, the identification of an optimal solution requires to put different weights for each objective and perform several experiments.

In [AT15], the authors used a multi-objective integer programming technique that provide the exact Pareto front. However, this technique presents limitations regarding scalability and support of non-linear problems.

Contrary to these approaches, we propose a multi-criteria DSE process based on a dedicated MOO metaheuristic technique, particularly MOEA method. MOEA methods are powerful random-based search techniques and well suited for solving MOOPs with combinatorial complexity, large search spaces, non-linear functions and multiple objectives.

The second category of approaches [RBP15b, KKR11, LEEC11] propose generic multi-criteria DSE frameworks. These frameworks can be used to solve different optimization problems with few customization efforts. These frameworks adopted population-based MOEAs. Nevertheless, using a population of solutions is not suited for MOOPs with computationally expensive evaluation and verification procedures. This is due to the computational cost and memory associated to the execution of each iteration [Deb08].

Unlike these frameworks, our DSE process is established on the basis of PAES (details in Section 4.2.1). PAES is a MOEA solver [CWC+17] which uses a straightforward evolutionary strategy (handling only one solution at each iteration), thus contributing to reduce the overall computational cost.

Table 3.2 summarizes the different multi-criteria DSE approaches described in this section. The approaches are highlighted according to the following criteria:

- The execution platform considered by the DSE approach;
- The scheduling test used to verify design alternatives;
- The modeling language required by the DSE approach to describe the application. The "-" mark means that no modeling language is required.
- The degree of freedom guiding the exploration procedure;
- The optimization technique adopted by the DSE approach, the objectives to be optimized and the Pareto characteristic. The latter is used to indicate if the approach generates a Pareto front either optimal (PF_{true}) or approximate (PF_{approx}); otherwise it only produces a single design solution ("✕").

TABLE 3.2: Multi-criteria design space exploration and optimization approaches

	Execution platform	Scheduling test	Specific modeling language	Degree of freedom	Optimization		
					technique	objectives	Pareto front
Maticu et al. [MPAI16]	multi-core ECUs	processor utilization	-	-runnable assignment -task assignment	simulated annealing + weighted sum	-communication bandwidth -core utilization	×
Annighofer et al. [AT15]	DIMA architecture	not mentioned	-	8 degree of freedom	MOIP	-mass -ship set cost -development cost	PF_{true}
Saidi et al. [SCCM15]	one multi-core ECU	processor utilization	-	-runnable assignment	ILP + weighted sum	-inter-core communication -core load	×
Rahmoun et al. [RBP15b]	any architecture	RTA	AADL	generic degree of freedom	meta-heuristic (MOEA)	-response time -reliability	PF_{approx}
Mehiaoui et al. [MWTP+13]	distributed ECUs	RTA	-	-runnable assignment -task assignment	MILP+weighted sum	-response time -memory	×
Wozniak et al. [WMM+13]	distributed ECUs	RTA	EAST-ADL2	-runnable assignment -task assignment	GA+weighted sum	-response time -memory	×
koziolek et al. [KKR11]	any architecture	×	PCM	generic degree of freedom	meta-heuristic (MOEAs)	-performance -cost -reliability	PF_{approx}
Li et al. [LEEC11]	any architecture	simulation	AADL	generic degree of freedom	meta-heuristic (MOEA)	-communication overhead -resource utilization -response time	PF_{approx}

3.6 Conclusion

In this chapter we first identified the challenges addressed in this thesis. Then, we exposed the assumptions of this work followed by our contributions. Afterwards, we described and discussed related work addressing function-to-task mapping and multi-criteria design space exploration. The next chapters provide functional and technical details related to our contributions.

Part II

Contributions

4

Multi-Criteria Design Space Exploration Process

Contents

4.1 Introduction

This chapter is devoted to present our multi-objective DSE process for RTE systems. Particularly, we are interested in mapping the system functional specification towards an operational design while optimizing multiple objectives. Obviously, all non-functional requirements regarding schedulability, consistency of shared data and also consistency between the functional and the design levels must be met. Functional and technical details related to the proposed DSE process are described in the remainder of this chapter.

Section 4.2 presents the adopted MOEA technique, namely PAES [KC00a], and provides an overview of the proposed DSE process. Next, Section 4.3 highlights the PAES underlying components designed and customized according to the addressed DSE mapping problem. Such a MOEA approach needs a specific formalization of candidate design solutions to enable their fitness evaluation and feasibility analysis. For such a purpose, we define a set of rules enabling the formalization of design alternatives by identifying task set and resource set (in terms of composition and timing parameters) from their chromosomal representation. These rules are provided in Section 4.4. Afterwards, Section 4.5 discusses the impact of our functions-to-tasks assignment method on design solutions feasibility (regarding schedulability and compliance with functions to tasks assignment requirements) and describes how to check the feasibility of design alternatives. Finally, Section 4.6 concludes the chapter.

4.2 Problem formulation using a MOEA approach

In this section, we first introduce the Pareto archived evolution strategy (PAES) MOEA technique used to establish the DSE process. Then we present an overview of the different steps composing the proposed DSE process.

4.2.1 Pareto archived evolution strategy (PAES)

The Pareto Archived Evolution Strategy (PAES) [KC00a] 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. The sequential PAES schema is outlined in the pseudo code of Algorithm 2.

PAES is based on a simple (1+1) evolution strategy instead of a population-based approach as the most part of existing MOEAs (e.g. NSGA-II, SPEA2, etc.). It has been suggested that PAES may outperform population-based methods,

Algorithm 2: 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
```

mainly for MOOPs with computationally expensive objective functions and/or constraints [COA11]. The (1+1) evolution strategy means that PAES maintains a single current solution (*parent*), and at each iteration, it generates a single new candidate (*offspring*) through a random mutation operator (line 5, Algorithm 2). The conceptual approach of PAES is confined to a local search, i.e. it performs only a small change through the mutation operator that moves from a current solution to a nearby neighbour. The candidate solutions are evaluated (line 6, Algorithm 2) according to a set of *objective functions* and compared using the Pareto dominance concept.

One of the key mechanisms in PAES is the maintenance of an archive of non-dominated solutions. This archive is used as reference set with respect to which each new candidate is being compared (lines 11-12, Algorithm 2). In order to foster the diversity aspect and maintain a limited archive size, PAES uses a crowding method based on the division of objective space into an adaptive grid. The latter allow to keep track of the degree of crowding in different regions of the solution space. When a new solution is generated, its grid location in the objective space is determined by a recursive subdivision [KC99]. The number of solutions currently residing in each grid location is also maintained in order to make decisions in the selection and in the archiving procedure when a location is

too crowded [LNA06].

PAES starts with a randomly initialized solution, which is evaluated and added to the archive and becomes the current solution (lines 1-2, Algorithm 2). At each iteration, the current solution is replaced by its mutated offspring if and only if the latter dominates or is in a less crowded region than its parent. Otherwise, the next iteration is realized keeping the same current solution as a basis for mutation.

A candidate solution m is put into the archive if (i) the archive is not full, (ii) m dominates another solution in the archive, or (iii) the archive is full but m is in a less crowded region than at least one other solution in the archive (in this case this other solution will be replaced by m).

The algorithm iterates (lines 4-18, Algorithm 2) until a termination condition will be satisfied. The termination condition can be defined regarding a certain goal quality (e.g. some convergence criteria of the objective function values) or based onto a finite number of iterations. The time complexity of PAES is in the order of $\mathbf{O}(a \cdot n)$, where a is the archive size and n is the number of iterations.

4.2.2 PAES adaptation for multi-criteria DSE process

As described in Figure 4.1, the entry point of our approach is the functional specification of the RTE system to be designed. This specification defines the functions of the system, their interactions and their real-time characteristics.

From this specification, an initial design alternative is proposed such as the 1-1 assignment solution, i.e. each function is assigned to a single task. A scheduling analysis is achieved on the initial design solution. 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 DSE process using PAES as MOEA.

At each iteration, an alternative design solution is generated from the current solution by changing the assignment of a random function to a *pseudo-random* task through the mutation operator (① in Figure 4.1). Feasibility checks 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 4.5 of this chapter). This candidate design is then evaluated according to the considered objective functions ②. Subsequently, other PAES steps are performed namely the comparison and ranking of solutions, then the update of the archive of non-dominated design solutions ③. At the end of each iteration, a design solution is selected as a current solution for the next iteration ④.

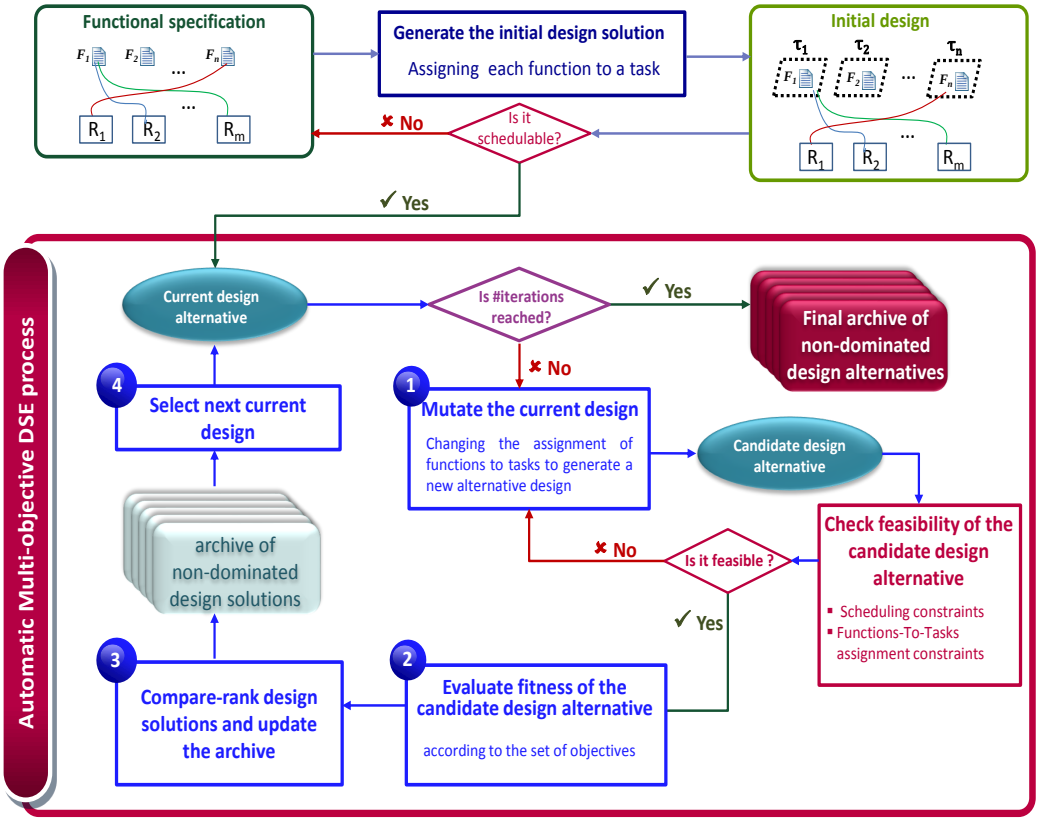


FIGURE 4.1: Proposed DSE process overview

The four steps are iterated until the termination condition of PAES is reached. In our work, we adopt a simple stopping criterion based on the number of iterations (i.e the algorithm stops when reaching a maximum number of iterations). The output of our method is a set of feasible design alternatives that approximates the optimal-Pareto set. From these solutions, software engineers would choose the suitable design.

4.3 Exploration operators

In this section, we present the key PAES components in terms of solutions data structure, initial solution, mutation operator and objective functions. These components are configured according to the functions-to-tasks assignment problem.

4.3.1 Encoding of solutions

The first step in the formulation of the MOEA technique (i.e. PAES) is to decide on an encoding scheme [HCFDC09]. This identifies the form of the chromosomal representation in which the design solutions will be manipulated.

In the literature, various kind of encoding scheme were defined such as *binary encoding*, *real number encoding*, *integer encoding*, etc. In the formulation of our problem, we adopt an *integer encoding* scheme, since authors of [GC97] showed that such an encoding is well suited for combinatorial problems. According to the integer encoding, a chromosome is defined as an integer vector with n positions. In our problem, 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*, is associated to a particular function, i.e. the i^{th} gene 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 $\{\tau_1, \tau_2, ..., \tau_k\}$.

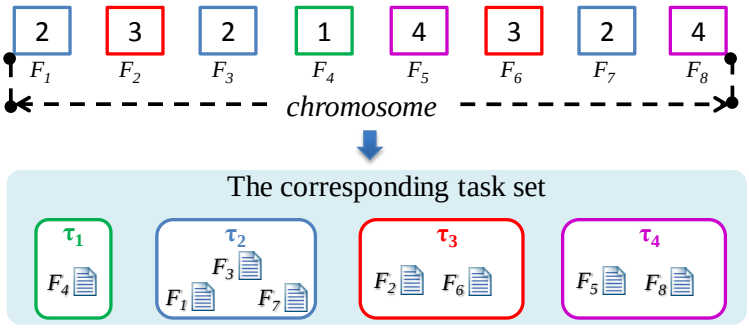


FIGURE 4.2: Chromosome representation of a particular functions to tasks assignment solution S

Figure 4.2 shows an example of a chromosome that corresponds to a particular functions to tasks assignment 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 formulated 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.

The used integer encoding is quite simple, however, it is redundant [HCFDC09], which means that the same solution can be represented by different chromosomes. For example the solution S encoded by the chromosome depicted in Figure 4.2 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 chro-

mosomes that model the same assignment solution independently of the indexes of tasks.

Normalization of the chromosomal representation

In order to reduce all of the equivalent assignment solutions to the same chromosome representation, we propose to *normalize* the chromosome representation. This is performed as follows: the index of the task to which the function F_1 is assigned, is always 1 (this means that the value held by the 1^{st} gene that is associated to the function F_1 in the chromosome, is always 1). Consequently, values of genes associated to functions that belong to the same task as F_1, are set to 1. The value of the 2^{nd} gene associated to F_2 is set to 2, except if F_2 is assigned to the same task with F_1, then all genes are set in the same way. Figure 4.3 depicts the normalized chromosome representation of the same assignment solution S given in Figure 4.2.

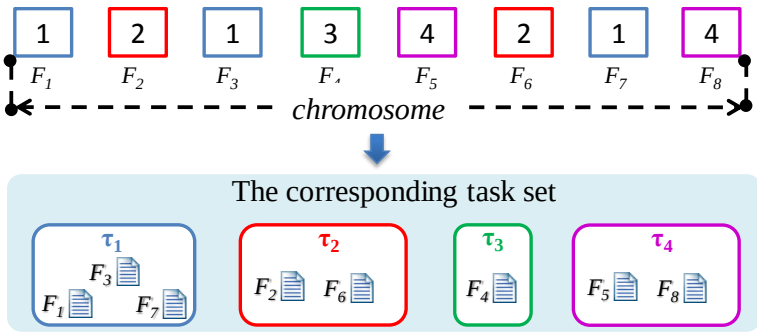


FIGURE 4.3: The normalized chromosome representation of the assignment solution S

The normalization procedure ensures a unique chromosomal representation of a given functions to tasks assignment solution. Using the normalization increases the speed of convergence and improves the quality of solutions [MJH13]. Thus, it is applied on each chromosome associated to a candidate solution manipulated by the algorithm.

4.3.2 Initial design solution

PAES needs an initial solution to start the search procedure. We investigate two ways to generate this initial solution: the first way is quite simple and it promotes the laxity objective whereas the second one is more sophisticated (i.e. requires a preprocessing) and it fosters the blocking time objective.

One function per task assignment initial solution (called 1-1 assignment solution)

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

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

Tasks laxities increase with the granularity of the functions to tasks assignment. This means that the 1-1 assignment solution provides the best overall system laxity thereby favoring the laxity objective.

Preprocessed initial solution oriented towards blocking-time objective

When considering the 1-1 assignment solution as initial solution, the latter is biased only towards laxity objective. In the case of systems with shared resources, this initial solution may be pessimistic regarding the blocking time criterion. A preprocessing of the initial solution is performed at baseline (i.e. before running the design exploration process using PAES) in order to promote the blocking time objective function of the initial solution. To do so, we develop an heuristic (Algorithm 3) that tries to gather dependent functions within the same tasks as much as possible.

This heuristic takes into account some functions to tasks assignment constraints defined in Section 4.4 and Section 4.5 (such as harmonic period assignment constraint, or task capacity/deadline constraint) to prevent as much as possible the generation of an infeasible solution. After delivering a first possible initial solution, feasibility checks are carried out on it by applying Algorithm 6. In the light of the feasibility checks result, we decide whether to accept it or not. In the case of an infeasible solution, we try to derive from it another feasible solution by applying our mutation operator (Algorithm 4).

Algorithm 3: Preprocessing for the generation of the initial solution

input : Functional specification (a set of functions that use a set of resources)
output: Preprocessed initial solution

```
1  begin
       /* Resources are sorted in decreasing order of number of
       functions that use resources.                                  */
2      Determine Res the set of resources in the functional specification, sorted in
         decreasing order of use (i.e. by the number of functions using each resource);
       /* By looking at sorted resources one by one, functions
       that use the same resource are assigned to the same task
       wherever possible.                                             */
3      foreach resource R_i in the resource set Res do
4          Determine Funct the set of functions that use R_i and are not yet assigned
             to tasks;
           /* Following functions-to-tasks assignment constraints
           and particularly harmonic period assignment constraint,
           only harmonic functions are assigned to the same task      */
5          Sort Funct in increasing order of function periods;
6          foreach function F_k in the function set Funct do
7              if (F_k is not assigned to a task) then
8                  Determine Harmonic_set the set of functions in Funct (other
                     than F_k) that are not yet assigned to tasks and harmonic with F_k;
9                  if (Harmonic_set is not empty) then
                       /* The task capacity must be less than its
                       deadline                                       */
10                     Assign F_k with functions in Harmonic_set to the same task
                         while ensuring that the task capacity does not exceed its
                         deadline;
11                 end if
12             end if
13         end foreach
14     end foreach
       /* Some functions may not be assigned to tasks.  These
       functions are either functions that do not use resources or
       those that use resources but not assigned to tasks due to
       constraints imposed on the assignment of functions to
       tasks.  We assign each of them to a task.                      */
15     Assign to a new task, each function not yet assigned;
       /* once a candidate initial solution (S_1) is produced, we
       check its feasibility (Algorithm 6) to decide whether to
       accept or rearrange it by applying the mutation operator
       such that a feasible solution is generated.                    */
16     Apply feasibility checks (Algorithm 6) on the design alternative relative to the
         candidate initial solution S_1;
17     if (S_1 is feasible) then
18         return S_1;
19     else
           /* S_1 is rearranged by mutation in order to produce a
           feasible solution                                          */
20         Apply the mutation operator (Algorithm 4) on S_1;
21     end if
22 end
```

4.3.3 Mutation operator

Algorithm 4: Mutation operator for the functions-to-tasks assignment problem

input : current solution (n functions assigned to k tasks ($n \geq k$))
output: mutated solution

```
1  begin
2      repeat
3          Choose randomly a function F_i (1 ≤ i ≤ n);        /* where F_i is
             assigned to the task τ_j in the current solution */
4          Determine the set of harmonic tasks with the chosen function F_i called
             Harmonic_task_set;
5          if (Harmonic_task_set = {τ_j}) then
               /* i.e. F_i is not harmonic with any task, only the
               task to which it is assigned                          */
6              Restart the algorithm with another function randomly chosen;
7          else
8              Choose randomly a task τ_m over tasks in the set
                 Harmonic_task_set (including τ_j = chrom[F_i]);
9              if (τ_m ≠ τ_j) then
10                 chrom[F_i] ← τ_m; /* F_i is moved to the task τ_m      */
11             else if (the function F_i is not alone in τ_j) then
12                 Create a new task τ_{k+1};
13                 chrom[F_i] ← τ_{k+1}; /* F_i is isolated in the new task
                     τ_{k+1}                                           */
14             else
                   /* i.e. F_i is the only function assigned to τ_j   */
15                 Restart the algorithm with another function chosen randomly;
16             end if
17         end if
           /* once the mutated solution is generated, we apply the
           assignment rules to generate the composition of the
           design alternative in terms of tasks and shared
           resources and their parameters                            */
18         Apply the assignment rules on the new candidate solution to generate
             the corresponding design alternative;
           /* Check the feasibility of the produced design
           alternative by applying Algorithm 6                       */
19         Apply feasibility checks (Algorithm 6) on the design alternative relative
             to the mutated solution;
20     until (A feasible solution is found) or (a maximum number of attempts is
         reached);
21 end
```

Instead of applying standard mutation operators that may not be fully suitable

to our problem, we propose a customized mutation operator (Algorithm 4). This mutation operator is designed based on characteristics of the addressed functions-to-tasks assignment problem. It chooses a random position in the chromosome and changes the value of the associated gene to a new pseudo-random value. The mutation produces a new alternative assignment solution from the current solution by reassigning one random function F_i to a task randomly selected from the set of harmonic tasks with F_i (lines 3-17 of Algorithm 4). Indeed, this restriction is made in order to follow functions-to-tasks assignment rules defined in Section 4.4. Furthermore, the mutation operator is implemented in order to generate only feasible solutions that satisfy constraints verified by Algorithm 6 (Section 4.5).

The mutation operator algorithm iterates until finding a feasible solution or reaching a maximum number of failed attempts. The latter condition is defined in order to guarantee the termination of the algorithm (i.e. avoid infinite loop). When we performed experiments to assess our proposals, we noted that our mutation operator always succeeds in generating a mutated solution. This is explained by the fact that in our DSE process, the current solution selected for mutation is always feasible. Accordingly, our mutation operator will be able to easily find from the current solution a nearby feasible solution. Nevertheless, this is not always the case with the preprocessed initial solution method (Algorithm 3) that calls the mutation operator when it fails to generate a feasible solution. In fact, the capability of the mutation operator to find a feasible solution from an infeasible solution decreases when resources contention and system size increase.

4.3.4 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 DSE process. Here, we list some possible objective functions.

- **Minimization of scheduling timing overheads**: A high number of context switches is one among scheduling factors that may introduce a significant timing overhead to the overall execution time of a task set. This is due to the fact that a context switch between two tasks implies the execution of several processor instructions in order to store the context[1] of the current running task and retrieve the context of the task selected to be executed. The number of context switches is closely related to the number of preemptions (since a preemption causes a context switch). Thus in order to reduce the scheduling

[1]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.

timing overheads, we can consider the minimization of either the preemption number (objective function f_1) or the context switch number (objective function f_2). We denote these objective functions as follows:

$$\textbf{minimize } (f_1 = \#\ preemptions) \tag{4.1}$$

$$\textbf{minimize } (f_2 = \#\ context\ switches) \tag{4.2}$$

It is worth noting that these two objectives are not to be considered simultaneously to guide the design exploration process, because they are obviously non-conflicting.

- **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 switches and requires extra memory allocations for the task execution stack. The objective function is denoted as follows:

$$\textbf{minimize } (f_3 = \#\ tasks) \tag{4.3}$$

- **Maximization of tasks laxities**: The maximization of the tasks laxities 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. Remind that laxity is the maximum time a task can be delayed on its activation to complete within its deadline (see Section 1.4.1).

  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.

$$\textbf{maximize } (\sum_{i=1}^{k} L_i) \qquad (where\ k = \#tasks)$$

  (B) Maximize the minimum task laxity

$$\textbf{maximize } (min_{i\in[1..k]}\{L_i\})$$

  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 of 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 4.4, we have arbitrarily chosen H (the hyperperiod of the 1-1 assignment solution) as a constant from which the original objective function will be subtracted. With such method, the minimization of the new expression (f_4) would result in the maximization of the current objective:

$$\textbf{minimize } (f_4 = H - \sum_{i=1}^{k} L_i) \tag{4.4}$$

- **Minimization of worst-case response time of tasks**: In the context of real-time systems, we have always an interest in minimizing the worst-case response time of tasks. As for the tasks laxity, there are two possible examples of objective function alternatives, either to minimize the overall worst-case response time (objective function f_5) or to minimize the maximum worst-case response time (objective function $f_5{}'$).

$$\textbf{minimize } (f_5 = \sum_{i=1}^{k} WCRT_i) \tag{4.5.1}$$

$$\textbf{minimize } (f_5{}' = max_{i\in[1..k]}\{WCRT_i\}) \tag{4.5.2}$$

- **Minimization of worst-case blocking time of tasks**: The synchronization of tasks to access shared resources causes latencies on the tasks execution defined by the blocking-time attribute (see Section 1.4.3). The intent when designing real-time applications is of course to minimize the worst case blocking time of tasks. Again, we can distinguish two possible examples of objective functions for the worst case blocking time metric. In the experiments, we will consider the objective function f_6.

$$\textbf{minimize } (f_6 = \sum_{i=1}^{k} B_i) \tag{4.6.1}$$

$$\textbf{minimize } (f_6{}' = max_{i\in[1..k]}\{B_i\}) \tag{4.6.2}$$

- **Minimization of the number of shared resources**: The minimization of the number of shared resources allows the minimization of semaphores and subsequently reduces both memory cost and computation time. We denote the associated objective function as follows:

$$\textbf{minimize } (f_7 = \#\ shared\ resources) \tag{4.7}$$

We note that not all the objectives combinations among the list given above, are relevant since some pairs of objectives are obviously non-conflicting such as the pairs (f_1 i.e. number of preemptions, f_2 i.e. number of context switches) or (f_4 i.e. laxity of tasks , f_5 i.e. WCRT of tasks). The objectives of each of these pairs shall not be considered together (with or without other objectives) to guide the design exploration process. Nevertheless, 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 7.3.2 to investigate through experiments the relationships (conflict or support) 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 framework (see Section 6.3).

4.4 Formalization of design alternatives

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

4.4.1 One function assigned to one task

First, we consider the 1-1 assignment solution, where each function F_i is assigned to one task τ_i 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 1-1 assignment design alternative 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.

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 [MSH11]. The Listing 4.1 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 4.1: 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   A_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 + A_period;
12    -- Time-based activation event
13    delay until Next_Time;
14  end loop;
15 end Tau_k;
```

4.4.2 Several functions assigned to the same task

In this section, we investigate assignment alternatives when more than one function can be assigned to the same task. Indeed, the mutation operator allows us to explore different design alternatives through reassignment of functions to tasks. As described in the mutation algorithm 4, 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 and (2) the parameters of resources and critical sections of an alternative assignment solution generated by mutation.

