COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université de Bretagne Occidentale

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE

ECOLE DOCTORALE N° 644
Mathématiques et Sciences et Technologies
de l'Information et de la Communication en Bretagne Océane
Spécialité : *Informatique et architectures numériques*

Par
Ill-ham ATCHADAM

Exploration d'architectures logicielles pour les systèmes critiques partitionnés sécurisés

Thèse présentée et soutenue à Brest, le 12/01/2023
Unité de recherche : Lab-STICC UMR CNRS 6285

Rapporteurs avant soutenance :

Liliana CUCU-GROSJEAN	Directrice de recherche, INRIA
Jérôme HUGUES	Senior Researcher, Carnegie Mellon University

Composition du Jury :

Président :	Emmanuel GROLLEAU	Professeur des Universités, ENSMA
Examinateurs :	Liliana CUCU-GROSJEAN	Directrice de recherche, INRIA
	Jérôme HUGUES	Senior Researcher, Carnegie Mellon University
	Antonio CASIMIRO	Associate Professor, Université de Lisbonne
	Yvon KERMARREC	Professeur, IMT Atlantique
Dir. de thèse :	Frank SINGHOFF	Professeur des Universités, Université de Bretagne Occidentale
Encadrant :	Laurent LEMARCHAND	Maître de conférences, Université de Bretagne Occidentale
Encadrant :	Hai Nam TRAN	Maître de conférences, Université de Bretagne Occidentale

Remerciements

Je souhaite exprimer ma gratitude envers tous ceux qui ont contribué à la réalisation de ma thèse.

Tout d'abord, je remercie les membres du jury pour leur temps, leur expertise et leur évaluation minutieuse de mon travail. Je remercie particulièrement Pr. Emmanuel Grolleau pour avoir accepté d'être le président du jury. Je tiens à remercier Dr Liliana Cucu-Grosjean et Senior Researcher Jérôme Hugues pour le temps qu'ils ont consacré à la relecture de ma thèse et leur contribution en tant que rapporteurs. J'exprime également ma gratitude envers Dr. Antonio Casimiro et Pr. Yvon Kermarrec pour avoir accepté d'examiner ma thèse.

Je tiens à exprimer toute ma gratitude et ma reconnaissance envers mon directeur de thèse, Pr. Frank Singhoff, et mes encadrants de thèse, Dr. Laurent Lemarchand et Dr. Hai Nam Tran, pour leur implication, leur patience, leur expertise et leur encouragement constant. Votre investissement personnel, votre expertise et votre soutien ont été inestimables tout au long de ce processus et ont constitué une source de motivation. Je suis reconnaissante de la chance que j'ai eue de travailler avec vous et j'ai énormément appris à vos côtés.

Je remercie tous mes collègues du Lab-STICC et de l'Université de Bretagne Occidentale, en particulier Bismark, Camélia, Grace, Julien, Libey, Morgan, Aymeric, Mayssa et Mourad. Je suis également reconnaissante envers les personnes avec lesquelles j'ai eu l'occasion de travailler : K. Bigou, D. Massé, J. Rivière, V. Marc, S. Rubini, P. Le Parc, et P. Ballet.

Je tiens à remercier tous mes amis qui ont été à mes côtés tout au long de ce parcours, pour leurs conseils et leur soutien moral. En particulier, Camara Sekou Oumar, Bouchra Ettahiri et Bachiratou Sahm pour leur aide lors de la préparation de ma soutenance.

Enfin, je tiens à exprimer ma gratitude envers ma famille pour leur amour indéfectible et leur soutien inconditionnel. En particulier, mes parents, mes frères Manaf et Alim qui n'ont jamais cessé de croire en moi. Vos encouragements et votre soutien moral ont été une source d'inspiration pour moi et m'ont aidé à atteindre ce jalon malgré les défis auxquels j'ai été confrontée.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

Real-time systems (RTS) are widely present around us in our daily life. They are widely integrated into devices of different domains from commonly used devices such as cellphones, car navigation, multimedia applications to critical industrial devices such as control command systems in aircraft, automotive systems, and robots. RTS are computing systems whose correctness depends not only on the logical results of the computation but also on the time at which the results are produced [12]. The results processed by RTS must not only be correct but also meet their timing constraints.

An RTS is an application or a set of applications made of tasks that correspond to execution units. Tasks are then subject to timing constraints. In RTS, tasks can be classified based on the criticality of the consequences that may occur if their timing constraints are not respected. Therefore, there are hard deadline tasks and soft deadlines tasks [13, 14]. Hard deadline tasks have stringent timing constraints that must be respected in all situations. Soft deadline tasks are more flexible and tolerate some delay.

With the evolution of technologies, RTS need to integrate more functionalities to satisfy as much as possible the users [15]. Historically in avionic domains, systems have been made based on federated architectures [16] where each function has its own dedicated computing system. In these systems, functions were isolated from each other with limited data exchange. With this architecture, a fault occurring in a function has fewer chances to affect another function on a different computing system. Modern avionics require more services (e.g. entertainment services), safety (e.g. at least a duplication of systems), and smartness (e.g. precise flight management systems, health management systems, smart sensors, and

actuators) [15]. This evolution leads to the integration of more functions. Then assigning each function to a dedicated computing system is no longer efficient since it implies an increase in the height, volume, installation, maintenance cost and power consumption of the systems. In [17], the authors shown that avionic software size is multipled by 2 very 4 years. Thus the proposal of integrated modular avionics (IMA) architecture [18] was a solution to the incompatibility of federated systems to the requirements of modern avionic systems. IMA architecture proposes computing resource sharing among different functions instead of dedicating a computing resource to each function as in the federated architecture.

Resource sharing solves the above-mentioned problems raised by the federated architecture, but it comes with some challenges. A fault occurring in a function can easily affect other functions sharing the same computing resource without a proper isolation. Thus, IMA architecture proposed time and space partitioning.

Time and space partitioned (TSP) systems allow the integration of applications with different criticality levels and potentially from different providers on the same shared banalized execution platform. In TSP systems, applications are assigned to partitions [15]. A partition is a logical software unit defining a boundary of isolation. TSP systems guarantee space and timing isolation between partitions. Space isolation may be brought by memory protection between partitions while timing isolation may be enforced by partition scheduling. These features contribute to master the growing complexity both in size and legacy of avionic software.

Although TSP concepts were first adopted by avionic stakeholders with the IMA architecture, they are gaining various domains such as aerospace [19] and railway [20]. For example, due to the similarity between the avionic and spacecraft domain, researchers such as authors of [19] address the integration of TSP in spacecraft. It comes as a solution to the complexity of the spacecraft onboard software due to the increasing number of mission functions implemented in software. In the railway domain, [20] investigates the optimization of partition-based distributed RTS on a real railway signaling application.

1.2 Problem statement

In TSP systems, tasks have to be assigned to partitions, their design implies deciding on how to do such assignment. This leads to multiple possibilities of tasks to partitions assignments. The number of options increases exponentially with the number of tasks, and partitions. It is a typical case of combinatorial explosion. Changing the tasks to partitions assignment has an impact on the schedulability of the system. It can make some tasks to miss their deadline. If a hard deadlines task misses its deadline, then the associated tasks to parti-

tions assignment is considered not feasible. Therefore, when designing a TSP system, tasks to partitions assignment and the respect of timing constraints of hard deadline tasks are important challenges to investigate.

As TSP systems may host an increasing number of applications provided by different stakeholders with a significant level of legacy, it then increases the probability of corrupted or malicious software deployment. There exists a multitude of attack and threat models for cyber-physical systems [17, 21]. For example, Man-in-the-Middle attacks [22] can threaten communications in and between partitions. Data can be intercepted during application communications which results in either confidentiality violations by disclosure of sensitive information or integrity violations by data alteration.

A standard approach to protect against attacks on confidentiality is the use of encryption with a symmetric private key [23]. It guarantees that the content of a message is intelligible only to the actual sender and receiver. To secure the integrity of communications, one can use message by a third party authentication codes (MACs) to make certain that a message has not been tampered and has been sent by the actual sender [22].

However, ensuring data confidentiality and integrity with the use of encryption and MACs incurs a significant computation overhead on banalized hardware. This overhead may impact the system schedulability. We propose that if TSP systems are made of hard deadline and soft deadline tasks, security can be optimized as much as possible by tolerating some missed deadlines of soft deadline tasks. It results in a problem of assigning tasks to partitions while optimizing schedulability and security, which is an NP-hard combinatorial problem with 2 conflicting objective functions: schedulability and security.

1.3 Contribution

In this thesis, we investigate the conflict aspect between schedulability and security TSP systems when assigning applications to partitions. Schedulability requires that there is no violation of timing constraints. Enforcing confidentiality and/or integrity of communications between applications introduces overhead and affects schedulability.

We propose a design space exploration process to address the combinatorial problem raised between schedulability and security of TSP systems. Our approach is based on Multi-Objective Evolutionary Algorithms (MOEAs), especially the Pareto Archived Evolution Strategy (PAES) [24] which is a metaheuristic that we have adapted for DSE problems with multiple and conflicting objectives functions. It helps to explore the search space to find an approximate set of optimal solutions in a suitable time for large-scale problems that would be time-consuming with an

exact method (i.e. exhaustive method). Our DSE approach proposes to explore the search space of TSP while investigating tasks and partitions assignment and communications security.

1.3.1 A PAES based DSE approach for unicore execution plateforms

The PAES is characterized by different operators that have to be defined according to the specific addressed problem. Our approach proposes an adaptation of PAES to jointly investigate schedulability and security.

First, it is important to ensure the feasibility of the proposed solutions. Then feasibility tests are performed to check the validity of the solutions.

Second, in order to find the best solutions, each solution explored has to be evaluated to be compared to others. Evaluations have to be performed to determine the solution's fitness toward the objectives. Feasibility tests and solutions fitness evaluations are performed through analysis. Our analyses are based on schedulability and security analysis. Our schedulability analysis proposes to identify hard and/or soft deadline tasks that missed their deadlines through scheduling simulations proposed in Cheddar [25] an open-source scheduling analyzer. For security analysis, we propose to implement the Bell-La Padulla (BLP) and Biba security rules to identify respectively confidentiality and integrity vulnerabilities. For this purpose, we integrated into the Cheddar tool, the security architecture Multiple Independent Levels of Security (MILS) that helps us to model our TSP systems with the Cheddar tool not only according to scheduling parameters (e.g. worst-case execution time, deadlines of tasks) but also to security parameters (e.g. confidentiality, integrity levels of tasks).

Third, the search space is explored through the generation of candidate solutions. Then is important to define how to proceed with the generation. We propose to explore the design space of TSP systems with different levels of granularity by three mutation algorithms coupled with a multi-objective meta-heuristic.

We assume that applications are composed of tasks. We start by the assignments of tasks to partitions. As this mutation algorithm leads to investigate a large design space, we propose a second mutation algorithm that consists of investigating groups of tasks constituting an application to partitions assignment. This second approach presents a less degree of freedom. We then propose a third mutation algorithm that consists of refining the results obtained at the application level (i.e. second algorithm) by applying on them a mutation algorithm at task granularity (i.e. first algorithm).

With each mutation algorithm, we evaluate four different means to implement security features in TSP systems.

1.3.2 Extension of the DSE approach to Multicore execution platforms

We show the extensibility of our DSE by applying it in another context. Then first, we investigate the impact of Multicore execution platforms on safe and secure TSP systems while considering not only tasks to partitions assignment but also tasks to cores assignment.

Indeed, TSP systems are safe from fault propagation from one partition to another through partitioning. However, They are still exposed to failure concerning the availability of tasks and partitions. Safety is a challenge to consider when designing a TSP. It can be addressed through active redundancy which means replication of tasks and partitions. Then as security, safety implies overheads that impact schedulability. Thus we proposed an extension of our DSE approach to Multicore execution platforms well-known to increase computation capability.

Second, we proposed a method to improve the diversity of the proposed solutions by the DSE with each of the three above-mentioned mutation algorithms. We implemented a prototype and performed experiments to show the extensibility of our DSE approach and evaluate the impact of shared hardware resources overheads on our addressed MOOP.

1.3.3 Implementation and evaluations

To summarize, we propose a DSE to compute trade-offs between security and schedulability considering four different security implementations in TSP systems. The DSE is computed with three different exploration algorithms (i.e. mutation algorithms) based on a formulation of a multi-objective problem, solved by an adaptation of PAES. The prototype of our DSE is implemented and integrated in the Cheddar tool. This prototype is reusable and extendable to any MOOP.

The security analysis part of this implementation has been integrated into AADL Inspector [26], a commercial scheduling analyzer from Ellidis Technologies.

We run multiple experiments with benchmarks or applications proposed by the community [27, 4, 28, 2, 29, 30, 31, 32] to identify the TSP architecture parameters which impact the trade-off between security and schedulability. They contribute to highlight guidelines that must be considered when designing secure TSP systems. Moreover, the experiments also contribute to evaluate our DSE approach.

1.4 Thesis organization

This thesis is organized in 11 chapters.

The chapters 2, 3, 4, and 5 cover the state of the art of this thesis. Chapter 2 presents key knowledge of real-time systems. Chapter 3 discusses the background about hierarchical systems while introducing integrated modular avionics (IMA) [18] concepts such as time and space partitioning. Chapter 4 gives a presentation of security concepts such as security properties, models, and architecture. Chapter 5 presents multi-objective optimization concepts by discussing multi-objective evolutionary algorithms (MOEA).

Chapter 6 depicts the motivations of this thesis. It presents the system model and taken assumptions. It also positions the contributions of the thesis by comparing it with related work.

The contributions are presented in chapters 7, 8, 9, and 10. Chapter 7 presents our DSE approach to investigate the schedulability and security trade-off in TSP systems when considering unicore platforms. Chapter 8 discusses the experiments we performed to evaluate our DSE approach and to identify key parameters that impact the trade-off between security and schedulability. Chapter 9 presents our DSE approach to investigate the impact of multicore platforms on TSP systems while addressing the conflicts between safety, security, and schedulability. It shows the adaptability of our DSE approach to a different context. Chapter 10 presents the implemented prototypes of this thesis. Since they are integrated into the Cheddar scheduling analyzer, the chapter also proposes a presentation of the Cheddar framework.

Finally, a conclusion of the thesis and some directions for future work are given in Chapter 11.

Part I

State of art

2

Real-time systems

This chapter is dedicated to the background on real-time systems (RTS). First, section 2.1 defines RTS by presenting their characteristics and classifications. Second, section 2.2 presents the architecture of RTS by depicting their hardware platform and their software structure, including the operating system. Third, section 2.3 discusses scheduling analysis and simulation to verify timing constraints. Finally, a conclusion of the chapter is given in section 2.4.

2.1 Definitions and classification

In this section, we define RTS, outline the classification and the characteristics of each category.

2.1.1 Definitions

Real-time systems are used in several domains such as avionic, space, automotive, and medicine. Applications in these domains may have different functionalities but are all characterized by timing constraints.

Definition 1. *(**Real-time systems**) A real-time system is defined as a computing system which the correctness depends not only on the logical results of the computation but also on the time at which the results are produced [12].*

A peacemaker [33] is an example of a real-time system placed down the heart. It measures the heartbeat and then provides pulsations to slow or accelerate the heart rate in case of an abnormal heartbeat. When the peacemaker receives the

information that the heartbeat is lower than normal, it sends electric pulsations to provide acceleration. The peacemakers correctness relies not only on the correctness of the heartbeat measurements but also on the time when the pulsations are sent. It is necessary that the sending of pulsations respects deadlines (i.e. at the right time, neither too early nor too late).

To summarize, a real-time system interacts with its environment by receiving information from the environment, processing them and returning the results that impact the environment while respecting timing constraints called deadlines [34]. Even if the computation results are correct, if the deadlines are not respected, the results can be considered incorrect. The deadlines misses do not lead always to disastrous consequences (e.g. loss of life). The consequences of deadlines misses are various depending on the addressed real-time system. Then real-time systems can be classified depending on the consequences of the deadlines misses.

2.1.2 Classification

RTS are classified into different categories. The classification can be made based on the criticality level i.e. the consequences of the missed deadline of the systems [13].

2.1.2.1 Hard real-time system

Hard real-time systems are systems that must imperatively respect their timing constraints [13, 14]. In hard real-time systems, the violations of deadlines have impacts on the system behavior with disastrous consequences. For these systems, the missed deadlines cannot be tolerated. Then a result provided after the expected deadline is considered not correct even if the computation is correct. Heart peacemaker, flight control systems are examples of hard real-time systems.

2.1.2.2 Soft real-time system

Soft real-time systems are systems that can tolerate timing constraints violations without leading to any disaster [13, 14]. Their timing constraints violations do not compromise the global functioning of the system. These systems are more flexible concerning timing constraints. Their deadline violations can lead to the deterioration of the results and system performances (e.g. quality of service [35]) but will not cause catastrophic consequences. Video streaming on internet is an example of soft real-time system.

2.1.2.3 Criticality of RTS

There are systems with programs/applications of multiple criticalities. Criticality is a designation of the level of assurance against failure needed for a system component [36]. It can refer to the classification (e.g. hard or soft) of programs/applications according to the consequences of their failures.

Aircraft is an example of a system with multiple criticalities. It contains flight management systems and passenger entertainment systems separated from each other. Flight management systems are hard real-time systems while passenger entertainment systems are soft real-time systems.

We distinguish mixed critical real-time systems and partitioned systems.

- Mixed critical real-time systems: are systems that have two or more distinct levels (for example safety-critical, mission-critical, and low-critical) [36].

  A mixed critical real-time system consists of integrating programs/applications of different criticality levels sharing resources together while making adaptations depending on the objectives (e.g. schedulability, security, safety). For example, in the case of schedulability objective, if a high-level task is susceptible to miss its deadline, the access to the resources of a low task can be modified to allow the high task to be executed at the time and to respect its deadline. As an advantage, the mixed critical approach allows better exploitation of the resources, but it is still at the research stage because it is difficult to apply in the industry.

- Partitioned systems are characterized by a necessary separation and independence of programs or applications to ensure that only intended coupling occurs [37].

  Each partition hosts programs or applications at the same criticality level. The objective is to limit the propagation of faults (missed deadlines, security attacks, safety failures) from one partition to another. As an inconvenience, it has a high resource requirement since resources have to be dedicated to each partition.

2.2 Real-Time architecture

RTS can be seen as layered systems. Most of them are composed of a software application layer deployed on an real-time operating system running on top of a hardware platform. RTS layers are described in the following sections.

2.2.1 Hardware platform

The hardware platform is characterized by components such as computing units, cores, networks, memory units.

2.2.1.1 Computing units

Many RTS are embedded and then require small computing units with minimum power consumption. There are different categories of hardware platforms depending on the contained components and their interactions.

- Uniprocessor systems: systems based on a single central processing unit (CPU) where the software applications execute. The programs of the applications request and compute concurrently to this computing resources.

- Multi-processor systems: systems that provide two or more central processing units for the execution of the software applications. These CPUs execute programs at the same time while sharing the main memory and the peripherals. There are different types of multi-processing systems depending on how the CPUs are used. Symmetric multiprocessing (SMP) systems are systems where the CPUs used are identicals. There are also Asymmetric multiprocessing (AMP) systems where the used processors are not used equally and may have different roles assigned each.

- Multi-core systems: systems based on single central processing with two or more executing units called cores.

2.2.1.2 Memory units

The memory units have many characteristics that should be well study to feet the systems requirements. Among those characteristics, there are power consumption, capacity/size, accessibility to perform for reading and writing operations, cost, etc. Memory is responsible for data and instructions storage. The capacity of storage should fit the needs of the systems. Memory speed is an important characteristic. If this speed is too low as compared to the processor's speed, requests to access data will be too long and lead to more power consumption. Among different existing kinds of memory systems, there are systems with memory partitioned where memory is split into different sections, each assigned to a different program.

2.2.2 Real-time software application layer

The software application layer of RTS runs on top of a real-time operating system (RTOS) and is composed of applications. Each application is a set of tasks that interact with each other.

Definition 2. *(**Task**)*

A task is an execution unit of a program and corresponds to a logical unit of computation in a processor [13].

A task is composed of a set of sequentially executed instructions that starts at the release of the task called job.

The software application layer may be composed of tasks that interact with each other. Hardware resources are allocated to tasks based on a multitasking process.

Definition 3. *(**Multi-tasking [38]**)*

Multi-tasking is the process of scheduling and switching tasks, making use of the hardware computing, or emulating concurrent processing using the mechanism of the task context switching (defined below).

Definition 4. *(**Context switch**)*

Context switch refers to the switching on the processor from one task to another [39].

The context switch is the process of interrupting a running task and then allocating the processing unit to another task. The state of the interrupted task is saved and the previously saved state of the other task is loaded.

2.2.2.1 Task life cycle

The life cycle of a task is composed of the four states detailed below in figure 2.1: inactive, ready, running and waiting. The switch between the states is handled by the RTOS.

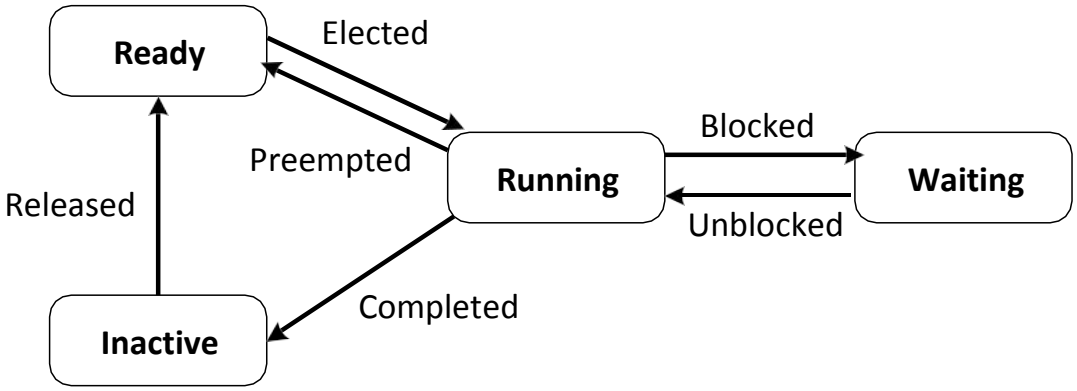


FIGURE 2.1: Task life cycle [3]

- Inactive: the inactive state of a task corresponds to its first state when it is created and its execution has not started yet. Thus the task is waiting for its release that can be activated by an event or message. A task that has completed its execution is also considered in the inactive state.

- Ready: after its release, the task enters its ready state and waits for computing resource assignment. The task waits for its election among the other tasks to be executed on the processor.

- Running: once the task is elected to access the processor for its execution, then it enters the running state and starts its execution. The task has access to all the shared resources. During the execution of a task, preemption may occur meaning another task has been elected to be executed. The current task which is at the running state stops its execution and switches to the ready state.

- Waiting: when a task execution is blocked by the unavailability of a resource (e.g. shared resources) except the processor, it enters the waiting state. As soon as the resource becomes available, the task switches to the running state.

2.2.2.2 Task properties

Each task is characterized by a set of properties that help to define its order of priority in the task set, its computational requirements and its timing constraints. This section presents the properties of a task.

Definition 5. ***(Offset)***

The offset of a task represents the time at which its first request occurs [40].

The offset represents the time at which the first job of a task is released. It is defined to model systems where tasks are not released at the same time. Thus, in those systems, tasks may have different offsets and then the first job of a task can be released later than another task's first job.

Definition 6. *(**Execution time**)*

The execution time of a task is defined as the time spent by the system executing a job of that task using processor resources.

The execution time of a task may vary from a job to another depending on the input data or different behavior of the environment [41]. For example, the changes of the input data may change the execution paths and the number of loop iterations and then lead to different execution times. Some works consider the upper-bound and the lower-bound execution time values respectively called worst-case execution time (WCET) and best-case execution time (BCET).

Definition 7. *(**Worst case execution time**)*

The WCET of a task is the longest execution time of all its jobs [41].

Definition 8. *(**Best case execution time**)*

The BCET of a task is the shortest execution time of all its jobs [41].

Hard real-time systems parameters are always fixed based on a pessimist approach that consists of considering the worst-case situations. They consider the WCET and assume the same execution time for all the jobs of a task. WCET of a task is sometimes refered as the task capacity. This pessimist approach helps to guarantee that the system meets its deadlines [14]. On the contrary, soft real-time systems are not necessarily built under pessimistic assumptions [14] because of their tolerance for deadline misses.