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

(i) The functional specification in terms of periodic activations of the functions.

(ii) The schedulability of the system.

A) Tasks parameters computation

Hereafter, 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.

We consider the task τ_k previously defined in Section 4.4.1. We show how the parameters of τ_k are set when it executes two functions F_i and F_j.

Computation of 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 F_j can be assigned to τ_k if and only if the greater period among ζ_j and T_k is divisible by the other. In this case, the function F_j is called *harmonic* with the task τ_k. Considering this assumption, we can set the period of τ_k as follows:

$$T_k = GCD(\zeta_i, \zeta_j) \tag{4.8}$$

Where 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 4.2. 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 frequencies.

Listing 4.2: Ada implementation of the task τ_k containing two functions

```
1  with Ada.Real_Time; use Ada.Real_Time;
2  ...
3  task body Tau_k is
4   Next_Time        : Time := Clock;
5   Number_Functions : Integer := 2;
6   Period_Functions : Integer_Array (1..Number_Functions) := (ζi, ζj);
7   Index_Functions  : Integer_Array (1..Number_Functions) := (i, j);
8   A_period         : constant Time_Span := Milliseconds(GCD(Period_Functions));
9   Counter          : Integer := 0;
10  Frequency        : Integer;
11 begin
12  loop
13   for i in 1 ..  Number_Functions loop
14    Frequency := Period_Functions(i)/Period;
15    if (counter mod Frequency = 0) then
16       Call_Function_by_index(Index_Functions(i));
17    end if;
18   end loop;
19   -- LCM : Least Common Multiple
20   counter :=
21        (counter + 1) mod (LCM(Period_Functions)/Period);
22   Next_Time := Next_Time + A_period;
23   delay until Next_Time;
24  end loop;
25 end Tau_k;
```

In order to illustrate the behavior at runtime of several functions assigned to the same task (Listing 4.2), 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 4.8, the period of τ_k is equal to $GCD(10, 5) = 5$.

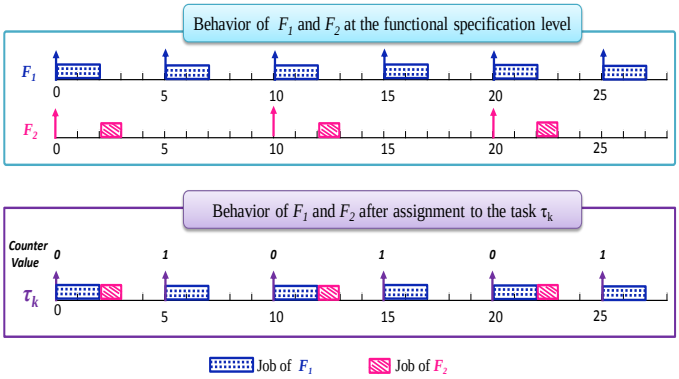


FIGURE 4.4: Behavior of functions F_1 and F_2 before and after assignment to the task τ_k

Figure 4.4 shows, first, the execution sequence of F_1 and F_2 at the functional specification level. 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 4.2 as well as Equation 4.8 ensure periodic executions of functions assigned to the same task.

Computation of task capacity

As shown in Figure 4.4, 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:

$$C_k = \gamma_i + \gamma_j \tag{4.9}$$

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

Computation of 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:

$$D_k = min(\delta_i, \delta_j) \tag{4.10}$$

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.

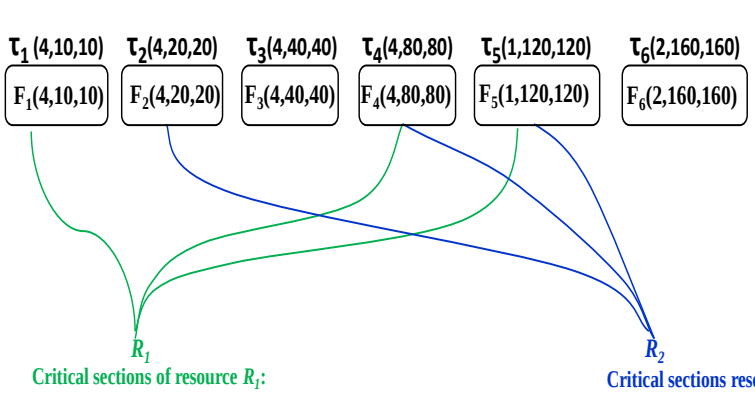
By considering the assignment method described above, the constraint (i) mentioned at the beginning of this section is preserved. As we aim also to ensure the constraint (ii), in Section 4.5.1, we discuss how the assignment may jeopardize schedulability of the design alternative solutions.

B) Resources parameters computation

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 5 shows the different steps and cases we deal with to set parameters of the resource set after mutation. This algorithm takes as input the assignment solution generated by mutation as well as the 1-1 assignment solution that will be used as reference for computing parameters.

To understand the different cases addressed in Algorithm 5, we illustrate them through an example. The 1-1 assignment solution of this example is depicted in Figure 4.5. It consists of 6 tasks and each one holds one function. In this example, there are two shared resources R_1 and R_2.



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	

FIGURE 4.5: 1-1 assignment solution model example: each function is assigned to one task

Supposing that the 1-1 assignment solution is given for mutation. In the new assignment alternative (i.e. mutated solution generated from the 1-1 assignment

Algorithm 5: Computation of resource set and critical sections of a mutated solution

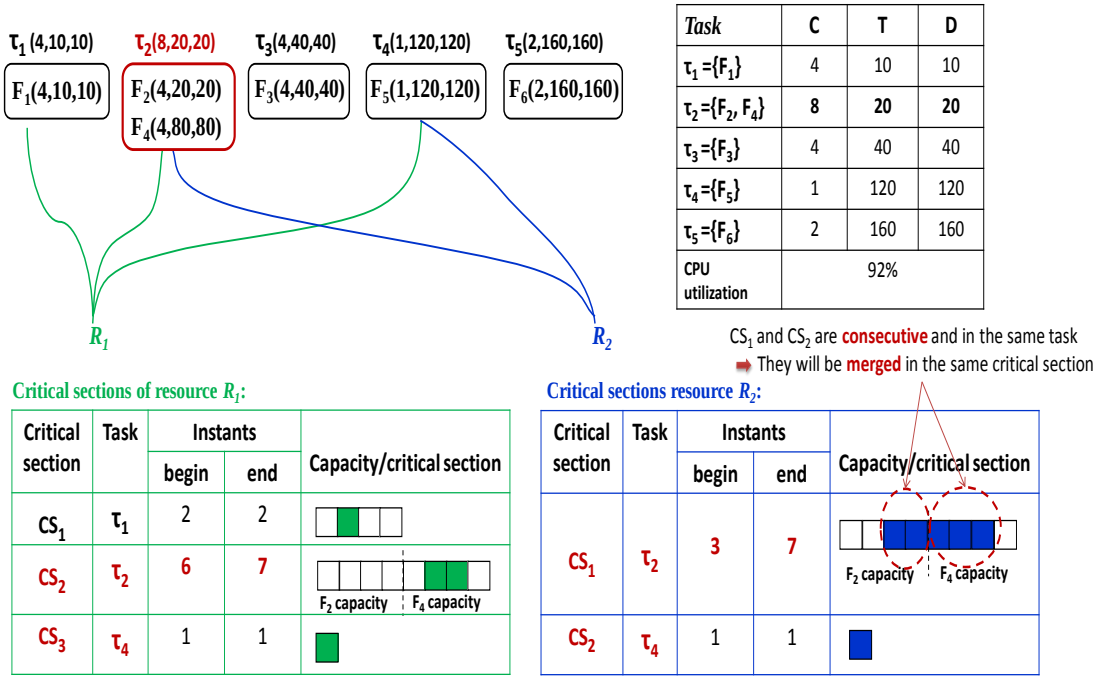
input : mutated solution, 1-1 assignment solution

- mutated solution: the assignment solution generated by mutation
- 1-1 assignment solution: each function is assigned to one task

output: resource set and critical sections relative to the mutated solution

begin
  foreach *resource* R_i *in the resource set of the 1-1 assignment solution* **do**
    foreach *critical section* CS_j *of the resource* R_i *in the 1-1 assignment solution* **do**
      /* Let consider F_h the function that uses the resource R_i on the critical section CS_j in the 1-1 assignment solution */
      Set the task that holds R_i on CS_j with the task to which F_h is assigned according to the mutated solution;
      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 $(F_l)_{l \leq h}$ the set of functions with lower indexes than F_h and which are grouped with it in the same task */
        $(CS_j)_{begin} \leftarrow (CS_j)_{begin} + \sum \gamma_l$;
        $(CS_j)_{end} \leftarrow (CS_j)_{end} + \sum \gamma_l$;
      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 */
      if *in the mutated solution,* R_i *is used by a single task* **then**
        Isolate R_i from the resource set of the associated design alternative;
      end if
      /* Case 2: Check if in the mutated solution there are two or more consecutive critical sections for the same task */
      if *in the mutated solution, there are two or more consecutive critical sections for the same task* **then**
        Merge these critical sections;
      end if
    end foreach
  end foreach
end

solution), the function F_4 is assigned together with the function F_2 into the task τ_2 since F_4 is harmonic with τ_2 (ζ_4 mod T_2 = 80 mod 20 = 0). We apply Algorithm 5 on this mutated solution and we get the resource set and critical sections as shown in Figure 4.6. We can notice that the change of parameters of



Task	C	T	D
τ_1={F_1}	4	10	10
τ_2={F_2, F_4}	8	20	20
τ_3={F_3}	4	40	40
τ_4={F_5}	1	120	120
τ_5={F_6}	2	160	160
CPU utilization	92%		

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_1	2	2	
CS_2	τ_2	6	7	
CS_3	τ_4	1	1	

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_2	3	7	
CS_2	τ_4	1	1	

FIGURE 4.6: A possible assignment solution: the resulting resource set and critical sections of the mutated solution

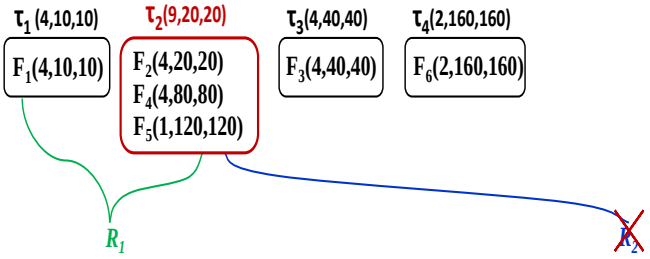
CS_2 of R_1 is stemmed from rules defined in Algorithm 5: the task index changes from τ_4 to τ_2 as F_4 is reassigned to τ_2 (line 6) and the begin/end instants are changed taking into account the capacity of F_2 (lines 8,9). We can also notice that the task accessing R_1 on CS_3 is set to τ_4 (instead of τ_5 in the 1-1 assignment solution before mutation) because in the mutated solution, the function F_5 that uses R_1 on CS_3 is assigned to the task τ_4 (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 1-1 assignment solution, as we can observe in Figure 4.5, the critical sections CS_1 and CS_2 of R_2 are held by τ_2 and τ_4 respectively. In the mutated solution (Figure 4.6) generated from the 1-1 assignment solution, the function F_4 is assigned together with F_2 into the task τ_2. The application of lines 6-9 of Algorithm 5 on this candidate solution gives the following parameters for critical sections 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 5 requires the merge of CS_1 and CS_2 in one critical section as illustrated in Figure 4.6.

Now, we suppose that the solution displayed in Figure 4.6 is given for mutation. In the produced mutated solution shown in Figure 4.7, the function F_5 is assigned together 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 5 on this mutated solution and we obtain the resource set and critical sections as shown in Figure 4.7. We can observe that the resource R_2 is used only by task τ_2. This means that the resource R_2 does not need protection, and thus contention to this resource does not need to be taken into account during schedulability analysis. It is hence abstracted from the design model relative to this mutated solution (lines 11-13).



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%		

Critical sections of resource R_1:

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_1	2	2	
CS_2	τ_2	6	7	F_2 capacity, F_4 capacity
CS_3	τ_2	9	9	F_2 capacity, F_4 capacity, F_5 capacity

Critical sections resource R_2:

Critical section	Task	Instants		Capacity/critical section
		begin	end	
CS_1	τ_2	3	7	F_2 capacity, F_4 capacity
CS_2	τ_2	9	9	F_2 capacity, F_4 capacity, F_5 capacity

FIGURE 4.7: Another possible assignment solution: the resulting resource set and critical sections of the mutated solution

In the following section, we address the verification of the feasibility of design alternatives.

4.5 Design alternatives feasibility verification

Feasibility checks are used to avoid the generation of non-feasible candidate solutions after mutation. We distinguish two kinds of feasibility checks: schedulability 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 explain the choice of the schedulability analysis method adopted in our work. Then, we show how the mutation and the assignment method may lead to unnecessary activations 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.

4.5.1 Impact of the assignment method on the schedulability

In Section 4.4, we proposed rules to manage the assignment of functions to tasks. We noted that the assumptions made in the assignment method may produce unschedulable design alternatives. 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 4.1. This task set is schedulable since we assume RM for the priority assignment and its processor utilization is equal to 65% ($< 69\%$).

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 4.2 and it is not schedulable (its processor utilization is equal to 110%).

TABLE 4.1: 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 4.2: 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 during the search procedure. In the next section, we present the scheduling analysis method adopted in order to verify the schedulability of design alternatives.

4.5.2 Schedulability analysis of design alternatives

As we mentioned, design alternatives explored by mutation have to be verified in terms of schedulability. The scheduling theory provides several scheduling analysis methods. The choice of the scheduling analysis method depends on assumptions on the target execution platform such as the scheduling policy, the hardware architecture, the temporal characteristics of tasks, etc.

Previously in Section 1.5.2, we reviewed and studied most popular schedulability tests that could be applied on systems under consideration, i.e. a set of synchronous and periodic tasks independent or interacting through a set of shared resources running on a uniprocessor architecture under a fixed priority and preemptive scheduling policy (assumptions of work are given in Section 3.3.1). Considering this study, we opt for the scheduling simulation as schedulability test for both independent task systems and systems with tasks sharing resources. This choice is justified by the fact that the scheduling simulation is used in the computation of some objective functions.

However, as previously mentioned in Section 1.5.2, given an online scheduler, the simulation schedulability test is not sustainable with respect to execution time in the context of dependent task systems such as tasks sharing resources. To implement a design alternative, the proposition of Goossens et al. [GGCG16] that consists in the use of an off-line scheduler has to be followed. Accordingly, for implementing design alternatives, the scheduling sequence produced by simulation during the DSE process will be executed infinitely by an off-line scheduler.

Nevertheless, it is possible to use another schedulability test when the computation of the considered objective functions does not require simulation of the schedule. For example, the RTA test can be used in both contexts since it is sustainable and its complexity is pseudo-polynomial.

We note that two obvious and necessary conditions for the schedulability of a design alternative have to be checked before performing the scheduling simulation. In other words, if these two conditions are not met by a given design alternative then it is not schedulable and must be directly discarded without performing the simulation. These conditions are expressed as follows:

1. Capacity/deadline constraint: the capacity of each task in the design alternative must be less than its deadline.

2. Processor utilization constraint: the overall processor utilization of the design alternative must be less than or equal 100 %.

4.5.3 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 4.8, 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 (T_4 mod ζ_1 $=$ 60 mod 20 $=$ 0).

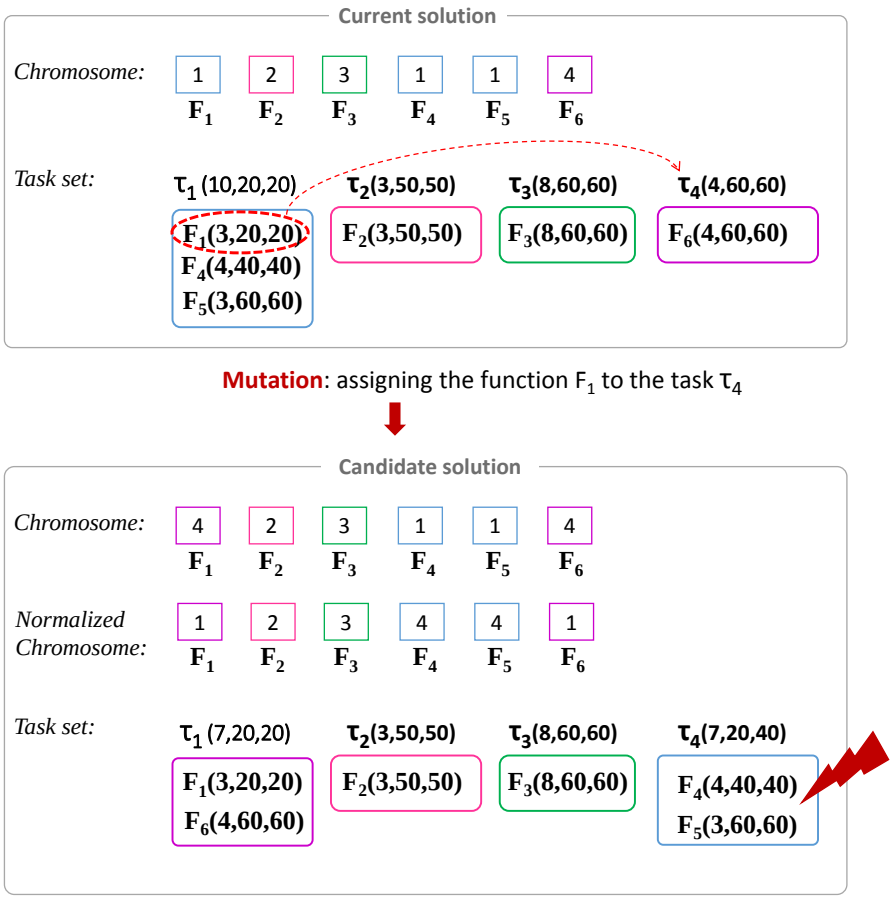


FIGURE 4.8: Example of non-feasible candidate solution according to the functions-to-tasks assignment constraint

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 (see Section 4.4). 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 4.2, 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 has (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.

4.5.4 Feasibility checks algorithm

In Sections 4.5.1 and 4.5.3, we showed situations for which a mutation action may produce infeasible 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 grouped in Algorithm 6.

Algorithm 6: 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

```
1  - false: infeasible solution
2  - true: feasible solution
3  begin
       /* 1) Check for each task that C_i < D_i                          */
4      if there is at least a task τ_i for which C_i ≥ D_i then
5          return false;
           /* 2) Check that the total processor utilization ⩽ 1          */
6      else if U > 1 then
7          return false;
           /* 3) Check that all tasks fulfill the functions to
           tasks assignment constraint                                   */
8      else if there is at least one task τ_i that violates the functions-to-tasks assignment
        constraint then
9          return false;
10     else
           /* 4) Check the schedulability by simulation                  */
11         Perform a scheduling simulation of the design alternative
12         if the design alternative is not schedulable then
13             return false;
14         else
15             return true;
16         end if
17     end if
18 end
```

4.6 Conclusion

In this chapter we have detailed the first part of our contributions addressing the key objective of this thesis which is: given the functional specification of a particular RTE system, explore architecture alternatives and produce those that meet all the requirements and answer at "best" to a set of conflicting performance criteria.

To tackle the combinatorial issue of the addressed problem, our method relies on a multi-objective optimization evolutionary algorithm (MOEA), namely the Pareto Archived Evolution Strategy (PAES). PAES specific components in terms of the encoding of solutions, the initial solution, the mutation operator and the objective functions are formulated regarding our specific MOOP. We defined rules that drive the assignment of functions into tasks. The exploration of possible assignment solutions is automatically performed by the DSE process. Design alternatives explored during the search process are checked towards a set of feasibility requirements such that the final selected designs enforce the correct system behavior and the timing constraints described in the functional specification.

In the next chapter, we focus on studying and improving the performance of the DSE method proposed in Chapter 4.

5

Towards Scalable and Efficient Design Exploration Process

Contents

5.1 Introduction

In the previous chapter, we proposed a multi-criteria DSE process to find optimal or near-optimal design alternatives in terms of functions to tasks assignment solutions towards a set of conflicting performance criteria.

In this chapter, we focus on enhancing and assessing the performance of the DSE process. We present changes introduced to this DSE process in order to improve its performance. The latter is determined in terms of scalability and effectiveness of the method. By scalability we mean the ability of such a method to handle computationally expensive problem instances with affordable execution time. While the effectiveness is related to the capability of the method to provide the most promising Pareto set. The developed changes are mainly the adaptation of a parallel processing for the DSE process and a new selection strategy for PAES.

The remainder of this chapter is structured as follows. Section 5.2 gives the main concepts related to the parallelization of MOEAs. Then, in Section 5.3 we discuss existing research activities addressing the parallelization of PAES. Section 5.4 introduces the parallel processing formulation proposed for our DSE process and the new selection strategy. Afterwards, we present in Section 5.5 the experiments achieved to assess the impact of our proposals on the performance of the DSE process. Finally, Section 5.6 concludes the chapter.

5.2 Basic background to parallel MOEAs (pMOEAs)

This section is devoted to present the basic philosophy of MOEAs parallelization and related underlying background material. In the remainder of this section, first we highlight the main motivations behind the use of parallel processing techniques in the context of MOEAs. Then we present background for designing pMOEAs in terms of mainstream parallel models. Finally, we discuss existing work addressing the parallelization of PAES.

5.2.1 Motivations for parallelizing MOEAs

Various techniques have been developed in the MOO community. MOO problems are often combinatorial, complex and CPU-time consuming. The use of evolutionary algorithms (MOEAs) for solving such kind of problems has been quite active over the past few decades. The rising popularity of MOEAs is justified by the following key reasons. In fact, these algorithms are able to estimate multiple members of the Pareto optimal set in one single run and with a practical and feasible time. They have proved their suitability and flexibility for solving MOO

problems compared to classical mathematical programming techniques [CLVV07] that generally assume small enumerable search spaces.

In spite of these advantages, some computational issues may hinder MOEAs from solving efficiently complex real-world MOO problems and adversely affect the scalability of these algorithms. Among these issues, the most relevant is the fact that real-world MOO problems typically involve computationally expensive methods (in terms of CPU-time) for computing objective functions and constraints (e.g. engineering design problems). This may induce a computational bottleneck and then limit the applicability of MOEAs. Additionally, real-world MOO problems tend to be large scale and to have high-dimensionality (i.e. large number of optimization criteria). These characteristics normally expand further the computational cost in order to efficiently approximating the Pareto optimal front. Finally, the use of archiving, clustering or niching techniques that are commonly adopted with MOEAs [KC00a, DPAM02] also emphasizes the computational overhead.

For these reasons, enhancing MOEAs performance has a rising interest reflected by the large number of research efforts attending to real-world MOO problems with very costly objective functions evaluations [CLVV07]. Two possible ways have been investigated in the literature to face these challenges [LA15]. One solution is to use an approximative *surrogate* evaluation of the objective functions instead of exact fitness function evaluations [SGNJ08, CB11]. Another line takes advantage of parallel computing techniques and platforms by decomposing the computational load across multiple processors. MOEAs are very suitable for parallelization because their main operations in terms of genetic operators (recombination and/or mutation) and objective functions evaluations can be performed independently on different candidate solutions. Thus, for computationally intensive objective functions, parallelizing their evaluations turns out to be a simple and potentially useful idea.

As Coello et al. [CLVV07] have pointed out, pMOEAs are the preferred MOEAs implementation for solving large scale MOO problems whose objective function evaluations and constraints verification are the major computational bottleneck. Parallelization of MOEAs have been widely studied in the literature. The reader is referred to [TMO+08, JC09, LA15] for surveys on this topic.

Using parallel computing in the design and implementation of MOEAs has proven to be an effective mechanism to improve not only the computational efficiency (i.e. reducing execution time) but also the quality of the obtained Pareto fronts. This second benefit stems from the fact that research activities on parallelizing MOEAs also aimed at evolving MOEAs algorithmic structures towards more advanced and effective search engines through proposed approaches on parallel models. A pMOEA may be more effective than the sequential MOEA counterpart, even when executed on a single processor [LA15].

Next, we present the most common parallel models used in pMOEAs.

5.2.2 Main parallel models used in pMOEAs

Parallel models represent the algorithmic design aspect for pMOEAs. Once the algorithmic model is explicitly designed, subsequently it is implemented and refined for deployment onto a given physical parallel architecture.

Four main parallel models are usually adopted in the parallelization of MOEAs: the master-slave, the diffusion, the island and the hybrid models [CLVV07, JC09, TMO+08, LA15]. Each of these models is briefly described in what follows.

- **Master-slave model**

  The *master-slave* or global parallelization scheme is known as the simplest way (from algorithmic and implementation management perspectives) to parallelize MOEAs, and hence, the most popular among practitioners [LA15, JC09].

  In this model, objective function evaluations are distributed among a number of slave processors. While one central processor referred to as master processor achieves miscellaneous tasks such selection operations, dominance/Pareto ranking of solutions, management of the Pareto front, assigning workload to and collecting results from slave processors, etc. The creation of offsprings using genetic operators can be performed either by slave processors (i.e. a slave processor applies genetic operators on a given solution, then it evaluates objective functions of the offspring solution) or by the master processor (i.e. offsprings are distributed to slave processors to be assigned with objective values).

  This parallel model is illustrated in Figure 5.1 where the master processor distributes solutions to slave processors, controls when/where the slaves complete the evaluations to finally gather and save evaluated solutions returned by slaves.

  The search space exploration process of a master-slave pMOEA is conceptually the same as the sequential MOEA. This means that the number of processors being involved in the parallel processing is independent of the number of solutions being evaluated. Thus, for the same number of solutions being evaluated, a master-slave pMOEA achieves identical results to those obtained by the MOEA sequentially executed, but it does affect the execution time. It is important to mention that objective functions need to be enough complex and computationally expensive so that communication/synchronization overheads don't affect/overwhelm the overall pMOEA execution time. Otherwise, the efficiency of the pMOEA drops and sometimes even worse the sequential MOEA may spend less execution time than pMOEA.