Besides the execution time, each job of task is characterized by its response time.

Definition 9. *(**Response time**)*

The response time of a job of a task is the time between the release of the job and its completion [42].

As for the execution times, the response time of a task varies from a job to another. They are computed based on scheduling analysis methods detailed later in section 2.3.

Upper-bound and lower-bound response time values can be computed and respectively called worst-case response time (WCRT) and best case response time (BCRT).

Definition 10. *(**Worst case response time**)*

The WCRT of a task is the longest response time of all its jobs [42].

Definition 11. *(**Best case response time**)*

The BCRT of a task is the shortest response time of all its jobs [42].

We highlight that a task execution time may be smaller than its response time since tasks are not executed immediately after their release. After their release, tasks may also be waiting for the availability of a shared resource or the processor used currently by higher priority tasks.

Definition 12. *(**Priority**)*

The priority of a task indicates its order of importance for the scheduling among other tasks of the system [13].

Tasks of a system can be ordered based on their priorities that define their order to access the processor. Tasks priorities can be fixed or dynamic depending on the assumed scheduling policy. The task in the ready queue with the highest priority level is usually the elected task to be executed by the processor. In case of preemptive scheduling (detailed in section 2.2.3), when a task is running, it can be suspended by a new ready task with a higher priority level.

Definition 13. *(**Deadline**)*

The deadline of a task is the maximal allowed response time to the jobs of its task [13].

We distinguish relative and absolute deadlines. The relative deadline is relative to the release time of a job in contrary to the absolute deadline. Figure 2.2 shows the difference between relative and absolute deadline of a task and some tasks properties such as release time, response time, deadline.

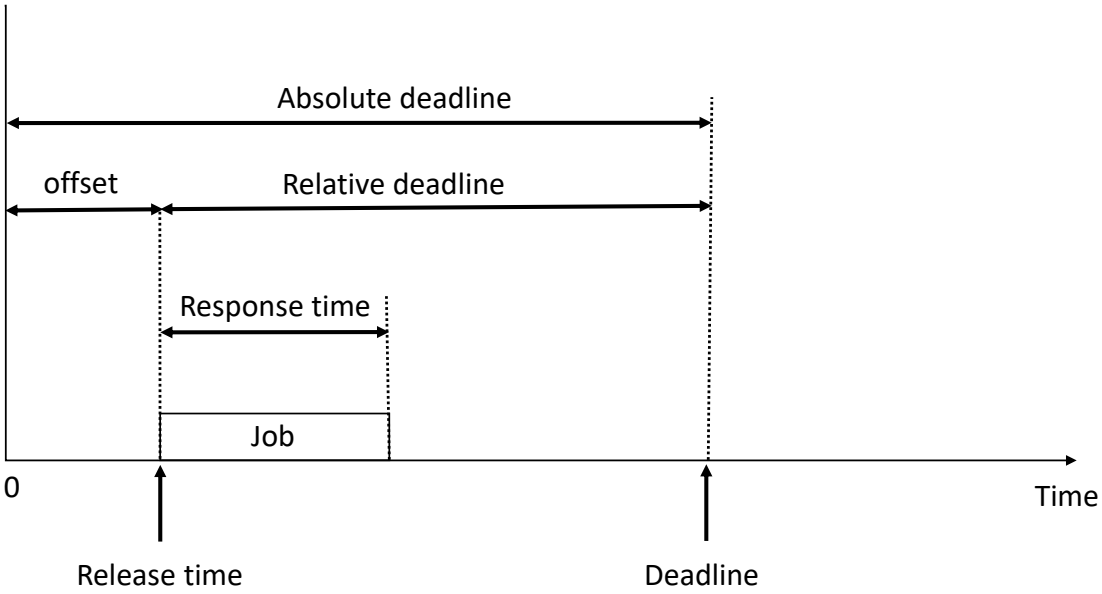


FIGURE 2.2: Illustration of some tasks properties

Task deadlines can be classified into two categories according to their necessity to be met. There are hard deadlines and soft deadlines.

Definition 14. *(**Hard deadline**) A hard deadline is a deadline that must imperatively be met, otherwise it can lead to severe damages [13, 14].*

Definition 15. *(**Soft deadline**) A soft deadline is a deadline that can be missed without compromising the integrity of the system [13]*

2.2.2.3 Task types

Tasks can be classified based on their release frequency and their activation circumstances (e.g. occurrence of an event). Tasks can be periodic tasks, aperiodic or sporadic.

Definition 16. *(**Periodic task** [43]) A periodic task is a task that is released at fixed and regular time interval.*

A periodic task is released repetitively at a fixed interval time called period. A task that acquires data periodically (e.g. every 1 second) from a sensor is an example of a periodic task.

Definition 17. *(**Aperiodic task** [44])*

An aperiodic task is a task that is not released at a regular time interval, with no minimum interval time between two releases.

Aperiodic tasks are activated by events and the events do not occur at regular intervals. They are usually soft real-time tasks with soft deadlines or no deadlines. It is complex to bound the resource utilization of aperiodic tasks. Thus the feasibility of a system with aperiodic tasks cannot be guaranteed.

Users interactive commands are examples of aperiodic tasks.

Definition 18. *(**Sporadic task** [44])*

A sporadic task is a task that is released with a minimum interval time between two releases.

A sporadic task has a minimum interarrival time (MIT) between two consecutives releases. The MIT of a sporadic task provides a safe upper bound to determine its resource utilization.

A security alert program is an example of a sporadic task. It is a high critical and its releases arrive arbritrary because arrival cannot be predicted.

2.2.2.4 Synchronous and asynchronous tasks

Tasks can also be classified based on their first release. Tasks can be synchronous or asynchronous.

Definition 19. *(**Synchronous tasks** [45])*

Tasks of a system are synchronous if the first jobs of all the tasks are released at the same time.

Synchronous tasks have the same offset. Thus tasks are simultaneously ready to be executed at a given time called critical instant [43].

Definition 20. *(**Asynchronous tasks** [45])*

Tasks of a system are asynchronous if the first jobs of at least two tasks are not released at the same time.

In asynchronous systems, there are at least two tasks with different offsets.

2.2.2.5 Tasks dependencies

Tasks can also be classified according to the relationships they have with each other.

Definition 21. *(**Independent tasks** [13]) An independent task is a task whose progress is not dependent upon the progress of other tasks of the task set.*

In a task set with only independent tasks, no task can be blocked by another task.

Definition 22. *(**Dependent tasks** [13]) A dependent task is a task whose progress is dependent upon the progress of other tasks of the task set.*

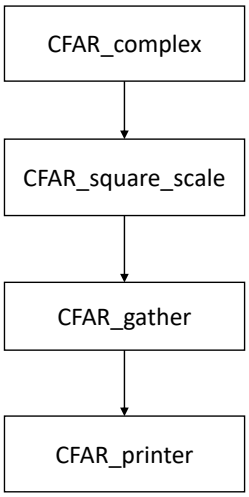
Dependent tasks are used to force the order in which communicating tasks execute [46]. They can interact in many ways including shared resources and precedence dependencies [13]. It is important to highlight that the competition between tasks to access a processor is not considered as a dependency.

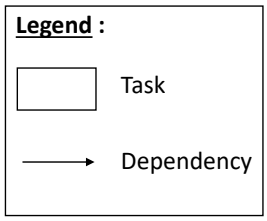
Definition 23. *(**Precedence dependency** [13]) Task τ_i precedes task τ_j in the task set, if the task τ_i has be executed before task τ_j.*

By considering two tasks τ_i and τ_j, τ_i precedes τ_j means that the jobs of task τ_i have to be executed before the jobs of task τ_j start their execution. Precedence dependency can be illustrated by a task τ_j waiting for a message or synchronization signal from another task τ_i.

Definition 24. *(**Shared resource** [13]) A shared resource is a resource accessed by several tasks, in an exclusive manner to enforce data consistency.*

Figure 2.3 shows an example of a real-time system composed of four periodic, synchronous and dependent tasks. This example is inspired from the Constant False Alarm Rate detection (CFAR) application proposed in the benchmark [4]. The boxes represent tasks and the arrows show the precedency dependencies between tasks. For example, the task CFAR_complex must be executed before task CFAR_square_scale. The table in figure 2.3 depicts the properties of each task such as the offset, WCET, priority, deadline, and period.





Properties / Tasks	Offset	WCET	Priority	Deadline	Period
CFAR_complex	0	90	1	1000	1000
CFAR_square_scale	0	50	2	1000	1000
CFAR_gather	0	340	3	1000	1000
CFAR_printer	0	30	4	1000	1000

FIGURE 2.3: Tasks properties and types illustration with CFAR application [4]

2.2.3 Real-time Operating System

An operating system (OS) is a software program that serves as an interface between the software application and the hardware platform by providing services such as memory management, process management, drivers management, communication management, etc.

A real-time operating system (RTOS) is an OS intended for RTS. The guarantee of timing constraints is one of the most significant differences between an RTOS and a general-purpose OS (GPOS) such as Windows, Linux, macOS. FreeRTOS [47] and RTEMS [48], RTLinux [49], PikeOS [50] are examples of RTOS.

Unlike GPOS, RTOS proposes additional services in order to guarantee systems predictability and timing constraints requirements.

Usually, RTOS support systems with tasks of different priority levels. They guarantee resources access based on tasks priorities. Low priority tasks can then be preempted to allow the execution of high priority tasks. RTOS ensures that whenever a task τ_i is running and a higher priority task τ_j arrives, the task τ_i is automatically interrupted in favor of the execution of τ_j.

RTOS proposes specific schedulers responsible of the order of the tasks executions.

Definition 25. *(**Scheduler** [13])*

A scheduler implements an algorithm or a policy for ordering the execution of the tasks on the processor according to some pre-defined criteria.

RTS scheduling is based on the scheduling policies provided by the scheduler.

Definition 26. *(**Scheduling** [13]) Scheduling is a method by which tasks are given access to computing resources (e.g. processors) based on a predefined scheduling policy.*

Definition 27. *(**Scheduling policy or scheduling algorithm** [13]) Scheduling policy is the algorithm that defines how tasks have access to computing resources (e.g. processors).*

A RTOS may support multiple scheduling policies that can be preemptive and non-preemptive, online and offline, fixed priority and dynamic priority.

Definition 28. *(**Preemptive scheduling** [13]) A task execution can be arbitrarily suspended and restarted later without affecting the behavior of that task.*

For example, a preemption occur if during the execution of a task, comes a higher priority task ready to be executed. In that case, the executing task is preempted in order to allow the execution of the higher priority task. The scheduler suspends an executing task through an interruption mechanism in order to allow another task to be executed.

Definition 29. *(**Non-preemptive scheduling** [13]) A task cannot be suspend during its execution.*

In the case of non-preemptive scheduling, once a task gets access to a computing resource (i.e. processor), it cannot be interrupted. Its execution must be completed before the resource access can be given to another task.

Definition 30. *(**Offline scheduling** [13]) A scheduler is static and offline if all scheduling decisions are made prior to the running of the system.*

Offline scheduling is characterized by the definition of all scheduling decisions at compile-time, before the execution of tasks. The scheduling, already confirmed to allow all tasks to meet their deadlines, is specified via a scheduling table. Thus the system will be scheduled based on the scheduling table. It is fully deterministic because all the tasks information such as the list of the tasks and their activation times are predefined in the scheduling table. This method suits well for high critical RTS with high determinism requirements. It also leads to a low runtime overhead since the same scheduling (i.e. specified in the scheduling table) is repeated till the end of the system's running time. The inconvenience of offline scheduling is that it requires complete knowledge of the system's behavior, requirements and environmental situations.

Definition 31. *(**Online scheduling** [13]) An online scheduler makes scheduling decisions during the runtime of the system.*

In contrary to offline scheduling, the online scheduling is more flexible since the decisions are taken during the runtime. The decisions are based on both tasks characteristics and the current state of the system. Thus, it is less predictable than offline scheduling and may increase significantly the runtime overhead. However, it provides more flexibility such as the possibility to add new entities (e.g. tasks) to the system design.

Definition 32. *(**Fixed priority scheduling**) In fixed priority scheduling, the priority assignment is done only once and hence the priority of a task will not change with time [51]. Tasks priorities are fixed offline at design [13].*

Tasks priorities decide the order in which tasks are executed. They are fixed based on tasks attributes such as periods or deadlines. There are multiple algorithms such as Rate-monotonic and Deadline-monotonic that implemented fixed priority scheduling.

Definition 33. *(**Rate-Monotonic**) In the Rate Monotonic algorithm (RM), all tasks are allocated a priority according to their periods. The shorter the period the higher their priority [52].*

Tasks priorities are inversely proportional to their periods. Tasks with the same period can be resolved in an arbitrary manner [51]. Thus, Rate-Monotonic is devoted to periodic tasks only.

Definition 34. *(**Deadline-Monotonic**)*

In the Deadline-Monotonic (DM), all tasks are allocated a priority according to their relative deadlines. The task with the shortest deadline is assigned the highest priority. The lowest priority is assigned to the longest deadline task [13].

Tasks priorities are inversely proportional to their deadlines. For a system where each task period value is equal to its deadline value, the DM and RM are similar.

Definition 35. *(**Dynamic priority scheduling**) In dynamic priority scheduling, tasks priorities may change when the system is running [13].*

Compared to fixed-priority scheduling, dynamic priority scheduling is more complex since tasks priorities are not static and fixed offline, but can vary during the runtime. This approach leads to an increase of the implementation complexity and the runtime overhead. However, it can allow systems with high processor utilization to become schedulable. The Earliest Deadline First (EDF) and Least Laxity First (LLF) are examples of dynamic priority scheduling.

Definition 36. *(**Earliest deadline first**) In Earliest Deadline First, the task with the (current) closest deadline is assigned the highest priority in the system and therefore executes [13].*

According to EDF, priorities are assigned dynamically and are inversely proportional to the absolute deadlines of the active jobs [53].

Definition 37. *(**Least Laxity First (LLF)**) The task which has the least laxity first is assigned the highest priority in the system and is therefore executed [52]. The laxity of a task is defined as the deadline minus remaining computation time [13].*

In LLF, the executing task has a constant laxity and will be preempted by a task whose laxity has decreased.

In the case of two tasks τ_1 and τ_2 with similar laxities, τ_1 will run for a short time and then will be preempted by τ_2 which will also run for a short time and will be preempted by τ_1. Thus, when a system has tasks with similar laxities, LLF may lead to thrashing which means the processor spends more time doing context switches than useful work [13].

2.3 Real-time scheduling analysis

The scheduling analysis aimed to check the timing constraints of RTS. It mainly consists of feasibility and schedulability tests. This section presents the concepts of feasibility and schedulability in the context of RTS.

2.3.1 Feasibility and schedulability

Definition 38. *(**Feasibility** [13]) Feasibility is the assessment of a task set to meet all its timing constraints.*

A task set is feasible if there is a scheduling policy guaranteeing that all the timing constraints are met. Thus, for a given system, if all the tasks have all their jobs scheduled without any missed deadline, the system can be qualified as feasible.

Definition 39. *(**Schedulability** [13]) Schedulability is the assessment of the feasibility of a tasks set under a given scheduling policy.*

A task set is schedulable with a particular scheduling policy if none of its tasks will ever miss its deadlines during execution [13]. Feasibility is a broader concept that includes schedulability. It means first, that a schedulable task set is feasible. Second, a feasible task set that is schedulable under a given scheduling policy A, is not necessarily schedulable under another given scheduling policy B.

To assess feasibility or schedulability, we may apply tests.

Definition 40. *(**Feasibility test** [54]) A feasibility test assesses whether a task set is feasible or not.*

Definition 41. *(**Schedulability test** [54]) A schedulability test assesses whether a task set is schedulable with a given scheduling policy or not.*

Tests that define the feasibility or the schedulability of a task set can be classified into three categories: sufficient, necessary and exact test.

- Feasible/schedulable sufficient test: a task set can be considered feasible/schedulable if the given conditions are fulfilled. Otherwise, nothing can be concluded. Therefore, the task set cannot be considered feasible/schedulable, if these tests are not fulfilled.
- Feasible/schedulable necessary test: a task set that does not fulfill the given conditions is automatically considered not feasible/schedulable. Otherwise, nothing can be concluded. Therefore, there is no guarantee for feasibility/schedulability of the task set, if these tests are fulfilled.

- Feasible/schedulable exact test: a task set that fulfills the given conditions are automatically considered feasible/schedulable. These tests are also called necessary and sufficient conditions.

Feasibility and schedulability tests depend on the parameters (i.e. attributes of tasks properties) of the system. It is difficult to have an exact estimation of these parameters. Usually, there are estimated in the worst-case scenarios. Therefore, in practice, there are always unpredictable deviations that can be covered by sustainability towards better scenarios.

Definition 42. *(**Sustainability** [55]) A given scheduling policy and/or a schedulability test is sustainable if any system that is schedulable under its worst-case specification remains so when its behavior is better than worst-case.*

The better scenario can be the decrease of the execution time or the jitter, or the increase of the period, the relative deadline of a task of the system. Thus sustainability can be categorized based on the parameters that changed for better scenario:

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

2.3.2 Scheduling analysis methods

Analysis can be performed to study the behavior of RTS in order to evaluate their schedulability. There are several methods to perform scheduling analysis of RTS. Analytical analysis and simulation are approaches used to perform such analysis. They required a modeling of the system with a level of abstraction that still presents a complete knowledge of the system architecture (e.g. hardware and software components, scheduling policy...) in order to maintain a certain level of accuracy [56]. This section presents these approaches.

2.3.2.1 Analytical methods

In scheduling analysis, analytical methods are mathematical equations that represent sufficient or exact schedulability tests. The analytical methods are based on the system model performance attributes such as processor utilization, or tasks response time. We assume that C_i, T_i, D_i and R_i represent respectively the capacity, the period, the deadline and the WCRT of task τ_i.

- Processor utilization based test: consists of the evaluation of the total processor utilization of the system model which consists of the sum of the processor utilization of the tasks.

  The utilization of the processor by a task τ_i is computed as follows:

$$U_i = \frac{C_i}{T_i} \tag{2.1}$$

  The total processor utilization of a task set that consists of n tasks is computed as follows:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.2}$$

  [43] has proposed the following theorems.

  Theorem 1 ([43]). *A task set of n synchronous independent periodic tasks, executing on a uniprocessor, and with $D_i \leq T_i$, is schedulable by EDF scheduling if and only if:*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.3}$$

  Theorem 2 ([43]). *In a fixed priority preemptive scheduling context, a task set of n synchronous independent periodic tasks with $D_i = T_i$, executing on a uniprocessor, is schedulable by RM if:*

$$U \leq n(2^{1/n} - 1) \tag{2.4}$$

  Theorem 3 ([43]). *In a fixed priority preemptive scheduling context, a task set of n synchronous independent periodic tasks with $D_i \leq T_i$ executing on a uniprocessor, is schedulable by DM if:*

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1) \tag{2.5}$$

- Response time analysis [57] is based on the WCRT of each task in the system model. It consists of computing the WRCT of each task and comparing it to the task relative deadline. The task set is considered schedulable if the following equation is respected:

$$\forall \tau_i,\ R_i \leq D_i \tag{2.6}$$

Analytic methods are rapid to perform since there are based on equations, but there are systems for which those equations cannot be applied. Analytic methods do not usually imply scheduling computation.

2.3.2.2 Scheduling simulation method

In scheduling simulation, the system model is executed by a scheduling simulator based on the specified scheduling policy during a finite interval which can be feasibility or simulation interval. Finally, the output of the execution is analyzed.

Definition 43. *(**Feasibility interval** [58, 59])*

A feasibility interval is a finite interval such that if all the deadlines of jobs released in the interval are met, then the system is schedulable [59].

For a given system model if no deadline is missed during its feasibility interval, no deadline will be missed latter and then the system schedulability is guaranteed.

The feasibility interval of a system model is related to the simulation interval.

Definition 44. *(**Simulation interval** [59]) A simulation interval is a safe interval such that the schedule repeats in a cycle [59].*

Knowing the length of the simulation interval is also required for capturing the whole behavior of a system when building a pre-run-time schedule known as offline schedule [59].

It is useful to capture the whole behavior of the system to characterize various metrics and evaluate during the simulation if the system can be considered as schedulable.

The litterature proposes several scheduling simulators such as Cheddar [1] [25] (detailed in section 10.2), Simulation Tool for Real-time Multiprocessor (STORM) [2] [60], Real-Time system SIMulator (RTSIM) [3].

The simulation method have advantages and inconveniences compared to the analytic methods. Simulation is more flexible and can be applied to a larger number of systems because the analytic methods always required that the analyzed system complies with assumptions of mathematical model. There are systems for which a mathematical model does not exist. As an advantage, for verification purpose, computing the simulation produces tasks scheduling.

As inconveniences, we cannot always rely on simulation as proof since it is conditioned by the accuracy of some parameters such as feasibility interval. It requires complete knowledge of the analyzed system and the parameters are often fixed based on the pessimist approach. Then, the scheduling simulation may lead to conclude that the analyzed system requires significantly more computing

[1] http://beru.univ-brest.fr/cheddar/
[2] http://storm.rts-software.org/
[3] RTSim, http://rtsim.sssup.it/

resources to be schedulable than it is in practice [61]. Furthermore, a combinatorial explosion is faced when performing exhaustive simulations for every possible system state.

As an example, figure 2.4 shows a fixed priority-based scheduling simulation of the CFAR application described above in figure 2.3 on its feasibility interval.

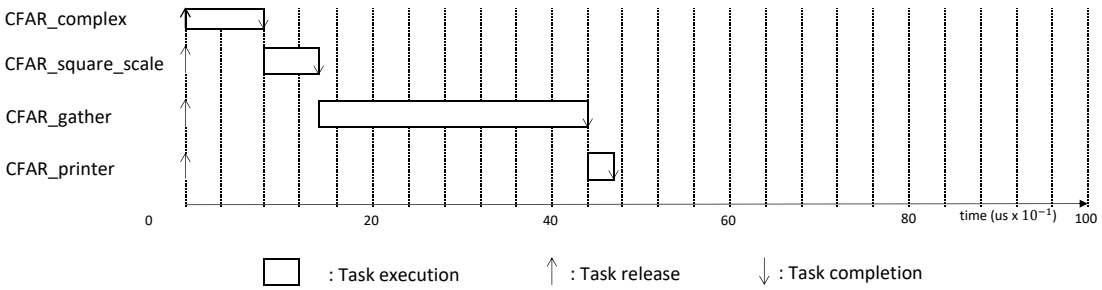


FIGURE 2.4: Illustration of a scheduling simulation

It shows the release and the completion time of each task represented by respectively an up arrow and a down arrow. The interval between these times represented by a box corresponds to the response time. We notice that no task missed its deadline. Then the task set is schedulable since it is computed on its feasibility interval.

2.4 Conclusion

This chapter is dedicated to the presentation of RTS. It presents the different categories of RTS, their components (hardware and software) and their properties. It also describes different approaches proposed in the literature to perform scheduling analysis. It presents background about analytics analysis methods, scheduling simulation, and schedulability and feasibility tests of RTS.

In domains such as avionics, automotive, there are RTS that require separation between tasks sharing the same computing resource. Then the next chapter presents hierarchical systems in the context of RTS.

3

Hierarchical real-time systems

This chapter discusses partitioned systems, which is the main target of this thesis. We start with hierarchical RTS in general and end with time and space partitioned systems in avionic. Section 3.1 gives definitions about hierarchical RTS and their characteristics. Section 3.2 presents examples of resources allocation modeling for hierarchical RTS. Section 3.3 introduces a well-known application of hierarchical RTS in the avionic domain called integrated modular avionics (IMA). Section 3.4 introduces the ARINC 653 standard. Finally, section 3.5 presents examples of scheduling, memory management, and communications mechanisms of several operating systems dedicated to TSP systems

3.1 Definitions and characteristics

With the evolution of technologies, RTS are integrating more and more functionalities that increase their complexity [15]. They need to integrate more and more applications with different requirements on each computing resource and sometimes with a high level of legacy. This coexistance of multiple application of different levels of criticality requires mechanisms to reduce fault propagation while not jeopardizing the performances.

An hierarchical approach provides a separation of a system into multiple subcomponents and then facilitates their design, their analysis, their verification, and their validation.

Virtualization [62, 63] is known to support separation between different subcomponents (e.g. operating systems, applications tasks) running concurrently on the same computing resource. It provides a software layer called hypervisor (vir-

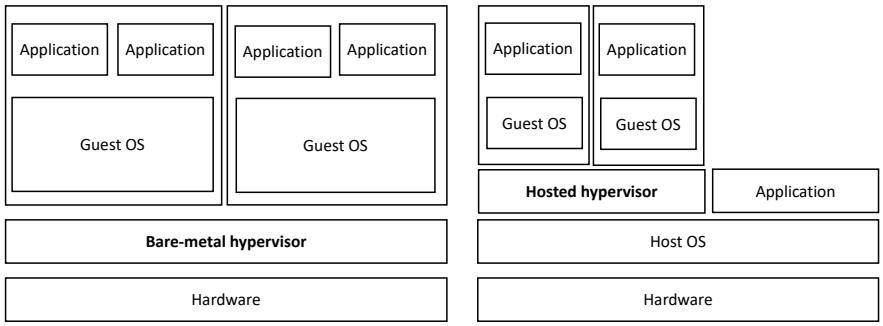


FIGURE 3.1: Bare-metal hypervisor Vs hosted hypervisor

tual machine monitor) with virtual machines that host operating systems and applications.

We notice two types of hypervisors: bare-metal hypervisors, and hosted hypervisors [63]. A bare-metal hypervisor (native or type 1 hypervisor) runs directly on the hardware, whereas a hosted hypervisor (embedded or type 2 hypervisor) runs on top of a parent OS as illustrated in figure 3.1.

In general, bare-metal hypervisors are faster because they have direct access to hardware resources. This direct access provides more security than hosted hypervisors by avoiding vulnerabilities such as malicious intrusions.

Further, there are two approaches to virtualization: full virtualization and paravirtualization [63]. In full virtualization, the guest OS can be hosted without any modifications while paravirtualization implies the modifications of the guest OS. Both virtualization approaches have their advantages. The modifications required in paravirtualization help to optimize the virtualization overhead. The advantage of full virtualization is that it does not require the modification of systems already verified and trusted.

Virtualization is commonly used to implement Hierarchical RTS. Hierarchical RTS [64] consists of hierarchical resource sharing between different subcomponents. It can be represented as in figure 3.2 where the resources of a subcomponent are shared between the subcomponents of the high level. Then resources of subcomponent 1 are shared between subcomponents 2 and 3. The resources of subcomponent 2 are shared between subcomponents 4 and 5.

Considering CPU resources, a hierarchical RTS can be considered as a scheduling hierarchy where each subcomponent consists of a real-time workload (i.e. tasks set model) and a scheduling policy. In figure 3.2, we notice two types of subcomponents. First, there are subcomponents (subcomponents 3, 4, and 5) that correspond to a set of tasks with a given scheduling policy. Then the tasks of each subcomponent are scheduled according to their scheduling policy. Second, there are subcomponents (subcomponents 1, and 2) that are composed of a set of the above-mentioned subcomponents. Each of them also includes a scheduling policy

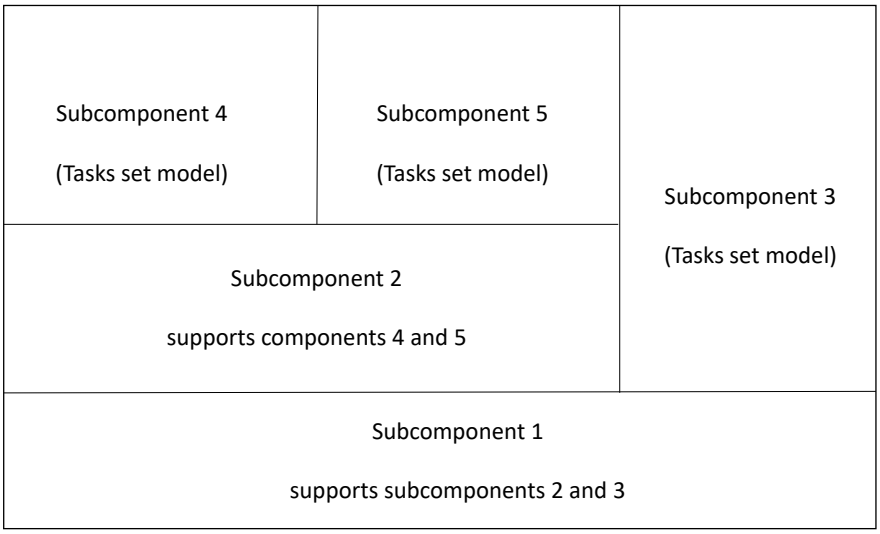


FIGURE 3.2: Hierarchical real-time systems

that defines how its subcomponents of the next level up should be scheduled.

Hierarchical scheduling is a confirmed mechanism to guarantee temporal isolation among partitions [64]. This can be established through a hierarchical scheduling framework (HSF) [65].

Definition 45. ***(Hierarchical scheduling framework (HSF))***

A hierarchical scheduling framework (HSF) is introduced to support CPU time-sharing among subcomponents under different scheduling services [65].

HSF is characterized by providing functional separation of subcomponents in order to guarantee not only their isolation during runtime but also to reduce the complexity of the whole system verification and validation through modularity [66]. With HSF, each subcomponent is executed independently and a failure in a subcomponent cannot affect another one. An HSF can be generally described as a two-level tree where each subcomponent is scheduled based on a local scheduler and all the subcomponents are scheduled according to the scheduling policy of a global scheduler. The scheduling policy of the global scheduler and all the local schedulers can be different.

Figure 3.3 illustrates two-level hierarchical scheduling.

- Global scheduler: provides the required CPU resources for the subcomponents of the HSF. To schedule all its tasks, each subcomponent requires an amount of CPU [66]. The global scheduler determines the order of execution of the subcomponents by deciding which subcomponent should have access to the CPU resources at a given time. It assigns each subcomponent

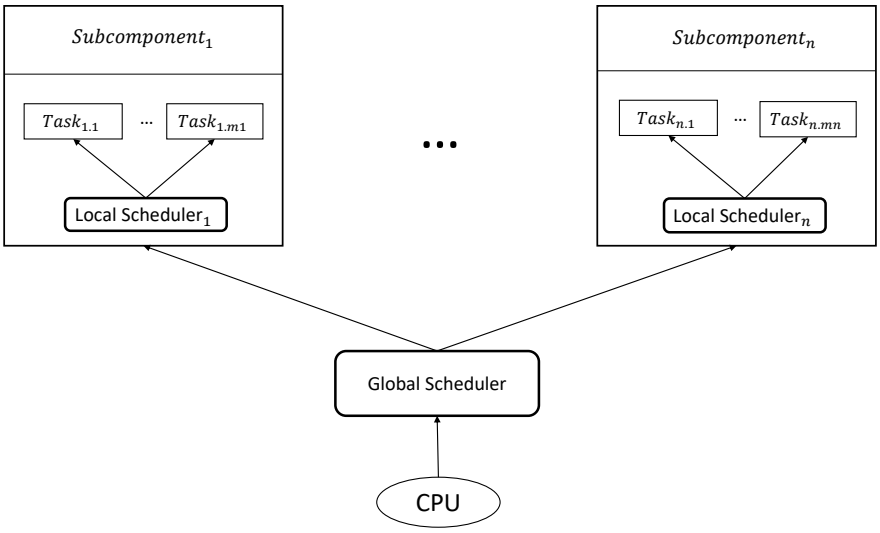


FIGURE 3.3: Hierarchical scheduling framework

a fraction of the total processor time distributed over the time line based on the assumed scheduling policy. The global scheduling can be decided off-line or online.

- Local scheduler: selects the next task in a subcomponent to be executed when the concerned subcomponent is selected by the global scheduler. It can be scheduled based on any scheduling algorithm presented in chapter 2.

To summarize, the global scheduler selects the subcomponent to be executed at a given time and then the local scheduler decides which task will be executed among the tasks of the selected subcomponent.

3.2 Resource model

This section presents resource models for resources allocation in hierarchical RTS. In an HSF, each parent node assigns resource allocations to its child nodes [66]. This can be referred to as a resource model also called virtual processor model [67].

Definition 46. *(**Resource model**) A resource model is a model for specifying the timing properties of resource supply [68].*

Definition 47. *(**Resource supply**) The resource supply of a resource is the amount of resource allocations that the resource provides [69].*

The literature proposes different resource models such as the periodic resource model and bounded delay resource model.

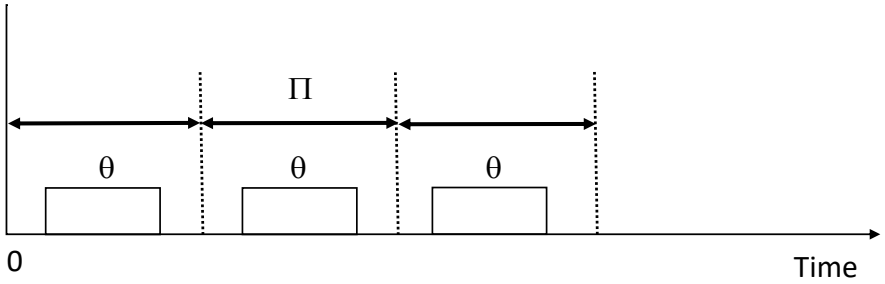


FIGURE 3.4: Periodic resource illustration

3.2.1 Periodic resource model

Definition 48. *(**Periodic resource model**) A periodic resource model* Γ*(*Π*,*Θ*) with* $(0 < \Theta < \Pi)$ *can characterize a resource allocation of* Θ *time units every* Π *time units with* Π *a positive integer and* Θ *a real number between (0,*Π*] [69].*

The periodic resource model is proposed to model resource allocation with a periodic approach. Figure 3.4 illustrates a periodic resource model $\Gamma(\Pi,\Theta)$.

For example, $\Gamma(6, 2)$ modelizes a resource model with a resource allocation of 2 times units every 6 times units.

In the cases where $\Theta = \Pi$, then the resource is available all the time.

3.2.2 Bounded delay resource

Definition 49. *(**Bounded delay resource partition model**)*

In a bounded delay resource partition model [67], a resource partition Π *is a tuple* (Γ, P) *where* Γ *is an array of* N *time pairs* $(S_1, E_1), (S_2, E_2), ..., (S_N, E_N)$ *that satisfies* $(0 \leq S_1 < E_1 < S_2 < E_2 < ... < S_N < E_N)$ *for some* $N \geq 1$*, and* P *is the partition period.*

This model defines the interval times in which a partitioned resource is available. It describes the behavior of a partitioned resource that is available at its full capacity at some times. The interval times of unavailability are referred to as blocking times. Figure 3.5 illustrates bounded delay resource $\Pi((S_1, E_1), (S_2, E_2), ..., (S_N, E_N), P)$.

For example, a bounded delay resource partition Π = (1,2), (4,6), 8) starts from time 1 to time 2 and from time 4 to time 6 with a period of 8 time units.

Especially, in traditional systems that we can consider as systems with only one partition, there is no blocking time and the model can be simplified by being represented by $\Pi = ((0, P), P)$ meaning that the partition starts at time 0 to time P every P period.

IMA architecture can be considered as an implementation of HSF in RTS.

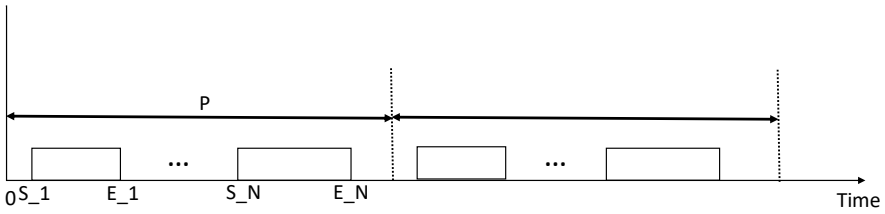


FIGURE 3.5: Bounded delay resource illustration

3.3 Time and space partitioning (TSP) and integrated modular avionics (IMA)

By the past, systems in avionics were implemented based on federated architectures [16]. The federated approach is based on a "one function = one computer" principle that consists of allocating to each function in the system its own dedicated computer system [15]. Each avionic function has its dedicated processing unit, sensors and actuators. Then an avionic system was considered as a set of standalone subsystems dedicated to each function of the system.

Since each function is physically isolated from others and data exchange is limited between functions, a fault that occurs in a function or/and any of its dedicated resources can difficultly affect other functions. Dedicating each function to a dedicated computing system improves fault tolerance but requires more hardware in the system. As avionics systems need to integrate more functions, the federated architecture was no more appropriate because it increases the height, the volume, the installation, maintenance cost and the power consumption of the aircraft [18]. To overcome those drawbacks, an alternative to meet the requirements of the modern avionics was proposed with the integrated modular avionics (IMA) [18].

Definition 50. *(**Integrated Avionic Modular systems**)*

An integrated Avionic Modular (IMA) [37, 18] system is a critical real-time system that uses multiple software modules called partitions to isolate applications with different levels of criticality through hardware sharing.

To reduce fault propagation, IMA is associated to a strict and robust partitioning.

Each function is considered as running on a dedicated virtual resource called partition.

Definition 51. *(**Partitioning**) Partitioning [37, 15] is an architectural technique to provide the necessary separation and independence of functions or applications to ensure that only intended coupling occurs.*

Definition 52. *(**Robust partitioning**) Robust partitioning [37, 15] is the requirement that ensures that any hosted application or function has no unintended effect on other hosted applications or functions.*

The robust partitioning aims to provide an isolation between applications sharing the same resources as much as close to the isolation provided if they were hosted on different computing units as in federated systems [70].

Spatial partitioning intervenes to guarantee that no function of one partition is able to access or change the memory (programs and data) private to another partition. Then a memory management unit (MMU) provided by the hardware is usually used to ensure the memory protection between partitions.

With multiple functions in different partitions integrated into a same hardware module, the system can face problems such as the monopolization of the CPU by one partition denying services to other functions in other partitions. Then intervenes the temporal partitioning to ensure that each partition access to hardware resources for defined period and duration through a deterministic scheduling.

IMA guarantees application portability by ensuring software abstracting. Airbus A380 and Boeing 787 are examples of IMA systems. By using its approach of IMA, Airbus has reduced half of the number of processors used for the new A380 suite [71]. For IMA systems, there are multiple standards that give specifications for their design and that also ease their certification. As examples we have ARINC 429 [72] and ARINC 664 [73] for communications between hardware modules, ARINC 653 that concerns time and space partitioning services.

3.4 ARINC 653

Definition 53. *(**ARINC 653**) ARINC 653 [74] is a standard that specifies time and space partitioning services to design safe and critical real-time systems.*

Figure 3.6 illustrates an IMA architecture based on ARINC 653 standard that we will describe in the sequel.

3.4.1 Hardware

In ARINC 653, multiple applications share a hardware module. Applications are clustered into partitions needed to be isolated. Then ARINC 653 standard requires the ability of the hardware to support services essential for the application isolation.

The hardware module contains resources such as processor, memory, I/O devices. Even if partitions share the same hardware module, each partition has restricted access to those resources [5]. This restriction is operated through an adequate OS and ensures that without changing the API, the hardware (resp. application software) can be changed with no effect on the application software (resp. hardware).

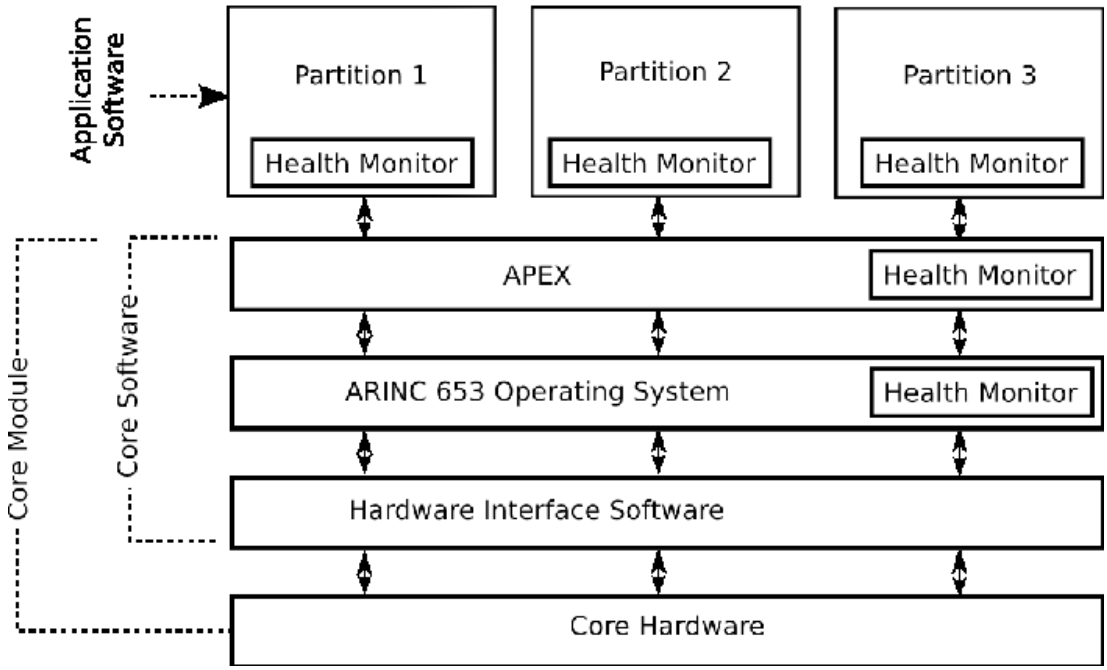


FIGURE 3.6: ARINC 653 architecture [5, 6]

3.4.2 Software

The applications run on top of a real-time operating system with the intermediate of an interface called Application/Executive (APEX) [75] provided by ARINC 653 standard.

Definition 54. *(**Application/Executive(APEX) interface**) Application/Executive(APEX) [75] interface is an application programming interface (API) defined by ARINC 653 standard for IMA systems to provide services for mechanisms such as partition and process management, time and memory management, inter and intra-partitions communications, error handling.*

APEX interface ensures the portability and modularity of the software applications and then favors the independence between software and hardware. It guarantees the reusability of the application code since it helps to reduce the customization effort of a reused component. It also allows flexibility in the choice of development tools and compilers by being independent of any high-level language. APEX interface enables independence between the development of applications and the operating system. APEX interface has the ability to host multi-criticality levels applications.

3.4.2.1 Partition and process management

With the APEX services, the operating system intervenes in the management of partitions and processes. In ARINC 653, software applications are defined as a set of processes that have to be hosted by partitions.

Each partition has a dedicated memory space protected by a memory management unit even if the partition is not active. Thus the space isolation is ensured because no active partition will be able to write in a nonactive partition. In ARINC 653 specification, each partition is characterized by the attributes defined below.

- A name and a unique identifier that help to constitute the identity of a partition in the system.
- A partition period that indicates the duration between two successive activations of the partition.
- A partition duration that represents the execution time of the partition. The time isolation is ensured by the allocation of disjoint time slot to each partition. No partition is allowed to exceed its allocated time slot.
- A memory space that delimits memory allocation of the partition. This helps in guaranteeing space isolation of partitions.

A process is a unit of programs allocated to a partition and that has to execute concurrently with the other processes within the partition. Both periodic and aperiodic processes are possible in ARINC 653 systems. The processes are characterized by some important attributes defined below.

- A name that identifies the process among others.
- A period that represents the duration between two successive activations of a periodic process.
- A time capacity that represents the execution time budget of the processes.
- A process priority that determines the order of execution of processes within a partition.

Processes inside a partition are scheduled with preemptive scheduling. In ARINC 653, a process can be dormant i.e. ineligible for scheduling, waiting, ready to be executed, or running. The access of a process to the processor resource depends on its priority level and its current state. Within a partition, the process in the ready state with the highest priority has first the access processor resource. In the case of equality in terms of priority, the resource is allocated to the process that has remained in the ready state the longest. Only higher priority processes have the ability to preempt a less priority process.

3.4.2.2 Scheduling of ARINC 653 systems

In ARINC653, the scheduling is a two-level scheduling: partitions scheduling and processes scheduling.

The scheduling of the partitions is fixed, off-line, and cyclic while the processes inside a partition are schedule with an online scheduling policy. The off-line partition scheduling is a configuration table system. In this table, time is organized as a regular time frame called major time frame (MAF).

Definition 55. *(**Major time frame**) MAF is a fixed duration periodically repeated throughout the runtime operation [5].*

MAF is repeated till the end of the runtime and corresponds to the multiple of the least common multiple of all partition periods in the system. The MAF is also divided into smaller regular time frames called minor time frames (MIF) that corresponds to the PGCD of the partitions periods.

Definition 56. *(**Minor time frame**) MIF [76] is defined by the PGCD of the partitions periods.*

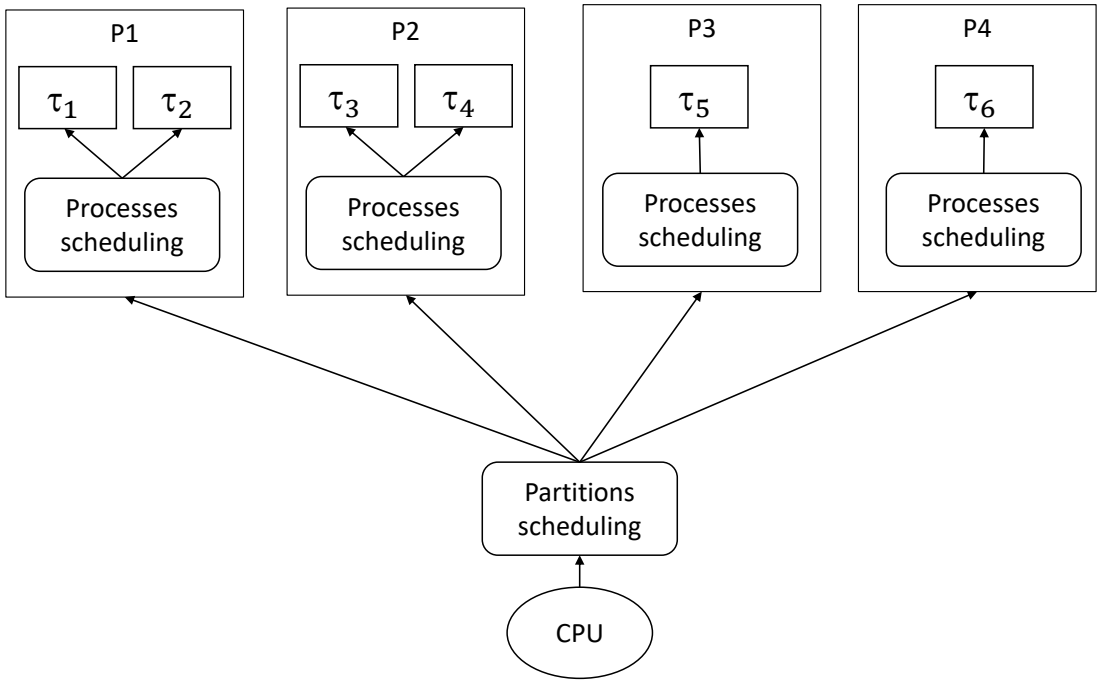
Each partition execution, during its period, is divided into time slot called partition windows. A MIF is composed of partition windows of several partitions.