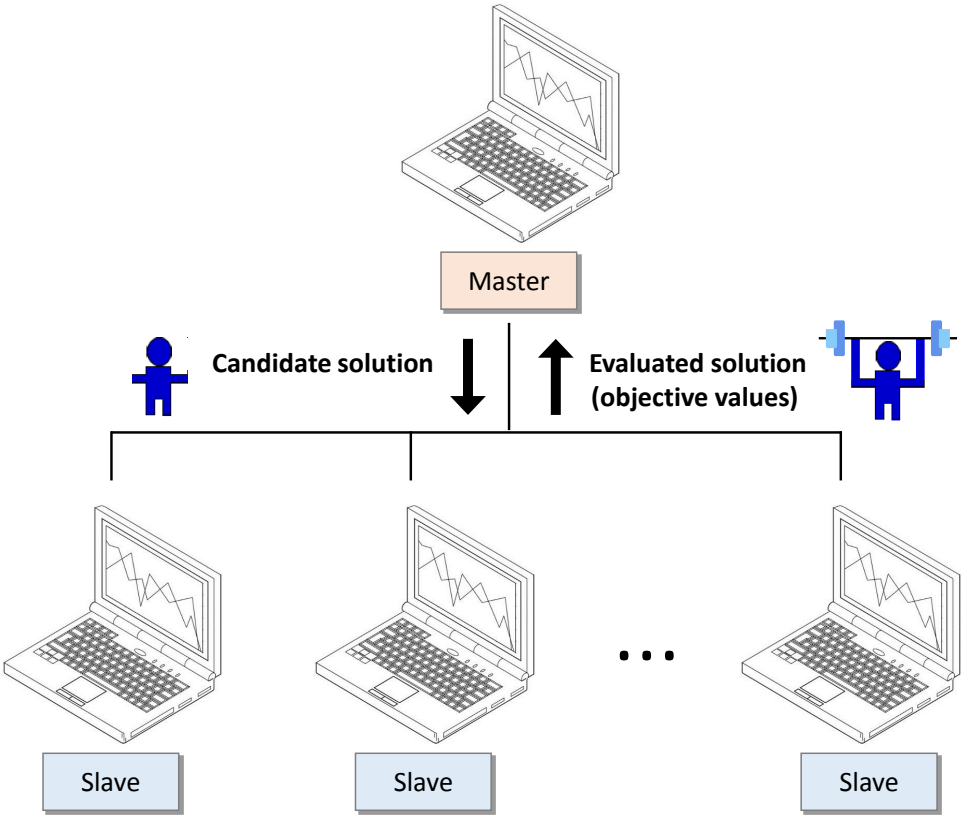


FIGURE 5.1: Master-slave parallel scheme

- **Diffusion model**

  In the *diffusion* pMOEA model [CLVV07], the population is spatially distributed over a set of processors structured onto a multi-dimensional grid (see Figure 5.2). Each grid point holds a processor and manages one individual solution of the population, that's why the diffusion scheme is sometimes referred to as *fine-grained* model. The main distinctive feature of this model is the dictation of a local neighborhood structure between grid points. The neighborhood of a given grid point is specified by a size and a shape that designate respectively the number of neighbors and their arrangement within the grid with respect to that grid point.

  Figure 5.2 illustrates an example of two-dimensional rectangular grid with a diamond-shaped neighborhood. As depicted by the dotted lines in Figure 5.2, each grid point has a neighborhood that overlaps with the neighborhoods of its nearby neighbors. The amount of overlap depends on the neighborhood size and shape. Note that all neighborhoods are of identical size and shape except on the edges.

  A given grid point can interact exclusively with its nearby neighbors to select

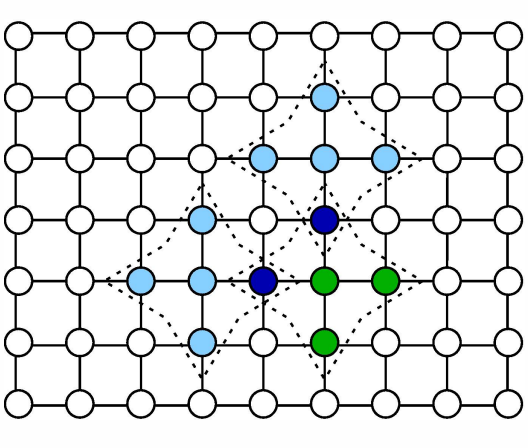


FIGURE 5.2: Diffusion parallel scheme (taken from [JC09])

parents and then produce offspring that will replace the current individual assigned to that grid point. The overlapping neighborhoods implicitly provides a migration mechanism of genetic material throughout the grid. This allows the expansion or diffusion of the "best" solutions (identified in different areas of the grid) throughout the entire population. The communication costs tend to be high, since the individuals who take part in the selection are distributed among neighbors processors [JC09]. This parallel scheme was designed for working in massively parallel computers [LA15].

- **Island model**

  In an *island* pMOEA, the overall pMOEA population is partitioned into a number of sub-populations called islands that evolve independently of each other (see Figure 5.3). Each island is isolated in a processor that runs a sequential MOEA for a number of generations/iterations called an epoch [JC09]. Such pMOEAs models are called also *coarse-grained* pMOEAs since each island maintains a large number of individual solutions. Additionally, they are called *distributed* pMOEAs as they are sometimes deployed on distributed memory computers.

  Although each island evolves in isolation for the majority of pMOEA execution, individual solutions migrate between neighboring islands at the end of each epoch, as shown in Figure 5.3. This migration mechanism requires a set of parameters and design decisions whose identification poses a number of key design challenges. These parameters are defined as follows:

  - the number of solutions to migrate
  - the migration frequency i.e. how often migration occurs (the number of generations between events)
  - the topology which defines the neighbors of each island

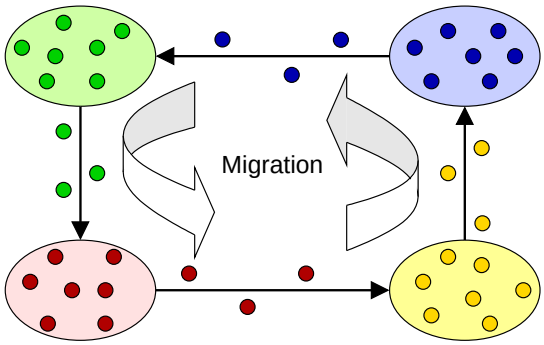


FIGURE 5.3: Island parallel scheme (taken from [JC09])

– the decision regarding the solutions selected for migration and those selected to be replaced by the immigrants

These parameters need to be set carefully as they strongly impact results of the entire pMOEA.

- **Hybrid model**

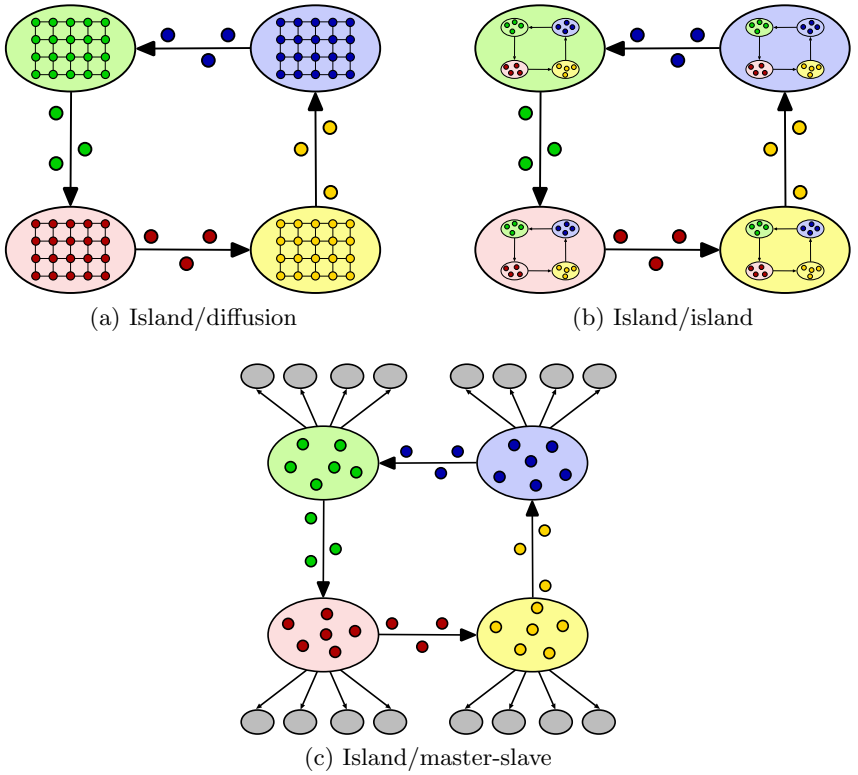
  Hybrid pMOEA models introduce another alternative to parallelize MOEAs that relies on the combination of two-level parallelization approaches. A coarse-grained parallel scheme is used at a high level of abstraction (e.g. island model) and each associated island runs a fine-grained scheme at a low level (e.g. diffusion model or master-slave model). Figure 5.4 illustrates three hybrid schemes proposed by Cantù Paz [CP00a] that use an island model at the high level:

  – Each island hosts a diffusion pMOEA (see Figure 5.4(a)),
  – Each island hosts a master-slave pMOEA (see Figure 5.4(c)) and
  – Each island hosts an island pMOEA (see Figure 5.4(b)).

  A hybrid approach allows the advantages of two pMOAE models to be combined.

Synchronous and asynchronous communication modes

All the aforementioned pMOEA models may be implemented in two different communication modes: either a synchronous or an asynchronous manner. Synchronous implementations lie in the use of some sort of synchronization mechanism (e.g. semaphores, locks or barriers) such that all processes are synchronized at the end of each generation/iteration or epoch. However, the need for synchronisation once per generation would increase the pMOEA execution time for fine-grained approaches. While asynchronous implementations maximize the exploitation of all available processors with no idle periods of time. That is all



(a) Island/diffusion (b) Island/island

(c) Island/master-slave

FIGURE 5.4: Hybrid parallel schemes (taken from [JC09])

available processors remain fully busy during the optimization. Asynchronous implementations are useful when processing time associated to parallel processors are different due to variation in processors speeds, memory sizes and/or workload decomposition between processors.

5.2.3 Discussion

After giving the basic concepts about the parallelization of MOEAs, we devote this section to explain our choices in terms of parallel scheme and communication mode.

The major computational bottleneck in the functions to tasks DSE problem we deal with in this thesis lies in objective functions evaluations and schedulability verification of design alternatives. According to our approach, the computation of these tasks relies on the scheduling simulation whose computation is non-linear and time-consuming. As mentioned before, the execution time of the scheduling simulation grows exponentially with respect to the designed system size (i.e number of functions or tasks) and complexity (i.e. timing constraints and/or shared resources contention). Additionally, design alternatives explored during

the search can be processed independently of each other. Thus parallelizing the mutation, evaluation and verification of many solutions will help to speed up the search.

Given the simplicity of the master-slave scheme and its suitability for solving MOO problems with computationally expensive objective functions and constraints [CLVV07, LA15], we opt for this scheme in the parallel formulation of our DSE process. Furthermore, the other parallel schemes are usually used for sub-population searching problems [LA15]. We deal with the parallelization of PAES algorithm that maintains a single current solution. Besides, the adaptation of one of these schemes is not straightforward for us because they require a thorough background knowledge regarding their key concepts like neighberhood, migration mechanism, etc.

As for the communication mode between processes, we adopt the asynchronous mode because with the parallel formulation that we propose, (see Section 5.4) execution times may vary from one slave to another. This is due to the time required to manage a given solution (i.e. mutate, check feasibility and evaluate fitness) which differs from one solution to another. Then, any synchronization mechanism established between slaves would increase the overall execution time without impact on results or the final Pareto front. These choices will be explained further in Sections 5.4 and 5.5.

5.3 Related work on PAES parallelization

Although significant research efforts and important advances have been achieved in the field of pMOEAs, however to the best of our knowledge, few work addressed the parallelization of PAES.

Authors of [LNA06, NLTA05] proposed a parallel PAES example based on the island model. This parallel PAES approach works as follows. Each island runs the sequential PAES and maintains its own local archive of non-dominated solutions. Periodically, islands exchange solutions according to a synchronous migration mechanism. The parameters used in the migration mechanism are set arbitrary by the authors. Once results from all islands become available, the last step consists in building the final Pareto-front by merging local fronts obtained in each island. Results of experiments conducted in this work showed that the proposed parallel PAES is not suitable for MOO problems with computationally expensive objective functions. Considering distinct sub-populations seems not computationally efficient for MOO design problems with costly objective function evaluations.

In [COA11], authors proposed another parallelization approach for PAES using a master-slave formulation. The proposed approach lies in running simultaneous

parallel evaluations at each iteration, and comparing results against the current solution at the end of each iteration. In order to generate candidate solutions to be evaluated at an iteration, authors consider a prediction tree taking into account the probability of a mutated solution to be better than its parent. This work tries to avoid useless evaluations, but still handles solutions synchronously. Again and in contrary, we intend to ensure optimal exploitation of available hardware resources by adopting an asynchronous parallel processing.

5.4 Parallel formulation of our DSE process

This section provides details about the parallel formulation of our DSE process in terms of a master-slave parallel asynchronous adaptation for PAES and a new selection strategy to guide the search procedure.

5.4.1 Master-slave parallel asynchronous adaptation for PAES

We consider here a master-slave asynchronous framework (see Figure 5.5). This model aims at distributing the processing of candidate solutions (i.e. mutation, objective functions evaluations and feasibility verification) on several slave processors while a single master processor manages the Pareto ranking, archiving and selection parts of PAES. In our approach we assume a central unique archive of non-dominated solutions handled by the master. As stated by the master-slave paradigm and also shown by Figure 5.5, slaves are started by the master, and interact exclusively with it, receiving solutions to process and sending back mutated solutions and associated objective values.

In the case of our DSE process, additional cost of the parallelization are expected to be reasonable as compared to the computation costs. This is due to the fact that the parallelization additional cost mainly lies in solution data transfers between the slaves and the master and a solution data consists only in a chromosome representation (i.e. an integer array carrying a set of function indexes) and a few of objective function values.

For simplicity, the proposed parallel asynchronous PAES is denoted `PA-PAES` in the rest of this chapter.

From an algorithmic design perspective, our parallel framework requires the following components:

- Selection strategy: it can be achieved based on one or many criteria such as: dominance or Pareto ranking scheme like the conventional PAES, crowded regions in objective space [KC00b], performance measure as hypervolume-based selection [BZ06], etc.

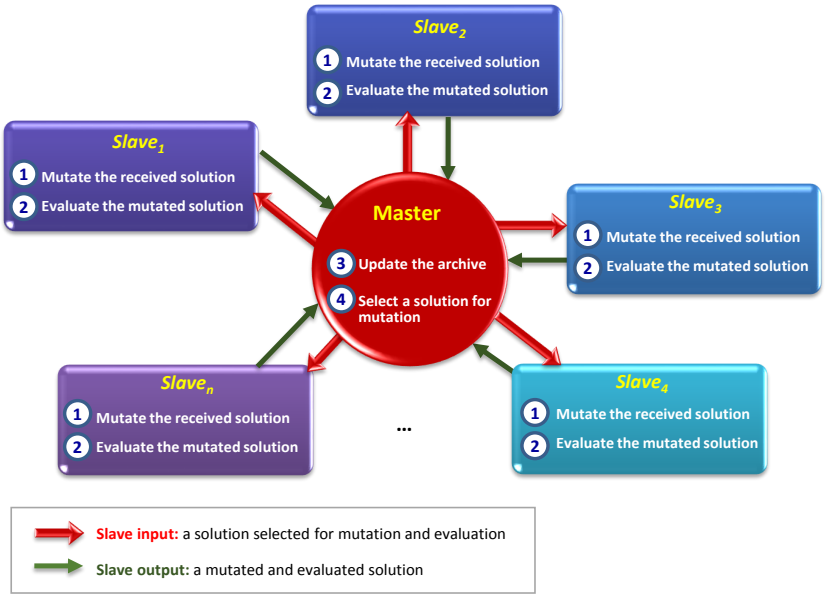


FIGURE 5.5: Master-slave parallel asynchronous scheme adapted to our DSE process

- Mutation and evaluation procedures.
- Local search procedure that outputs one or more neighbours (optional).

We also propose a new selection strategy that will be explained in Section 5.4.2. We maintain the mutation and evaluation procedures developed with the formulation of conventional PAES to our DSE process (see Chapter 4). Besides, the local search procedure of conventional PAES is kept the same in the parallel algorithm.

This parallel algorithm also takes as input a number S of slaves and a number I of iterations. We only use I for lack of simplicity. The master and slave computation processes for our parallel framework are described in Algorithms 7 and 8 respectively.

As stated before, parallel operations or workload achieved by slave processors are realized asynchronously. This means that the master processor does not have to wait for all of the evaluated solutions returned by all slaves. The results of the slave computations are taken into account as soon as they become available: when a slave sends an evaluated solution, the master archives it using the original PAES approach, and goes on the optimization by selecting a new solution to send to the slave for mutation and evaluation. This way, we save the overall computation time and resource usage, especially as the processing time of different solutions by slaves may be different. This is because time spend in the mutation may differ from one solution to another: for a given mutation execution the number of solutions generated and checked until having a feasible solution is not know

in advance. This fact in turn is due to the randomness involved in the mutation procedure. Furthermore, execution time associated to one feasibility verification run may vary from one candidate solution to another, since for some cases the solution to be checked is rejected without processing a scheduling simulation (see Section 4.5).

Algorithm 7: Master process algorithm

```
1  begin
2      start S slaves;
3      send to each of them a seed unevaluated solution;
4      iteration := 1;
5      while iteration ≤ I do
6          receive a set or a single evaluated solution(s) from any slave s;
7          compare it/them with the archive A;
8          update the archive A with non-dominated solutions;
9          if iteration < I − S then
10             select a candidate solution according to a selection strategy;
11             send it to a slave s;
12         end if
13         iteration := iteration + 1;
14     end while
15     terminate all slaves;
16     return the final archive A;
17 end
```

Algorithm 8: Slave process algorithm

```
1  begin
2      while not terminated do
3          receive a candidate solution from master;
4          mutate it;
5          evaluate it;
6          if local search then
7              generate and evaluate neighbours;
8              discard locally dominated neighbours;
9          end if
10         send back to the master the evaluated solution;
11     end while
12 end
```

5.4.2 Global selection: a new selection strategy for PAES

The parallel framework needs a selection strategy for the search procedure. The conventional PAES uses a straightforward selection strategy that relies on a form of Pareto ranking mechanism that in its turn is based on dominance and crowding

criteria comparisons (details are given in Section 4.2.1). This selection strategy would either accept the mutated solution m to become the new current solution or keep the current solution c as a basis for mutation. We should also mention that the selection strategy of PAES is fast since it requires only few comparisons to perform.

As opposed to the sequential version of PAES, in the parallel framework there is no current solution maintained by the master process. Observe that the selection operation (achieved by the master) is overlapped with the slave computations that are expected to be time-consuming, we can spend more time in the selection mechanism while intending to enhance the search engine. The main idea of our proposed selection strategy is to use the archive of non-dominated solutions as a basis for selecting a new candidate solution for mutation, instead of using solely the received solution from the slave.

The solution selected to be sent to a slave for mutation and evaluation (line 10, Algorithm 7) is picked from the archive as follows. We randomly choose a selection criterion among the followings:

- **random criterion**: a random solution from the archive is selected, to ensure the exploration aspect.
- **one of the objective function**: one of the archive solutions that is within the 10% best values of this objective function is picked. This criterion fosters the *exploitation* of most promising solutions already identified.
- **crowding criterion**: a solution from the archive within the less crowded area is chosen. This criterion helps in maintaining good diversity and spread of the non-dominated solution set. Thus, it supports the *exploration* of undiscovered regions of the search space.

These choices are equiprobable. This new selection strategy is designed to balance between exploration and exploitation of the solution space. The proposed selection strategy and that of conventional PAES are referred to as `global selection` and `local selection` respectively.

5.5 Experiments and evaluation

In this section, we present experiments that assess the efficiency of the above proposals. We perform two evaluations. The first evaluation aims at comparing the proposed `global selection` strategy against the `local selection` of the original PAES through comparing Pareto fronts produced using each of these selection strategies. The second evaluation intends to (1) assess the efficiency of

the parallel framework regarding to the temporal behavior, and (2) check that the parallelization does not induce lost on fronts quality as compared to the sequential counterpart.

Experiments are performed on synthetically generated test instances through our problem instance generator described in Section 7.3.1. For both evaluations, we assume systems with independent functions, i.e. no shared resources in generated test cases. In that respect, the assessment of our proposals with respect to the complexity of systems (given as input to our DSE process) is confined to the variation of test cases sizes (i.e. the number of functions). Furthermore, as part of the performed experiments, the DSE process is executed with two objective functions namely $Min\ (f_1 = \#\ preemptions)$ and $Min\ (f_4 = (H - \sum_{i=1}^{k} L_i))$.

While configuring experiments, certain experimental parameters like `#test instances per experiment`, `#PAES independent runs per test instance` and `#PAES iterations` are set to relatively small values. The reasons behind these experimental setup are discussed in Appendix A.

Experiments are conducted on a SMP machine with 48 processors at 2.2 GHz frequency and 125 GBytes of RAM, running Linux CentOS. The parallel framework is developed in Ada programming language within our prototype (see Chapter 6). Ada concurrency features (tasking, synchronization and communication with *Rendezvous* or protected objects, ...) are used in order to implement parallel formulation of our DSE process.

In the following, first we introduce the performance metrics used for measuring the results and assessing our proposals (section 5.5.1). Then, sections 5.5.2 and 5.5.3 are devoted to present the two performed evaluations. For each evaluation, we provide the experiments protocol and configuration, then the obtained results and their analysis.

5.5.1 Performance assessment metrics

In the conducted experiments, we are interested in analyzing two main performance concerns by means of two different metrics:

1. Execution-time efficiency of our parallel framework that is assessed using the most common parallel performance measure i.e. the *speed-up*.

   Speed-up metric

   The speed-up metric is a classical way for measuring the efficiency of parallel algorithms [AT02]. This metric captures the relative benefit of solving a particular problem in parallel by comparing the sequential against the parallel time required to solve that problem. The speed-up denoted S_p is defined as the

ratio of the execution time of the sequential MOEA on one-processor (denoted T_1) to the execution time of a given pMOEA using p processors (denoted T_p).

$$S_p = \frac{T_1}{T_p}$$

Because of the random nature of PAES, one single run is not statistically significant. Thus, we compute speed-up values using average values of execution time computed over a given number of independent runs. Larger values of S_p are considered better and ideally the speed-up is greater or equal to the number of processors involved in the parallel computations ($S_p \simeq p$, or $S_p \geq p$).

2. Effectiveness of our parallel framework in terms of quality of produced Pareto fronts, which is measured by applying the *hypervolume* indicator (see Section 2.3.4).

5.5.2 Solution sets quality evaluation: global selection Vs. local selection

This evaluation aims at qualitatively comparing `global selection` regarding to `local selection`. For that purpose, a number of experiments were run on a set of synthetically generated problem instances. We intend to investigate and compare the effectiveness of both selection strategies within our DSE process while varying the scales (in terms of the number of functions) of test instances under consideration.

Experiments setup

Here we present how we have set parameters of the function sets generator in order to generate test cases for the current experiments. In these experiments, the total processor utilization factor of generated function sets is fixed at 80%. Different function set sizes are used that range from 20 to 100 by step of 20 functions. For each size, we generate 5 different test instances and each test instance is processed by our DSE process with the following configurations:

(1) Local selection with sequential PAES (i.e. the conventional PAES),

(2) Local selection with PA-PAES$_4$ (i.e. using 4 slaves),

(3) Global selection with sequential PAES,

(4) Global selection with PA-PAES$_4$ (i.e. using 4 slaves).

The number of iterations of the DSE process is fixed at 2000, for all the above configurations. Each test instance is executed by each of these configurations for 5 independent runs. For each test instance, we consider the average hypervolume value computed over the 5 runs of a particular configuration.

Results interpretation

For each considered test instances size, Figure 5.6 reports hypervolume values associated to each of the 4 configurations. For a given configuration, the presented hypervolume value is the average over all processed test instances with same size.

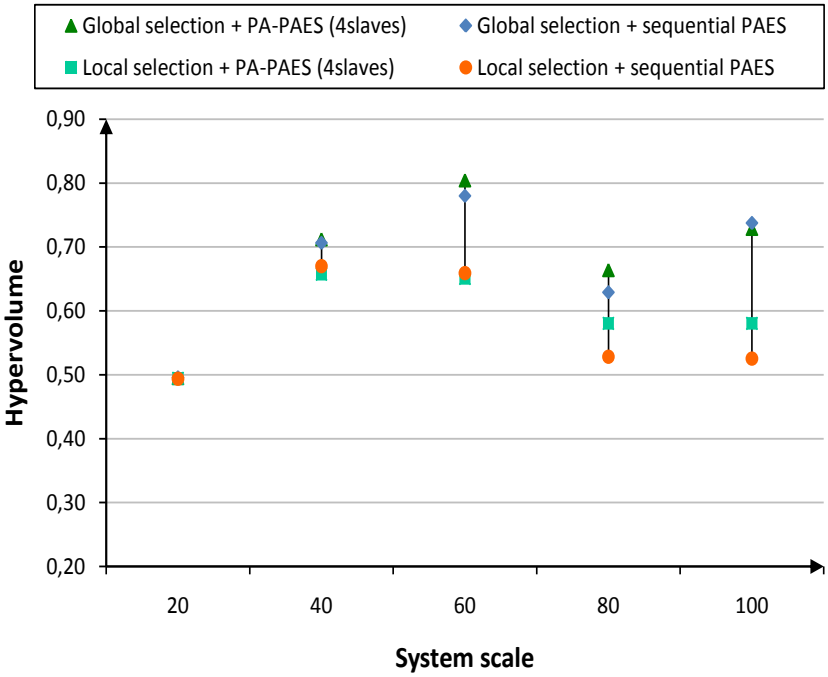


FIGURE 5.6: Hypervolume comparison between the global selection and the local selection with sequential-PAES and PA-PAES$_4$ configurations for different test instances scales

As we can observe in this figure, for almost all considered sizes, best hypervolume values are associated to the following configuration: global selection with PA-PAES$_4$. In addition, we note that efficiency of the global selection when compared with the local selection is more relevant for larger sizes test instances ($\geq 60 functions$). This conjecture is reinforced by the hypervolume improvement achieved with the global selection with respect to the local selection under both sequential and parallel executions.

Table 5.1 reports, for both the sequential and parallel executions, the hypervolume improvement rate related to each considered test instance size. For a given size and a particular execution configuration (sequential or parallel), the

TABLE 5.1: Average hypervolume improvement rates of the global selection against the local selection in sequential and parallel execution configurations

Size (#functions)	hypervolume improvement rate (%)	
	sequential execution	parallel execution (4 slaves)
20	0.43	0.06
40	5.43	8.28
60	18.35	23.59
80	19.09	14.32
100	40.43	25.48

hypervolume improvement rate of the global selection against the local selection is computed as follows: $(\frac{HV_{GS}}{HV_{LS}} - 1) \cdot 100$ where HV_{GS} is the hypervolume value with the global selection and HV_{LS} is the hypervolume value with the local selection. According to values provided in Table 5.1, the hypervolume improvement rates increase when test instances sizes raise. We can notice also that the improvement rate is fairly negligible for test instances with 20 functions. This can be explained by the fact that all the considered configurations perform almost equally for small-size test instances, since the latter involve reasonable and manageable search spaces.

Moreover, for the parallel execution configuration i.e. PA-PAES$_4$, we computed the speed-ups for the performed experiments. The average speed-up for the overall experiments is about 3.8 which represents a promising speed-up value for 4 parallel slaves. In the next section, we carefully analyze speed-up values while varying the number of slave processes.

5.5.3 Scalability and effectiveness evaluation of the parallel DSE process

The second evaluation is performed in order to assess the efficiency of the parallel formulation of the DSE process (PA-PAES) compared to its sequential counterpart.

Experiments setup

We run a set of experiments in a range of different number of slaves (4, 8 and 16) to investigate the speed-up of the parallel approach. These experiments are carried out on three problem instances scales: 50, 80 and 100 functions. The selection strategy is set to the proposed global selection. For each scale, we generate 5 different test instances using our function set generator. Each test instance is processed 5 times by the sequential formulation as well as by the PA-PAES$_4$,

PA-PAES$_8$ and PA-PAES$_{16}$. The termination condition is set to 2000 iterations for all performed runs with the sequential and the parallel configurations.

Results interpretation

Figure 5.7 points out the average speed-up against the number of slaves involved in the parallel computation. In this figure, we can notice that even if they are not linear, speed-ups increase regularly with the number of involved slaves, reaching up to 11.7 for 16 slaves with test cases having 100 functions. The decrease marked in the speed-up values with more slave processors may indicates a decrease in the efficiency of the parallel method with the number of parallel processors. A more thorough investigation of this issue will be performed as part of our future work. However, this cannot deny the improvement achieved (in term of execution time) with the parallel formulation regarding the sequential counterpart.

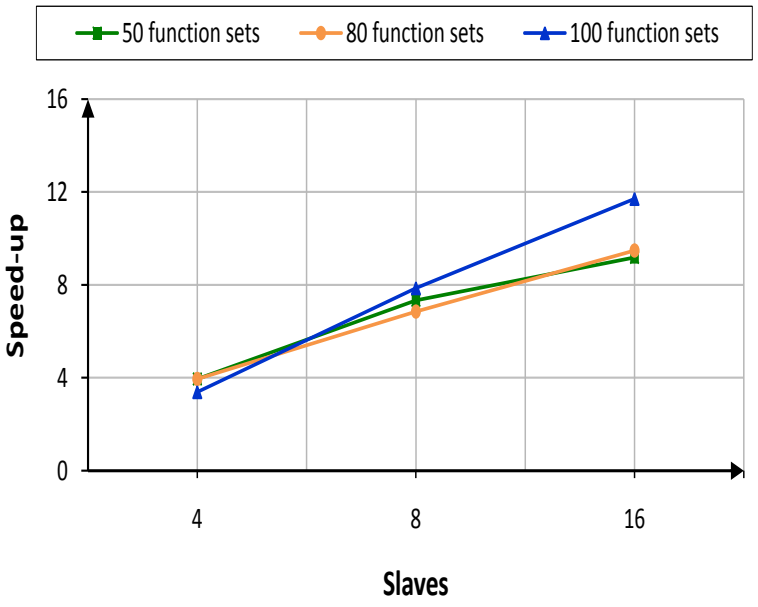


FIGURE 5.7: Speed-up values against the number of slaves involved in the parallel execution

As shown in Figure 5.8 quality is maintained, with almost roughly constant hypervolume values from 4 to 16 slaves.

To conclude, the results of this evaluation provide evidence that the parallel formulation of the DSE process allows to master large-scale and more complex problem instances with affordable execution time, while preserving quality.

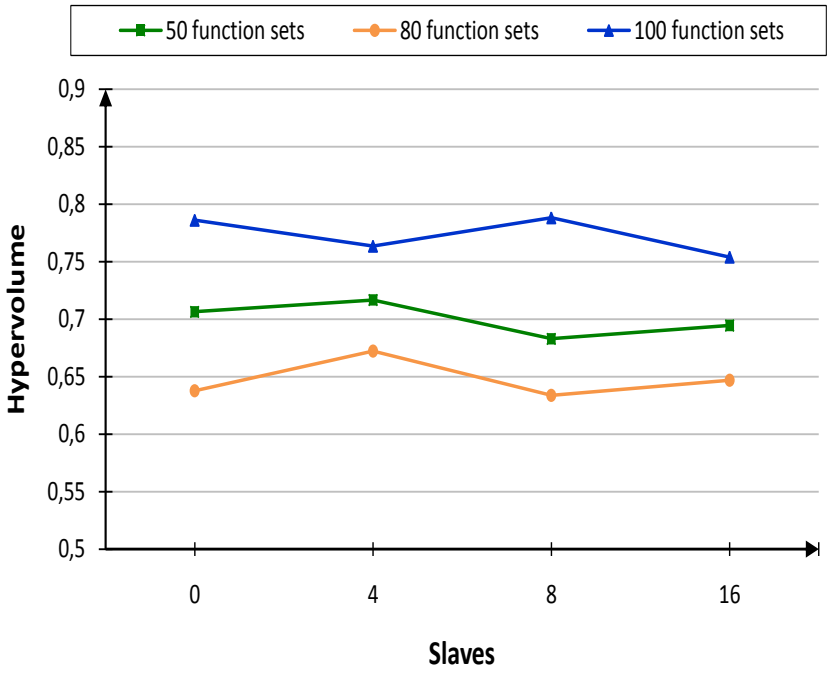


FIGURE 5.8: Average hypervolume against the number of slaves involved in the parallel execution

5.6 Conclusion

In this chapter, we proposed a parallel formulation to the DSE process. The parallel framework intends to improve the applicability of our DSE process to more complex and larger problem instances. This parallel framework relies on a master-slave parallel asynchronous scheme. In order to enhance the search mechanism of our method, we proposed and investigated a new selection strategy for the PAES algorithm (called global selection strategy). Our parallel approach is particularly suitable for multi-objective optimization problems having expensive and with variable time objective functions computations.

The parallel framework was evaluated by a set of experiments on synthetically generated problem instances. Results show an improvement not only in the execution time (e.g. with 4 slave processors, the average speed-up is about 3.5 for test cases with 100 functions) but also in the quality of obtained solution sets when compared to the previous sequential formulation (e.g. with 4 slave processors, the average improvement rate in hypervolume is about 38.7% for test cases with 100 functions). The quality improvement has been achieved thanks to the new global selection strategy.

In the next chapter, we will describe the prototype provided as part of this thesis and that implement all elements of our approach.

6

Prototype Implementation of the Design Exploration Process

Contents

6.1 Introduction

In previous chapters, we presented the theoretical contributions and research activities performed in order to establish our method for the design exploration of the functions-to-tasks assignment solutions space towards the optimization of a

set of conflicting performance criteria. These contributions consist in the definition of all the approach elements and details required to implement our solution for the addressed problem. All these elements are structured and developed in a prototype easing the exploitation of the proposed multi-criteria DSE process.

This chapter is dedicated to expose the design and features of the prototype. It is organized as follows. Section 6.2 gives the features covered by the prototype. The latter is supported by the Cheddar scheduling framework that provides an extensible toolset. Section 6.3 is devoted to introduce the Cheddar framework. Details related to the design of our prototype are presented in Section 6.4. Last section concludes the chapter.

6.2 Prototype overview

The prototype developed to implement elements of our approach covers the following aspects:

- The implementation of the multi-criteria DSE process based on PAES technique (details are given in Chapter 4).
- The parallel implementation and the new selection strategy for the PAES search procedure proposed to achieve better scalability and efficiency of the DSE process (See Chapter 5).
  The prototype provides various options which allow users to run the DSE process while configuring the method parameters. The user can set the number of processors to be involved in the execution (by default, the method is executed in a sequential way). He can also select one of the selection strategies (either the original selection strategy or the one that we propose).
- Problem instance generator: the prototype gives the user the opportunity to test the method through a synthetically generated problem instance. A problem instance represents a customized functional specification that includes a set independent or dependent functions through shared resources. The user specifies parameters in order to configure the problem instance generator according to the method detailed in Section 7.3.1. Thanks to the customizable problem instance generator, we are able to perform a broad range of experiments while varying the generator parameters following the experiment requirements.
- The implementation of an exhaustive search method defined in Section 7.2.1 (see Algorithm 9), mostly for evaluation purposes.

We continue by presenting the Cheddar framework and explaining how this framework was extended to support our prototype.

6.3 Cheddar framework

The Cheddar framework is a *GPL open-source* scheduling analyzer and simulation toolset freely available to researchers and practitioners for checking the temporal behaviour of real-time applications. It is developed and maintained by the MOCS/CACS team of Lab-STICC laboratory (*Université de Bretagne Occidentale, France*).

The Cheddar framework is built around an ADL called *Cheddar-ADL* devoted for real-time scheduling theory. This framework provides a panoply of scheduling analysis features.

In the next sections, detailed information about the Cheddar-ADL and scheduling analysis features are presented. Then, we explain the manner of using Cheddar to perform a scheduling analysis of a given real-time application.

6.3.1 Cheddar-ADL for modeling RTE systems

Cheddar-ADL is an analysis-specific[1] ADL. It has been designed in the specific perspective of scheduling analysis. The main concepts manipulated through Cheddar-ADL stem from the real-time scheduling theory. This ADL defines various entities (such as a processor, core, task, shared resource, critical-section, etc.) which enable to model main concepts required to perform a scheduling analysis of a given RTE application. Two kinds of entities are supported by Cheddar-ADL, namely, hardware components and software components.

- ***Hardware components in Cheddar-ADL***

  Hardware entities model resources provided by the execution platform. Similarly to the real-time scheduling theory that usually deals with relatively straightforward hardware architectures, Cheddar-ADL provides limited capabilities to model hardware components. As Figure 6.1 depicts, hardware entities are of two types:

  - **Core entity**: it models a resource unit that read and execute program instructions [KTJ06]. According to Cheddar-ADL, a core entity involves attributes that permit the setting of the scheduling parameters (e.g the scheduling policy, preemptive or not, etc.).
  - **Processor entity**: it may involve one or multiple cores which enable the modelling of both mono-core and multicore architectures.

[1] An analysis-specific ADL means that this ADL focuses on concepts related to a particular analysis problem

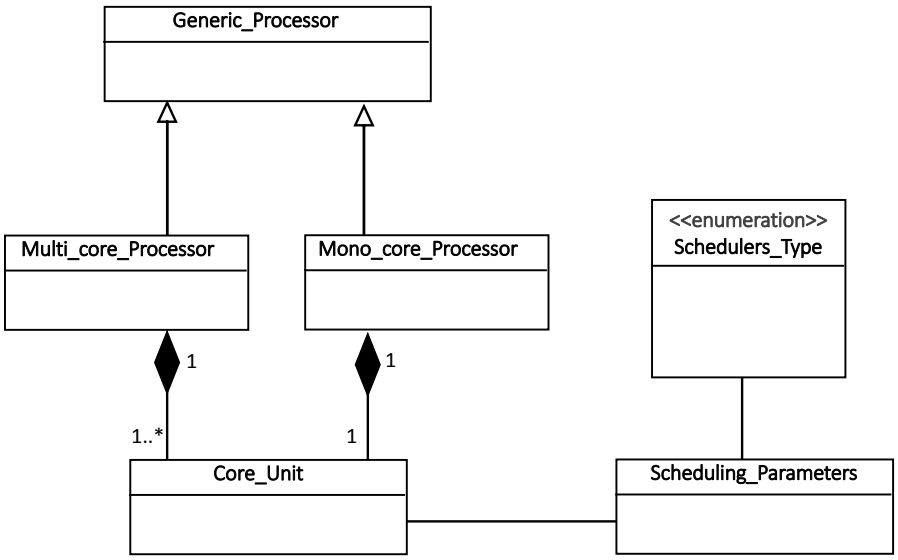


FIGURE 6.1: Main hardware components in Cheddar-ADL (adapted from [TRA17])

- ***Software components in Cheddar-ADL***

  As illustrated in Figure 6.2, Cheddar-ADL provides a set of software components allowing to model the software part of a real-time system. By looking at both Figures 6.1 and 6.2, we can notice that the software parts of a system are modeled in a more detailed way. The Cheddar-ADL user guide [FSP+14] defines these software components as follows:

  - **Address space**: an address space entity refers to either physical address or virtual address. It may include a range of addresses that can be accessed by several components e.g. processors, tasks, etc. An address protection mechanism may be assigned to an address space.
  - **Task**: a task component models a control flow (i.e. run any type of program). It is statically attached to an address space component.
  - **Resource**: a resource specifies any data structure that can be shared between tasks assigned to the same address space. It may be accessed through one of classical resources access synchronization protocols (e.g. PCP, PIP or IPCP). A resource is statically defined in an address space.
  - **Buffer**: it models queued asynchronous data exchanges between tasks belonging to the same address space.
  - **Message**: unlike a buffer component, a message entity allows to model queued asynchronous data exchanges between tasks located in **different** address spaces.

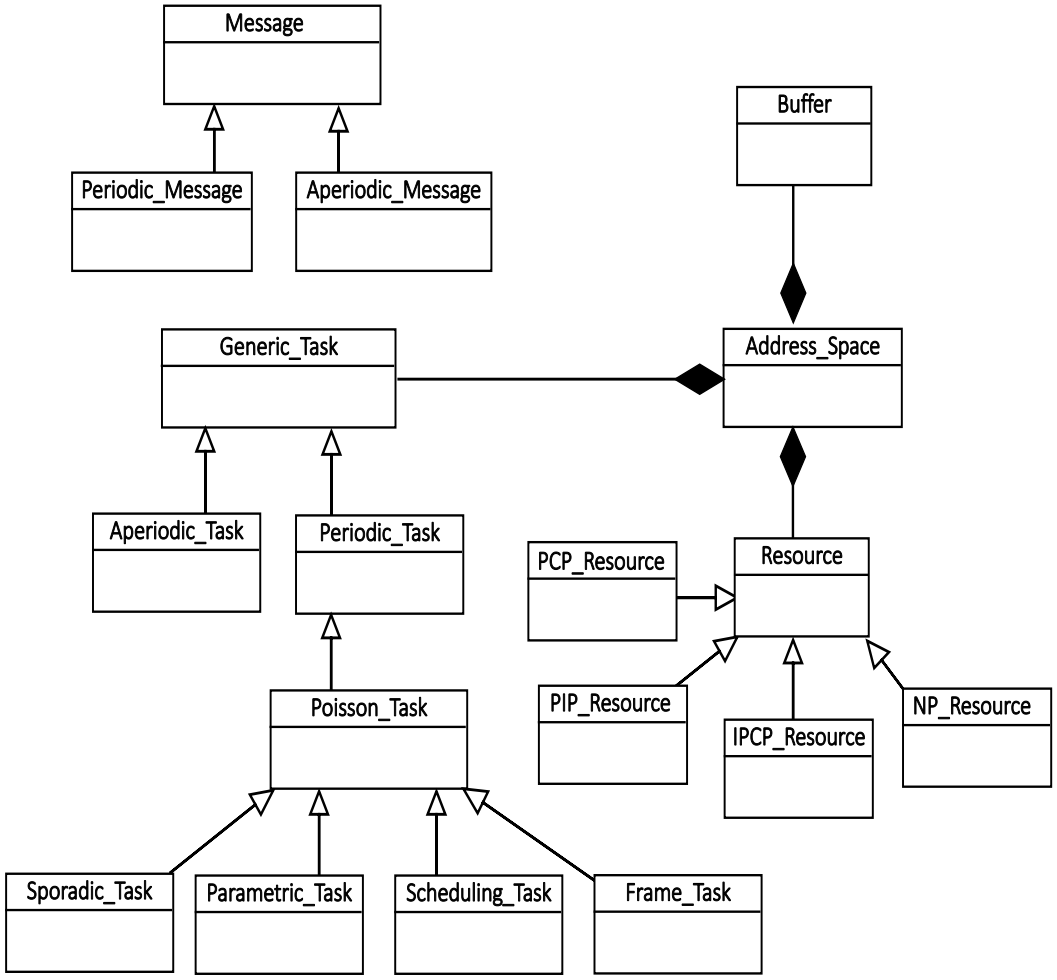


FIGURE 6.2: Main software components in Cheddar-ADL (adapted from [TRA17])

Resource, buffer and message components enable the design of the different kinds of dependencies between tasks supported by the scheduling theory.

An exhaustive list of software and hardware entities and their attributes is provided in the Cheddar-ADL user guide [FSP+14].

Listing 6.1 illustrates an extract of a Cheddar-ADL model. The latter describes a system running on a uniprocessor platform under a fixed priority (according to the Rate-Monotonic) and a preemptive scheduler. The system tasks shown in this listing are periodic. They interact through shared resources that are synchronized using PCP protocol. As shown in this listing, the Cheddar-ADL formalism is based on XML.

Cheddar users can use the Cheddar-ADL associated graphical-editor in order to easily build their real-time application models.

Listing 6.1: Example of a Cheddar-ADL model

```
<?xml version="1.0" encoding="utf-8"?>
<cheddar>
  <core_units>
    <core_unit id="id_2">
      <name>core1</name>
      <scheduling>
        <scheduling_parameters>
          <scheduler_type>RATE_MONOTONIC_PROTOCOL</scheduler_type>
          <preemptive_type>PREEMPTIVE</preemptive_type>...
        </scheduling_parameters>
      </scheduling>...
    </core_unit>
  </core_units>

  <processors>
    <mono_core_processor id="id_3">
      <name>processor1</name>
      <core ref="id_2">...
    </mono_core_processor>
  </processors>

  <address_spaces>
    <address_space id="id_1">
      <name>addr1</name>
      <cpu_name>processor1</cpu_name>...
    </address_space>
  </address_spaces>

  <tasks>
    <periodic_task id="id_4">
      <name>Task1</name>
      <task_type>PERIODIC_TYPE</task_type>
      <cpu_name>processor1</cpu_name>
      <address_space_name>addr1</address_space_name>
      <capacity>6</capacity>
      <deadline>140</deadline>
      <period>140</period>...
    </periodic_task>
    <periodic_task id="id_5">
      <name>Task2</name>
      <task_type>PERIODIC_TYPE</task_type>...
      <capacity>4</capacity>
      <deadline>70</deadline>
      <period>70</period>...
    </periodic_task>...
  </tasks>

  <resources>
    <pcp_resource id="id_6">
      <name>R1</name>
```

```
51        <protocol>PRIORITY_CEILING_PROTOCOL</protocol>
52        <critical_sections>
53          <task_name>Task1</task_name>
54          <critical_section>
55            <task_begin>2</task_begin>
56            <task_end>4</task_end>
57          </critical_section>
58          <task_name>Task2</task_name>
59          <critical_section>
60            <task_begin>1</task_begin>
61            <task_end>1</task_end>
62          </critical_section>
63        </critical_sections>...
64      </pcp_resource>...
65    </resources>
66    ...
67  </cheddar>
```

6.3.2 Cheddar scheduling analysis features

Given a system model (expressed with Cheddar-ADL or any input format supported by Cheddar) to be analyzed, the Cheddar scheduling analyzer offers different schedulability tests that can be driven on this system model. Basically, the schedulability tests provided by Cheddar are either feasibility tests or scheduling simulations on the feasibility interval. Cheddar supports classical methods of both scheduling analysis techniques. In the following, we first present some of the feasibility tests implemented into the Cheddar scheduling analyzer and then, we introduce its scheduling simulation main features.

- ***Scheduling analysis with feasibility tests***

  We recall that 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. A feasibility test is defined according to a particular set of assumptions (see Section 3.3.1). Many feasibility tests provided by the scheduling theory are available with the Cheddar scheduling analyzer. For example, Cheddar implements the processor utilization based test that is one of the most popular feasibility tests in the context of Liu and Layland models. Moreover, the Cheddar scheduling analyzer involves WCRT based tests that can be applied in the context of periodic tasks independent or subject to mutual exclusion constraints due to shared resources. In addition, other feasibility tests for hierarchical architectures have also been implemented.

  Since feasibility tests don't cover all real-time contexts on the one hand and some of them are known as being too pessimistic on the other hand, additional

techniques such as simulation are then involved in the Cheddar scheduling analyzer.

- ***Scheduling analysis with simulations***

  The Cheddar scheduling simulator supports most of conventional schedulers such as RM, DM, EDF or LLF, under both preemptive and non preemptive policies. Those scheduling algorithms have been implemented in the context of both uniprocessor architectures and global scheduling with multiprocessor architectures. Furthermore, the Cheddar simulator illustrates the hierarchical scheduling by implementing the ARINC653 scheduling policies and several classical aperiodic servers (deferrable, sporadic, polling).

  From a task model standpoint, the Cheddar simulator manages usual task types such as periodic, aperiodic and sporadic, etc. Besides, it allows to simulate the scheduling sequence of a task set while taking into account dependencies between tasks related to shared resources, precedence or communication task relationships. Again, several classical resources access synchronization protocols (namely PCP, PIP and IPCP [BW97]) are implemented into the Cheddar simulator.

  Various performance criteria associated with the system to be simulated are computed from the scheduling sequence and delivered with the scheduling simulation analysis result. These performance criteria are: worst/best/average response time, probability distribution of response time, worst/best/average shared resource blocking time, number of context switches, number of preemptions, deadlocks, priority inversion or specific properties defined with a domain specific language [SPR+15].

  Although Cheddar supports a great number of scheduling methods, there are cases where existing methods don't fit the characteristics of a particular system. For those cases, Cheddar provides the possibility to extend the scheduling analysis capabilities without a deep understanding of the internal design and implementation of the Cheddar framework. This feature allows users to easily tailor the scheduling analysis tool to their needs (i.e. implementing a scheduling method that does not exist yet in Cheddar).

6.3.3 Utilization scenarios of Cheddar framework

The main utilization scenario of Cheddar scheduling analyzer consists of three steps as displayed in Figure 6.3.

Step 1: Designing the system model
At a first stage, designers/users should specify the design model of the application to be analyzed by creating the application components (software and hardware),

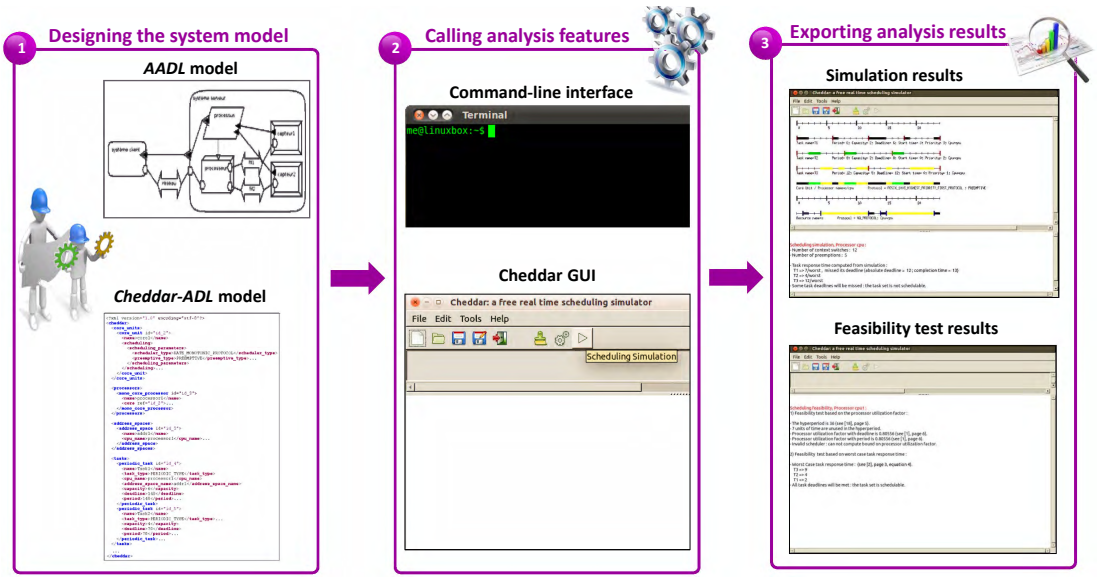


FIGURE 6.3: Utilization scenario of Cheddar

defining their attributes and how software components are deployed on the hardware components. Cheddar offers different ways to design the input system model and introduce it to the framework:

- AADL model: Cheddar supports importing and also exporting a design model that is expressed in AADL [FG12].
- Cheddar-ADL model: Cheddar supports importing and also exporting a system model that is defined in Cheddar-ADL.
- Cheddar GUI: Cheddar offers a graphical-editor that provides elementary actions required to build/edit a system model, namely "select", "add", "delete" and "modify" actions. Once the system model is built, it can be saved and exported in Cheddar-ADL model format.
- Ada program: Users can manually create a system model by writing an Ada program. This method can be used when the system model is created inside a program that calls Cheddar analysis features. In the implementation of the DSE process, we typically use this method in order to create design alternatives explored during the search process and that need to be analyzed. However, such a way could be laborious and not user-friendly especially for ordinary users because it requires to well understand the implementation of Cheddar.

STEP 2: CALLING ANALYSIS FEATURES

Analysis features can be called by using a command line interface, Cheddar GUI

or inside an Ada program. The latter alternative is the manner used in the implementation of the DSE process prototype in order to apply the scheduling simulation on design alternatives.

STEP 3: EXPORTING ANALYSIS RESULTS
When Cheddar GUI is used to perform scheduling analysis, then analysis results are displayed in the Cheddar GUI. Cheddar allows to export analysis results in XML file format.

6.4 Prototype implementation

Figure 6.4 illustrates an overview of the software architecture of our prototype and a subset of Cheddar framework libraries.