Definition 57. *(**Partition window**) A Partition window represents an integral time duration during which the Operating System exclusively schedules processes belonging to a given partition [77].*

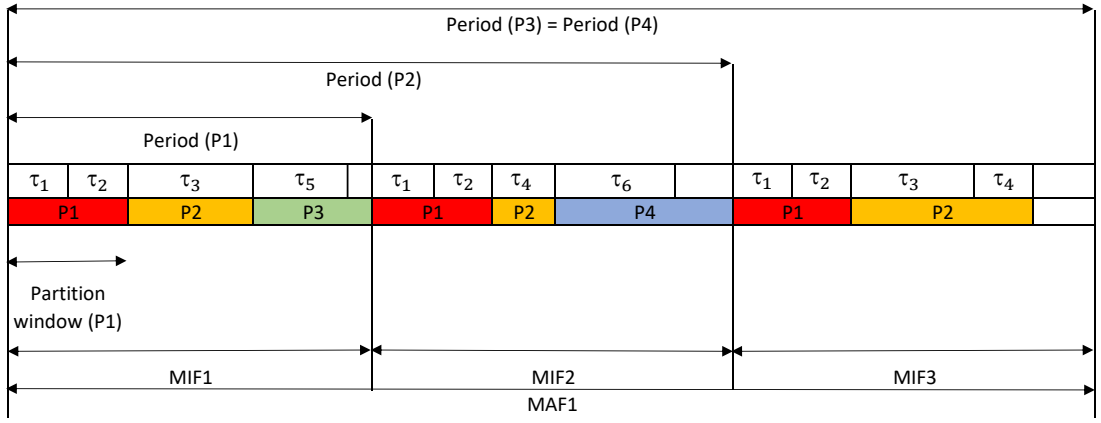
Figure 3.7a shows an example of ARINC 653 system. We have six processes (τ_1,τ_2,τ_3,τ_4,τ_5, and τ_6) assigned to four partitions ($P1$, $P2$, $P3$, and $P4$). Figure 3.7b presents an example of scheduling of this partitioned system. This example shows scheduling of the system during a MAF divided into three MIF. Tasks are executed only during the partition window of their assigned partitions based on their priority level. Then even if tasks τ_1 and τ_2 are both assigned to the same partition $P1$, task τ_1 being executed before task τ_2 implies that task τ_1 has higher priority level than task τ_2. Each partition has its own period that indicates its activations. We remark that the MAF contains three partition windows of partitions $P1$ and $P2$ while it only has one partition window of partition $P3$ and partition $P4$.

3.4.2.3 Communications

In ARINC 653 specifications, communications are operated based on inter and intra-partition communications concepts.



(A) Example of an ARINC 653 architecture



(B) Illustration of an ARINC 653 scheduling

Definition 58. *(**Intra-partition communication**) The intra-partition communications represent communications between processes within a same partition [74, 63].*

Intra-partition communications are completed via buffers or/and blackboards. Buffers prevent data overwriting by maintaining a message queuing that stores messages that will be transmitted by FIFO. Blackboard maintains a message until its transmission or overwriting with a new message. Semaphores and events are also used to synchronize processes inside a partition.

Definition 59. *(**Inter-partition communication**) Inter-partition communications are communications between partitions [74, 63].*

Inter-partition communications include communications between a partition on a core module and any other component non-compliant to ARINC 653 and external to this core module.

They are performed via messages through ports and channels. Each partition has its own port to send and receive messages from other partitions. A channel is considered as a link between ports of the communicating partitions. In a partition, a port can be used by many processes of this partition. Each port has to be configured in sampling or queuing ports mode. At the sampling mode, the message is kept at the source port until it is transferred or overwritten by a new message. In the queuing mode, messages are maintained in a FIFO message queue. It is important to mention that in ARINC 653 systems, inter-partition communication sinks are partitions, not processes.

3.5 Examples of hypervisors/operating systems for TSP systems

In TSP systems, space and time isolation are most of time enforced by an hypervisor. This section introduces some hypervisors/operating systems dedicated to TSP systems summarized in Table 3.1. The table provides for each hypervisor, its authors, supported platforms, and guest OS. Further, the section presents for each hypervisor, its characteristics such as their scheduling, memory management and communications mechanisms.

3.5.1 PikeOS

PikeOS [78, 50] is a real-time operating system and a hypervisor for safety and security-critical systems that supports several processor architectures such as

Hypervisor/ RTOS	**Authors**	**License**	**Supported platform**	**Supported Guest OS**
PikeOS	Sysgo AG	Proprietary	ARM, NXP, X86, PowerPC, LEON, SPARC	PikeOS native, Linux, POSIX, AUTOSAR, Android, RTEMS, OSEK, ARINC 653 APEX, ITRON
Xtratum	fentISS	Open-source	PowerPC, LEON2/3/4, SPARC v8, ARM v7	Linux, LithOS, RTEMS
POK	Julien Delange and al.	Open-source	x86, PowerPC, LEON 3	ARINC 653 APEX
Lynxsecure	LynuxWorks	Proprietary	x86	LynxOS, Linux, Windows

TABLE 3.1: Summary of hypervisors and RTOS for TSP systems

ARM, NXP, X86, and Leon/Sparc. It ensures system application protection through partitioning and supports mixed-criticality systems with different levels of security and safety. Each partition can hold a different operating system, API, or runtime environment such as PikeOS native, ARIN653, Linux, Android, Posix, RTEMS, AUTOSAR. PikeOS has its development environment named CODEO [78] based on Eclipse.

3.5.1.1 Scheduling

PikeOS considers that applications are composed of tasks made of threads. Threads are divided into sets named time partitions. Each time partition contains only threads with the same priority level. Inside a time partition, PikeOS considers priority-based scheduling where round-robin is applied for the thread of the same priority list.

The execution of a thread assigned to a time partition can be performed only in the activation time of the time partition.

PikeOS authorizes the simultaneous activation of two time partitions. The first one called background partition is always actived. The remaining partitions called

foreground partitions are activated one at a time. A foreground partition is activated based on a cyclic and predefined scheduling configuration.

PikeOS proposes a more general scheduling process than the scheduling in the ARINC 653 standard. The scheduling of the ARINC 653 standard is a particular case of PikeOS scheduling that is obtained by always activating at a time only one-time partition for its duration [50].

3.5.1.2 Memory management

PikeOS divides the global memory into subsets called kernel resource partitions. The resource partitions are configured through a specific system call. At creation time, each task is assigned to a kernel resource partition. Then each partition is defined by a set of kernel resource partitions assigned to its tasks. After initialization, the memory assignment to partitions in PikeOS is static (i.e no extra resource is can be allocated to a partition). Inside a partition, every guest system can establish its memory management.

3.5.1.3 Communications

In PikeOS, basically, tasks can only communicate with their parents. However, each task has a communication right that defines the tasks they are allowed to communicate with. The right to communicate with another task different from the parent can be granted by the parent. Two threads of different tasks can communicate only if their tasks have the right to communicate with each other [50].

3.5.2 Xtratum [1]

Xtratum [1] is a bare-metal hypervisor based on paravirtualization. It is designed to meet the time and space requirements of safety-critical systems. Xtratum can guest simultaneously several operating systems and supports Leon2/3/4 (Sparc v8) and ARM architectures. Natively, Xtratum is not compliant with the ARINC 653 standard. For example, a partition in Xtratum architecture is a virtual machine, not a set of processes as defined in the ARINC 653 standard. However, Xtratum partitions can be adapted to the standard for providing services like the ARINC 653 scheduling policy for example.

3.5.2.1 Scheduling

Since partitions cannot run directly on top of Xtratum. Each partition is composed of an operating system on top of which applications are run. Partitions

are scheduled based on a fixed and cyclic scheduling policy similar to ARINC 653 scheduling that consists of a periodic repetition of the MAF. As in ARINC 653, each partition is characterized by a slot time defined by a start time and the duration for its execution. The scheduling of processes inside a partition is not handled by Xtratum. Internally, each partition defines its own scheduling policy.

3.5.2.2 Memory management

In Xtratum, memory protection is implemented by the use of MMU or Write Protection Register (WPR) provided by the hardware. To ensure spatial isolation of partitions, each partition is characterized by a starting address and a size.

For the systems deployed on a processor with a MMU, the MMU only acts as a Memory Protection Unit (MPU) where there is no difference between virtual and physical address spaces and each partition executes in its designated addresses. In this case, memory is divided into several segments allocated to partitions. A partition may host several segments.

For the systems without MMU, the WPR is required to define some characteristics of the address spaces. Thus, each address space allocated to each partition has to be contiguous, should not exceed a size of 32KB and the start address has to be a multiple of the size. Unfortunately, WPR cannot completely guarantee memory isolation since it does not control read memory operations.

3.5.2.3 Communications

Communications between partitions are performed by messages through ports and channels as in the ARINC 653 standard. Xtratum implements inter-partition communication using sampling and queuing ports and manages the encapsulation of the messages. The communications inside partitions are managed by the partition developers.

3.5.3 POK

POK [79] is a partitioned operating system for safety-critical real-time systems, compliant with ARINC 653 standard, and designed for x86, PowerPC, and Leon 3 platforms. It is composed of a microkernel and a partition runtime. The microkernel intervenes in time and space partitioning of applications of different domains such as avionic, automotive.

3.5.3.1 Scheduling

The scheduling is similar to ARINC 653 scheduling. A fixed time slot is assigned to each partition. POK defines the major time frame as the sum of the time slots of the partitions.

3.5.3.2 Memory management

A unique memory segment is assigned to each partition. In the microkernel, each partition stores information about its processes. This information is used by the microkernel to enforce memory isolation.

3.5.3.3 Communications

In POK, inter-partition communications use sampling and/or queuing port. Intra-partitions communications are performed with buffers and blackboards as in the ARINC 653 standard.

3.5.4 LynxSecure

LynxSecure [80] provided by Lynx Software Technologies is an hypervisor mostly used for military applications. It is based on the MILS architecture (detailed in section 4.3 of the chapter 4) to guarantee high assurance requirements by partitioning data and resources and ensuring information control. LynxSecure can host several applications and operating systems in different secured partitions that prevent from risky interactions.

It supports full virtualized operating systems and para-virtualized operating systems. For example, it supports para-virtualized Linux and LynxOS real-time operating systems, full virtualized Windows operating system.

3.5.4.1 Scheduling

By default, in LynxSecure, partitions are scheduled as in ARINC 653 standard using fixed and cyclic off-line scheduling; but for more flexibility, it allows dynamic scheduling policies for tasks.

3.6 Conclusion

This chapter consists of a presentation of hierarchical RTS. It starts by describing hierarchical RTS through definitions and characteristics. It introduces the

resources allocation modeling in hierarchical RTS while presenting the periodic resource model and the bounded delay resource model. It also presents some operating systems on which hierarchical RTS can be deployed to enforce protection between partitions. We describe the IMA and the ARINC 653 standard applied in avionic and space domains to decrease weight and power consumption through isolation of applications sharing hardware resources on different partitions. Finally, the scheduling, memory management and communications mechanisms are described for each operating system.

Considering that intra-partition and inter-partition communications between tasks may present some security vulnerabilities, the next chapter is dedicated to discuss confidentiality and integrity vulnerabilities and mechanisms to ensure communications security.

4

Security

This chapter presents security concepts that we address in this thesis. Section 4.1 presents the most addressed security properties in this thesis: confidentiality and integrity. Section 4.2 discusses security models based on data access control with these security properties. Section 4.3 gives a description of the Multi Independent Levels of Security (MILS) architecture. It is an architecture model designed for security properties based data access control with separation mechanisms that support both untrusted and trustworthy components. It provides time and space partitioning that corresponds to partitioned systems. MILS is an architecture model based on the partition concept. Finally, a conclusion of the chapter is given in section 4.4.

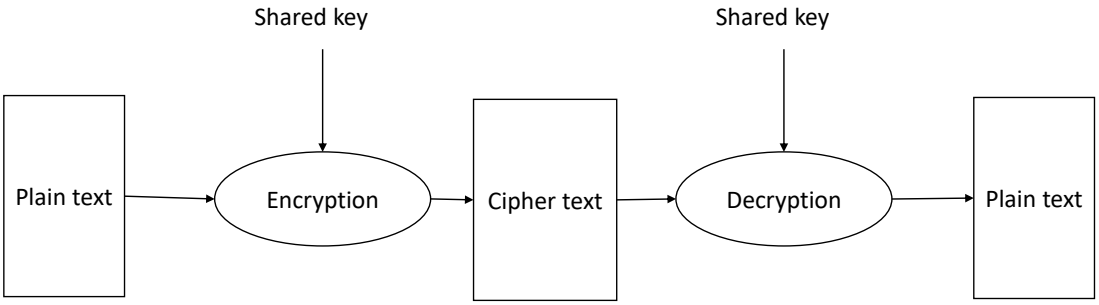
4.1 Security properties

In this section, we introduce two security properties: confidentiality and integrity [81, 82].

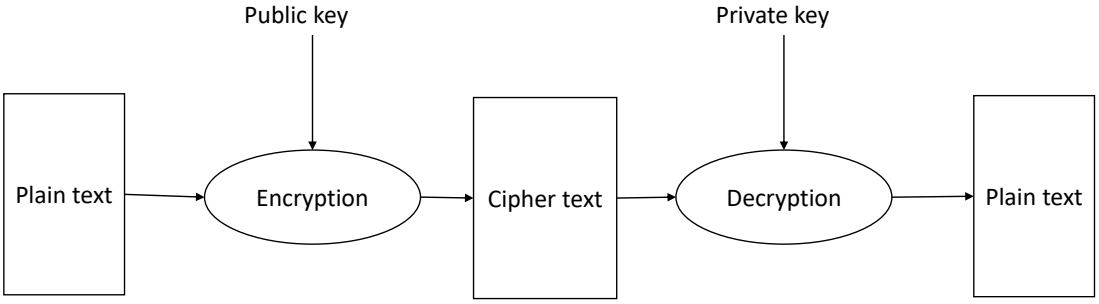
Definition 60. *(**Confidentiality**) Confidentiality is the guarantee that information is not made available or disclosed to unauthorized individuals, entities, or processes [83].*

The confidentiality requires that the information remains intelligible only to authorized entities. There are many threats to confidentiality, such as the interception of data by an intruder after sensitive data leakage or disclosure.

Confidentiality can be achieved through two principal ways. The first one consists of enforcing access control to sensitive data. This is achieved by definition of access control policies. The policies depend on definitions of subjects, objects,



(A) Symmetric encryption illustration



(B) Asymmetric encryption illustration

FIGURE 4.1: Encryption illustration

and the privileges/permissions that the subjects have on the objects. Objects represent data while subjects are components that manipulate objects by performing on them operations such as read, write, or execute. The permissions are defined with denied and allowed objects access (e.g. data access) to subjects.

The other way is to control data access through encryption. Encryption [23] is a means of securing data by encoding it mathematically such that it can only be read, or decrypted, by authorized entities. Data are encrypted with a key and only entities who possess the appropriate key can decrypt and understand the information. Depending on how the data can be encrypted or decrypted, there are two types of encryption:

1. Symmetric key cryptography [23]: uses the same key for data encryption and decryption, which must be secretly shared between the sender and the receiver.

   Figure 4.1a illustrates symmetric encryption where a same key shared between the sender and the receiver, is used to encrypt (resp. decrypt) a plain text (resp. cipher text) to a cipher text (resp. plain text). The strength of this cryptography depends on the length of the key. The longer the number of bits that represent the key, the more difficult it becomes for the external actor to guess the key and then access the information. Sometimes the key needs to be renewed to enforce the security. Blowfish [31], AES [84], DES [84] are examples of symmetric cryptography.

2. Public key or asymmetric key cryptography [85]: depends on a public key accessible by anyone and a private key possessed only by the owner. Figure 4.1b illustrates asymmetric encryption where a public (resp. private) key is used to encrypt (resp. decrypt) a plain text (resp. cipher text) to a cipher text (resp. plain text). The data is encrypted by the public key while only the private key can be used for decryption. Diffie Hellman [86], RSA [84] are examples of well-known asymmetric key methods.

Data divulgation addressed by confidentiality is not the only challenge in data protection. Unauthorized entities can corrupt or change data intentionally or accidentally if their actions are not properly controlled. Then it is also important to guarantee data integrity when needed.

Definition 61. *(**Integrity**) Integrity is the ability to prevent data from unauthorized modifications (i.e. tampering) [87, 88].*

Data integrity can be achieved through data access control policies. It can also be achieved through data hashing cryptography. Hashing [89] is a mean that ensures data integrity through mathematical algorithms that transform data into a hash value. The sender transmits the data with its hash value. Then on receipt, the receiver will hash the data and compare its hash value with the one computed by the sender. If both hash values are equal, data integrity can be confirmed. Otherwise, the received data has been modified by an attacker. There are many cryptographic hash algorithms: MD5 [90], SHA-1 [91, 92], SHA-2 [92, 93].

These hash algorithms can be associated with a secret shared cryptographic key to enforce integrity with authentication [94] by not only ensuring that data has not been modified but also ensuring to the receiver that the received data originates from the claimed source. This association is referred to as hash-based message authentication code (HMAC) [95]. HMAC is based on a message authentication code which is a value computed by the sender (resp. receiver) at the emission (resp. reception) with the data to send (received data), the hashing algorithm, and the secret key shared between the sender and the receiver.

As illustrated in figure 4.2, at the emission, the sender sends the data and the produced HMAC to the receiver. At the reception, the receiver computes its HMAC with the received data and the secret key. Then it compares its HMAC and the received HMAC. If both hash values are equal, data has not been modified and the sender is confirmed to be part of entities sharing the same key with the receiver [95]. Then the data integrity and authentication can be confirmed. Otherwise, the received data has been modified by an attacker or not sent by the claimed sender.

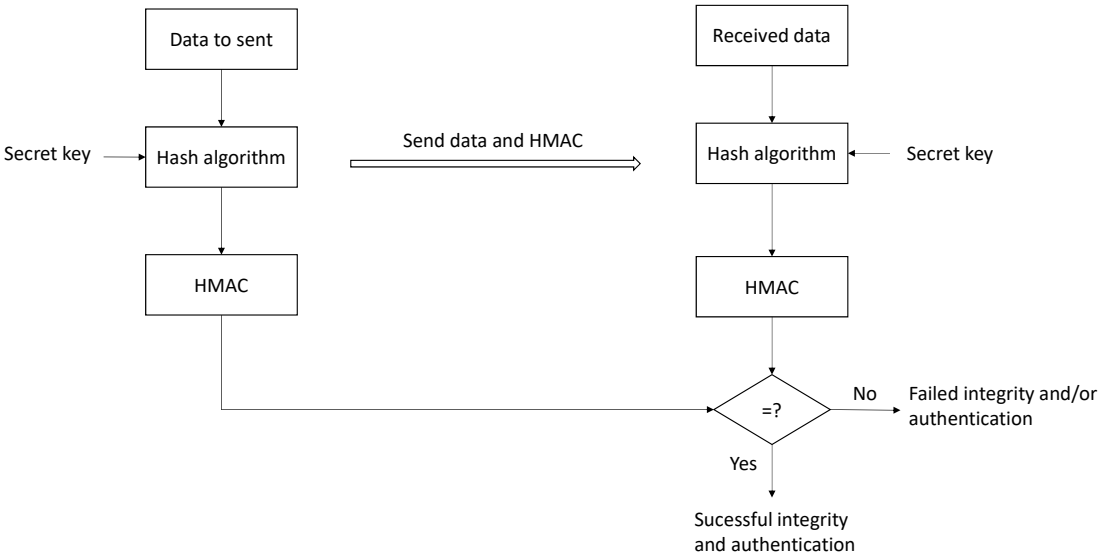


FIGURE 4.2: HMAC illustration

4.2 Security models

Security properties are enforced by security models.

Definition 62. *(**Security model**) A security model [96] describes the security strategy for a system to ensure security properties (e.g. confidentiality, integrity).*

It is an implementation of mathematical and analytical assumptions mapped to a system specification to resolve security issues. As introduced in the previous section, confidentiality and integrity can be achieved through data access policies where subjects have granted permissions that permit or deny them data access. Then security labels are assigned to the subjects and data to define these permissions. For this purpose, systems are composed of subjects of different levels of security that access data of different levels of security. The levels of security of subjects and data are defined based on the data access policies.

These systems may use classifications, such as the United States government classification system [97], which is based on the degree of secrecy and level of sensitivity. Classification levels can be confidential, top-secret, and secret. They are applied to subjects or objects. Objects can be data classified at different levels and subjects make operations such as read, write or execute on objects.

The literature proposes several security models such as Information Flow Control (IFC) models [96], Graham-Denning model [98], State-Machine model [99], non-Interference model [100].

Bell-La Padula [87] and Biba [101] are concrete examples of IFC models.

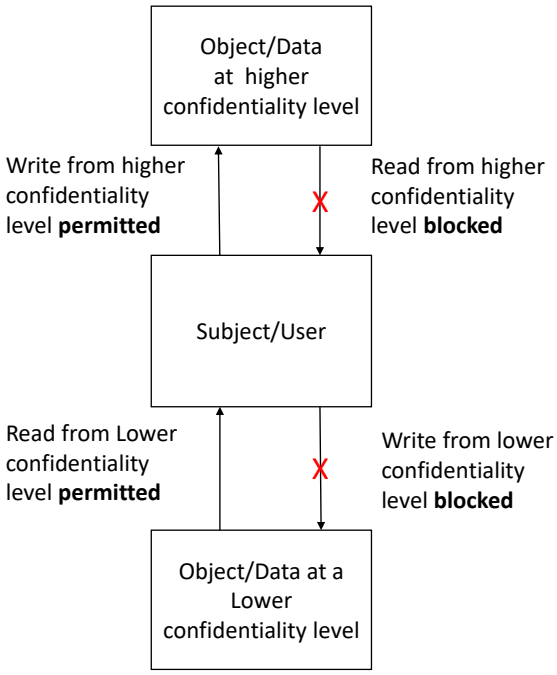


FIGURE 4.3: Bell-La Padula model illustration [7]

Bell-La Padula (BLP) model [87] was introduced to formalize the U.S. Department of Defense (DoD) multilevel security [97].

Definition 63. *(**Bell-La Padula model**) Bell-La Padula (BLP) model is a security model intended for confidentiality and based on the No read up-No write down principle [87].*

It specifies that a subject at a given confidentiality level is forbidden to read data tagged with a higher confidentiality level. It cannot also write information to a lower confidentiality level.

Figure 4.3 illustrates the BLP model by showing allowed and forbidden information flow between a subject and two objects at different confidentiality levels. The first object at the top has a higher confidentiality level than the subject while the second object at the bottom has a lower confidential level than the subject. The arrows with red crosses represent the forbidden data access defined by the No read up-No write down principle. The remaining data accesses represented by arrows without crosses are allowed.

Definition 64. *(**Biba model**) Biba model is a security model developed towards data integrity and based on the No read down-No write-up concept [101].*

The Biba model describes a set of access control rules designed to ensure data integrity. It defines an integrity policy that a subject may not read data of lower

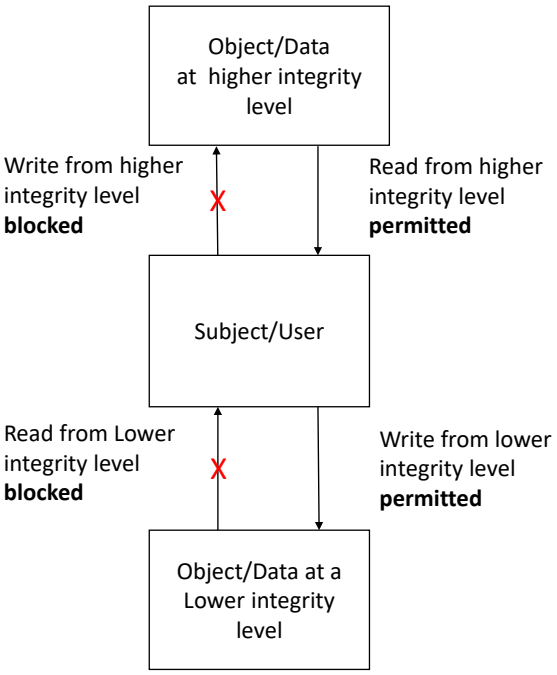


FIGURE 4.4: Biba model illustration [7]

integrity level than its own, and a policy that a subject may only write data whose integrity levels are equal or lower than its own [7].

With the Biba model, a subject at a given integrity level is forbidden to read data from a lower integrity level and to write data to a higher integrity level.

Figure 4.4 illustrates the Biba model by showing allowed and forbidden information flow between a subject and two objects at different integrity levels. The first object at the top has a higher integrity level than the subject while the second object at the bottom has a lower integrity level than the subject. The arrows with red crosses represent the forbidden data access defined by the No read down-No write-up principle. The remaining data accesses represented by arrows without crosses are allowed.

4.3 Security architecture

Security architecture uses an architectural view of the system to comply with security properties. Then in this section, we present an example of security architecture called the Multi Independent levels of security (MILS) architecture.

Definition 65. *(**Multi Independent levels of security (MILS) architecture**) MILS is a high-assurance security architecture characterized by untrusted and trusted components, based on security models such as information control by*

ensuring that systems are non-bypassable, evaluable, always invoked and tamper-proof [102].

MILS is based on divide and conquer to make easier systems manipulation and evaluation [81, 82]. Therefore, MILS adopted a classification of its components based on the degree of criticality by assigning to them security levels. According to that classification, MILS applications are tagged as Single Level of Security (SLS), Multiple Levels of Security (MLS), or Multiple Single Level of Security (MSLS) applications [103, 81, 82].

4.3.1 MILS classification

For security purposes, in MILS architecture, each object/data is characterized by a security level. Subjects are also classified based on the security levels of the objects/data they manipulate:

Definition 66. *(**Single Level of Security (SLS)**) A Single Level Secure Component is a component that every time processes data of one security level [103].*

Definition 67. *(**Multiple Level of Security (MLS)**)*

A Multi-Level Secure Component is a component that handles information with different security levels concurrently during one runtime instance [103].

MLS components process data at different levels of security without a separation between security levels. A device that downgrades an object at a given level of security to a lower level of security, is an example of an MLS component.

Definition 68. *(**Multi Single Level of Security (MSLS)**) A Multiple Single-Level Secure Component is a special kind of SLS component that processes data of multiple security levels, but always maintains separations between classes of data by exclusively processing only one security level during its runtime instance [103].*

MSLS components process data at different levels of security with a separation between security levels. Therefore, data will not be downgraded or upgraded.

4.3.2 MILS architecture

MILS architecture based systems are layered systems. Most of them are composed of an application layer, a middleware service layer, and a separation kernel. Figure 4.5 gives an overview of the MILS architecture. It shows the application layer deployed on a middleware service layer running on top of a separation kernel. The separation kernel divides the system into separate partitions where the

middleware and applications are located. The middleware provides an interface to applications or a virtual machine enabling operating systems to be executed within partitions. MILS prevents information leakage from one partition to another. It also provides controlled information flow between partitions represented with arrows.

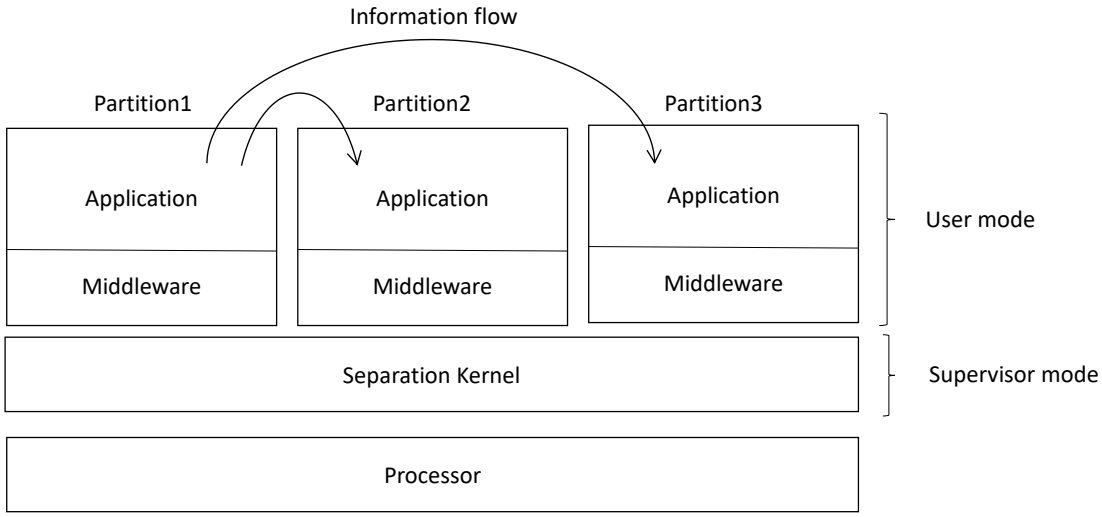


FIGURE 4.5: MILS overview [8]

4.3.2.1 Separation Kernel

Definition 69. ***(Separation Kernel)***

The separation kernel (partitioner layer) is the base layer of the system, and is responsible for enforcing data separation and information flow controls within a single microprocessor; providing both time and space partitioning [102, 104].

The separation kernel ensures data separation by partitioning the memory base on the hardware memory management unit [105]. Information in a partition is only accessible for subjects of this partition, but by controlling information, the separation kernel allows information flow between partitions by ensuring that it has been explicitly authorized and configured. Any other communication is considered as violating security rules.

The separation kernel guarantees fault isolation by ensuring that faults detected in one partition do not spread to other partitions. It ensures that all shared resources are cleaned before being used by subjects of another partition.

A separation kernel has to be as small and simple as possible to easies its verification. PikeOS [106], Xtratum [1], POK [79], Lynxsecure [80] are examples of separation kernel.

4.3.2.2 Middleware service

In a MILS architecture, it is expected to have a small separate kernel. Therefore many of the services traditionally provided by conventional operating systems such as file systems, device drivers, I/O services are located in the middleware layer running in non-privileged mode [107, 108].

The middleware layer contains end-to-end data processing services as well. It can provide services that help to enforce the information flow security through labeling, filtering, and the maintenance of only authorized communications specified by the system designer.

Middleware services are responsible for verifying and filtering data that are not correctly labeled in the communication between two partitions. A guard is an example of a middleware service destined to enforce information flow security.

Definition 70. *(**Guard**) A guard in the MILS architecture is a process that intervenes between communications to verify if they respect application-level security rules [109].*

4.3.2.3 Application layer

In a MILS architecture, the application layer intervenes to enforce security policies by hosting some applications that implement specific security rules [110].

Information control in MILS architecture is based on communications between components in the same security domain, or communications through security monitors such as MILS Message Router (MMR), encryption devices, downgraders, upgraders, collators that are part of the application layer.

Definition 71. *(**MILS Message Router**)*

MILS Message Router (MMR) [111] is a MILS component that enforces communication classification between partitions by receiving data at different levels of security from multiple partitions and routing them to the correct recipient partition.

During the message transmission, when the MMR concludes that the communication between the involved partitions is allowed, the message is routed to the appropriate guard which verified if the concerned processes are allowed to communicate. Each protocol required a specific guard responsible for analyzing the received message and checking the verification of the communication protocol. When the guard notice that the communication is not allowed or the message does not respect the communication protocol, a message is sent to the MMR to discard the message. Then an error message is eventually notified to the sender.

Each communication between partitions shall pass through the MMR that verifies if this communication is allowed. MMR contains a static multi-dimensional array that specifies the allowed communications between partitions. It also contains an internal memory package with a set of pointers that helps each process to own a part of the memory. The MMR does not need to know the content of the message. It only analyzes the message header to identify the sender and the destination. The action of the MMR may require additional security components for the message to arrive at the final destination.

Definition 72. *(**Downgrader**) Downgrader is a component that transmits data from a process at a given security level to another process at a lower security level [102] [112].*

Information flow from a higher subject to a lower subject can be considered as information disclosure and is not allowed. Sometimes that operation can be needed and then explicitly authorized in the security policies. This information flow is sensitive and then has to pass through a trusted component. Therefore, the communication needs to operate through a downgrader. A downgrader can be a filter that restricts the information that can be transferred [112]. An encrypter that transforms information into an unintelligible form before transferring it to the lower security subject, can also be considered as a downgrader.

There are also components called upgraders.

Definition 73. *(**Upgrader**) Upgrader is a component that that transform data from a process at a given security level to another process at a higher security level.*

Decrypters are examples of upgraders.

4.4 Conclusion

This chapter introduces security concepts that may be considered when designing partitioned systems. It describes confidentiality and integrity properties and the means to achieve them such as encryption and hashing cryptography. It also presents security models based on MLS that define rules to enforce data access control. Finally, it presents an example of high-assurance security architecture called MILS that characterizes untrusted and trusted components based on security models in partitioned systems.

5

Multi-objective optimization

This chapter presents a background of multi-objective optimization, which is the main technique used to solve the scheduling and security problem in our thesis. Section 5.1 describes multi-objective optimization problems (MOOP) mathematical formulation. It also presents the key concepts in multi-objective optimization such as dominance concept, Pareto set, solutions feasibility. Section 5.2 presents the scalarization methods used traditionally to solve MOOP. They only propose a single solution rather than a set of solutions as proposed by the direct approaches such as multi-objective evolutionary algorithms (MOEA). Section 5.3 describes MOEA. It also presents some metrics to evaluate the solution sets proposed by the MOEA for a given MOOP. Finally, a conclusion of the chapter is given in Section 5.4.

5.1 Definitions and characteritics

MOOP addresses problems with multiple mutually conflicting objectives to be optimized simultaneously. The complexity of these problems lies in the fact that the optimization of one objective can lead to the degradation of another one. On contrary, single-objective optimization addresses problems with only one objective that can be optimized to provide optimal solution. It is often difficult or impossible to find an optimal solution that optimizes all the objectives simultaneously. Then it is necessary to propose solutions with trade-offs between objectives. The decision maker has to choose a solution among the good generate tradeoffs, maybe according to external and/or subjective criteria not taken into account in the optimization model.

MOOP are faced in multiple domains such as economics, logistics, chemistry,

engineering, industry. We also faced them in our daily life. For example, buying a car with maximum comfort at a minimum cost is a MOOP. On contrary, buying a car with the minimum cost is a single objective optimization problem. An MOOP can be formulated mathematically as follows [113, 114]:

$$\begin{cases} Optimize\ F(X) = (f_1(X), f_2(X), ..., f_k(X)), & \text{where } X = (x_1, x_2, ..., x_n) \\ & \text{and } k \geq 1 \\ g_j(X) \geq 0, & \text{where } j \in 1, ...m \end{cases} \tag{5.1}$$

$X = (x_1, x_2, ..., x_n)$ represents the vector of decision variables. $f_i(X)$ corresponds to the i^{th} objective function to optimize (i.e. to minimize or maximize) with $i \in 1, ..., k$. k is the number of objective functions to optimize. We highlight that the maximization of an objective function is equivalent to the minimization of its negative. $g_j(X)$ is a function that formulates the inequality constraints that must be applied to each solution. These constraints conditioned the feasibility of the vector of decisions.

In the resolution of MOOP, we are interested in the best solutions among the feasible solutions in regard of the objective functions. The search for the best solutions implies comparing solutions with each other. Then the comparison between solutions is often performed based on the Pareto dominance principle [115].

Definition 74. *(**Pareto dominance principle**) Considering a minimization problem, a solution X_1 dominates a solution X_2 (i.e. $X_1 \prec X_2$) , if and only if $\forall i \in 1, ..., k, f_i(X_1) \leq f_i(X_2)$, and $\exists j \in 1, ..., k$ such that $f_j(X_1) < f_j(X_2)$ [115, 116].*

It states that a solution X_1 dominates a solution X_2 if and only if the following conditions are both respected:

1. For all the objectives functions, X_1 is at least as good as X_2.
2. At least for one objective function, X_1 is strictly better than X_2.

If $X_1 \not\prec X_2$ and $X_2 \not\prec X_1$ then X_1 (resp. X_2) is not dominated by X_2 (resp. X_1). The solution X_1 is better than solution X_2 at least for one objective function and X_2 is better than X_1 for at least another objective function. Then X_1 and X_2 are considered as non-dominated to each other.

Definition 75. *(**Non dominated solution**) A solution X is a non-dominated solution [113, 114], if and only if $\nexists X' \in D, X' \prec X$.*

Where D represents a set of feasible solutions (i.e. solutions that respects the defined contraints). It means that a solution is non-dominated in a set of solutions, if there is no solution in this set that dominates this solution.

For a given MOOP, in the set of feasible solutions, there is a smaller set of solutions that are not dominated by any of the feasible solutions. This smaller set is called the Pareto set and contains the non-dominated solutions of the set of feasible solutions.

Definition 76. *(**Pareto set**) A Pareto set PS is defined as follows* $PS = X \in D, \nexists X' \in D$ *such that* $X' \prec X$ *[116].*

Definition 77. *(**Pareto front**) The representation of a Pareto set's solutions through their objective functions is called a Pareto front. It corresponds to the image of the Pareto set. A Pareto front (PF) is defined as follows* $PF = F(X)$ *such that* $X \in PS$ *[116, 114].*

Figure 5.1 shows an illustration of a Pareto front of a MOOP with two conflicting objectives f_1 and f_2 to minimize. Each point p_i corresponds to a solution X_i of the design space. The black (resp. red) dot represents the non-dominated (resp. dominated) solution associated points. For example solution X_i dominates solution X_j since X_i has a lower value of f_1 compared to X_j and both have the same value of f_2. Since both objectives are conflicting, the transition from one solution to another on the Pareto front is characterized by some sacrifices on one objective in order to optimize the other [117].

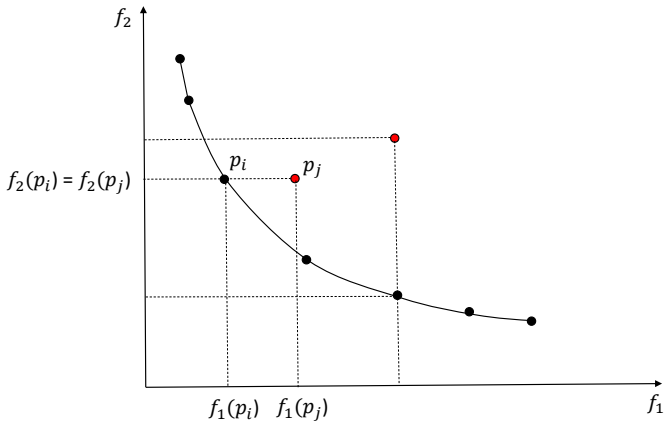


FIGURE 5.1: Illustration of a Pareto front [9]

The space of feasible solutions in large practicle instances of MOOP is usually so large that it becomes unmanageable to explore all the solutions and compare them to each other. Then it can become practically impossible to provide the Pareto set. There are different methods (detailed in Section 5.3.1) proposed to explore the space of solutions in order to find a set of non-dominated solutions

that represents as much as possible the Pareto set. This set is called an approximative Pareto set. We associated to this set, the approximative Pareto front that corresponds to its image.

As MOOP is often faced in several domains, different methods have been proposed to provide solutions. In general, there are two mains approaches proposed in the literature: scalarization and direct approaches.

5.2 Scalarization based multi-objective optimization

The scalarization methods [118] were traditionally used to solve MOOP. The scalarization consists of formulating a single objective function that combines the objective functions of a MOOP. There are multiple scalarization methods such as the weighted sum method [119, 117], the $\varepsilon - constraints$ method [120, 121], the goal programming method [122, 123, 124].

5.2.1 Weighted sum method

The weight sum method is the most popular approach. It is based on defining a weight w_i for each objective function f_i. The weight of an objective function represents its importance for the decision maker. It combines the objective functions into the following linear function:

$$\begin{cases} Optimize\ F(X) = \sum_{i=1}^{k} w_i \cdot f_i(X), & \text{where } k \geq 0,\ and\ w_i \geq 0 \\ g_j(X) \geq 0, & \text{where } j \in 1, ...m \end{cases} \tag{5.2}$$

The weighted sum method seeks Pareto optimal solutions one by one by systematically changing the weights among the objective functions. This method is efficient in generating non-dominated solutions in convex regions of Pareto front [125] and simple to use. Indeed, the weighted sum method only considers positive weights and their sum must be constant. Moreover, it is based on a convex combination of objective functions. Therefore, it cannot provide non-dominated solutions in the non-convex regions of the Pareto front [119]. The another challenge resides in the determination of the weights because they impact the solution set.

5.2.2 $\varepsilon - constraints$ method

Another well-known method is the $\varepsilon - constraints$ method. It consists of choosing one of the multiple objective functions, says i^{th} as the main objective function

and formulating the remaining objective functions as constraints [121]. It can be formulated as follows:

$$\begin{cases} Optimize\ f_i(X), & \text{where } k \geq 0,\ and\ i \in 1, ...k \\ subject\ to\ f_t(X) \geq \varepsilon_t, & \text{where } t = 1, ...k,\ and\ t \neq i \\ g_j(X) \geq 0, & \text{where } j \in 1, ...m \end{cases} \tag{5.3}$$

As an advantage, it is applicable to either convex or non-convex problems. Multiple rounds of searching for solutions using a different set of constraints can identify trade-off points among multiple objectives [117]. As inconvenient, it can be difficult to formulate objectives in constrained forms.

5.2.3 Goal programming

The goal programming approach [122, 123, 124] is a multi-objective optimization that proposes solutions that tend towards targets fixed for the objective functions up to a satisfaction level. It can be formulated as follows

$$\begin{cases} Optimize\ F(X) = \sum_{i=1}^{k} |f_i(X) - T_i|, & \text{where } k \geq 0 \\ g_j(X) \geq 0, & \text{where } j \in 1, ...m \end{cases} \tag{5.4}$$

T_i corresponds to the target value fixed for each objective function f_i. The goal programming objective is then to minimize the deviation to the targets. The deviation represents the difference between the achievement f_i and the targets values T_i.

There are also the preemptive or lexicographical goal programming [123, 126] where goals are ranked. Priorities are then assigned to each goal. Goals are then classified from the highest priority to the least priority. The goals are met successively according to their priorities. In the first step, the principle is to attempt as much as possible the higher priority goal. In the second step, the second-highest priority goal is attempted while not degrading the solution obtained in the previous step. This process is repeated until the meeting of the least priority goal [127].

Goal programming focuses on providing a solution that satisfies as much as possible the goals instead of providing an optimal solution. It also allows the decision-maker to incorporate environmental, organizational, and managerial consideration into the model through the ranking of goals (e.i. goal priorities). As a drawback, it requires a priori very detailed information on the preferences of the decision-maker.

In general, scalarization based multi-objective optimization methods are simple to use, but they propose a single solution rather than a set of solutions. They

have to be run many times to provide a set of non-dominated solutions. Then direct approaches can be preferred in some cases since they are alternatives that provides a set of solutions and do not require a priori bias of objectives.

5.3 Direct approaches for multi-objective optimization

These approaches provide a set of solutions that represents trade-offs between the objective functions. Then depending on the situation, the decision-makers can choose the solutions that fit more their requirements.

The Pareto set of a MOOP can be fully provided by an exact method such as the exhaustive method. This one enumerates all the solutions of the MOOP and check their non-dominance to provide the non-dominated solutions set. As a drawback, it can be time and resource-consuming for large-scale problems since MOOP are often NP-hard [128]. In this case, the exhaustive method becomes difficult and impractical for large size instances.

Then as alternatives to this approach, there are the approximation methods. They propose a non-dominated solutions set as close as possible to the Pareto set for limited computational time and resources. This set is qualified as near-optimal or sub-optimal set. These methods are often based on metaheuristics.

Definition 78. ***(Metaheuristic [129])***

A metaheuristic is a high-level algorithmic strategy for exploring the search space of a problem and identifying optimal and near-optimal solutions.

For a given MOOP and a multi-objective method, the challenge for heuristic resides in the fact that the computed front (i.e. approximative Pareto front) has to respect the following properties [117]:

- The approximative Pareto front must be as much as possible close to the Pareto front (i.e. the objective functions values have to converge to optimal ones).
- The approximative Pareto front should contain numerous solutions that are uniformly distributed in order to be as much as possible representative of the Pareto front. Then it must also consider the extreme solutions (i.e. bounds of the objective functions).

There are multiple multi-objective metaheuritics methods such as evolutionary algorithms [130], ant colony opimization method [131], particle swarm optimization method [132], simulated annealing [133] and tabu search [134].

In the next section, we focus on the description of the multi-objective evolutionary algorithms (MOEA).

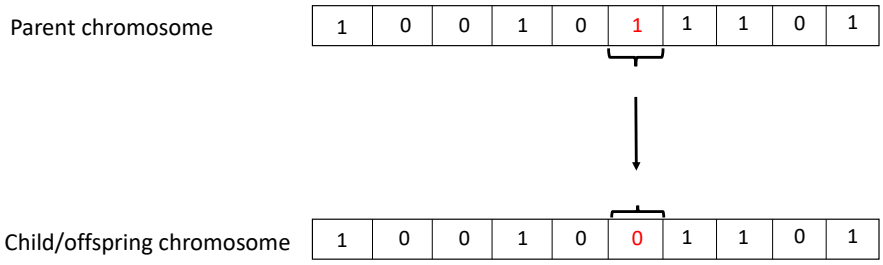


FIGURE 5.2: Illustration of mutation

5.3.1 Multi-objective evolutionnary algorithms

MOEA are widely used and known as random-based search strategies since they are among the first proposed metaheuristics. MOEA are inspired by the biological evolutionary process characterized by steps such as reproduction, random variation, selection of Darwin's theory of evolution. The reproduction consists of producing new individuals (children or offspring) from an existing population. The reproduction implies some random variation such as mutation or crossover.

Individuals correspond to solutions of the addressed MOOP. They are encoded to ease their manipulation. Often they are defined by a vector called chromosome or genotype and each position in the vector is called a gene. The encoding of solutions depends on the addressed MOOP and then can differ from a MOOP to another.

The mutation consists of producing a chromosome from a parent chromosome by changing an arbitrary gene value or exchanging gene values of the parent. Figure 5.2 shows an example of mutation of a chromosome represented by bit strings. The parent chromosome made of ten genes has a mutation operation on its sixth gene to generate the child chromosome. This operation has been performed by changing the sixth gene value from 1 to 0.

The crossover consists of combining two parent chromosomes to generate new chromosomes (offspring, children). Figure 5.3 shows an illustration of a crossover between two parents that generates two children by swapping their genes to the right of a random point (crossover). This point divides the parent chromosomes into two sections (sections A and B for the parent 1 chromosome and sections C and D for the parent 2 chromosome in figure 5.3). Then child 1 (resp. child 2) chromosome is generated with section C (resp. A) of parent 1 chromosome and section B (resp. D) of parent 2 chromosome. This example of crossover is called one-point crossover. Mutation is an operation on one individual while crossover is an operation involving several individuals.

It is important to highlight that not all the generated solutions are feasible. Some of them could not respect the constraints of the MOOP. As in the Darwinian principle, the fittest individuals are selected as survivors and the weakest

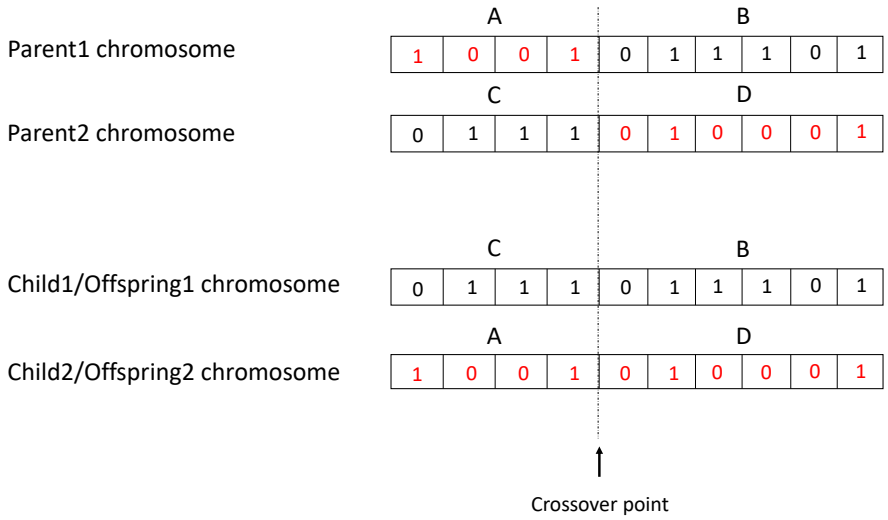


FIGURE 5.3: Illustration of crossover

individuals do not survive. Fitness functions corresponds to objective functions of the MOOP.