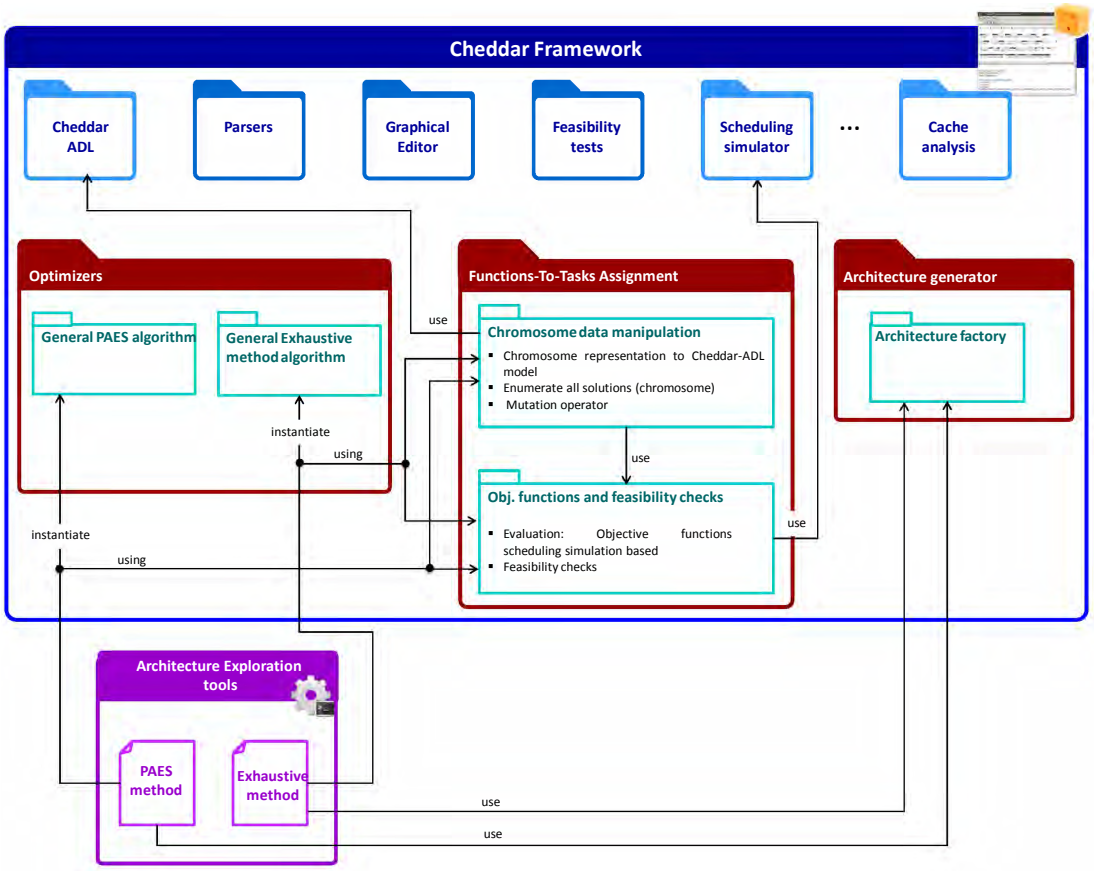


FIGURE 6.4: Prototype design overview

As shown by Figure 6.4, in order to implement our prototype, we extend the Cheddar framework as follows. We add two libraries called `Optimizers` and `Functions-To-Tasks Assignment` and we extend an existing library called `Architecture generator`.

6.4.1 Optimizers library

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

6.4.2 Functions-to-tasks assignment library

The `functions-to-tasks assignment` library defines components that are specific to our optimization problem domain. As shown in Figure 6.4, this library contains two packages. The first package called `chromosome data manipulation` provides functions for manipulating the chromosome representation adapted to the functions-to-tasks assignment. For example, it enables to transform a chromosome representation associated to a given candidate solution to the corresponding Cheddar-ADL model using the `Cheddar ADL` library (already exists in the framework). This package 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 4.4). The mutation operator proposed for our problem (details in Section 4.3.3) is also implemented in this package. It 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 called `objective functions and feasibility checks` is dedicated to the evaluation of candidate solutions by way of objective functions (Section 4.3.4) and the verification of design alternatives feasibility (Section 4.5). This package uses the scheduling simulator library of Cheddar framework in order to perform the scheduling analysis and compute the objective functions from the scheduling simulation results (e.g. the number of preemptions, the number of context switches, worst case response time of tasks, worst case blocking time of tasks, etc).

6.4.3 Problem instance generator

The implementation of our problem instance generator (details are given in Section 7.3.1) 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. The problem instance generator implementation involves both the function sets generator and the resource sets generator described in Section 7.3.1.

6.4.4 Tools

As we can see in Figure 6.4, our prototype provides two new tools ensuring the execution of the DSE process and the exhaustive method for the DSE mapping problem that we address. Each one of these programs instantiates the general form of the corresponding technique 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 as argument the Cheddar-ADL model of the initial solution to the method to run. Second, the user can configure the problem instance generator through a set of parameters that he provides to the program in order to execute the desired method on synthetically generated problem instances.

6.5 Conclusion

In this chapter, we presented the prototype conceived and implemented as part of research activities conducted in this thesis. This prototype was implemented and integrated in Cheddar, an existing scheduling framework. The key benefits of this implementation are the reusability and extensibility of its software artefacts. For example, the optimizers library could be reused in different MOO 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 DSE-specific problem with other MOO solvers, etc.

Experiments achieved using our prototype in order to assess our proposals are presented in the next chapter.

7

Evaluation and Empirical Studies

Contents

7.1 Introduction

In the previous chapter, we presented the prototype that implements our approach to the addressed problem. Thanks to this prototype, we are able to

perform experimental studies and evaluate our theoretical and technical proposals.

This chapter presents experiments achieved to assess the proposed DSE process and drive empirical studies regarding certain concerns, namely the conflicting relationships between performance criteria and the impact of the initial design on the DSE process behavior/performance. Experiments are structured into two major sets according to the system models assumed in this thesis.

As stated before, in some of the following experiments, certain experimental parameters like `#test instances per experiment`,`#PAES independent runs per test instance` and `#PAES-iterations` are set to relatively small values. The reasons behind these experimental setting are discussed in Appendix A.

The remainder structure of the chapter is the following. Section 7.2 provides experiments performed to assess our proposals on systems with independent tasks. Sections 7.3 presents experiment sets achieved on systems with shared resources. Finally, Section 7.4 summarizes results of experiments and concludes the chapter.

7.2 Experiments for independent tasks systems

In this section, we present experiments to assess the effectiveness of our DSE process approach while considering independent tasks systems. In these experiments, the DSE process is executed with the conventional PAES (i.e. sequential execution and the default local selection strategy, see Sections 4.2 and 4.3) and two objective functions namely $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$.

Experiments are achieved by means of synthetically generated test cases i.e. function sets with the following configuration:

- Function periods ζ_i, are uniformly generated between 10 and 150 units of time. The periods generator is implemented so that generated values are multiples of 10.
- Processor utilization factor U_i for each function F_i is tuned with the UUnifast method of Bini and Buttazzo [BB05], so that the function utilizations added up to the desired total utilization factor for the function set.
- Function deadlines are implicit, i.e $\forall\ i$: $\delta_i = \zeta_i$.
- Function capacities γ_i are set based on the generated periods and processor utilization factors, $\forall\ i$: $\gamma_i = Max(1, Round(\zeta_i \cdot U_i))$ where $Round(x)$, is a function which returns the nearest integer to x. By this expression we intent to warrant that generated capacities are non-zero integer values. In fact, the generated processor utilization factor per function is likely to be smaller when

the number of functions increases. Small utilization factor of functions may lead to zero values capacities (if they are computed by $\gamma_i = Round(\zeta_i \cdot U_i)$). However, the resulting overall processor utilization value of the generated function set may be slightly different of the one given as a parameter to the generator.

We perform two experiments. The first experiment aims at comparing the Pareto front attained by our DSE process against the optimal Pareto front. This experiment is applied on small-size test cases. Furthermore, in order to assess the effectiveness of the proposed method for larger sizes test cases, we carry out a second evaluation that lies in analyzing the quality of produced Pareto sets using the hypervolume metric (description of this metric is provided in Section 2.3.4).

Experiments are conducted on a personal computer with a 2.4GHz Intel Core i7 processor, 8 GByte of memory and running Ubuntu 12.04 OS. In the following, we present experiments protocols and results of both evaluations.

7.2.1 Experiment 1.1: Accuracy/convergence evaluation for small-sizes test instances

In the first evaluation, we are interested in comparing Pareto fronts generated by our DSE process against optimal Pareto fronts. To do so, we design and implement a method that for a given problem instance determines the optimal Pareto front through an exhaustive search among all functions-to-tasks assignment solutions. The exhaustive method procedure is outlined in Algorithm 9.

Algorithm 9: Exhaustive method algorithm

input : a functional specification of a particular application: function set (n functions) sharing or not a set of resources (m resources)
output: optimal Pareto front

1 **begin**
2 Enumerate all possible assignment solutions; /* The number of all enumerated solutions is equal to the *Bell* number of the processed function set size (B_n). */
3 Determine feasible solutions among all the enumerated solutions by applying Algorithm 6;
4 Evaluate feasible solutions according to the considered objective functions;
5 Compute the Pareto front from feasible solutions, by discarding dominated solutions;
6 **end**

An exhaustive search as the above proposed method can be applied only for small-size instances of our problem. This is due to the fact that the search space size increases exponentially with the problem instance size (i.e. the number of functions) and the evaluation of one candidate solution is not immediate. That's why the current experiment is carried out on a test case with 11 functions.

Experiment protocol and parameters settings

The 11 functions test case was generated by the above described function sets generator. The total processor utilization given as input to the generator is set to 50%. Generated timing parameters of functions are given in Table 7.1.

Function	Capacity	Period	Deadline
F_1	2	60	60
F_2	10	110	110
F_3	8	120	120
F_4	1	30	30
F_5	15	120	120
F_6	2	110	110
F_7	2	60	60
F_8	3	120	120
F_9	4	60	60
F_{10}	1	100	100
F_{11}	2	90	90

TABLE 7.1: Generated timing parameters of test case with 11 functions

We first compute its exact Pareto front using the exhaustive method, and we then also apply to it our DSE process, with 2000 PAES-iterations. We notice that the number of feasible solutions examined by the DSE process is always lower than the number of performed iterations. In fact, at each PAES-iteration the mutation procedure (Section 4.3.3) generates a feasible solution that can have been already investigated in previous PAES-iterations.

Results interpretation

For a problem instance of 11 functions, the number of all possible assignment solutions to proceed with the exhaustive method is equal to the 11^{th} *Bell number*, $B_{11} = 678570$ solutions. For the considered 11-functions test case (Table 7.1), the exhaustive method outputs 2530 feasible design alternatives among all possible assignment solutions. 7 of those solutions are non-dominated, and constitute the exact Pareto front.

Figure 7.1 shows the projection in the objective space of all feasible solutions found by the exhaustive method as well as the exact Pareto front. For the test case under consideration, our DSE process succeeds at identifying the exact Pareto front. It is also important to mention that the DSE process has achieved 226 iterations in about 10 minutes to get the entire exact Pareto front. Whereas, the exhaustive method took about 7 hours and 25 minutes. In the performed experiments, we run the DSE process for a fixed number of iterations. However,

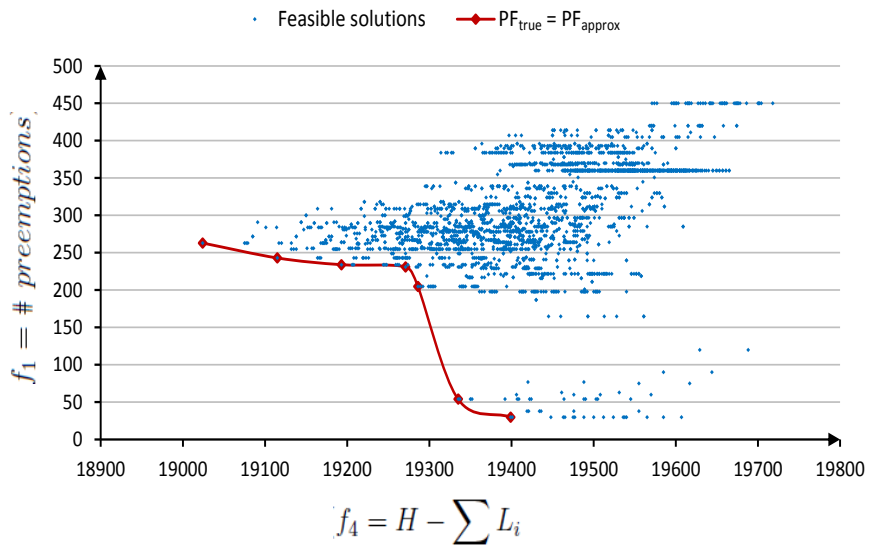


FIGURE 7.1: Projection in the objective space of all feasible solutions and the exact Pareto front for the 11-functions test case

seeing that for the considered test case, the DSE process didn't need all the 2000 iterations to reach the exact front, it should be interesting to investigate some convergence-based termination criteria (which is out of the scope of this thesis).

This experiment shows the effectiveness of our method to determine the exact Pareto front ("best" trade-offs of design alternatives), for a small-size test case, in a short time as compared to the exhaustive method. Finally, although pessimistic assumptions and choices we have made in Section 4.4 to compute parameters of design alternatives, experiments show that there are many schedulable solutions in the search space.

7.2.2 Experiment 1.2: Solution sets quality evaluation for different test instance sizes

A number of experiments were run in order to investigate the quality of solution sets for larger-size systems. The hypervolume quality indicator (see Section 5.5.1) is used to analyze and compare Pareto fronts obtained by our DSE process.

Experiment protocol and parameters settings

We consider different systems sizes that range from 15 to 40 functions by step of 5 functions. For each size, we generate 10 different test instances by means the above described generator. The total processor utilization factor of all generated test instances is set to 80%. The number of PAES-iterations for each run is fixed

at 3000. Each test instance is processed with our DSE process for 5 independent runs. For each test instance, we consider the average hypervolume value computed over the 5 runs.

Results interpretation

The hypervolume values of test instances for each size are depicted in Figure 7.2.

In this figure, for each size, the *cross* denotes the average hypervolume value over all test instances and the ends of the vertical lines indicate the maximum and minimum obtained hypervolume values. Remind the maximum value of the hypervolume metric is 1.

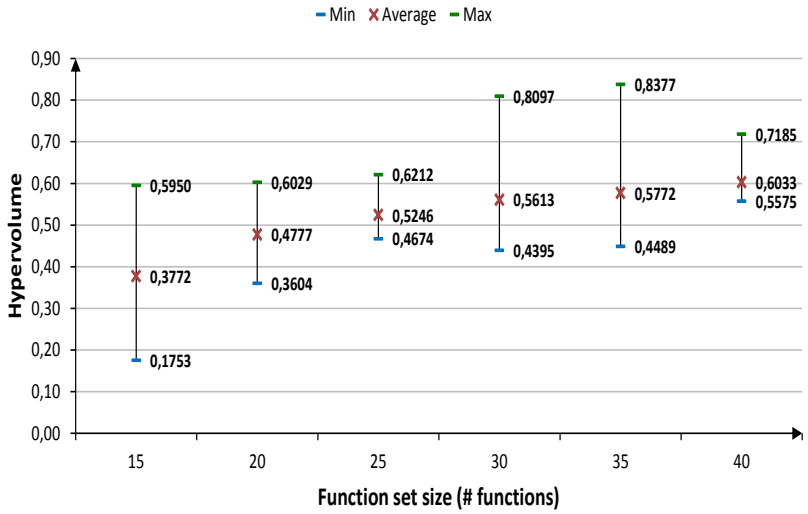


FIGURE 7.2: Hypervolume values of solution sets for different system sizes

As we can observe in this figure, the average hypervolume values increase with the size of test instances. For example, the average hypervolume value for 40-functions test instances is equal to 0.6 against 0.47 for 20-functions test instances[1]. This observation can be explained as follows. All test instances for the different studied sizes are generated with the same total processor utilization and the same period interval. Thus, function capacities of test instances with larger sizes (i.e. $\geq$ 25 functions) are rather small compared to function capacities of smaller size test instances. This means that timing constraints are increasingly acute with test instances of smaller size. In other words, the overall system laxity is more narrow with smaller size test instances, which would lead to less schedulable design alternatives associated to these test instances.

[1]Remind that larger hypervolume values point out better quality of Pareto sets (in terms of good spread, diversity and closeness to the exact Pareto front).

Rather, for larger size test instances, timing constraints become moderate (larger overall system laxity) which involves larger spaces of schedulable solutions. These facts are reflected in the obtained front shapes and hypervolume values as well. Indeed, front shapes associated to smaller size test instances are likely to be compact (few non-dominated solutions) and with non-convex portions, as the front shown in Figure 7.1 (in Experiment 1.1). For example the hypervolume of this front is equal to 0.23, which is a low value despite that was the optimal Pareto front. In [VVL98a], the authors do note that the hypervolume metric may be misleading in the case of non-convex PF_{approx}. Hence, reduced hypervolume values associated to smaller size test instances did not necessarily imply inefficiency of the DSE process. As for larger size test instances, their Pareto fronts are likely to be convex, extensive and well spread in the objective space (their fronts shapes are similar to the one shown in Figure 2.7) which would explain their high hypervolume values.

This conjecture is consolidated by the average number of solutions in produced fronts (PF_{approx}): we can observe in the second row of Table 7.2 that there are more non-dominated solutions with larger size test cases.

Function set size	15	20	25	30	35	40
Avg. #solutions	5	9	12	15	13	14
HV avg. std-dev.	0.0033	0.0133	0.0365	0.035	0.0431	0.0725

TABLE 7.2: Average number of solutions in produced fronts and average standard deviation of the hypervolume between runs

Furthermore, we have studied the stability of the DSE process between different runs of the same problem instance. The stability means the capability of the method to generate fronts with close qualities over different runs of the same test instance. To do so, we compute the standard deviation of the hypervolume between runs of each one of the considered test instances. The third row of Table 7.2 provides the average standard deviation over all test instances of each size. We can observe that the standard deviation raises when the size of test cases increases. Again, this can be explained by the fact that larger size test instances have larger spaces of feasible solutions, which may lead the DSE process to behave differently from one run to another for the same test instance. Thus the stability of the DSE process decreases with larger size test instances.

Finally, these experiments show that the DSE process makes effectively the architecture exploration and provides a set of promising trade-offs for the designers.

After accomplishing the above experiments we have realized that the proposed test instance generator is unable to generate large-size, schedulable and with

bounded simulation period problem instances. In fact, this issue has been addressed in [GM01, BFO14]. Thus, in next section we improve the test cases generator by following the method proposed in [GM01].

7.3 Experiments for systems with shared resources

In this section, the focus is set to RTE systems with Ravenscar compliant shared resources. We present experiments and empirical studies carried out as part of the assessment and the investigation of some concerns related to the proposed DSE process while considering systems with shared resources.

In those experiments, we use the parallel formulation with the global selection strategy proposed in Chapter 5. Experiments are conducted on a SMP machine with 48 processors at 2.2 GHz frequency and 125 GBytes of RAM, running Linux CentOS.

The content of this section is structured as follows.

- First, in Section 7.3.1, we describe the test instance generator proposed as part of the current experiments. Details about performed experiments (in terms of experiments protocol, parameters setup, results and their analysis) are presented afterwards.
- Section 7.3.2 provides experiments achieved to explore the correlation between three objective functions selected from the list given in Section 4.3.4.
- Section 7.3.3 presents experiments intended to evaluate the accuracy (in terms of convergence and coverage ability) of the DSE process. This evaluation is performed on small-size problem instances using IGD and CR metrics (that are introduced in Section 2.3.4).
- Section 7.3.4 is devoted to the assessment of the DSE process effectiveness for more complex systems by investigating the quality of produced Pareto fronts for different resources contention levels using the hypervolume metric.
- Last, Section 7.3.5 shows the impact of the initial solution choice (see Section 4.3.2) on the DSE process convergence towards the Pareto-optimal front.

7.3.1 Test instance generator

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 m shared resources (in accordance with the functional specification model given in Section 3.3.2).

Functions parameters generation method

- **Functions period (ζ_i)**: by following the method of Goossens and Macq [GM01], the period of each function (among the n functions) is randomly selected from a set of n_k ($n_k < n$) different periods per function set. Then, in the produced function set, we have at most n_k different periods. With this method the *lcm* of all generated periods (i.e the associated *hyperperiod*) is bounded by a given upper bound. This implies a bounded simulation duration required to study the schedulability. In our implementation, the upper bound of the hyperperiod is set to 27720 time units.

- **Functions processor utilization factor (u_i)**: given a processor utilization U of a function set, the utilization factor u_i of each function F_i is generated using the UUnifast method of Bini and Buttazzo [BB05].

- **Functions capacity (γ_i)**: computed in the same way as with the generator for independent function sets (see Section 7.2), $\forall\ i$: $\gamma_i = Max(1, Round(\zeta_i \cdot U_i))$. As stated before, the resulting overall processor utilization $U_{current}$ may be different of the one given as a parameter to the generator U_{target} (from which u_i are generated): $U_{current} = \sum_{i=1}^{n} \frac{\zeta_i}{\gamma_i} \geqslant U_{target}$. The generator is implemented so that the difference between $U_{current}$ and U_{target} is bounded by a certain tolerance value. For example, our generator accepts only values of $U_{current}$ such as $U_{current} \in [U_{target} - 0.09, U_{target} + 0.09]$. That is, the generator repeats the generation of parameters (i.e. u_i, ζ_i and γ_i, $\forall\ 1 \leqslant i \leqslant n$) until this condition is met.

- **Functions deadline (δ_i)**: implicit, i.e $\forall\ i$: $\delta_i = \zeta_i$.

Resources and critical sections parameters generation method

Each function set handles a set of shared resources between the functions. Each resource R_j is accessed by a random number of functions in the range $[2, rsf \cdot n]$ 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 following the experiment purpose.

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 capacity of F_i: $Length(\omega_k) = csr \cdot \gamma_i$. In addition to the resource sharing factor, the critical section ratio allows us to customize the resources contention in our experiments. The dates of begin and end of ω_k, i.e. $(\omega_k)_{begin}$ and $(\omega_k)_{end}$ are set as

follows: $(\omega_k)_{begin}$ is randomly selected within the range $[1, \gamma_i - Length(\omega_k) + 1]$ and obviously $(\omega_k)_{end} = (\omega_k)_{begin} + Length(\omega_k) - 1$.

Note that all random variables are generated with a uniform distribution.

7.3.2 Experiment 2.1: Empirical study of the correlation between objectives

In all the previous experiments performed as part of the assessment of either the parallel approach (Section 5.5) or the DSE process when applied on systems with independent tasks (Section 7.2), we have considered only two objective functions, namely $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ Although in these experiments we did not investigate the conflict between this pair of objectives, results showed that they were effectively conflicting.

In the present experiment, functional input models to our methodology involve interactions between functions by means of shared resources. We would like to include at least one performance metric directly related to resource sharing as an optimization objective. We have already mentioned in Section 4.3.4 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: $Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ and $Min(f_6 = \sum_{i=1}^{k} B_i)$. 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 for efficient design space exploration, we have to 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.

Pearson correlation coefficient 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 [PF07].

Formally speaking, as defined by [CF95], two conflicting objectives are referred to as *negatively correlated*. But, if they support each other, then they are said to be *positively correlated*. Measuring the conflict or the correlation among objectives is a key research line in work proposing objectives reduction approaches to cope with many-objective MOO problems [DS06, LJCCC08, LJCUB09, JCAT14, SDT+13, WY16].

In order to estimate the correlation between the objectives, we suggest to use a simple method similar to those applied in [DS06, LJCCC08, LJCUB09, JCAT14]. 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 DSE process, as the data set. Each solution in PF_{approx} is an observation. The computed correlation coefficient is associated to a $p - value$ [WY93], representing the statistical risk of error on the correlation result significance (confidence value). In our experiments, $p - values$ are interpreted regarding a 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}.

Experiment protocol and parameters settings

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 test instances. The parameters settings of the test instance generator are given in Table 7.3. Experiments are performed on test instances with 20 and 30 functions. For each size, 50 different test instances are generated. In these experiments, certain parameters are set for both test instances sizes, namely: (i) the overall processor utilization factor is fixed at 80% (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 7.3. Hence, there will arise a sufficient number of resource conflicts.

Each generated test instance is processed by the DSE process for 5 independent runs. The number of PAES iterations for each run is fixed at 2000. For a given

TABLE 7.3: Experiment 2.1: test instance generator parameters settings

Common parameters	
Parameters	Values
# generated test instances per function set size	50
overall processor utilization	80%
# 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

test instance, the correlation coefficients of objectives pairs are computed over all the PF_{approx} generated by all the runs.

Results interpretation

Figure 7.3 reports for each pair of objectives: $[Min(f_1 = \#preemptions), Min(f_4 = (H - \sum_{i=1}^{k} L_i))]$, $[Min(f_1 = \#preemptions), Min(f_6 = \sum_{i=1}^{k} B_i)]$ and $[Min(f_4 = (H - \sum_{i=1}^{k} L_i)), Min(f_6 = \sum_{i=1}^{k} B_i)]$, the rates of each correlation kind (i.e. negative, positive or insignificant) captured over the 100 test instances.