Figure 5.4 presents the general process of MOEA. A basic MOEA algorithm is an iterative process that starts with a given population composed of random solutions (step 1 in Figure 5.4). From this population, some solutions are selected for reproduction (step 2 in Figure 5.4). These solutions are called parents. Then the reproduction is performed through mutation and/or crossover to generate children as new candidates solutions (step 3 in Figure 5.4). The candidate solutions are evaluated according to the fitness functions that represent the objective functions of the MOOP addressed (step 4 in Figure 5.4). Non-feasible solutions among the new solutions are automatically rejected (step 5 in Figure 5.4). A subset is selected for next generation with a randomized process that flavors the ones with good fitness values. Then the population is updated with the selected solutions. New parents are selected to perform the next generation (step 2 in Figure 5.4). The process is repeated till the prefixed termination criteria are satisfied. The reaching of a number of iterations or a convergence to a stable Pareto set are examples of termination criteria.

The literature proposes multiple MOEA such as nondominated sorting genetic algorithm (NSGA) [135],and Pareto archived evolution strategy (PAES) that we propose to describe in the next section.

5.3.1.1 Nondominated sorting genetic algorithm

NSGA differs from other MOEA through its selection process characterized by a nondominated sorting and crowding distance calculation [136] (defined below).

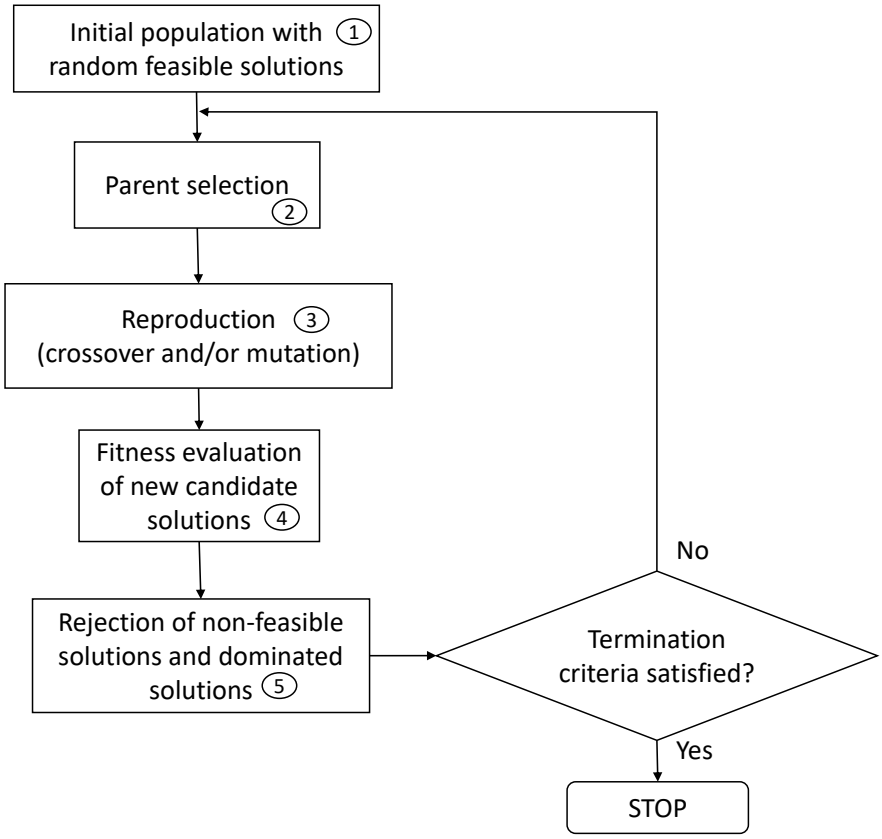


FIGURE 5.4: Multi-objective evolutionnay algorithm

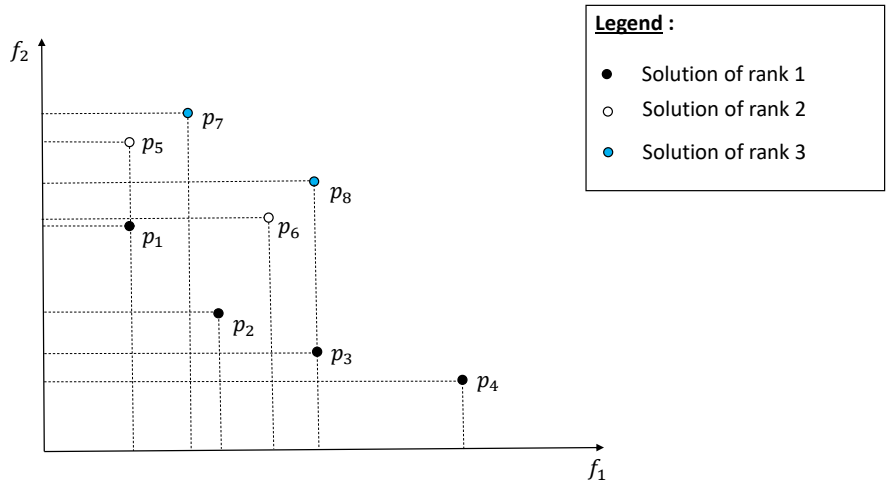


FIGURE 5.5: NSGA ranking Illustration [10]

Then it starts with a population of size N where each solution has a non-domination rank level (i.e. a solution at level 1 is dominated by no solution, but a solution at level 2 is dominated by one or more solutions at level 1, and so on).

Figure 5.5 proposes an illustration of a NSGA ranking with two objectives functions f_1 and f_2 to minimize. It shows three levels of rank (rank 1, rank 2, and rank 3). Each rank is composed of points that correspond to solutions. There is no solution that dominates the solutions of rank 1 ($p1$, $p2$, $p3$, $p4$), but solutions of the others ranks are dominated by solutions of lower ranks. For example, the solution $p5$ of the rank 2 is dominated by the solution $p1$ of rank 1. However, solutions at the same rank are non-dominated among each other.

From this population, it is generated a children population of size N through mutation operation for example. Both populations are merged and the resulting population is divided into several mutually exclusive equivalent classes. These classes are ordered based on the degree of Pareto dominance. The solutions of each class form a front characterized by a non-domination rank. From the new population of size $2 \cdot N$, the next generation parent population is created by selecting solutions from the rank 1 to the i^{th} rank such as the number of solutions do not exceed the size N.

In the case where all the solutions of the i^{th} rank cannot be kept, the crowding distance calculation mechanism is applied to select some solutions while preserving the diversity among the population.

Definition 79. *(**Crowding distance calculation**) Crowding distance calculation of a point p_i corresponds to evaluating the size of the largest cuboid enclosing p_i without including any other point in order to estimate the density of solutions surrounding this point in the population [137, 136].*

For each front, a crowding calculation is conducted to compute the distance between neighboring solutions. Two rules are defined for solutions selection. First, between two solutions of the same front (i.e. same non-domination rank), the solution with the high crowded distance should be preferred. This means that the solution with the less crowded region is preferred. Second, between two solutions on different fronts (i.e. different non-dominant ranks), the solution with the lower non-domination ranks should be preferred.

NSGA ensures diversity of the solutions in the provided Pareto set. However, the crowded distance calculation mechanism can restrict the convergence. It also has a high computational complexity of $O(M \cdot N^3)$ where M and N are respectively the number of objectives functions and the size of the population. Then an improvement of NSGA called NSGA-II [138] has been proposed with a fast non-dominated sorting procedure with $O(M \cdot N^2)$ computational complexity.

5.3.1.2 Pareto Archived Evolution Strategy (PAES) overview

PAES is an iterative evolution strategy. It is a $(1 + 1)$ local search evolution strategy because it is based on a current solution c that is mutated to a candidate solution m at each iteration. If during an iteration, m is better than c, then m becomes the current solution of the next iteration. Otherwise c remains the current solution until a mutation provides a better solution. An archive of a limited size stores non-dominated solutions found at each iteration.

Figure 5.6 depicts the PAES process. The exploration starts with an empty archive and the generation of an initial solution that is evaluated according to the objective functions of the addressed MOOP (steps 1 in Figure 5.6). The initial solution is added to the archive (step 2). The archive is characterized by a predefined maximal number of solutions that it can contain. At the beginning, the initial solution is considered as the current solution c that is mutated to a candidate solution m (step 3). Then m is evaluated and compared to c based on the Pareto dominance principle. We highlight that at the first iteration, the archive only contains solution c.

If m is not dominated by c or any other solution in the archive, then m is added to the archive and all the solutions in the archive dominated by m are removed from the archive (step 4). If m dominates c, then m becomes the current solution. Else if c dominates m, c remains the current solution. Otherwise (i.e. m is not dominated by c and c is not dominated by m), a solution in the archive is chosen to become the current solution (step 5).

Thus the mutation, comparison, and archive update are repeated until the condition of the end of the exploration is met. This condition can be a number of iterations or a targeted value of objective functions.

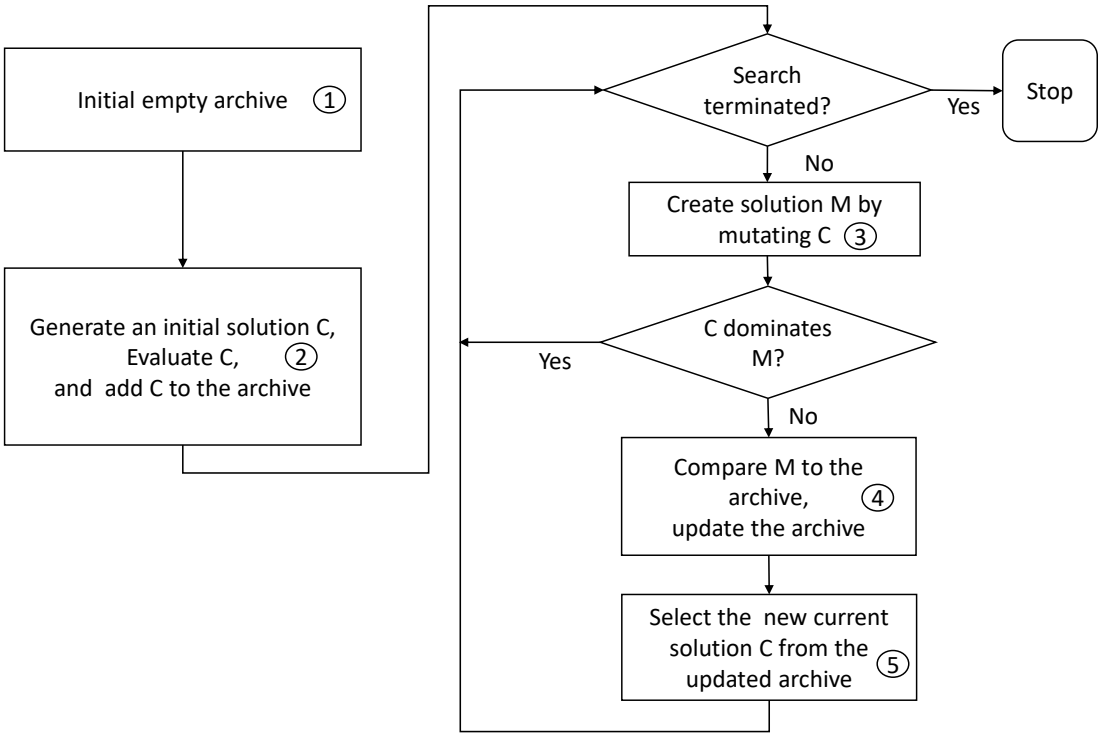


FIGURE 5.6: PAES process

In order to ensure diversity of solutions, PAES calls a crowding technique [139] based on a recursive subdivision that splits the objective space into a grid. The grid is used to verify if a solution is located in a crowded region. It helps to provide diversity among proposed solutions. It consists first by computing a grid location for each solution by bisecting recursively the space of each objective function and finding the side on which the solution resides. Each time that the number of solutions in the archive changes, the grid location of the solution should be updated.

PAES is simple to use and reduces the computational effort. PAES does not need crossover, which is an advantage for MOOPs where crossover is difficult to design when crossing of solutions often generate non-feasible solutions.

5.3.1.3 MOEA metrics

Since MOEA proposes non-dominated solutions set supposed to approximate the Pareto set, it is important to be able to evaluate the quality of the proposed approximative Pareto set.

Considering an approximate Pareto front F and a true Pareto front $F*$, there are multiple indicators such as generational distance (GD) [140], inverted generational Distance (IGD) [141], hypervolume (HV) [142, 143], that help to measure quality criteria as convergence, diversity and/or number of solutions. In the

sequel, we assume that the true Pareto front is known.

GD [140] measures the distance between an approximate Pareto front and the true Pareto front. It determines the gap between these fronts. It consists of computing the distance between each $p_i \in F$ and its closest $p_i \in F*$, averaged over the size of F as follows:

$$GD = \frac{\sqrt{\sum_{i=1}^{|F|} d_i^p}^{1/p}}{|F|} \tag{5.5}$$

Where d_i represents the euclidian distance between the image p_i of the solution $X_i \in F$ (i.e. $F(X_i) = p_i$) and the image of the nearest solution in $F*$. $|F|$ represents the cardinality of the set F. GD formula in its original form assumes $p = 2$, but later for more simplicity of interpretation and computation it was updated with $p = 1$. [141].

The lower this distance, the better the approximate Pareto front. In the best case, the solutions in this set are a subset of the Pareto set (i.e. $F \in F*$); then $GD = 0$.

GD evaluates the convergence of the approximate Pareto front since the convergence [144, 145] represents the distance between F and $F*$ [143]. As a drawback, GD is sensitive to the size of the approximate Pareto front. Thus, large approximated fronts of poor quality may be ranked highly by GD [141].

Then IGD metric is proposed as an improvement of the GD metric. IGD is similar to GD but it computes the distance between each $p_i \in F*$ and its closest $p_i \in F$, averaged over the size of $F*$. It considers every points $p_i \in F*$ on contrary to GD which considers only the points $p_i \in F*$ that are closer to the points of $p_i \in F$. Thus the IGD evaluates not only the convergence but also the diversity of F and is computed as follows:

$$IGD = \frac{\sqrt{\sum_{i=1}^{|F*|} d_i^p}^{1/p}}{|F*|} \tag{5.6}$$

The diversity of a Pareto set estimates the extent of spread among the solutions in the set [136]. The literature proposes also a diversity metric that helps to evaluate the diversity of a front. The diversity Δ can be computed as follows:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{|F|-1} |d'_i - \bar{d}|}{d_f + d_l + (|F| - 1)\bar{d}} \tag{5.7}$$

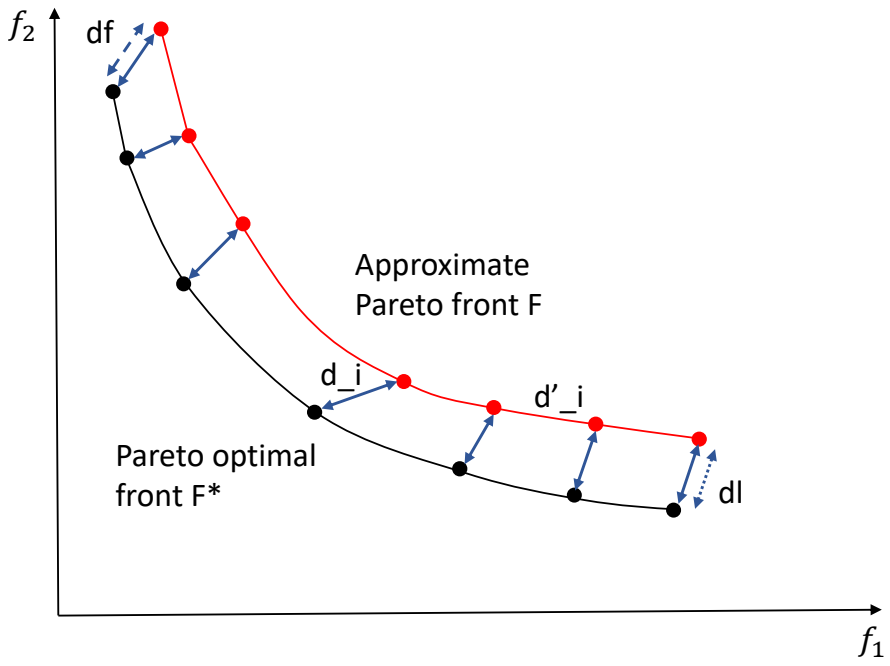


FIGURE 5.7: Convergence and diversity metrics

Where d'_i ($\forall i \in [1, |F|]$) represents the euclidian distance between the image p_i of the solution $X_i \in F$ and the nearest neighbor. $\bar{d}$ is the average of the d_i. $|F|$ represents the cardinality of the set F. d_f and d_l are the euclidean distances between the extreme solutions of F and $|F*|$ (illustrated in figure 5.7).

A low diversity metric value implies a better distribution of the solutions. In the best case, where $\Delta = 0$, the extreme points in F corresponds to the extreme points in $F*$ and then all the distances d_i equal $\bar{d}$. Thus solutions in F are considered widely and uniformly spread.

Figure 5.7 illustrates the parameters in the convergence and diversity equations. It shows a Pareto front and the optimal front of a given MOOP with two objective functions f_1 and f_2. It illustrates the distances d_f, d_l, d_i and d'_i described above.

Among the metrics, there is also the hypervolume metric.

Definition 80. *(**Hypervolume metric**) The hypervolume metric [143, 142] calculates the hypervolume enclosed by the approximated front and a reference point.*

The reference point is fixed to correspond to a solution dominated by all the front. It can be the anti-ideal point of the Pareto front also called the nadir point as proposed in figure 5.8. It corresponds to the solution that worse all the

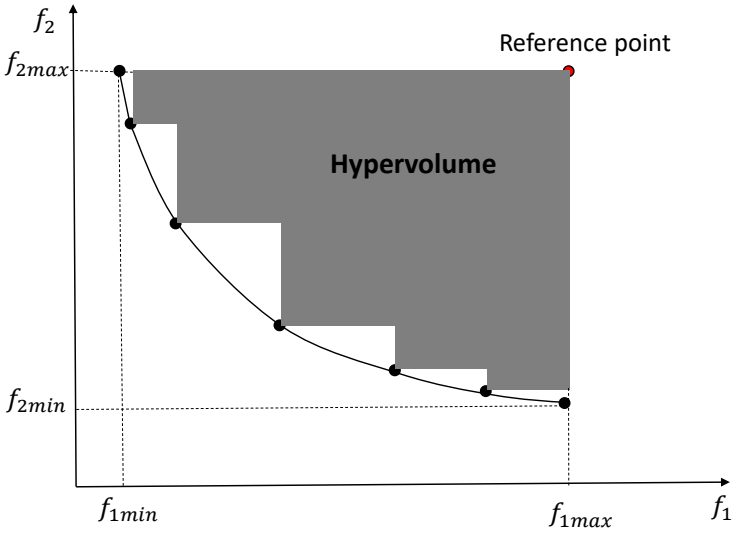


FIGURE 5.8: Hypervolume illustration

objective functions. For a MOOP with objective functions to minimize, between two fronts the front with the larger hypervolume should be preferred [143]. Two fronts can be compared by computing their hypervolume value with respect to the same reference point.

Figure 5.8 shows an illustration of a hypervolume computation of an approximate Pareto front. We can observe that the hypervolume value is influenced by indicators such as convergence, diversity, and number of solutions. They characterize the spread of solutions, then the extent of the hypervolume. This example considers a commonly used linear normalization that helps to have a small hypervolume value between 0 and 1 since objective functions values can of different orders of magnitude. Thus to each objective function f_i of each solution including the reference point solution, the following normalization is applied:

$$f_i = \frac{f_i - f_i min}{f_i max - f_i min} \tag{5.8}$$

5.4 Conclusion

MOOP are faced in multiple and diverse domains, even in our daily life. In this chapter, multi-objective optimization definitions and characteristics are presented. The chapter describes different methods proposed by the literature to solve MOOP. More details are given for MOEA especially PAES widely used in MOOP context. PAES proposes a set of solutions that approximate the optimal solutions for a limited time and resources. These solutions are trade-offs between the objective functions of the addressed MOOP. Then the decision-maker can choose the solutions according to his requirements. Finally, the chapter describes some metrics that can help the decision-maker to evaluate the quality of the solutions proposed by an MOO approach or to compare results of different algorithms.

Part II

Work orientations and positioning

6

Work orientations and positioning

This chapter is devoted to the presentation of the orientations and the positioning of our work. Sections 6.1 and 6.2 present the system model and assumptions taken in our work. Section 6.3 motivates the interest of proposing a DSE for secure TSP systems by illustrating the conflict between schedulability and security in TSP systems. Section 6.4 positions our work by comparing different approaches on security and schedulability optimization for real-time systems including our proposal. Finally, a presentation of our expected contributions and a conclusion of the chapter are given respectively in sections 6.5, and 6.6.

6.1 System model, security and schedulability assumptions

In this section, we present the assumed system model and hypothesis. We define a TSP system as a set of m applications $(A_1, ..., A_m)$. An application consists of a set of n periodic tasks noted $\tau_1, ..., \tau_n$. Each task τ_i is characterized by 8 parameters: A_i, C_i, T_i, D_i, CI_i, CL_i, IL_i and P_i.

- A_i specifies that task τ_i is part of application A_i.
- C_i, T_i, D_i are scheduling parameters. C_i is the capacity, or worst-case execution time (WCET) of task τ_i. T_i is the period of the task, i.e. the fixed duration between two consecutive releases of the task. Each task has a deadline D_i which is less then or equal to T_i. We also assume that all tasks are synchronous, i.e. they have all their first release at time 0.
- CI_i represents the tolerance of τ_i to meet its timing constraints. The possible values of CI_i are hard and soft. Tasks with hard timing constraints must

meet their deadlines while tasks with soft timing constraints may tolerate missed deadlines.

- IL_i and CL_i represent the level of integrity and confidentiality of a task, respectively. Possible values of IL_i are Low, Medium, or High, and possible values of CL_i are Unclassified, Secret, or Top_Secret.

- P_i represents the partition to which the task is assigned to. We assume r partitions (noted $P_1, ..., P_r$). Each partition is characterized by an execution time duration and a period.

The DSE we propose and the computed trade-off are an early verification of the system we design. We assumed an offline partition scheduling executed during a cyclic interval similar to major time frame (MAF) described in section 3.4.2. The MAF is supposed to be known. The MAF here can be seen as a legacy element or a budget that may be revisited after trade-off analysis. Inside each partition, tasks are scheduled based on a fixed priority and preemptive scheduling.

Tasks are assumed to communicate with each other through intra or inter-partition communications. Intra-partition communications are implemented by mechanisms similar to ARINC 653 blackboards while inter-partition communications are implemented by ARINC 653 sampling ports. We assume that applications are deployed on uniprocessor platforms.

6.2 Assumptions on security implementation

In this section, we present the security hypothesis taken in this thesis. A communication is said to be vulnerable if it violates defined security rules. We consider the security rules defined by BLP and Biba models as described in section 4.2.

In our work, a communication from task τ_i to τ_j is seen according to BLP and Biba as if τ_i writes to τ_j and τ_j reads from τ_i. We then define confidentiality and integrity violations as follow:

Definition 81. *(**Confidentiality violation**) A communication from task τ_i to τ_j is considered as a confidentiality violation if $CL_i > CL_j$. In this communication, τ_i performs a write down and τ_j performs a read up, which violate BLP's rules.*

Definition 82. *(**Integrity violation**) A communication from τ_i to τ_j is considered as an integrity violation if $IL_i < IL_j$. In this communication, τ_j performs a read down and τ_i performs a write up, which violate Biba's rules.*

We consider attacks on data integrity or confidentiality that can be operated on TSP intra-partition and or inter-partition communications.

For intra-partition communications, we assume that an attacker can access any data stored in the memory of attacked partition. With ARINC 653, there is no memory protection between tasks within the same partition, which makes all data of the partition vulnerable from any of its tasks. The attacker cannot only access data but also modify them depending on his purpose. This can be a result of a code injection attack [146] where a malicious employee injects malicious code inside a partition.

For inter-partition communications, where data are sent via ports connected by channels, we assume that an attack can be operated on ports and/or channels. An attacker can eavesdrop and get access or even modify data stored in ports. These can be achieved through attacks such as eavesdropping attack [147], spoofing attack [148] and a man in the middle attack [22].

Securing a TSP system may be made by a 2 steps process. First, BLP and Biba are used to evaluate the communications in the TSP system, and to identify those that are vulnerable. Communications that are vulnerable are those that do not respect confidentiality or integrity rules. Second, communication vulnerabilities are mitigated by adding security features.