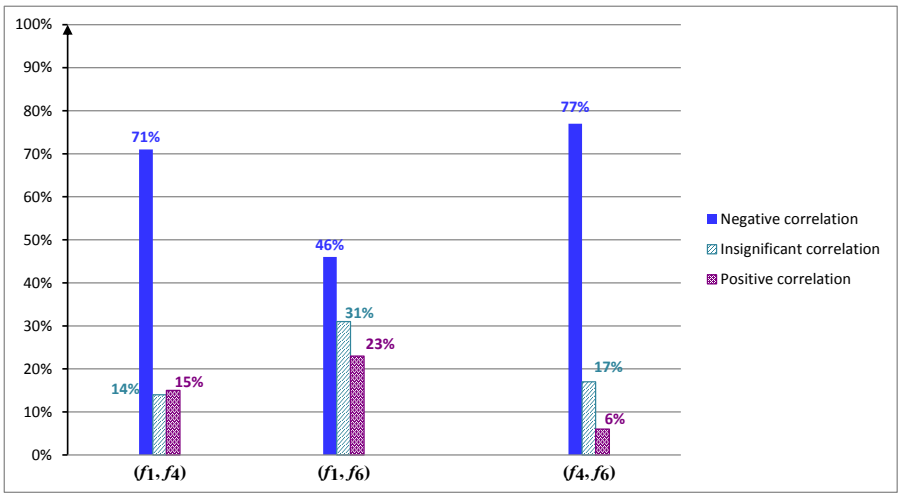


FIGURE 7.3: Negative, positive and insignificant correlation rates between the objectives pairs over all the generated test instances

We can observe that the presence of the conflict relationship among pairs of ob-

jectives is more important for some pairs than others. As shown in this figure, both pairs of objectives $[Min(f_1 = \#preemptions),\ Min(f_4 = (H - \sum_{i=1}^{k} L_i))]$ and $[Min(f_4 = (H - \sum_{i=1}^{k} L_i)),\ Min(f_6 = \sum_{i=1}^{k} B_i)]$ present a conflict relationship for most of the studied test instances (more than 70% of the test instances), whereas for the objectives pair $[Min(f_1 = \#preemptions),\ Min(f_6 = \sum_{i=1}^{k} B_i)]$ only 46% of the test instances present a conflict relationship. This means that running the design exploration method while considering as objectives one of the couples $[Min(f_1 = \#preemptions),\ Min(f_4 = (H - \sum_{i=1}^{k} L_i))]$ or $[Min(f_4 = (H - \sum_{i=1}^{k} L_i)),\ Min(f_6 = \sum_{i=1}^{k} B_i)]$ will be relevant. But, considering solely the pair of objectives $[Min(f_1 = \#preemptions),\ Min(f_6 = \sum_{i=1}^{k} B_i)]$ may be impractical since these two objectives reveal a positive or insignificant correlation for 54% of the test instances. $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ show a support relationships. This observation may be explained by the fact that for these test instances the added objective $Min(f_6 = \sum_{i=1}^{k} B_i)$ is in conflict with each one of the former objectives. In other words, for all the obtained trade-offs (i.e. all elements in the Pareto front) in each one of these test instances, the objectives $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ are either improved simultaneously or deteriorated simultaneously.

Nevertheless, 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 $[Min(f_1 = \#preemptions),\ Min(f_6 = \sum_{i=1}^{k} B_i)]$. 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).

Seeing these results, it would be interesting to apply a method that uses the conflict information during the search/optimization process like in [LJCUB09, JCAT14, SDT+13, WY16]. Such a method allows us to involve many objectives and subsequently the set of non-redundant objectives will be determined during the algorithm progress. This research line will be addressed as part of the current work perspectives.

7.3.3 Experiment 2.2: Accuracy/convergence evaluation for small-sizes test instances

This experiment aims at assessing the convergence and coverage of our DSE process. The inverted generational distance (IGD) and the optimal Pareto front coverage ratio (CR) quality indicators (see Section 2.3.4) are employed for this experiment. The computation of these metrics requires the knowledge of the theoretical optimal Pareto front. To this end, we use the exhaustive method previously proposed in Section 7.2.1 to perform Experiment 1.1 (see Algorithm 9).

Unlike Experiment 1.1 that deals with a unique test instance, in the current experiment we evaluate numerous test instances. The size of studied test instances is restricted to 10 functions at most. As a matter of fact, achieving the current experiment with test instances having 11 functions or more is impractical as their processing by the exhaustive method requires a great amount of time.

Experiment protocol and parameters settings

In these experiments, we take into account problem instances with 9 and 10 functions. For each size, 30 different test instances are generated using our generator. Parameters used in the configuration of the generator are pointed out in Table 7.4.

The low test instances size and the constraints dictated on these test instances (timing requirements and mutual exclusion constraints) will result in very few feasible solutions which turns out a single optimal solution (i.e $|PF_{true}| = 1$). These cases are ignored and we consider only test instances with PF_{true} containing at least two non-dominated solutions.

TABLE 7.4: Experiment 2.2: test instance generator parameters settings

Parameters	Values
size of test instances (i.e. # functions per function set)	9 - 10
# 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

In the light of results from Experiment 2.1 (Section 7.3.2), the current evaluation is achieved while considering the three objectives $Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ and $Min(f_6 = \sum_{i=1}^{k} B_i)$. 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 DSE process for 30 independent runs. Each IGD and CR value shown in this experiment is the average over the 30 runs. The number of PAES iterations for each run is fixed at 3000.

Results interpretation

Table 7.5 and Table 7.6 show the results for test instances of 9 and 10 functions respectively. For each studied test instance, we compute a set of performance

metrics, namely: the IGD value, the size of PF_{true} (i.e. the number of non-dominated solutions), the size of PF_{approx} (average, minimum and maximum number of solutions found over all the runs) and finally the CR value.

TABLE 7.5: Results relative to 9-functions test instances

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$		Avg. CR
			Avg.	Min - Max	
1	**0.0**	3	3.0	3 - 3	1.0
2	**0.0**	2	2.0	2 - 2	1.0
3	**0.0**	3	3.0	3 - 3	1.0
4	**0.0**	2	2.0	2 - 2	1.0
5	**0.0**	12	12.0	12 - 12	1.0
6	**0.0**	2	2.0	2 - 2	1.0
7	**0.0**	7	7.0	7 - 7	1.0
8	**0.0**	2	2.0	2 - 2	1.0
9	**0.0**	12	12.0	12 - 12	1.0
10	**0.0**	3	3.0	3 - 3	1.0
11	**0.0**	4	4.0	4 - 4	1.0
12	**0.0**	4	4.0	4 - 4	1.0
13	**0.0**	2	2.0	2 - 2	1.0
14	**0.0**	5	5.0	5 - 5	1.0
15	**0.0**	4	4.0	4 - 4	1.0

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$		Avg. CR
			Avg.	Min - Max	
16	**0.0**	5	5.0	5 - 5	1.0
17	**0.0**	2	2.0	2 - 2	1.0
18	0.0002	12	12.0	12 - 12	0.9861
19	0.0022	18	17.86	17 - 18	0.9814
20	0.0084	14	13.16	12 - 14	0.9404
21	0.0131	6	6.0	6 - 6	0.9222
22	0.0327	8	7.23	7 - 8	0.8958
23	0.1237	9	7.0	7 - 7	0.7777
24	0.2589	4	3.06	3 - 4	0.7666
25	0.2654	15	6.0	6 - 6	0.4000
26	0.3106	7	9.0	9 - 9	0.5714
27	0.3334	3	2.0	2 - 2	0.6666
18	0.3535	4	2.0	2 - 2	0.5000
29	0.7043	6	2.0	2 - 2	0.3333
30	0.7071	2	1.0	1 - 1	0.5000

TABLE 7.6: Results relative to 10-functions test instances

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$		Avg. CR
			Avg.	Min - Max	
1	**0.0**	2	2.0	2 - 2	1.0
2	**0.0**	3	3.0	3 - 3	1.0
3	**0.0**	3	3.0	3 - 3	1.0
4	**0.0**	5	5.0	5 - 5	1.0
5	**0.0**	12	12.0	12 - 12	1.0
6	**0.0**	3	3.0	3 - 3	1.0
7	**0.0**	2	2.0	2 - 2	1.0
8	**0.0**	3	3.0	3 - 3	1.0
9	**0.0**	2	2.0	2 - 2	1.0
10	**0.0**	4	4.0	4 - 4	1.0
11	**0.0**	2	2.0	2 - 2	1.0
12	**0.0**	9	9.0	9 - 9	1.0
13	**0.0**	9	9.0	9 - 9	1.0
14	**0.0**	6	6.0	6 - 6	1.0
15	**0.0**	5	5.0	5 - 5	1.0

Test instances	Avg. IGD	$\lvert PF_{true}\rvert$	$\lvert PF_{approx}\rvert$		Avg. CR
			Avg.	Min - Max	
16	**0.0**	3	3.0	3 - 3	1.0
17	0.0005	26	25.93	25 - 26	0.8104
18	0.0454	10	9.03	9 - 10	0.7633
19	0.0431	18	16.3	15 - 18	0.7629
20	0.1209	6	5.03	5 - 6	0.8388
21	0.1659	7	4.16	4 - 5	0.5952
22	0.1670	14	8.33	8 - 14	0.5857
23	0.1742	12	9.40	6 - 12	0.7833
24	0.1786	11	7.33	7 - 9	0.6363
25	0.2214	4	3.16	3 - 4	0.6333
26	0.3479	9	4.1	4 - 7	0.4518
27	0.4886	7	3.0	3 - 3	0.2000
28	0.5909	5	3.03	3 - 4	0.2000
29	0.6224	4	2.0	2 - 2	0.5000
30	0.7071	2	1.0	1 - 1	0.5000

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 values and we divide the results into 3 classes according to IGD ranges:

- **First class**: it encloses test instances holding a *zero* IGD value, which means that the DSE process method succeeds in converging to the exact Pareto front PF_{true} of these test instances in all achieved runs. This class represents 55% of all the presented test instances (17 test instances in Table 7.5 and 16 test instances in Table 7.6). 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 DSE process to handle instances with diversified Pareto front.

- **Second class**: it 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 of this second class may be asserted by the relatively high CR values: the average CR over test instances of the second class in both Table 7.5 and Table 7.6 are about 0.94 and 0.77 respectively. This means that for all test instances of this second class (i.e for both test instances sizes), our method is able to attain optimal non-dominated solutions by an average rate about 85%.

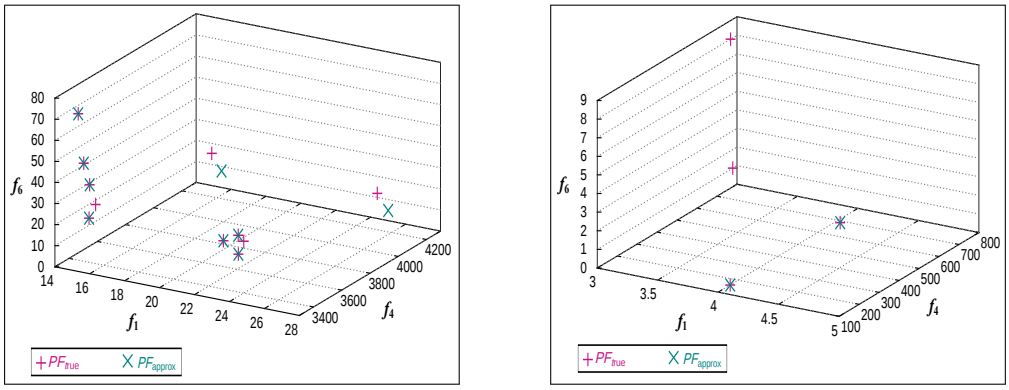
- **Third class**: it is defined for test instances with relatively *medium* and *high* IGD values that belong to the range $[0.1, 0.7]$. We find in this class around 32% of the studied test instances. The average CR over test instances of the third class in both Table 7.5 and Table 7.6 are about 0.56 and 0.53 respectively. The comparison of these CR values with those obtained in the second class justifies higher IGD values obtained in the third class with regard to the second one (remind that low IGD values are preferable and indicate better convergence and accuracy of the method).

  Furthermore, the range of IGD values of this class which is between 0.1 and 0.7 (i.e. a non-tight range) may be due to the variation of the coverage ability of the DSE process among test instances. For example, let consider from Table 7.6, test instance 24 having an IGD of 0.1786 and test instance 29 with an IGD of 0.6224. As we can see for both test instances, our DSE process misses 2 solutions: for test instance 24 it can reach at the best runs 9 solutions among 11 in PF_{true} and for test instance 24 it attains at best 2 solutions among 4 in PF_{true}. Even though the number of missed solutions is the same for both test instances, their IGD values are different. As shown in Figure 7.4, for test instance 24 (Subfigure 7.4a) the DSE process covers the overall Pareto area, whereas for test instance 29 (Subfigure 7.4b) it misses a region of the search space. That is the rationale for which test instance 29 is penalized in its IGD value (0.6224).

  The observation with problem instance 29 may due to constraints imposed on this problem instance. Indeed, constraints may create isolated feasible regions, which are unattainable directly by a mutation operation on a feasible solution from other feasible regions. In such constrained search spaces, a local search method (like the one used by PAES) could eventually fail to effectively explore the search space [AM15].

The results obtained with the DSE process for first and second classes (that represent 68% of the studied test instances) can be considered as fairly 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 execution time of both methods for the 60 studied test instances. For the DSE process, we consider the average execution



(A) PF_{true} and PF_{approx} of test instance 24 of Table 7.6

(B) PF_{true} and PF_{approx} of test instance 29 of Table 7.6

FIGURE 7.4: Comparison of the PAES coverage ability between two test instances

time over all the runs. Then, for each test instance we computed the execution time reduction rate of the DSE process against the exhaustive method: $(1 - \frac{T_{PAES}}{T_{Exhaustive}}) \cdot 100$. The average execution time reduction rate over all the test instances is about 74% which shows that the PAES method outperformed the exhaustive method with respect to the execution time. Furthermore, we notice that the execution time comparison results between the two methods are better for instances with 10 functions than 9 functions. This may be justified as follows. On one side, the exhaustive method spends more time for test instances with 10 functions than 9 functions, since the solutions 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.

7.3.4 Experiment 2.3: Solution sets quality evaluation for different resources contention levels

In Section 5.5 and Section 7.2.2, we have already evaluated our architecture exploration method for independent functions systems by assessing the quality of produced solution sets while varying the systems size up to 100 functions. We have shown that our method not only scales well but also makes effectively the architecture exploration and provides a set of promising trade-offs for the designer. The average hypervolume was 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 # non-dominated solutions in PF_{approx})

and qualitatively (using the hypervolume performance indicator defined in Section 2.3.4).

Experiment protocol and parameters settings

The parameters of the test instance generator are outlined in Table 7.7.

TABLE 7.7: Experiment 2.3: test instance generator parameters settings

Common parameters settings			
Parameters	Values		
test instance size (# functions per test instance)	30		
# generated test instances per resources contention level	20		
overall processor utilization	80%		
# 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

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, namely: (i) each test instance contains 30 functions, (ii) the overall processor utilization factor is set at 80%, (iii) function periods of a particular test instance are 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 7.7, the test instances associated to $level_1$ (respectively $level_2$, $level_3$) will have *low* (respectively *medium*, *high*) resources contention level.

As Experiment 2.2 (Section 7.3.3), the current experiment also considers the three objectives $Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ and $Min(f_6 = \sum_{i=1}^{k} B_i)$ to be optimized. Test instances of each level are processed by the DSE process for 10 independent runs. The number of PAES iterations for each run is set at 3000.

Results interpretation

Table 7.8, Table 7.9 and Table 7.10 present the results for the processing of 20 test instances relative to each of the three considered resources contention levels. Each

test instance is assessed 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.

TABLE 7.8: Results relative to test instances with *low* resources contention level ($level_1$)

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

Test instances	Hypervolume		$\lvert PF_{approx} \rvert$	
	Avg.	Std.	Avg.	Min - Max
11	0.0	0.0	4.00	4 - 4
12	0.0	0.0	3.00	3 - 3
13	0.0	0.0	8.00	7 - 9
14	0.0	0.0	3.80	3 - 4
15	0.0	0.0	2.40	2 - 5
16	0.0	0.0	7.00	7 - 7
17	0.0	0.0	8.10	7 - 9
18	0.0	0.0	3.00	3 - 3
19	0.0	0.0	5.80	5 - 6
20	0.0	0.0	8.30	7 - 10

TABLE 7.9: Results relative to test instances with *medium* resources contention level ($level_2$)

Test instances	Hypervolume		$\lvert PF_{approx} \rvert$	
	Avg.	Std.	Avg.	Min - Max
1	**0.8337**	0.0154	25.80	19 - 36
2	**0.8336**	0.0332	**79.20**	**59 - 101**
3	**0.8142**	0.0173	**90.60**	**74 - 119**
4	**0.7896**	0.0208	**60.00**	**51 - 72**
5	**0.7835**	0.0375	15.00	11 - 22
6	**0.7723**	0.0220	35.50	33 - 40
7	**0.7699**	0.0589	**103.60**	**68 - 132**
8	**0.7420**	0.0324	34.20	26 - 42
9	**0.7360**	0.0315	19.30	14 - 28
10	**0.7275**	0.0096	5.50	5 - 6

Test instances	Hypervolume		$\lvert PF_{approx} \rvert$	
	Avg.	Std.	Avg.	Min - Max
11	**0.6976**	0.0396	41.80	24 - 56
12	**0.6559**	0.0276	25.40	22 - 28
13	**0.6527**	0.0498	39.40	22 - 46
14	**0.6474**	0.0335	**64.30**	**50 - 99**
15	**0.6164**	0.0400	**115.20**	**68 - 147**
16	**0.6089**	0.0313	19.70	16 - 23
17	**0.5970**	0.0505	34.40	28 - 44
18	**0.5801**	0.0257	**82.90**	**62 - 98**
19	0.4886	0.0015	21.00	21 - 21
20	0.4838	0.0339	21.50	16 - 28

TABLE 7.10: Results relative to test instances with *high* resources contention level ($level_3$)

Test instances	Hypervolume		$\lvert PF_{approx} \rvert$	
	Avg.	Std.	Avg.	Min - Max
1	**0.8802**	0.0041	**120.70**	**97 - 150**
2	**0.8551**	0.0165	**145.20**	**117 - 158**
3	**0.8507**	0.0109	**163.20**	**148 - 186**
4	**0.8311**	0.0155	30.80	23 - 35
5	**0.7814**	0.0252	**111.10**	**75 - 138**
6	**0.7725**	0.0539	**96.50**	**50 - 148**
7	**0.7667**	0.0129	**102.90**	**67 - 155**
8	**0.7546**	0.0197	**75.90**	**63 - 88**
9	**0.7407**	0.0453	39.60	28 - 57
10	**0.7328**	0.0488	**141.50**	**128 - 160**

Test instances	Hypervolume		$\lvert PF_{approx} \rvert$	
	Avg.	Std.	Avg.	Min - Max
11	**0.7298**	0.0381	**113.70**	**42 - 150**
12	**0.7195**	0.0558	46.40	30 - 60
13	**0.6866**	0.0875	**89.50**	**57 - 109**
14	**0.6832**	0.0337	36.70	28 - 43
15	**0.6700**	0.0617	**173.00**	**146 - 196**
16	**0.6654**	0.0298	**102.00**	**81 - 127**
17	**0.6538**	0.0515	47.90	41 - 54
18	**0.6311**	0.0164	**146.80**	**123 - 186**
19	**0.5723**	0.0361	**144.80**	**131 - 165**
20	**0.5222**	0.0626	**64.80**	**53 - 77**

Let consider Table 7.8, 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. Accordingly, each solution in PF_{approx} associated to these test instances has for the objective $Min(f_6 = \sum_{i=1}^{k} B_i)$ a zero value. For these test instances, only the objectives $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ present a conflict relationship, which induces the reduction of the search space dimension. For test instances with non-zero hypervolume values (i.e. the first part of Table 7.8), we can see that the hypervolume values are greater than 0.5 which mean that the obtained solution sets are well spread in the part of the objective space that has been explored.

The average hypervolume over test instances displayed in Table 7.9 and Table 7.10 are about 0.6915 and 0.7250 respectively. 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 7.9 and 196 in Table 7.10.

Besides, when we inspect the size of PF_{approx} in the three tables, we notice that the number of solutions is different between runs associated to each test instance. This variation is more acute in Table 7.9 and Table 7.10. 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.

Furthermore, we have computed the execution time of the 20 test instances of each resources contention level. Table 7.11 shows the average, the minimum and the maximum execution time 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 execution time values of each level.

We can see in this table that for each level, the difference between the minimum and the maximum execution time values is about $2h\ 26min$, (respectively $2h\ 44min$ and $3h\ 45min$) for $level_1$ (respectively $level_2$ and $level_3$). This execution time variation is caused by the variation of the simulation period between the generated test instances as shown in Table 7.11. In addition, we can observe that the execution time increases when the resources contention level raises. Let consider the maximum execution time values for each resources contention level. We note that even though the simulation periods of $level_1$ ($SP = 27720$) and $level_2$ ($SP = 27720$) are greater than the one of $level_3$ ($SP = 13860$), the maxi-

TABLE 7.11: Execution time computed over all test instances generated for each resource contention level

Resources contention level	Execution time		
	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)

mum execution time relative to $level_3$ is more important[2]. The rationale here is 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 execution time 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.

To conclude, although this experiment does not address explicitly the accuracy of the generated fronts (contrary to Experiment 2.2, where optimal Pareto fronts can be computed), the results in terms of hypervolume and number of solutions show that our method performs effectively the architecture exploration for systems with shared resources by generating a set of promising design trade-offs. For instance, for the studied system instances with high resources contention, we have obtained $HV \in [0.52, 0.88]$ and $|PF_{approx}| \in [30, 173]$.

7.3.5 Experiment 2.4: Impact of the initial design solution choice on the DSE process performance

The initial solution is one among key factors that influence the performance of the MOEA [HGT05]. In all the previous experiments, we run the DSE process by setting the initial solution to the 1-1 assignment solution. The latter corresponds to an extreme design solution involving the maximum number of tasks which promotes the laxity objective $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$. The 1-1 assignment initial solution is quite simple and defined at very low computational cost. However, starting the search from an extreme solution may hinder the algorithm from reaching all solutions in PF_{true}. For that, in Section 4.3.2, we

[2]Logically, for two test instances with the same size (i.e. # functions), the test instance that have larger simulation period will take more time to be processed by DSE process.

investigate another initial solution called preprocessed solution that assigns functions to tasks following a strategy based on dependencies among functions (see Algorithm 3). The preprocessed solution includes fewer number of tasks than the 1-1 assignment solution and is rather oriented towards the blocking time objective $Min(f_6 = \sum_{i=1}^{k} B_i)$.

In the current experiment, we want to investigate the impact of the initial solution choice on the DSE process. We drive a set of experiments aiming at comparing the fronts generated by running the DSE process while setting the initial solution first to the 1-1 assignment solution and secondly to the preprocessed solution. In the sequel of this section, first we give the experiment protocol. Then, we present results and their interpretation.

Experiment protocol and parameters settings

This experiment is performed on the same test instances generated as part of Experiment 2.2 (Section 7.3.3).

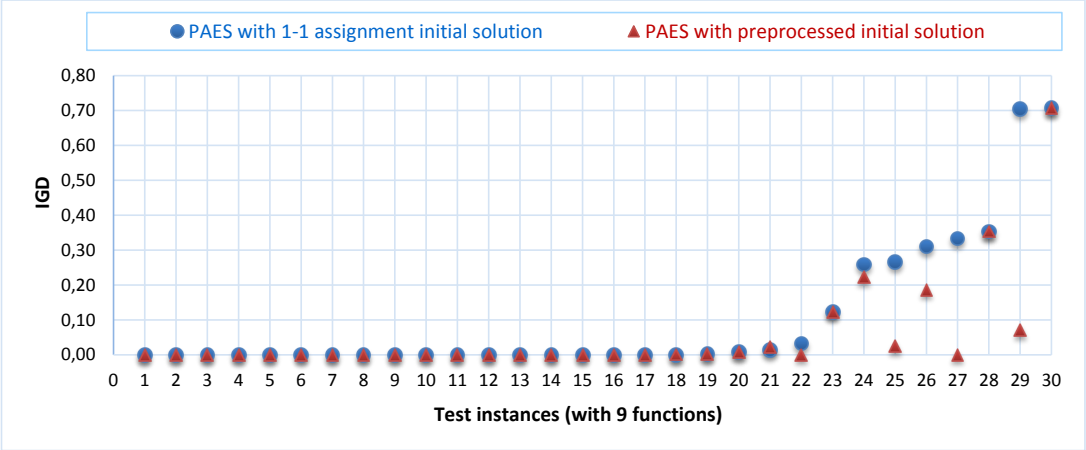
First, we apply the preprocessed initial solution method on each of these test instances in order to generate a preprocessed initial solution. Then, these test instances are proceeded by the DSE process while considering the preprocessed initial solutions. Thereafter, IGD values are computed on the obtained fronts. As for the 1-1 assignment initial solution, we exploit IGD values stemmed from Experiment 2.2. Each IGD value shown in this experiment is the average over 30 independent runs of the DSE process with the considered initial solution. Remind that IGD values are normalized (i.e. IGD $\in [0..\sqrt{2}]$) and smaller values are preferred for this metric.

Results interpretation

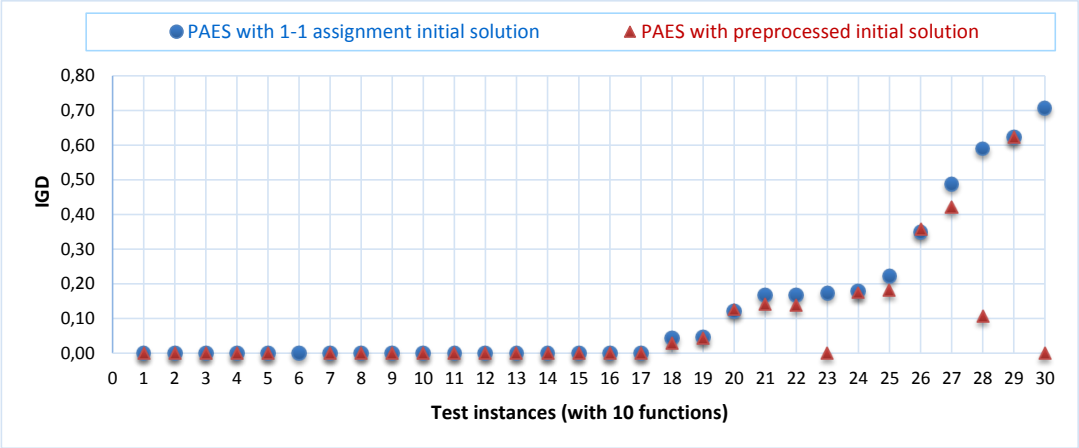
Results of this experiment are illustrated through Figure 7.5 and Table 7.12. IGD values obtained when considering 1-1 assignment initial solution are compared against those obtained when considering the preprocessed initial solution.

Figure 7.5 provides IGD values for 9-functions test instances (Subfigure 7.5a) and 10-functions test instances (Subfigure 7.5b). Both subfigures illustrate IGD values associated to (i) fronts produced when considering the 1-1 assignment initial solution (marked with blue-filled circles) and (ii) fronts produced when considering the preprocessed initial solution (marked with red-filled triangles).

Table 7.12 indicates the number of test instances for which (1) IGD values are the same for both initial solutions, (2) IGD values are degraded (i.e. IGD values obtained when considering the preprocessed initial solution are larger than those relative to the 1-1 assignment solution), and (3) IGD values are enhanced (i.e.