Different implementations of security features can be investigated based on the combination of intra-partition and inter-partition communications. A security feature can be implemented through (1) function calls of a security library, or (2) dedicated tasks implementing security features.

Implementation with function calls is used in [149] where each task needing security features calls functions of a library providing confidentiality and/or integrity. These libraries implement encryption, decryption or hash functions.

Implementation of security features by dedicated tasks was proposed by [150] and the MILS architecture [102] by extending the system architecture with extra tasks implementing security algorithms.

In figure 6.1, we illustrate these implementations with tasks τ_i and τ_j. τ_i sends data to τ_j. Due to the confidentiality levels of the tasks, the communications from τ_i to τ_j are considered vulnerable. So the data must be encrypted to avoid potential disclosure between its emission and its reception.

For function calls, a function that represents the key set up is added to both sending and receiving tasks. We assume the worst-case situation where the encryption key is set up at each task release. For the dedicated security tasks, a task that represents the key setup is added. We add communications between sending task and key task and between receiving task and key task.

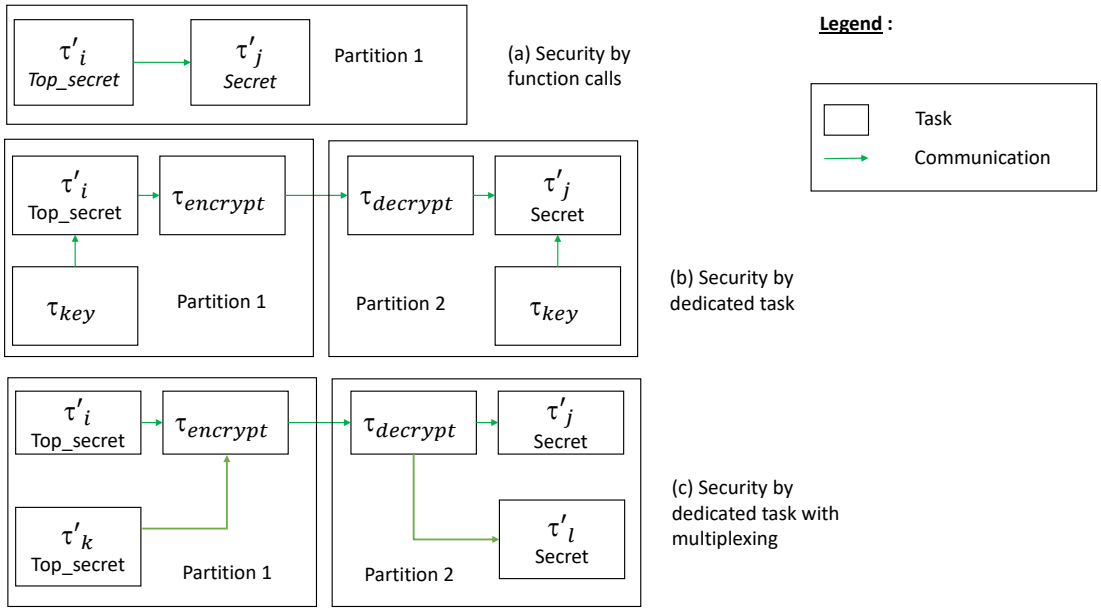


FIGURE 6.1: Illustration of security implementations

6.2.1 Securing communications through function calls

In this implementation, a task makes function calls to secure its data before sending/receiving. For communications with confidentiality vulnerabilities, we add to the sending (resp. receiving) task a call to an encryption (resp. decryption) function. A call to the key setup is also added to both tasks. For communications with integrity vulnerabilities, we add a call to a hash function in the sending and the receiving tasks.

Figure 6.1(a) presents, in the top, the task set when such security implementation is applied.

τ'_i (resp. τ'_j) is task τ_i (resp. τ_j) with a capacity changed as follows:

$$\begin{aligned} C'_i &\leftarrow C_i + C_{encryption\_function} + C_{encryption\_key\_function} + C_{hash\_function} \\ C'_j &\leftarrow C_j + C_{decryption\_function} + C_{encryption\_key\_function} + C_{hash\_function} \end{aligned} \quad (a)$$

C'_i and C'_j are the original task τ_i and τ_j execution time plus the execution time of the called security functions (integrity or confidentiality). $C_{hash\_function}$, $C_{decryption\_function}$ and $C_{encryption\_function}$ are respectively the worst-case execution time of the hash, decryption and encryption functions called by τ_i and τ_j. We highlight that in case of only confidentiality (resp. integrity) vulnerabilities, hashing is not needed. Then $C_{hash\_function} = 0$ (resp. $C_{decryption\_function} = 0$, $C_{encryption\_function} = 0$ and $C_{encryption\_key\_function=0}$). In the sequel, when applying this security implementation during DSE, we will run scheduling analysis with the new/updated parameters of tasks τ'_i and τ'_j.

For intra-partition communications, function calls can only be used if it exists a memory protection between tasks within a partition. Otherwise, the security vulnerability remains unresolved: for instance, a malicious task operating inside the partition has the possibility to access data of other tasks at a time when they are not encrypted. For inter-partition communications, function calls guarantee that data sent over communication channels are encrypted and not vulnerable from external attacks.

6.2.2 Securing communications through dedicated tasks

To secure communications between tasks, extra tasks dedicated to security functions can be added. Data are sent through these tasks before being read by the receiving task. This leads to new communications as illustrated in figure 6.1(b) for a confidentiality vulnerability.

For a communication that have confidentiality vulnerabilities, we add one task dedicated to encryption (called $\tau_{encrypt}$) and another task for decryption (called $\tau_{decrypt}$). Then direct communication between the sending and the receiving tasks is replaced by three communications: sending task to encrypting task, encrypting task to decrypting task, and decrypting task to receiving task.

For a communication that have integrity vulnerabilities, we add two hash tasks called τ_{hash} and $\tau_{hash'}$. In this solution, τ_{hash} intervenes to hash the data of the sender while $\tau_{hash'}$ is used to hash the data at the receiver side. Then we add new communications from the sender to τ_{hash}, from τ_{hash} to $\tau_{hash'}$, and from $\tau_{hash'}$ to the receiver.

During DSE, scheduling analysis will be done by considering the extra tasks $\tau_{encrypt}$, $\tau_{decrypt}$, and τ_{key} which have a capacity equal to the worst-case execution time of the security functions, as shown by the following equations:

$$\begin{aligned} C_{encrypt} &\leftarrow C_{encryption\_function} \\ C_{decrypt} &\leftarrow C_{decryption\_function} \\ C_{key} &\leftarrow C_{encryption\_key\_function} \end{aligned} \tag{b}$$

In the case of integrity violation, two extra tasks τ_{hash} and $\tau_{hash'}$ are added, as defined by the following equations:

$$\begin{aligned} C_{hash} &\leftarrow C_{hash\_function} \\ C_{hash'} &\leftarrow C_{hash\_function} \end{aligned} \tag{c}$$

We also assumed that for each communication that has vulnerabilities, all the introduced security tasks inherit the periods of the sender tasks.

This implementation can only be applied to inter-partition communications. Adding extra tasks to secure an intra-partition communication is inefficient because the attacker can intercept data between the task sender and the encryption task since they are located in the same partition. In that case, we assume that all intra-partition communications are not vunerable. Otherwise, we should assume memory protection between tasks within the same partition. This is out of the scope of this thesis.

Furthermore, in the sequel, we call communication multiplexing when security features are shared by several sending/receiving tasks. In the security dedicated task implementation, multiplexing consists of assuming for each partition, with confidentiality and/or integrity vulnerabilities, the use of only one encrypter task for all the tasks sending data and one decrypter task for all the tasks receiving data through confidentiality vulnerable communications; and/or one hash task for all the integrity vunerable communications.

In each partition for the communications that have confidentiality vulnerabilities, there will be only one task used for encryption and another one for decryption. For communications with integrity vulnerabilities, we have only one hash task. Figure 6.1(c) presents an illustration of this process assuming that $data_1$ ($data_2$) is sent from task τ_i (resp. τ_k) to τ_j (resp. τ_l). Then the equations b and c become:

$$\begin{aligned} C_{encrypt} \leftarrow & C_{encryption\_function}(data_1) + C_{encryption\_function}(data_2) \\ & + C_{encryption\_key\_function} \\ C_{decrypt} \leftarrow & C_{decryption\_function}(data_1) + C_{decryption\_function}(data_2) \\ & + C_{encryption\_key\_function} \end{aligned} \tag{d}$$

$$\begin{aligned} C_{hash} \leftarrow & C_{hash\_function}(data_1) + C_{hash\_function}(data_2) \\ & C_{hash'} \leftarrow C_{hash\_function} \end{aligned} \tag{e}$$

For inter-partition communications, the encrypting (resp. decrypting) and hash task are added in the partition of the sending (resp. receiving) task.

To summarize, from the security implementations presented above, we propose to investigate during DSE the 4 combinations that are outlined in Table 6.1. Each security implementation is characterized by its identification label (ID) and the publications (Ref) that motivated its definition.

For intra-partition communications, securing through dedicated tasks implies extra communications. Then it first requires communications from the sending (resp. decrypting) tasks to the encrypting (resp. receiving) tasks and communications from encrypting tasks to decrypting tasks. The communications from encrypting tasks to decrypting tasks are secured, but data are vulnerable before being encrypted and after being decrypted. Then communications from the sending tasks to the encrypting tasks and from decrypting tasks to receiving tasks

	Intra-partition communication security implementation	**Inter-partition communication security implementation**	**ID**	**Ref.**
Intra-partition communication are vulnerable	Function calls	Function calls	F-F	[149] [151]
Intra-partition communication are secured	Not investigated	Function calls	X-F	[149] [151]
		Dedicated tasks without multiplexing	X-T	[150] [151]
		Dedicated tasks with multiplexing	X-TM	[150] [151]

TABLE 6.1: Security implementations considered in this thesis

which are also intra-partition communications will present the same vulnerabilities that we are trying to resolve.

Then, assuming memory protection boundary is partition, in the case of intra-partition communications, adding security dedicated tasks do not resolve the security vulnerabilities. Only security through functions calls can ensure the security without implying extra communications with extra vulnerabilities.

For inter-partition communications, when adding security dedicated tasks, the extra intra-partition communications (sending tasks to encrypting tasks, and decrypting tasks to receiving tasks) are still vulnerable. Then when considering intra-partition vulnerable, security dedicated tasks are not suitable for inter-partition communications. Applying function calls on these extra communications after adding the security dedicated tasks can also be an alternative but it will considerably increase the security overheads. This explained our decision to investigate only security through function calls for intra and inter-partition communications when considering intra-partition communications as vulnerable.

When intra-partition communications are considered non-vulnerable, the inter-partition communications can be safely secured through security dedicated tasks since the extra intra-partition communications will be implicitly secured.

6.3 Security and scheduling: trade-off in TSP systems

When designing a TSP system, tasks to partitions assignment and the respect of timing constraints of hard deadline tasks are important challenges to investigate. TSP systems present communications between tasks that may present confidentiality and/or integrity vulnerabilities. However, ensuring data confidentiality

Task	A_i	C_i	T_i	D_i	CL_i	CI_i	P_i
τ_1	1	2	24	24	Top Secret	Hard	1
τ_2	1	6	24	24	Top Secret	Soft	1
τ_3	1	3	24	24	Secret	Hard	1
τ_4	2	4	24	24	Unclassified	Soft	2
Encrypt Function	-	1	-	-	-	-	-
Decrypt Function	-	1	-	-	-	-	-

Partition	Length	Period
1	12	24
2	12	24

TABLE 6.2: Task and partition configuration

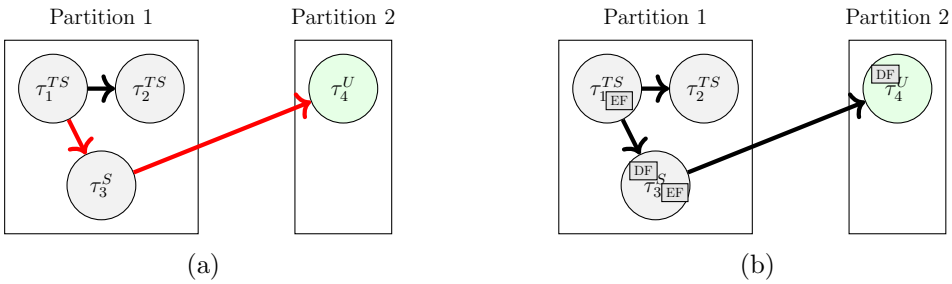


FIGURE 6.2: Partitioning and communications without/with security functions

and integrity with the use of encryption and hashing incurs a significant computation overhead [152]. This overhead impacts the system schedulability and may lead some tasks to miss their deadlines.

In this section, we illustrate the conflict between schedulability and security in TSP systems. Table 6.2 and figure 6.2a present an example of a task set and its partitioning.

The system consists of four tasks and two partitions. Communications between tasks are illustrated in figure 6.2a. An arrow from τ_i to τ_j models a communication from τ_i to τ_j. We only illustrate vulnerabilities, which violates to BLP rules in figure 6.2a.

Without considering any security constraint, the task set is scheduled as illustrated in figure 6.3a. All tasks can meet their deadlines at time $t = 24$, which is the end of the first MAF. This schedule is then repeated indefinitely for the next MAFs.

Considering security constraints, the vulnerable communications, which violate the BLP rules, are marked in red in figure 6.2b. There are two confidentiality violations: one from τ_1 (Top Secret) to τ_3 (Secret) and the second from τ_3 (Top Secret) to τ_4 (Unclassified). To secure these communications, one solution may use encryption and decryption functions to ensure that the data exchange cannot be exposed. This is illustrated in figure 6.2b: for each vulnerable communication, an encryption function (EF) is called to the sender and an decryption function (DF) is called to the receiver. Secured communications are shown in black in figure 6.2b.

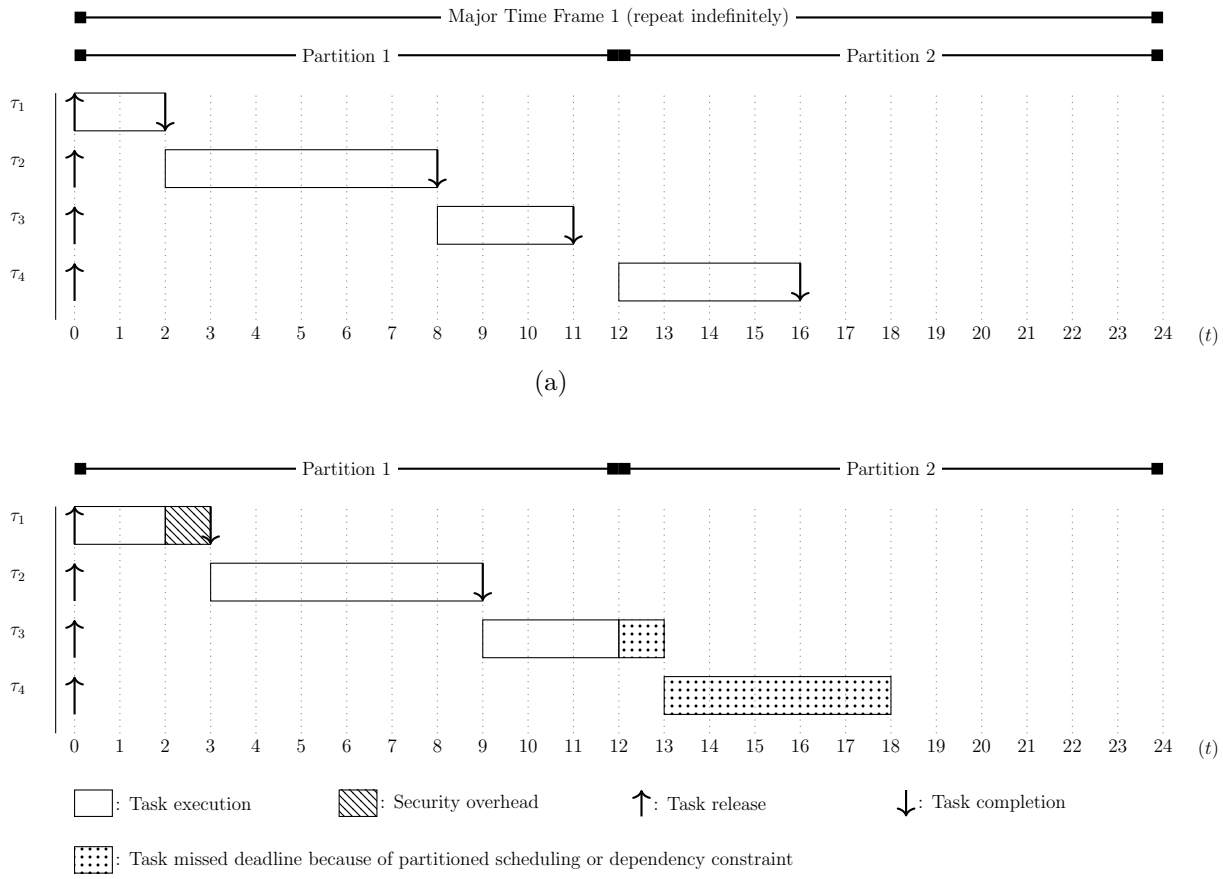


FIGURE 6.3: Partitioned scheduling without/with security functions

The task scheduling with secured communications is illustrated in figure 6.3b.

We assume an execution time of one unit of time for encryption or decryption function. Then we consider a security overhead of one unit of time for tasks τ_1, τ_4, and two units of time for task τ_3. First, with the overhead due to security functions, τ_3 cannot complete its execution in the time slot reserved for partition 1. As partition 1 stops at time $t = 12$, τ_3 misses its deadline at time $t = 24$. Second, as τ_4 depends on τ_3, it cannot start executing at time $t = 12$. Eventually, τ_4 also misses its deadline at time $t = 24$.

To remove the missed deadlines, the first solution is to assign τ_2 or τ_3 to partition 2. Another solution is to only secure the communication between τ_3 and τ_4 to reduce the security overhead. Finally, as τ_2 has a soft deadline, we can also choose to not meet its deadline by prioritizing τ_3 over τ_2.

With this example, we illustrated the possible trade-offs when enforcing schedulability and security. We have to simultaneously consider the scheduling constraints of tasks and partitions, the costs of securing vulnerable communications, and task

	[153]	[154]	[155]	[150]	[22]	[156]	[149]	[152]	[157]	[158]	Ours
RTS security	X		X	X	X	X	X	X		X	X
Schedulability optimization	X	X	X						X	X	X
Security optimization								X		X	X
Trade-offs between security and schedulability						X					X
MOEA/DSE		X		X						X	X
TSP									X		X
Exploration with different levels of granularity											X
Multiple security implementations											X

TABLE 6.3: Related work

assignments on partitions. In some cases, a fully schedulable and secured solution cannot be achieved and trade-offs have to be proposed, which motivates the need of a DSE.

6.4 Related work

In this section, we compare different approaches on security and schedulability optimization for real-time systems including our proposal.

As shown in Table 6.3, security of real-time systems has been addressed by many works [153, 155, 150, 22, 156, 149, 82, 152]. Considered security criteria can be related to confidentiality [153, 155, 156, 149, 152], integrity [155, 150, 22, 156, 149, 152], and authentication [22, 149, 152]. Many of these works only focus on improving the security of the systems. For example, [22] proposes to guarantee integrity and authentication of information transmitted from sensors to controllers in real-time systems. [149] provides a model for clustered real-time systems to evaluate the overhead required to ensure confidentiality and integrity requirements.

As timing constraints are one of the characteristics of real-time systems, numerous works propose approaches of schedulability optimization without considering security in the constraints or objective functions [153, 154, 155, 157, 159]. Proposals in [154, 157] focus on objective functions related to the deadlines the tasks must meet. [154] addresses the functions to tasks assignment in real-time systems while optimizing tasks preemption number and task laxities. The work in [157] designs a heuristic to minimize the worst-case response time of tasks in TSP

architecture.

Fewer works have investigated both timing and security constraints. [153] and [155] propose to fully guarantee the security of the systems while allowing few tasks to miss their deadlines. They optimize the schedulability in terms of missed deadlines for real-time database systems. When previous papers optimize schedulability while sometimes guaranteeing security requirements, [152] optimizes security while guaranteeing schedulability. They propose a security-aware scheduling for embedded systems called SASES to improve the security of real-time systems without allowing any task to miss its deadline. The security requirements addressed in this approach are about confidentiality, integrity and authentication.

Computing trade-off between security and schedulabilty has raised less interest. Instead of guaranteeing schedulability of all functions of the system (resp. security) at a cost of assuming only a decrease of security (resp. schedulability), [156], describe a proposal to find trade-offs between security and schedulability based on a concurrency protocol named 2PL-HP. A minimal percent of missed deadlines is assumed. Security and schedulability are both optimized while tolerating a mutual decrease. Then, partial security requirements violations are allowed to respect at least a minimal rate of missed deadlines.

Most of the papers cited above produce a single design decision. DSE is performed in [154, 150, 158]. The authors propose approaches to find trade-offs for real-time systems based on a multi-objective evolutionary algorithm (MOEA) for different multi-objective optimization problems. [150] proposes a DSE named Hydra based on a heuristic that investigates security tasks to cores assignments and architecture parameters (e.g. task period) to improve the schedulability of the system. [158] proposes a DSE based on Tabu meta-heuristic applied on a directed acyclic graph (DAG) model. The DAG expresses task communications and the DSE explores solutions with a given level of communication security, processor voltage and task frequency, in order to ensure schedulability and to minimize energy consumption. Jiang does not propose a set of solutions as DSE trade-off and does not explore TSP systems.

As far as we know and specifically for TSP real-time systems, existing works address schedulability but not security requirements. For example, the work in [157] proposes a scheduling approach to optimize the schedulability of integrated modular avionics (IMA) systems characterized by a set of tasks to execute on multiple partitions, but does not investigate security.

To sum up, multi-objective optimization of real-time systems and security have been studied by several researchers. Fewer have worked on both optimizing security and schedulability. Even if there are many existing DSE approaches, as far as we know, none has worked on trade-off and at different levels of granularity while in this thesis, we propose a DSE with three mutation algorithms and four security implementations based on different combinations leading to trade-off.

Finally, as far as we know, none has considered exploring TSP systems with such different options jointly investigated.

6.5 Summary of expected contributions

This thesis addresses the conflict between schedulability and security considering TSP systems and the combinatorial problem raised by tasks to partitions assignment. To address the gap in the state of the art, we explore TSP systems while considering different granularity of tasks to partitions assignment, different security implementations to propose tradeoffs between schedulability and security based on a multi-objective evolutionary algorithm.

We propose a DSE approach to address the combinatorial problem raised between schedulability and security of TSP systems. We formulate our multi-objective problem and adapt the PAES to it in order to explore the search space and propose trade-offs for safe and secure TSP systems. Our approach proposed to explore the search space of TSP while investigating tasks and partitions assignment and communications security. We proposed feasibility tests based on schedulability and security analysis to check the validity of the solutions during the DSE. We performed evaluations to find the best solutions by comparing candidate solutions to each other during the DSE.

Considering the generation of candidate solutions, we propose to explore the design space of TSP systems with different levels of granularity by three mutation algorithms. The first algorithm considers moving only one task to another partition at each mutation. This implies an investigation of a large design space. Then the second algorithm proposes to move at each mutation, an application composed of a set of tasks to another partition. It reduces the design space size but presents a less degree of freedom. We proposed the third mutation algorithm that mixed the two above algorithms. It consists of refining the results obtained at the application level (i.e. second algorithm) by applying them a mutation algorithm at task granularity (i.e. first algorithm). For mutation algorithms, we assume that the considered applications have similar criticalities allowing us to move tasks to other partitions with tasks of different applications. Finally, we proposed a fourth algorithm to improve the diversity of the proposed solutions based on a better choice of the current solution during the DSE.

With each mutation algorithm, we evaluate four different means to implement security features in TSP systems (detailed in Table 6.1) when computing the trade-offs between schedulability and security.

Furthermore, we experiment the extensibility of our DSE by investigating the impact of multicore execution platforms on safe and secure TSP systems while

considering not only tasks to partitions assignment but also tasks to cores assignment.