(A) IGD values for 9-functions test instances



(B) IGD values for 10-functions test instances

FIGURE 7.5: IGD values associated to fronts produced by applying the DSE process on test instances while setting the initial solution to (i) the 1-1 assignment solution and then (ii) the preprocessed solution

IGD values relative to preprocessed initial solution are smaller than those relative to the 1-1 assignment solution). Table 7.12 provides also the average IGD degradation and enhancement.

First, in both Figures 7.5a and 7.5b, we can see that test instances having a *zero* IGD value with the 1-1 assignment initial solution maintain also a *zero* IGD value with the preprocessed initial solution.

Besides, by looking at IGD values given in these two figures, we can notice that the DSE process performs better when the initial solution is set to the preprocessed solution rather than considering the 1-1 assignment as initial solution. This is reinforced by the results given in Table 7.12 in terms of the number of test instances for which we record an IGD enhancement (17 test instances) and the

TABLE 7.12: Comparison between IGD values computed by considering the 1-1 assignment initial solution and those associated to the preprocessed initial solution

	Average $(IGD_1 - IGD_2)$	# test instances
Same IGD	0.0	35 (about 58% of test instances)
IGD degradation	−0.0038	7 (about 11% of test instances)
IGD enhancement	0.1871	17 (about 27% of test instances)

IGD_1: IGD relative to 1-1 assignment initial solution
IGD_2: IGD relative to preprocessed initial solution

average IGD enhancement (0.1871) as well. As we can observe, for test instances 22 and 27 in Figure 7.5a and test instances 23 and 30 in Figure 7.5b, IGD values that are non-zero values with the 1-1 assignment initial solution become zero values with the preprocessed initial solution. This means that for these test instances, the DSE process by using the preprocessed initial solution converges to PF_{true} in all runs while it fails (for certain runs among the 30 performed runs) when the initial solution is set to the 1-1 assignment solution.

However, for exactly 7 test instances we perceive a slight degradation of IGD values (by an average of 0.0038) in the case of preprocessed initial solution with regards to the 1-1 assignment initial solution.

We would like to underline that the preprocessed initial solution method fails to generate an output for test instance 6 of Figure 7.5b.

The current experiment shows that the preprocessed initial solution method is efficient to handle small-sizes systems. In order to assess its capability to handle larger (i.e. having more functions) and more complex systems (i.e. subject to high resources contention in addition to the timing constraints) we apply the preprocessed initial solution method on test instances of Experiment 2.1 (Section 7.3.2) and those generated as part of Experiment 2.3 (Section 7.3.4).

Table 7.13 and Table 7.14 exhibit results derived from test instances of Experiment 2.1 and Experiment 2.3 respectively. Each table specifies (1) the number of test instances for which the preprocessed initial solution method succeeds to produce an output (i.e. a feasible solution) and (2) the number of test instances when it fails.

Seeing results in these two tables, we note that the efficiency of the preprocessed initial solution method decreases when the number of the system functions and the resources contention grow. As pointed out in Table 7.13, the rate of test instances that cannot be handled by the preprocessed initial solution method swells from 10% for test instances generated according to the first configuration parameters (see second row of Table 7.13) to 68% for test instances of the second configuration having larger number of functions and more resources con-

TABLE 7.13: Preprocessed initial solution method applied on test instances of Experiment 2.1

Test instances configuration	# test instances associated to the **success** of preprocessing initial solution	# test instances associated to the **failure** of preprocessing initial solution
50 test instances: - 20 functions - 6 resources - rsf = 0.2 - csr $\in$ {0.1, 0.3, 0.5}	45 (90 % of test instances)	5 (10 % of test instances)
50 test instances: - 30 functions - 10 resources - rsf = 0.25 - csr $\in$ {0.1, 0.3, 0.5}	16 (32 % of test instances)	34 (68 % of test instances)

TABLE 7.14: Preprocessed initial solution method applied on test instances of Experiment 2.3

Resources contention levels	# test instances associated to the **success** of preprocessing initial solution	# test instances associated to the **failure** of preprocessing initial solution
$Level_1$	18 (90 % of test instances)	2 (10 % of test instances)
$Level_2$	13 (65 % of test instances)	7 (35 % of test instances)
$Level_3$	6 (30 % of test instances)	14 (70 % of test instances)

tention (see third row of Table 7.13). It is further shown in Table 7.14 that for test instances with the same number of functions (30 functions), the ability of the preprocessed initial solution method to generate a feasible solution falls as resources contention increases.

The failure of the preprocessed initial solution method to produce a feasible solution can be explained as follows. Indeed, according to Algorithm 3, when a large and complex system is given as input, the first determined solution (i.e. before checking the feasibility) will involve a design with few tasks and with high resources contention. Unfortunately, such a design would lead to very narrow tasks laxities and increased blocking-time of tasks, and worse, it may jeopardize the system schedulability. This will lead to an infeasible solution. Additionally, it may be difficult to reproduce a feasible solution through a mutation applied on this infeasible solution. Accordingly, the failure rate of the preprocessed initial solution method raises with acutely constrained systems.

Results of this experiment let us to conclude that a proper choice of the initial solution would improve the accuracy and convergence to optimal results of our design exploration method. Furthermore, these results provide evidence that the method that we propose to preprocess an adequate initial solution for systems with shared resources, is efficient for small-sizes systems. However, the efficiency of that method decreases as resources contention and number of system functions increase.

7.4 Conclusion

In this chapter, we have presented results of experimental studies performed to investigate some aspects stemmed from the addressed problem and assess the effectiveness of our approach.

Experiments are divided into two groups according to the system model configuration. A first experiments group targeted systems with independent functions following Liu & Layland task model. The second experiments group has been achieved for systems with interacting functions through shared resources. All the experiments were performed on synthetically generated problem instances. For each group of experiments we have proposed a problem instance generator dedicated to the system model under consideration.

For independent task systems, the evaluation of the proposed DSE process is twofold. The first experiment (Experiment 1.1) assessed the accuracy of our method by comparing its result against an exhaustive method providing exact results. This experiment was performed on a small size test case (a system with 11 functions), since the exhaustive method can not handle problem instances with large search spaces. It showed that the DSE process is able to converge towards the exact Pareto front for the considered test case. For larger test instances, the second experiment (Experiment 1.2) evaluated the quality of produced Pareto fronts. The obtained results validate the capability of our framework to explore a design space of assignment solutions, and provide sets of promising trade-offs with respect to two objectives, namely $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$.

As for systems with interacting tasks through shared resources, four experiments were carried out. In Experiment 2.1, we performed an empirical study aiming at investigating the correlation between three pairs of objectives. Results of this experiment showed that: two pairs of objectives namely $[Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))]$ and $[Min(f_4 = (H - \sum_{i=1}^{k} L_i))$, $Min(f_6 = \sum_{i=1}^{k} B_i)]$ present a conflicting relationship for more than 70% of the studied test instances whereas for the objectives pair $[Min(f_1 = \#preemptions)$, $Min(f_6 = \sum_{i=1}^{k} B_i)]$ only 46% of test instances present a conflicting relationship.

Besides, Experiment 2.2 was achieved in order to assess the accuracy of the DSE process. The solution sets produced by our method were compared against an exhaustive method results. The experiments for this evaluation were realized on small size test instances. Results of this experiment showed that our 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 solution set (i.e. test instances with low IGD values, see Section 7.3.3). Furthermore, an execution time comparison study pointed out that our DSE process outperforms the exhaustive method by about 74% of an execution time reduction.

Experiment 2.3 was conducted for more complex systems with different resources contention levels in order to assess the quality of produced solution sets. Results of this experiment 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.

Finally, Experiment 2.4 was derived to explore the impact of the initial solution choice on the DSE process performance. This experiment was performed on small size problem instances. The DSE process using the 1-1 assignment solution that biases the search towards the laxity objective was compared against the DSE process while using a preprocessed solution. The latter is built through a method called preprocessed initial solution method that we proposed. This method assigns functions to tasks following a strategy based on dependencies among functions through shared resources. The goal is to guide the search towards a region that promotes also the blocking-time objective. Results of this experiment revealed that running the DSE process by using the preprocessed solution for the initial solution performed better or equally well in terms of IGD values than using the 1-1 assignment initial solution for 85% of the studied test instances. This leads to conclude that spending some computational effort in building an adequate initial solution is encouraged as it helps to enhance the convergence in optimization, thereby improving the efficiency of the design exploration method.

As part of Experiment 2.4, we have assessed the robustness of the preprocessed initial solution method towards more complex systems. This method was run on test instances with larger size and more resources contention. However, through results of experiments, we realized that the efficiency of that method goes down when resources contention and system size increase. For this reason and considering further the low computational cost to build the 1-1 assignment solution, the latter is used, so far, as initial solution for the DSE process.

Conclusion

The work presented in this thesis contributed to verification of constraints and optimization of multiple performance criteria at early design stages of critical real-time and embedded (RTE) systems.

In order to present our contributions in this field, we introduced, in Chapters 1 and 2, the required background knowledge about RTE systems underlying concepts related to their timing verification, development and design. We also presented an overview about multi-objective optimization (MOO) methods that are of great interest when dealing with the exploration of design alternative space regarding multiple conflicting performance criteria. Chapter 3 synthesized the work orientation by highlighting the challenges, the work assumptions, and the proposed contributions, then it described and discussed the related work.

Afterwards, we detailed, in Chapter 4, elements of our solution regarding the automated multi-objective design space exploration (DSE) process for the functions-to-tasks mapping problem. In order to better master the scalability of this DSE process, we presented, in Chapter 5, a parallel implementation of the DSE process combined with a new selection strategy for PAES, the multi-objective evolutionary algorithm used by the DSE process. All the approach elements are structured and developed in a prototype described in Chapter 6. Furthermore, in Chapter 7, we presented several experiments conducted in order to evaluate our proposals and drive some empirical studies.

This last chapter concludes this thesis by recalling the problem statement followed by our key contributions. Then, it provides the main experimental results obtained in this work. Finally, some directions for future work are outlined.

Reminder of the problem statement

This thesis dealt with mapping the functional specification (i.e. function level) of a given RTE system towards an operational design (i.e. RTOS task level), so as to guarantee the optimization of the system performance and its correctness with respect to the timing properties. The system performance criteria are always conflicting: an attempt to enhance one criterion leads to the degradation of others. Accordingly, RTE systems designers are faced with several design decisions/choices and they struggled to make the most suitable decisions in order

to define trade-offs between conflicting performance criteria. They need to explore the design space of mapping alternatives, in order to identify those meeting the timing constraints and answering at best to a trade-off among the conflicting performance criteria. During this exploration process, designers must check the feasibility of each design alternative regarding all requirements. Unfortunately, performing this exploration process manually by enumerating all the possible design alternatives is costly, error-prone, tedious, unmanageable for a human. This is due to the ever-increasing complexity and size of RTE systems (that may include up to several hundreds of functions) leading to a very large design space of mapping alternatives. Indeed, the number of mapping alternatives increases exponentially with the system size (i.e. the number of functions).

Thus, the problem we dealt with in this thesis is identified as a combinatorial multi-objective optimization problem (MOOP) and can be summarized as follows: *How to assign the functional specification of a particular RTE application into a specific execution platform while taking into account the timing constraints and trade-offs among multiple conflicting performance criteria*?

Contributions summary

Before outlining our contributions, we recall the real-time scheduling context and assumptions delineating the scope of our solutions.

In this thesis, we considered hard critical real-time systems where all task deadlines must be met. The proposed architecture exploration method is applied on systems that consist of a set of concurrent tasks interacting through shared resources. We relied on a conventional task model based on the well-known *Liu & Layland* model [LL73]. We assumed periodic synchronous tasks with implicit deadlines running on top of a uniprocessor platform under a preemptive and fixed-priority scheduling policy. We targeted Ravenscar compliant systems, i.e. shared resource accesses are governed by the priority ceiling protocol (PCP) in order to ensure the synchronization of tasks and their mutual exclusion.

To tackle issues raised by design exploration, notably the combinatorial explosion of functions-to-tasks assignment alternatives and the conflicting character of performance criteria, we automated the exploration of assignment alternatives using a multi-objective metaheuristic in the class of Multi-Objective Evolutionary Algorithms (MOEAs), namely the *Pareto Archived Evolution Strategy* (PAES). This kind of methods, including a part of randomness, aims at finding optimal or near-optimal solutions without exploring the whole search space, thereby producing results at reasonable computational cost. Although a metaheuristic method does not guarantee the optimality, it is of great interest when exact methods fail to handle large search spaces and can not find optimal solutions.

From the functional specification of the system under consideration, an initial assignment alternative solution is generated manually, such as the 1-1 assignment solution where each function is assigned to one task. This initial design alternative is given as input to the proposed automatic multi-criteria DSE process, from which the search procedure is launched. The PAES underlying components in terms of encoding of solutions, mutation operator and evaluation of solutions by means of objective functions are designed and customized according to the addressed DSE mapping problem.

Such a MOEA-based approach needs a specific formalization of candidate design solutions to enable their evaluation and their feasibility verification. To do so, we defined a set of rules driving the assignment of functions into tasks. These rules enable the formalization of candidate design alternatives by identifying task set and resource set in terms of composition and timing parameters from their chromosomal representation. The DSE process iterates different steps (evaluation, mutation, archiving and selection) for a certain number of times in order to automatically explore the space of assignment alternatives. It is worth mentioning that design alternatives explored during this DSE process are checked towards a set of feasibility requirements such that the final produced design alternatives enforce the correct system behavior and the timing constraints described in the functional specification. The DSE process produces a Pareto set of design alternatives representing the "best" trade-offs regarding the considered objectives. From these Pareto design alternatives, designers of the system under consideration would choose the most suitable one.

The second part of our contributions is about the scalability and the effectiveness of the DSE process. First, to cope with the scalability issue, we adapted a parallel asynchronous schema namely the well-known *Master-Slave* parallel paradigm to our DSE process. With this coarse-grained approach, multiple candidate design solutions are processed in parallel for checking constraints and evaluating objective functions. Second, we defined the so-called *global selection*, a new selection strategy that aims at improving the search procedure of our DSE process, thereby achieving better overall effectiveness, as compared to the default local selection embedded in PAES. These proposals are particularly suitable for MOO problems with large and variable time objective functions computation and/or constraints verification.

To evaluate our solutions, we implemented a prototype that was embedded in Cheddar, an existing scheduling framework. This prototype implements several functions, each one implementing a part of our solutions. The main benefits of the provided prototype are its reusability and the extensibility of its software artefacts. For instance, it could be reused in different MOO 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 DSE-specific problem with other MOO solvers, etc.

Main experimental results

In order to evaluate our theoretical and technical proposals and investigate some points related to the addressed DSE mapping problem, we conducted different experiments.

In Chapter 5, a set of experiments was devoted to assessing our proposals contributing to the improvement of scalability of the DSE process. These experiments were performed on synthetically generated problem instances. Their results show improvement not only in the execution time but also in the quality of produced Pareto sets when compared to the previous sequential version of the DSE process. For instance, with 4 slave processors, the average speed-up is about 3.5 and the average improvement rate in hypervolume is about 38.7% for problem instances with 100 functions. The improvement captured in the produced Pareto sets quality has been achieved thanks to the global selection strategy that we have proposed.

In Chapter 7, experiments were divided into two groups according to the system model configuration. A first set of experiments focused on systems with independent functions. The second experiment group has been achieved for systems with interacting functions through shared resources. All the experiments were performed on synthetically generated problem instances. For each group of experiments we have proposed a problem instance generator dedicated to the considered system model.

For independent task systems, the evaluation of the proposed DSE process was twofold. The first experiment assessed the accuracy of our method by comparing its results against an exhaustive method results. This experiment was performed on a small size test case presenting small enumerable search space since the exhaustive method can not handle problem instances with large search spaces. Results of this experiment show that the DSE process is able to converge towards the exact Pareto front for the considered test case. For larger test instances (with #functions ≥ 15), the second experiment evaluated the quality of produced Pareto fronts when varying the number of functions. The obtained results validate the capability of our framework to explore a design space of assignment solutions, and to provide a set of promising trade-offs with respect to two objectives, namely $Min(f_1 = \#preemptions)$ and $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$. For example, for test cases with 40 functions, hypervolume[3] values are between 0.5 and 0.7.

For systems with interacting tasks through shared resources, four experiments were achieved. In the first experiment, we performed an empirical study aiming at investigating the correlation between three pairs of objectives. Results of this experiment showed that: two pairs of objectives namely $[Min(f_1 = \#preemptions)$,

[3]Remind that the maximum value of the hypervolume metric is 1.

$Min(f_4 = (H - \sum_{i=1}^{k} L_i))]$ and $[Min(f_4 = (H - \sum_{i=1}^{k} L_i))$, $Min(f_6 = \sum_{i=1}^{k} B_i)]$ present a conflict relationship for more than 70% of the studied test instances whereas for the objectives pair $[Min(f_1 = \#preemptions)$, $Min(f_6 = \sum_{i=1}^{k} B_i)]$ only 46% of test instances present a conflict relationship. Nevertheless, none of the objective pairs presents a support relationship for a major part of the studied test instances, since the best supporting ratio is 23% for the couple $[Min(f_1 = \#preemptions)$, $Min(f_6 = \sum_{i=1}^{k} B_i)]$. Thus, in the following introduced experiments, we have considered the three objectives $Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ and $Min(f_6 = \sum_{i=1}^{k} B_i)$.

Moreover, we assessed the accuracy of the DSE process for systems with interacting tasks through shared resources. A set of experiments was carried out, where results produced by our DSE process were compared against an exhaustive method results. These experiments were realized on small size test instances (i.e. with 10 functions at most). Their results exhibit that our DSE process can find the exact Pareto front for 55% of the studied instances. And for other instances (about 13%) results were very close to optimal solution sets since their IGD values were in the range $[0.0002, 0.05]$. Additionally, an execution time comparison study points out that our DSE process outperforms the exhaustive method by about 74% of an execution time reduction.

For systems with higher number of functions and more resources contention, another evaluation was achieved in order to assess the quality of produced Pareto sets when varying the resources contention level. Results of experiments approved that our method made effectively the design exploration and generated meaningful trade-offs with respect to three objectives, namely $Min(f_1 = \#preemptions)$, $Min(f_4 = (H - \sum_{i=1}^{k} L_i))$ and $Min(f_6 = \sum_{i=1}^{k} B_i)$. For instance, with high resources contention level, the average hypervolume and the average number of non-dominated solutions over 20 test instances were about 0.725 and 100 non-dominated solutions respectively.

Finally, we derived an experiment aiming at investigating the impact of the initial solution choice on the DSE process performance. In this experiment, we compared IGD results of the DSE process by using the 1-1 assignment solution against the same method while using a preprocessed initial solution. Results of this experiment revealed that running the DSE process by using the preprocessed solution for the initial solution performs better or equally well (in terms of IGD values) than using the 1-1 assignment initial solution for 85% of the studied test instances. This leads to conclude that spending some computational effort in building an adequate initial solution is encouraged as it helps to enhance the accuracy/convergence of our DSE process.

Future work

This work can be extended by several means.

First of all, in this thesis we have considered a uniprocessor architecture for the execution platform with synchronous periodic tasks having implicit deadlines. Thus, the next major improvement is to extend the proposed approach in order to take into account characteristics of today's RTE systems: (1) from execution platform point of view such as: multi-core/multi-processor architecture, hierarchy of shared memories, etc (2) and from task model point of view such as: asynchronous and/or sporadic tasks, arbitrary deadlines, mixed criticality, precedence constraints, etc. These extensions on the targeted systems require further investigations regarding the optimization objectives, degree of freedom in the exploration, as well as the schedulability test to check the design alternatives.

When analyzing results of experiments we have noticed that for some test instances our DSE process misses a region of the search space (e.g. see Figure 7.4b). This may due to constraints imposed on these problem instances. Indeed, constraints may create isolated feasible regions, which are unreachable from other feasible regions using the proposed mutation operator. In such constrained search spaces, a single-current solution method (like the one used by PAES) would fail to effectively explore the search space [AM15]. Therefore, it will be interesting to formulate the DSE mapping problem with MOO techniques other than PAES. In this case, elements of our approach proposed in chapter 4 in terms of solutions encoding, search operators, formalization of design alternatives and feasibility checks can be reused when formulating our problem with other MOEAs. These approaches can also be mixed with local search method.

Additionally, in all the performed experiments, we run the DSE process for a fixed number of iterations. However, it should be interesting to investigate some convergence-based termination criterion.

As for the DSE process parallel implementation, we have observed through results of experiments that the speed-up decreases with more processors involved in the parallel computation. This indicates a decrease in the efficiency of the parallel method with the number of parallel processors, which may be due to the parallel approach adopted in our work. Thus, we plan to improve the efficiency of the parallel search model for high number of processors by investigating the solution choice like in [COA11]. In the same regard, another direction to improve the scalability of our DSE process concerns the reduction of the computation time associated to the scheduling simulation. For instance, authors of [GB01] have investigated this issue and attempted to speed up the preprocessing phase of the simulation by adapting it for implementation in parallel environments.

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 [SDT+13, WY16]. 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 [LJCUB09, JCAT14, SDT+13, WY16]. 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 4.3.4 or even other user defined objectives and then non-redundant objectives will be determined during the algorithm progress.

Furthermore, through experiments, we have realized that the initial solution (from which the DSE process starts the search) significantly influences the performance in terms of accuracy/convergence of the DSE process. Hence, we expect to thoroughly investigate the choice of the initial solution in order to improve the performance of our DSE process. This work can be also used for creating the initial population of alternative MOEAs.

Part III

Appendices

A

Experimental Setups and Threats to Validity

This appendix is devoted to discuss/explain the setup of certain empirical parameters in experiments presented in Chapter 5 (Section 5.5) and Chapter 7 (Sections 7.2 and 7.3). We acknowledge that, for some experiments, certain parameters such as # `test instances per experiment`, # `PAES independent runs per problem instance` and # `PAES-iterations` (i.e. the stopping condition of the algorithm) are set to relatively small values which may compromise the statistical relevance of experiments. The rationale behind adopting reduced values of these parameters is closely related to the combinatorial complexity issue of the problem we deal with. In fact, the design exploration problem addressed in this thesis falls into computationally intensive MOO problems.

Despite the adoption of the parallel implementation for the DSE process (proposed to cope with computational costs issue), when achieving experiments we have realized that they require considerable time. We can refer to Table 7.11 that exhibits some values about execution time. For instance, we can see in this table that DSE process could take more than 5 hours to run a fairly complex problem instance (e.g. having 30 functions and high resources contention). In addition, given the significant number of experiments that we have planned either to evaluate our proposals or to perform empirical studies, considering a large number of test instances per experiment (and/or a large number of runs per test instance and/or a large number of PAES-iterations) will hinder us from driving a broad range of experiments in affordable time duration.

While studying the experimental setups, we explored the literature in order to see how experimental parameters are set in research work belonging to our fields of interest (i.e. constrained software design exploration for RTE systems, MOO

problems using evolutionary methods, computationally expensive problems, etc). In Table A.1, we give experimental setups from some related work evaluating optimization frameworks/approaches based on heuristic methods. For these related work, we report the tackled problem, the used optimization technique, the number of problem instances, the number of runs (or trials) for each problem instances and the number of iterations (or generations) when the stopping condition is based on a fixed number of iterations.

TABLE A.1: Experimental setups from related work evaluating optimization frameworks/approaches

Reference	addressed problem	optimization technique	# problem instances or case studies	# independent runs	# iterations (or generations)
Rahmoun et al. [RBP15a]	generic MOO design exploration for RTE systems	NSGA-II	a case study	ignored	100 generations
Sayuti et al. [SI15]	task mapping and priority assignment for RTE NoCs systems	GA	20 test benches	ignored	500 generations
Aleti et al. [AM15]	component deployment and architecture optimization for RTE systems	MOEAs	30 test cases	30 runs	stopping condition not based on # iterations
Wozniak et al. [WMM+13]	task to runnables mapping optimization for AUTOSAR systems	GA	11 test cases generated by replication	ignored	stopping condition not based on # iterations
Koziolek et al. [KKR11]	generic MOO design exploration for RTE systems	MOEAs	2 case studies	10 runs	200 iterations
Li et al. [LEEC11]	generic MOO design exploration for RTE systems	MOEAs	2 case studies	15 runs	500 iterations

By observing the 4^{th} column in Table A.1, we noted that the evaluation of each work was performed through experiments that either rely on a relatively small set of synthetically generated problem instances (at most 30 problem instances) or use one or two case studies. Experiments conducted on a limited number of problem instances that are created based on a sampling process may cause the "threat to validity". The complexity dictated on constrained real-world scientific and engineering MOO problems poses a major challenge that hinders researchers and/or practitioners to cope with this threat to validity.

The random nature of heuristic method may lead to different results in different runs of the same problem instance. This causes another threat to validity. In order to reduce this threat, work dealing with standard optimization techniques (i.e. didn't address a specific optimization problem), usually used 30 independent runs for each problem instance. It is worth noting that experiments achieved to evaluate this kind of research work are often performed on simple and small problems known as "toy" problems. When optimization techniques are applied on larger and more complex and realistic problems, sometimes for simplification reasons, researchers and/or practitioners used a less number of runs for each problem instance. For that reason, in some of our experiments, we restrict the number of independent runs to 5 or 10 runs.

As for the maximum number of iterations used as a stopping condition for the evolutionary method, again in real-world and complex problems, experiments are usually executed with a relatively limited number of maximum iterations (or generations). For example in [SI15, LEEC11], the number of maximum generations is set to 500. In our experiments, the number of PAES-iterations is set to 3000, but when experiments involve a large set of problem instances with reasonable complexity the number of iterations is limited to 2000.

To conclude, although the above mentioned threats make difficult the generalization of the obtained results, the experimental results we provided already demonstrate the feasibility of our approach on significant DSE problems. In addition, the most complex problem instances we considered in experiments represent software architectures of a reasonable complexity in the domain of real-time embedded systems.

B

Publications

International Journal Article

RTS 2018

Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, Mohamed Jmaiel. Multi-Objective Design Exploration Approach for Ravenscar Real-time Systems. *Real-Time Systems*, 54(2):424-483, 2018, Springer.
https : *//doi.org*/10.1007/*s*11241 − 018 − 9299 − 6

International Conference Articles

RSP 2016

Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, Mohamed Jmaiel. 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*, pages 1-7, 2016, New York, NY, USA, ACM.

ICECCS 2015

Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, Mohamed Jmaiel. 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*, pages 1-10, 2015, Gold Coast, Australia, IEEE.

Bibliography

[ABD+95] Neil Audsley, Alan Burns, Rob Davis, Ken Tindell, and Andy Wellings. Real-time system scheduling. In *Predictably Dependable Computing Systems*, pages 41–52. Springer Berlin Heidelberg, 1995.

[ABR+93] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sep 1993.

[AM10] Konstantinos P Anagnostopoulos and Georgios Mamanis. A portfolio optimization model with three objectives and discrete variables. *Computers & Operations Research*, 37(7):1285–1297, 2010.

[AM15] Aldeida Aleti and Irene Moser. Fitness landscape characterisation for constrained software architecture optimisation problems. In *Proceedings of the* 20^{th} *International Conference on Engineering of Complex Computer Systems*, ICECCS '15, pages 11–20, Washington, DC, USA, 2015. IEEE Computer Society.

[AMK98] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 178–189. ACM, 1998.

[AT02] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[AT15] B Annighöfer and F Thielecke. A systems architecting framework for optimal distributed integrated modular avionics architectures. *CEAS Aeronautical Journal*, 6(3):485–496, 2015.

[Bal97] F. Balarin. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer international series in engineering and computer science: VLSI, computer architecture, and digital signal processing. Springer US, 1997.

[Bar03] Sanjoy Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[BB05] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, May 2005.

[BB06] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the* 27^{th} *IEEE International Real-Time Systems Symposium*, pages 159–168. IEEE, 2006.

[BB08] Alan Burns and Sanjoy Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[BBB01] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 59–66. IEEE, 2001.

[BDV04] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the ada ravenscar profile in high integrity systems. *ACM SIGAda Ada Letters*, XXIV(2):1–74, june 2004.

[BFO14] Antoine Bertout, Julien Forget, and Richard Olejnik. Minimizing a real-time task set through task clustering. In *Proceedings of the* 22^{nd} *International Conference on Real-Time Networks and Systems*, pages 23–31. ACM, 2014.

[BHM+05] Marco Bekooij, Rob Hoes, Orlando Moreira, Peter Poplavko, Milan Pastrnak, Bart Mesman, Jan David Mol, Sander Stuijk, Valentin Gheorghita, and Jef Van Meerbergen. Dataflow analysis for real-time embedded multiprocessor system design. In *Dynamic and robust streaming in and between connected consumer-electronic devices*, pages 81–108. Springer, 2005.

[BHP+08] Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro, and Victor Jégu. A framework for distributing real-time functions. In *International Conference on Formal Modeling and Analysis of Timed Systems*, volume 5215, pages 155–169. Springer, 2008.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[Bla04] Benjamin S Blanchard. *System engineering management*. John Wiley & Sons, 2004.

[BLAC05] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency (Series in Computer Science)*. Plenum Publishing Co., 2005.

[BLDN05] C. Bartolini, G. Lipari, and M. Di Natale. 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*, pages 458–467, March 2005.

[BMR90] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the* 11^{th} *Real-Time Systems Symposium*, pages 182–190. IEEE, 1990.

[BNE07] Nicola Beume, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[Bur99] Alan Burns. The ravenscar profile. *ACM SIGAda Ada Letters*, 19(4):49–52, 1999.

[But11] Giorgio Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[BW94] Alan Burns and Andy J. Wellings. Hrt-hood: A structured design method for hard real-time systems. *Real-Time Systems*, 6(1):73–114, 1994.

[BW97] Alan Burns and Andrew J Wellings. *Real-time systems and programming languages*. Harlow : Addison-Wesley, second edition edition, 1997.

[BW07] Alan Burns and Andy Wellings. *Concurrent and Real-Time Programming in Ada*. Cambridge University Press, New York, NY, USA, 2007.

[BZ06] Matthieu Basseur and Eckart Zitzler. Handling uncertainty in indicator-based multiobjective optimization. *International Journal of Computational Intelligence Research*, 2(3):255–272, 2006.

[CB11] Rajan Filomeno Coelho and Philippe Bouillard. Multi-objective reliability-based optimization with stochastic metamodels. *Evolutionary computation*, 19(4):525–560, 2011.

[CDJ10] Carlos A Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. Multi-objective combinatorial optimization: Problematic and context. *Advances in multi-objective nature inspired computing*, 272:1–21, 2010.

[CF95] Christer Carlsson and Robert Fullér. Multiple criteria decision making: The case for interdependence. *Computers & Operations Research*, 22(3):251–260, 1995.

[CGG04] Annie Choquet-Geniet and Emmanuel Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theoretical computer science*, 310(1-3):117–134, 2004.

[Che03] Albert MK Cheng. *Real-time systems: scheduling, analysis, and verification*. John Wiley & Sons, 2003.

[CLVV07] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

[COA11] José C Calvo, Julio Ortega, and Mancia Anguita. Comparison of parallel multi-objective approaches to protein structure prediction. *The Journal of Supercomputing*, 58(2):253–260, 2011.

[CP00a] Erick Cantu-Paz. *Efficient and accurate parallel genetic algorithms*, volume 1. Springer Science & Business Media, 2000.

[CP00b] Antoine Colin and Isabelle Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2-3):249–274, 2000.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[CWC+17] Jin-Hee Cho, Yating Wang, Ray Chen, Kevin S Chan, and Ananthram Swami. A survey on modeling and optimizing multi-objective systems. *IEEE Communications Surveys & Tutorials*, 2017.

[DB11] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):35, 2011.

[Deb01] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[Deb08] Kalyanmoy Deb. Introduction to evolutionary multiobjective optimization. In *Multiobjective Optimization*, pages 59–96. Springer, 2008.

[Der74] Michael L Dertouzos. Control robotics: The procedural control of physical processes. *IFIP Congress*, 1974.

[DH92] K. Driscoll and K. Hoyme. The airplane information management system: an integrated real-time flight-deck control system. In *IEEE Real-Time Systems Symposium*, pages 267–270, 1992.

[DMG97] G De Michell and Rajesh K Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[DS06] Kalyanmoy Deb and D Saxena. 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*, pages 3352–3360, 2006.

[EAP17] Andrea Enrici, Ludovic Apvrille, and Renaud Pacalet. A model-driven engineering methodology to design parallel and distributed embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):34, 2017.

[EG00] Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *Or Spectrum*, 22(4):425–460, 2000.

[EG04] Matthias Ehrgott and Xavier Gandibleux. Approximative solution methods for multiobjective combinatorial optimization. *Top*, 12(1):1–63, 2004.

[EGP16] Matthias Ehrgott, Xavier Gandibleux, and Anthony Przybylski. *Exact Methods for Multi-Objective Combinatorial Optimisation*, pages 817–850. Springer New York, 2016.

[FG12] Peter H Feiler and David P Gluch. *Model-based engineering with AADL: An Introduction to the SAE architecture analysis & design language*. Addison-Wesley, 2012.

[FKTZ05] Carlos M Fonseca, Joshua D Knowles, Lothar Thiele, and Eckart Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Proceedings of the* 3^{rd} *International Conference on Evolutionary Multi-Criterion Optimization*, volume 216, page 240, 2005.

[FMB+09] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. Autosar–a worldwide standard is on the road. In *the* 14^{th} *International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, 2009.

[FSP+14] Christian Fotsing, Frank Singhoff, Alain Plantec, Vincent Gaudel, Stéphane Rubini, Shuai Li, Hai Nam Tran, Laurent Lemarchand, Pierre Dissaux, and Jérôme Legrand. Cheddar architecture description language. *Lab-STICC technical report.*, 2014.

[FSV99] Alberto Ferrari and Alberto Sangiovanni-Vincentelli. System design: Traditional concepts and new paradigms. In *Computer Design, 1999.(ICCD'99) International Conference on*, pages 2–12. IEEE, 1999.

[GB01] Joël Goossens and Sanjoy Baruah. Multiprocessor preprocessing algorithms for uniprocessor on-line scheduling. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 219–226. IEEE, 2001.

[GC97] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. John Wiley & Sons, 1997.

[GF00] Xavier Gandibleux and Arnaud Freville. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: the two objectives case. *Journal of Heuristics*, 6(3):361–383, 2000.

[GGCG16] Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Systems*, 52(6):808–832, 2016.

[GH99] Tomas Gal and Thomas Hanne. Consequences of dropping nonessential objectives for the application of mcdm methods. *European Journal of Operational Research*, 119(2):373–378, 1999.

[GM01] Joel Goossens and Christophe Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the Real-Time Embedded Systems*, 2001.

[GMCH07] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. *European Journal of Operational Research*, 180(1):116–148, 2007.

[GSP+11] Vincent Gaudel, Frank Singhoff, Alain Plantec, Stéphane Rubini, Pierre Dissaux, and Jérôme Legrand. An ada design pattern recognition tool for aadl performance analysis. In *ACM SIGAda Ada Letters*, volume 31, pages 61–68. ACM, 2011.

[GTT02] Sébastien Gérard, François Terrier, and Yann Tanguy. Using the model paradigm for real-time systems development: Accord/uml. *Advances in object-oriented information systems*, pages 260–269, 2002.

[HCFDC09] Eduardo R Hruschka, Ricardo José Gabrielli Barreto Campello, Alex Alves Freitas, and AC Ponce Leon F De Carvalho. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics Part C*, 39(2):133–155, 2009.

[Hea02] Steve Heath. *Embedded systems design*. Newnes, 2002.

[HGT05] Christian Haubelt, Jürgen Gamenik, and Jürgen Teich. Initial population construction for convergence improvement of moeas. In *Proceedings of the* 3^{rd} *International Conference on Evolutionary Multi-Criterion Optimization*, EMO'05, pages 191–205. Springer-Verlag, 2005.

[JC09] Antonio Jaimes and Carlos Coello. Applications of parallel platforms and models in evolutionary multi-objective optimization. *Biologically-inspired optimisation methods*, pages 23–49, 2009.

[JCAT14] Antonio López Jaimes, Carlos A Coello Coello, Hernán Aguirre, and Kiyoshi Tanaka. Objective space partitioning using conflict information for solving many-objective problems. *Information Sciences*, 268:305–327, 2014.

[JP86] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[KC99] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages 98–105. IEEE, 1999.

[KC00a] Joshua D. Knowles and David W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[KC00b] Joshua D Knowles and David W Corne. M-paes: A memetic algorithm for multiobjective optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 325–332. IEEE, 2000.

[KDVVDW97] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter Van Der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, pages 338–349. IEEE, 1997.

[KDWV02] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pages 18–37. Springer-Verlag, 2002.

[KKR11] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. 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*, pages 33–42. ACM, 2011.

[KM91] T-W Kuo and Aloysius K Mok. Load adjustment in adaptive real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 160–170. IEEE, 1991.

[KNRSV00] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1523–1543, 2000.

[Kop97] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer US, 1997.

[KTJ06] Rakesh Kumar, Dean M Tullsen, and Norman P Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the* 15^{th} *international conference on Parallel architectures and compilation techniques*, pages 23–32. ACM, 2006.

[LA15] Francisco Luna and Enrique Alba. Parallel multiobjective evolutionary algorithms. In *Springer Handbook of Computational Intelligence*, pages 1017–1031. Springer, 2015.

[LEEC11] Rui Li, Ramin Etemaadi, Michael TM Emmerich, and Michel RV Chaudron. An evolutionary multiobjective optimization approach to component-based software architecture design. In *IEEE Congress on Evolutionary Computation*, pages 432–439. IEEE, 2011.

[LHG13] Juan C López, Román Hermida, and Walter Geisselhardt. *Advanced techniques for embedded systems design and test*. Springer Science & Business Media, 2013.

[LJCCC08] Antonio López Jaimes, Carlos A Coello Coello, and Debrup Chakraborty. Objective reduction using a feature selection technique. In *Proceedings of the* 10^{th} *Annual Conference on Genetic and Evolutionary Computation*, pages 673–680. ACM, 2008.

[LJCUB09] Antonio López Jaimes, Carlos A. Coello Coello, and Jesús E. Urías Barrientos. *Online Objective Reduction to Deal with Many-Objective Problems*, pages 423–437. Springer Berlin Heidelberg, April 2009.

[LL73] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, January 1973.

[LLP+09] Rongshen Long, Hong Li, Wei Peng, Yi Zhang, and Minde Zhao. An approach to optimize intra-ecu communication based on mapping of autosar runnable entities. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages 138–143. IEEE, 2009.

[LLS07] Insup Lee, Joseph YT Leung, and Sang H Son. *Handbook of real-time and embedded systems*. CRC Press, 2007.

[LM80] Joseph Y-T Leung and ML Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115–118, 1980.

[LNA06] Francisco Luna, Antonio J Nebro, and Enrique Alba. Parallel evolutionary multiobjective optimization. In *Parallel Evolutionary Computations*, pages 33–56. Springer, 2006.

[LSD89] J. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *IEEE Real Time Systems Symposium*, pages 166–171, Dec 1989.

[LT99] Panta Lučic and Dušan Teodorovic. Simulated annealing for the multi-objective aircrew rostering problem. *Transportation Research Part A: Policy and Practice*, 33(1):19–45, 1999.

[LVDWVD01] Paul Lieverse, Pieter Van Der Wolf, Kees Vissers, and Ed Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 29(3):197–207, 2001.

[LW82] Joseph Y-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[LW08] Y. Li and C. Wen. Task construction with temporal consistency for embedded control software design. In *International Symposium on Computer Science and Computational Technology*, volume 1, pages 76–79, Dec 2008.

[MJH13] Hamid Masoud, Saeed Jalili, and Seyed Mohammad Hossein Hasheminejad. Dynamic clustering using combinatorial particle swarm optimization. *Applied intelligence*, 38(3):289–314, 2013.

[MKT09] Andreas Menychtas, Dimosthenis Kyriazis, and Konstantinos Tserpes. Real-time reconfiguration for guaranteeing qos provisioning levels in grid environments. *Future Generation Computer Systems*, 25(7):779–784, 2009.

[MLF08] Hugo Macedo, Peter Larsen, and John Fitzgerald. Incremental development of a distributed real-time model of a cardiac pacing system using vdm. *FM 2008: Formal Methods*, pages 181–197, 2008.

[MNBSL12] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multisource software on multicore automotive ecuscombining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, 2012.

[Mok83] Aloysius K. Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD dissertation, Massachusetts Institute of Technology, 1983.

[MPAI16] Florin Maticu, Paul Pop, Christian Axbrink, and Mafijul Islam. *Automatic Functionality Assignment to AUTOSAR Multicore Distributed Architectures*. Society of Automotive Engineers, Incorporated, 2016.

[MSH11] John W. McCormick, Frank Singhoff, and Jrme Hugues. *Building parallel, embedded, and real-time applications with Ada*, volume 1. Cambridge University Press, 2011.

[MTPG11] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum: a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[MWTP+13] Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, Chokri Mraidha, Marco Di Natale, Haibo Zeng, Jean-Philippe Babau, Laurent Lemarchand, and Sébastien Gerard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. *ACM SIGPLAN Notices*, 48(5):121–132, 2013.

[NIS02] *The Economic Impact of Inadequate Infrastructure for Software Testing*. Number Planning Report 02-3. National Institute Of Standards & Technology, May 2002.

[NLTA05] A.J. Nebro, F. Luna, E.-G. Talbi, and E. Alba. *Parallel Multiobjective Optimization*, pages 371–394. John Wiley & Sons, Inc., 2005.

[OBM14] Melih Ozlen, Benjamin A Burton, and Cameron AG MacRae. Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 160(2):470–482, 2014.

[OJS03] Tatsuya Okabe, Yaochu Jin, and Bernhard Sendhoff. A critical survey of performance indices for multi-objective optimisation.

In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 878–885. IEEE, 2003.

[OMG08] OMG. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems. available at www.omgmarte.org/Documents/Specifications/08-06-09.pdf, June 2008.

[Osy78] Andrzej Osyczka. An approach to multicriterion optimization problems for engineering design. *Computer Methods in Applied Mechanics and Engineering*, 15(3):309–333, 1978.

[Ouh13] Yassine Ouhammou. *Model-based framework for using advanced scheduling theory in real-time systems design*. Phd thesis, ISAE, Toulouse, France, 2013.

[PF07] Robin C. Purshouse and Peter J. Fleming. On the evolutionary optimization of many conflicting objectives. *IEEE Transactions on Evolutionary Computation*, 11(6):770–784, 2007.

[PFB+11] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011.

[PH98] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[Pin04] Alessandro Pinto. *Metropolis design guidelines*. Electronics Research Laboratory, College of Engineering, University of California, 2004.

[RBP15a] S. Rahmoun, E. Borde, and L. Pautet. Multi-objectives refinement of aadl models for the synthesis embedded systems (muramses). In *Proceedings of the* 20^{th} *International Conference on Engineering of Complex Computer Systems*, pages 21–30, Dec 2015.

[RBP15b] Smail Rahmoun, Etienne Borde, and Laurent Pautet. Automatic selection and composition of model transformations alternatives using evolutionary algorithms. In *Proceedings of the* 9^{th} *European Conference on Software Architecture Workshops*, page 25. ACM, 2015.

[Rot64] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, May 1964.

[RS04] Christine Rochange and Pascal Sainrat. Vers une prédictibilité temporelle des processeurs haute-performance. *RTS Embedded Systems*, 2004.

[RSC06] Margarita Reyes-Sierra and CA Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

[RVLB15] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *Computing Conference (CLEI), 2015 Latin American*, pages 1–11. IEEE, 2015.

[SAA+04] Lui Sha, Tarek Abdelzaher, Karl-Erik Arzen, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2-3):101–155, Nov-Dec, 2004.

[SCCM15] Salah Eddine Saidi, Sylvain Cotard, Khaled Chaaban, and Kevin Marteil. An ilp approach for mapping autosar runnables on multi-core architectures. In *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, page 6. ACM, 2015.

[Sch06] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, Feb 2006.

[SDT+13] Dhish Kumar Saxena, João A Duro, Ashutosh Tiwari, Kalyanmoy Deb, and Qingfu Zhang. Objective reduction in many-objective optimization: Linear and nonlinear algorithms. *IEEE Transactions on Evolutionary Computation*, 17(1):77–99, 2013.

[SGNJ08] Anna Syberfeldt, Henrik Grimm, Amos Ng, and Robert I John. A parallel surrogate-assisted multi-objective evolutionary algorithm for computationally expensive optimization problems. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3177–3184. IEEE, 2008.

[SI15] M Norazizi Sham Mohd Sayuti and Leandro Soares Indrusiak. Simultaneous optimisation of task mapping and priority assignment for real-time embedded nocs. In *The 23rd Euromicro International Conference onParallel, Distributed and Network-Based Processing (PDP)*, pages 692–695. IEEE, 2015.

[SKW00] Manas Saksena, Panagiota Karvelas, and Yun Wang. Automatic synthesis of multi-tasking implementations from real-time object-oriented models. In *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computin (ISORC)*, pages 360–367. IEEE, 2000.

[SLNM04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: A flexible real time scheduling framework. *Ada Lett.*, XXIV(4):1–8, November 2004.

[SLS95] J.K. Strosnider, J.P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[SLS05] Françoise Simonot-Lion and Ye-Qiong Song. Design and validation process of in-vehicle embedded electronic systems. In Richard Zurawski, editor, *The Embedded Systems Handbook*. CRC Press - Taylor&Francis, 2005.

[SPR+15] Frank Singhoff, Alain Plantec, Stéphane Rubini, Hai-Nam Tran, Vincent Gaudel, Jalil Boukhobza, Laurent Lemarchand, Shuai Li, Etienne Borde, Laurent Pautet, et al. Teaching real-time scheduling analysis with cheddar. In *9ème édition de lEcole dEté Temps Réel*, 2015.

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, Jun, 1989.

[Sta88] John A Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.

[SV02] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, 2002.

[SVDN07] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 40(10), 2007.

[TDB+14] S Tucker Taft, Robert A Duff, Randall L Brukardt, Erhard Ploedereder, Pascal Leroy, and Edmond Schonberg. *Ada 2012 reference manual. Language and standard libraries: International standard ISO/IEC 8652/2012 (E)*, volume 8339. Springer, 2014.

[Tei12] Jürgen Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1411–1430, 2012.

[TKK+98] Jorma Taramaa, Munish Khurana, Pasi Kuvaja, Jari Lehtonen, Markku Oivo, and Veikko Seppanen. Product-based software process improvement for embedded systems. In *Proceedings of the* 24^{th} *Euromicro Conference*, volume 2, pages 905–912. IEEE, 1998.

[TMO+08] El-Ghazali Talbi, Sanaz Mostaghim, Tatsuya Okabe, Hisao Ishibuchi, Günter Rudolph, and Carlos A Coello Coello. Parallel approaches for multiobjective optimization. In *Multiobjective Optimization*, pages 349–372. Springer, 2008.

[TRA17] Hai Nam TRAN. *Cache memory aware priority assignment and scheduling simulation of real-time embedded systems*. PhD dissertation, Bretagne Occidentale University, 2017.

[UT94] Ekunda Lukata Ulungu and Jacques Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1994.

[UTFT99] EL Ulungu, JFPH Teghem, PH Fortemps, and D Tuyttens. Mosa method: a tool for solving multiobjective combinatorial optimization problems. *Journal of multicriteria decision analysis*, 8(4):221, 1999.

[VVL98a] David A Van Veldhuizen and Gary B Lamont. Evolutionary computation and convergence to a pareto front. In *Late breaking papers at the genetic programming 1998 conference*, pages 221–228, 1998.

[VVL98b] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer, 1998.

[WMM+13] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and Sébastien Gerard. An optimization approach for the synthesis of autosar architectures. In *The* 18^{th} *IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–10. IEEE, 2013.

[Wol03] Wayne Wolf. A decade of hardware/software codesign. *Computer*, 36(4):38–43, 2003.

[WS06] Shige Wang and Kang G Shin. Task construction for model-based design of embedded control software. *IEEE Transactions on Software Engineering*, 32(4):254–264, 2006.

[WY93] Peter H Westfall and S Stanley Young. *Resampling-based multiple testing: Examples and methods for p-value adjustment*, volume 279. John Wiley & Sons, 1993.

[WY16] Handing Wang and Xin Yao. Objective reduction based on nonlinear correlation information entropy. *Soft Computing*, 20(6):2393–2407, 2016.

[XP00] Jia Xu and David Lorge Parnas. Priority scheduling versus pre-run-time scheduling. *Real-Time Systems*, 18(1):7–23, 2000.

[ZG11] Ming Zhang and Zonghua Gu. Optimization issues in mapping autosar components to distributed multithreaded implementations. In *The* 22^{nd} *IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 23–29. IEEE, 2011.

[Zit01] Eckart Zitzler. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. *EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, 2001.

[ZT98] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithmsa comparative case study. In *International Conference on Parallel Problem Solving from Nature*, pages 292–301. Springer, 1998.

[ZTL+03] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.

[ZZZ+08] Qingfu Zhang, Aimin Zhou, Shizheng Zhao, Ponnuthurai Nagaratnam Suganthan, Wudong Liu, and Santosh Tiwari. 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, 2008.

République Tunisienne
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique
et de la Technologie

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Ecole Doctorale
Sciences et Technologies

Thèse de *DOCTORAT*
Ingénierie des Systèmes Informatiques
N° d'ordre: 68–17

Multi-Objective Optimization and Design Space Exploration of Critical Real-Time Systems

Rahma BOUAZIZ FRIKHA

الخلاصة: النمو المستمر في حجم وتعقيد الأنظمة الآنية بالغة الأهمية يؤدى الى عدد كبير من التصاميم الممكنة، الأمر الذي يشكل تحديا كبيرا للمصممين لاتخاذ أفضل القرارات عند التصميم. يجب على هذه التصاميم أن تلبّي مجموعة من القيود الغير وظيفية للأنظمة مع الحرص على تحقيق التوازن بين العديد من معايير الأداء المتضاربة. نقترح في هذه الاطروحة، عملية لاستكشاف التصاميم القابلة للتنفيذ والتي تبرز أفضل توفيق بين معايير الأداء. لتحقيق هذا، قمنا بتوظيف تقنية الاستخدام الأمثل متعدد الأهداف والقائمة على الخوارزميات التطورية. كما قمنا بإرساء مجموعة من القواعد لتحديد مؤشرات عناصر التصاميم. وبالإضافة إلى ذلك، تطرّقنا إلى معالجة رفع قابلية وكفاءة عملية استكشاف التصاميم باقتراح صياغة لهذه العملية قائمة على أساس البرمجة المتوازية.

Résumé : La croissance de la taille et de la complexité des systèmes temps-réel critiques conduit à un nombre important d'architectures possibles ce qui pose un défi majeur pour les concepteurs lors de la prise des décisions de conception. Ces architectures doivent vérifier les contraintes non-fonctionnelles du système tout en tenant en compte différents critères de performances généralement orthogonaux. Nous proposons dans cette thèse, un processus d'exploration architecturale qui fournit un ensemble d'architectures réalisables et répondant au mieux aux compromis entre les critères de performance. Pour ce faire, nous formulons le problème à l'aide d'une technique d'optimisation multi-objectif basée sur les algorithmes évolutifs. Afin de formaliser les alternatives d'architectures explorées pendant le processus, un ensemble de règles est défini permettant de déterminer les paramètres des entités de l'architecture. En outre, nous abordons le problème du passage à l'échelle et l'efficacité du processus d'exploration en l'adaptant pour une implémentation dans des environnements parallèles.

Abstract: The increasing complexity and scale of critical real-time systems leads to a large number of possible architectures which poses a major challenge for designers when making design decisions. These architectures must meet the system non-functional constraints while balancing multiple conflicting performance criteria. We propose in this thesis, a design space exploration process that provides a set of feasible architectures answering at best to trade-offs between performance criteria. To do so, we formulate the problem using multi-objective optimization technique based on evolutionary algorithms. In order to formalize design alternatives explored during the process, a set of rules are defined allowing to compute parameters of the design entities. Furthermore, we address the scalability and the effectiveness of design exploration process by adapting it for implementation in parallel environments.

المفاتيح: الأنظمة الآنية بالغة الأهمية، استكشاف فضاء التصاميم، الاستخدام الأمثل متعدد الأهداف، الخوارزميات التطورية، القيود الزمنية، تقييم الجدول الزمني للمهام.

Mots clés: Systèmes temps-réel critiques, Exploration de l'espace des architectures, Optimisation multi-objective, Algorithmes évolutif, Contraintes temporelles, analyse d'ordonnancement

Key-words: Critical real-time systems, Design space exploration, Multi-objective optimization, Evolutionary algorithms, Timing constraints, Scheduling analysis