We integrate the prototypes of our DSE approaches into the Cheddar scheduling analyzer. We conducted multiple experiments to evaluate these approaches and identified guidelines that must be considered when designing safe and secure TSP systems towards unicore or multicore execution platforms.

6.6 Conclusion

In this chapter, we propose to depict the orientation and positioning of our work. Then we first present the system model and the assumptions on which our proposal is based. Second, we discuss the conflict between security and schedulability through a synthetic example. It shows the motivation behind our proposal. Third, we position our work by presenting some related works. Finally, we present the expected thesis contributions which are detailed in the next four chapters.

A presentation of our DSE approach to investigate the schedulability and security trade-off in TSP systems for unicore platforms is proposed in Chapter 7. Chapter 8 presents the experiments performed to evaluate this approach. An extension of this approach to multicore platforms on TSP systems while addressing the conflicts between safety, security, and schedulability is presented in chapter 9. Finally, the implemented prototypes of the approaches integrated into the Cheddar scheduling analyzer are presented in the chapter 10.

Part III

Contributions

7

Design space exploration to secure unicore TSP systems

As this thesis addresses the conflict between schedulability and security in TSP systems and the combinatorial problem raised by tasks to partitions assignment, we propose to explore the solutions space of secure TSP systems to identify trade-offs. This chapter is therefore dedicated to the presentation of our DSE approach.

We have adopted PAES multi-objective metaheuristic to address our MOOP. Section 7.1 presents an overview of how we have adapted the general framework provided by PAES to our MOOP. The adaptation of PAES implies the specifications of each PAES operator based on the addressed problem. Then, Section 7.2 presents the objectives functions, and constraints defined to evaluate the solutions. This helps to define the feasibility tests that validate or invalidate candidate solutions. This section also proposes a chromosomal representation that defines the solutions and makes them manipulable by evolutionary algorithms. It also presents the mutation operator for new solutions generations, the initial solutions, and the archiving process. Finally, Section 7.3 concludes the chapter.

7.1 PAES general framework for schedulability and security trade-off

This section gives a general view of our framework resulting of our adaptation of the PAES to the MOOP raised by the conflict between schedulability and security. We adopt PAES which is adapted to DSE problems with multiple and conflicting

objectives. Figure 7.1 presents an adaptation of the PAES to schedulability and security optimization problem.

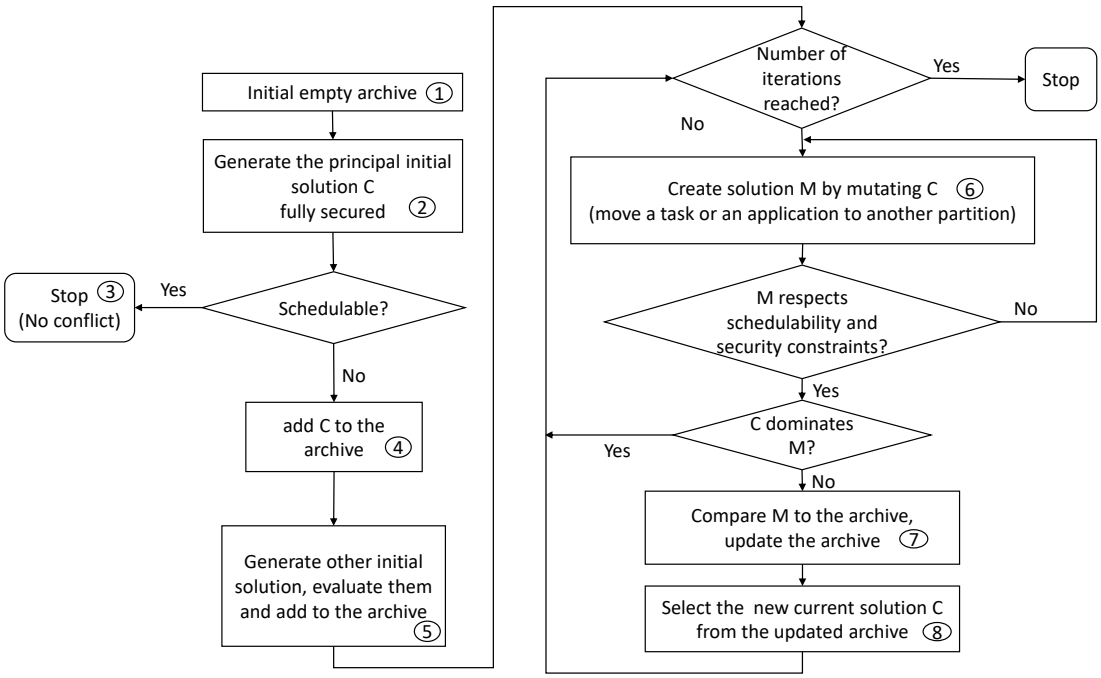


FIGURE 7.1: PAES process

The PAES starts with an initial solution (step 2 in figure 7.1) that considers only one partition to which all the tasks are assigned with all security vulnerabilities resolved. This solution is built to determine the existence of conflicts between security and schedulability and then the necessity of proceeding with the DSE. Its schedulability analysis determines the necessity of operating a DSE. If the initial solution is schedulable, this solution is already an optimal solution that optimizes both schedulability and security. Thus there is no need to proceed with a DSE (step 3 in figure 7.1). Otherwise, if this initial solution is not schedulable (i.e. some tasks missed their deadlines), a DSE is useful to provide a set of trade-offs between schedulability and security. So the solution is added to the archive initially empty (steps 1 and 4 in figure 7.1).

During the DSE, at each iteration, a mutation operator that consists of moving a task or an application (a set of tasks) to another partition is applied to the current solution to generate a candidate solution (step 6 in figure 7.1). At the first iteration, the current solution is the initial solution which consists of the entry point of the DSE process. After the generation of the candidate solution, it goes through feasibility tests in order to determine if it respects the schedulability and security constraints detailed later in Section 7.2.1. Each generated solution (including the initial solution), has to pass through an evaluation of objective functions detailed later in Section 7.2.1.

The candidate solution is compared to the current solution and the other solutions in the archive (step 7 in figure 7.1). The archive is updated in order to keep only non-dominated solutions as described in section 5.3.1.2. Then a solution is selected to become the current solution of the next iteration (step 8 in figure 7.1) as described in the section 5.3.1.2.

The mutation, the solutions comparison, the current solution selection, and the archive update are repeated till the end of the DSE which we defined with a prefixed number of iteration. When this number is reached, then DSE stops and the non-dominants solutions in the archive are proposed as the trade-offs between schedulability and security. Then the designer has a set of solutions from which he can choose the model best suited to his requirements.

We highlight that after the generation of the initial solution that represents the first current solution, we filled the archive with other initial solutions (step 5 in figure 7.1) that may correspond to extreme solutions that each maximize one of the objective. This procedure is adopted to make the PAES faster and to boost the diversity of solutions in the archive at the end of DSE.

7.2 PAES adaptation to the MOOP of schedulability and security

In this section, we specify how the operations such as the encoding of the solutions, the mutation operator, the security and schedulability constraints, the objectives functions, and the archiving process are conducted in the context of this thesis.

7.2.1 Objective functions and constraints

The multi-objective optimization of a problem requires the definition of functions that fit the objectives to optimize. Since our objective is to optimize schedulability and security, we define fitness functions that model these goals.

We also define constraints considered for schedulability and security issues. It is important to highlight the difference between objective functions and constraints in order to avoid confusion. The objective functions are functions that have to be optimized during the DSE. They constitute the criteria of solutions evaluation. However, constraints are conditions that determine the validation or invalidation of a solution. They can be conditions made on some objective functions (e.g. a condition on their values) or any other criteria or event necessary to confirm the validation of a system based on the designer requirements. As example, we can refer to hard deadline tasks that must imperatively meet their deadlines.

7.2.1.1 Objective functions and constraints concerning schedulability

We remind that the necessity for the DSE came from the impossibility to propose for some systems a model which is fully schedulable and fully secure. Then concessions has to be made on both sides in order to find trade-offs.

Therefore, we distinguish tasks with hard and soft deadlines. We define the first constraint by requiring that no task with hard deadline should be allowed to miss its deadline. Then any model should be automatically considered invalid and then rejected if one of its tasks with hard deadline does not respect its deadline.

In the search of trade-offs, the requirement of schedulability can be relaxed for tasks with soft deadlines in order to introduce security while maintaining schedulability of hard deadline tasks. The number of missed deadlines of soft deadline tasks can be used to evaluate the schedulability of a solution. We deduce an objective function reflecting the quality of the schedule function note as :

$$F1 = \#missed\_deadlines$$

The number of missed deadlines represents the number of soft deadline tasks that have worst-case response times higher than their deadlines. To assess such a metric, we simulate the scheduling of the task set on the feasibility interval [59] with Cheddar scheduling simulator. Computing the schedulability simulation over the feasibility interval provides a proof of schedulability. The entry point of Cheddar is a model composed of partitions, tasks and communications between tasks. Notice that this model is generated from the solution representation in figure 7.2 and can include extra tasks dedicated to security, depending on the security implementation chosen in the solution.

7.2.1.2 Objective functions and constraints concerning security

Our models contain communications between tasks. These communications can present security vulnerabilities described in Section 6.2.

We divide communications into two categories: weakly sensitive communications and strongly sensitive communications. Any strongly sensitive communications that have vulnerabilities must be secured. By considering BLP (resp. Biba) rules, we assumed as constraints that a task with $Unclassified$ confidentiality (resp. *Low* integrity) level is not allowed to communicate with a task at higher confidentiality (resp. integrity) level.

Table 7.1 resume the security constraints. A model with communication that violates a security constraint is invalid and should be automatically rejected.

Since weakly sensitive communications are not concerned by these constraints, their security vulnerabilities can be tolerated.

Tasks	**Write access violation**	**Read access violation**
Confidentiality	Top_secret → Unclassified Secret → Unclassified	Unclassified → Top_secret Unclassified → Secret
Integrity	Low → Medium Low → High	Medium → Low High → Low

TABLE 7.1: Communications concerned by security constraints

Tasks	**Write access violation**	**Read access violation**
Confidentiality	Top_secret → Secret	Secret → Top_secret
Integrity	Medium → High	High → Medium

TABLE 7.2: Communications concerned by security objective functions

These tolerated vulnerabilities are resumed in Table 7.2. Thus, we can tolerate that a task with a $Top - secret$ confidentiality (resp. $Medium$ integrity) level can send information to a task at secret confidentiality (resp. $High$ integrity) level.

We identify two objectives functions to characterize security optimization. First, the number of confidentiality vulnerabilities that represents the number of weakly sensitive communications that violate BLP's rule in a TSP system:

$$F2 = \#Bell\_violations$$

Second, the number of integrity vulnerabilities which is the number of weakly sensitive communications that violate Biba's rules in a TSP system:

$$F3 = \#Biba\_violations$$

Both metrics can help to perform a security evaluation of a given solution. They are computed through BLP and Biba rules implemented in Cheddar.

The objective of our work being to optimize both schedulability and security of models, the DSE should be operated by minimizing the number of soft deadline tasks missing their deadlines and the numbers of weakly sensitive communications that have confidentiality or integrity vulnerabilities.

7.2.2 Feasibility tests

As we defined constraints (section 7.2.1) to evaluate the feasibility of generated solutions, we implemented these constraints through feasibility tests as sketched in the algorithm 1.

The algorithm takes as input a solution and returns a boolean to confirm or not the feasibility of this solution. The algorithm starts by checking if the schedulability constraints are respected. For this purpose, it requires a scheduling simulation to compute the WCRT of each task (line 3). From line 4 to line 10, it proceeds by checking if no hard deadline task has missed its deadlines. If a hard deadline task misses its deadline, the solution is considered non-feasible and the feasibility test stops. Otherwise, the feasibility test can continue by checking if the security constraints are respected by the strongly sensitive communications of the solution. Then from line 11 to line 20, the algorithm verifies if there is a strongly sensitive communication that violates BLP or Biba rules. If it is the case, then the solution is considered non-feasible. Otherwise, we can confirm that the solution is feasible (line 21).

During the DSE, as soon as a solution is generated, feasibility tests are applied. If it results that the solution is feasible then it is validated as a candidate solution and the PAES process continues with the comparison of this solution with the current solution as defined in figure 7.1. Otherwise, if the solution is confirmed non-feasible, then it is rejected and another solution has to be generated. The feasibility tests help to proposed an archive with only feasible solutions for the designer.

7.2.3 Solutions encoding

An evolutionary algorithm implies the definition of solutions encoding that helps to represent solutions and eases their manipulation during the exploration process. Solutions are represented in chromosomal formal with an encoding method. There are multiple encoding methods such as binary encoding [160] [161], real encoding [162], integer encoding [162]. All the methods present advantages and drawbacks described in [162]. Since integer encoding is widely used in combinatorial optimization problems, we have chosen an adhoc encoding based on integer encoding.

The chromosome in integer encoding is represented by a vector of n genes where n represents the number of objects. Each gene indexes a position in the vector and has a value.

In this work, as we address secured TSP systems with tasks communicating with each other and assigned to partitions, we assume a vector of $(n + 1 + m)$ genes. n corresponds to the number of tasks and m to the number of communications. The chromosome is divided into three parts. The first part of the chromosome models the assignment of tasks to partition where a gene indexes a task and the attributed value is the partition to which this task is assigned. Then $chrom[i] = j$ $(1 \leq i \leq n$ and $1 \leq j \leq r)$ with r the number of partitions in the model, reveals that the i^{th} task is assigned to the j^{th} partition.

Algorithm 1: Feasibility tests algorithm

Input: A solution with n tasks and m communications
Output: A boolean that confirm with $True$ or infirm with $False$ the feasibility of an input
// schedulability constraint on hard task
Perform scheduling simulation on the solution
i=0
while $i < n$ **do**
    i=i+1
    if *(tolerance(τ_i)=hard) and ($WCRT(\tau_i) > deadline(\tau_i)$)* **then**
        return False
    end if
end while
$k = 0$
while $k < m$ **do**
    $k = k + 1$
    // confidentiality constraint on strongly sensitive communication
    if *(confidential_level(Task_source(k^{th} communication))= $unclassified$) Or (confidential_level(Task_sink(k^{th} communication))= $unclassified$)* **then**
        return False
    end if
    // integrity constraint on strongly sensitive communication
    if *(integrity_level(Task_source(k^{th} communication))= low) Or (integrity_level(Task_sink(k^{th} communication))= low)* **then**
        return False
    end if
end while
return True

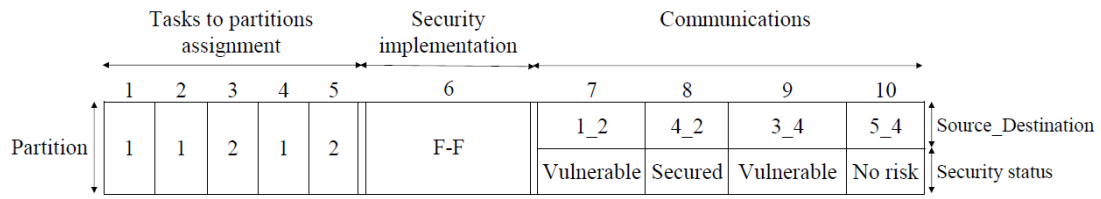


FIGURE 7.2: Example of chromosome

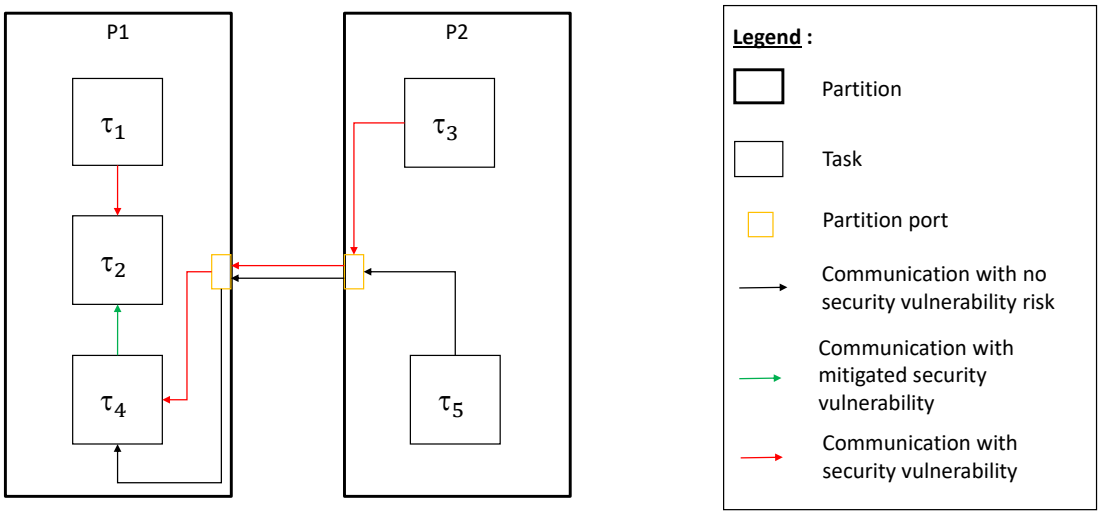


FIGURE 7.3: Model of the illustrated chromosome

Figure 7.2 shows an illustration of our chromosomal representation of the model presented in the figure 7.3 composed of five tasks assigned to two partitions and four communications.

The first part of the chromosome illustrates the tasks to partitions assignment encoding. The 1^{st}, 2^{nd}, and 4^{th} (resp. 3^{th}, 5^{th}) positions in the chromosome indicate that tasks τ_1, τ_2, τ_4 (resp. τ_3, τ_5) are assigned to partition P_1 (resp. P_2).

Since we assume different methods to secure vulnerable communications, there is a gene that specifies the security implementation of each solution. Then the second part of the chromosome is a single value that defines the security implementation chosen to secure the vulnerable communications of the chromosome. It can have value as F-F, X-F, X-T, or X-TM that we defined previously in table 6.1.

The model presented in figure 7.3 presents intra and inter-partition communications. Each partition has its own port (represented with yellow box) for inter-partition communications. Some of these communications represented with red arrows present security vulnerabilities. For this model, we assume to resolve security vulnerabilities with the security implementation F-F which refers to function calls. Then the 6^{th} position of the chromosome (figure 7.2) shows that the security implementation F-F is chosen for communications to be secured.

The third and last part of the chromosome representation concerns communications in the TSP system. As described in the section 7.2.1.2 communications are decomposed into two categories. A model of m communications has mu weakly sensitive communications and mc strongly sensitive communications with $m = m_c + m_u$.

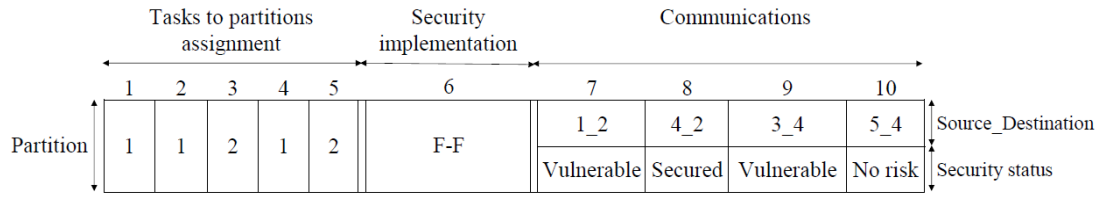
Then the $chrom[k]$ with $(n+2) \leq k \leq (n+m+1))$ that corresponds to the k^{th} communication is associated with two values. The first value indicates the task source that initiates the communication and the task sink of the communication (e.g. value 1_2 specifies that a communication is initiated by the task τ_1 towards the task τ_2). The second value indicates the status of the communication. It shows if the communication presents non-resolved vulnerabilities. Its possible values are "vulnerable", "secured" and "no risk". "Vulnerable" is for communications with security vulnerabilities. "Secured" is for communications that have vulnerabilities mitigated through security features such as encryption and/or hashing functions. "No risk" is for communications that present no security vulnerability. Notice that m_c communications are fixed to secured.

This encoding is illustrated by the final slice of the chromosome in figure 7.2. As example, the position 7^{th} represents a communication initiated by task τ_1 to task τ_2 that presents security vulnerabilities. The 8^{th} position shows a secured communication from task τ_4 to task τ_2.

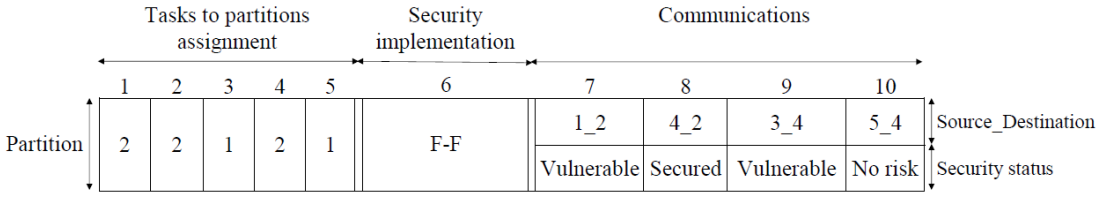
By observing only the task to partitions assignment part of our chromosomal representation and assuming identical partitions, we faced a redundancy problem well known in integer encoding. A solution can be encoded by multiple chromosomes different from each other. For example, the tasks to partition assignment in figure 7.2 represented with the vector [1 1 2 1 2] can also be represented by the vector [2 2 1 2 1] as illustrated in figure 7.4.

Figure 7.4 shows two representations of the same chromosome. Figure 7.4a shows normalized representation of the chrosome and figure 7.4b shows a non-normalized representation of the chrosome.

Both representations correspond to the same solution. They represent a solution where tasks τ_1, τ_2, and τ_4 are assigned to one partition, and tasks τ_3 and τ_5 are assigned to a second partition. By referring to [162], in our context, for a model with k partitions, there are $k!$ chromosomes that encode the same solution. It is not efficient to consider the redundant chromosomes during the DSE. Then we adopted a normalization of solutions in order to eliminate the redundancy, which reduces the search space size and increases the quality and the diversity of solutions proposed at the end by the DSE. For this purpose, we assumed that task τ_1 should always be assigned to partition P_1, and task τ_2 is assigned to P_2 if and only if task τ_1 and τ_2 are assigned to different partitions. Task τ_3 is then assigned to P_3 if it is not embedded with τ_1 nor τ_2, etc.



(a) Normalized representation of a chromosome



(b) Non-normalized representation of a chromosome

FIGURE 7.4: Normalization illustration

7.2.4 Mutation operator

PAES is (1+1) evolution strategy: the DSE exploration is operated by generating a new solution from a current solution at each generation. We proposed a random mutation search operator while considering the MOOP that we addressed and the solutions encoding that we defined. Our mutation process is divided into two steps: the mutation of tasks to partitions assignment and of the communications.

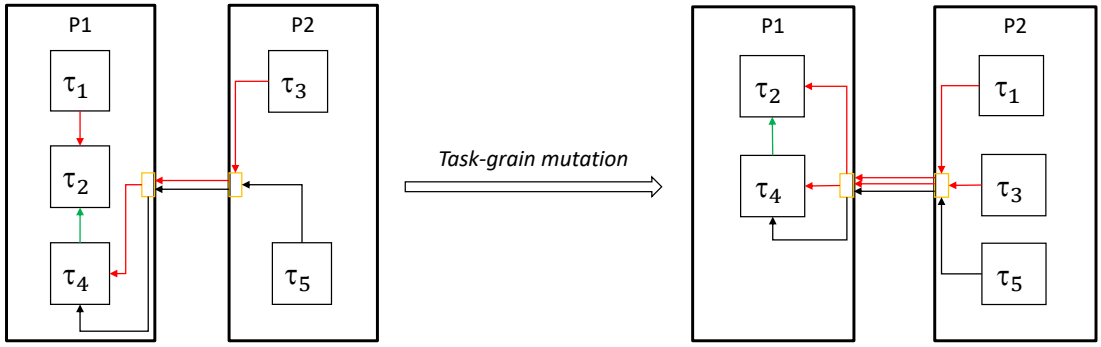
7.2.4.1 Mutation process: Tasks to partitions assignment

The first step of our mutation process is dedicated to the tasks to partitions assignment which corresponds to the first slice of the chromosome. We proposed three possible mutation algorithms that defined how a mutation can be made. One of the proposed algorithms has to be chosen by the designer as a parameter of the DSE.

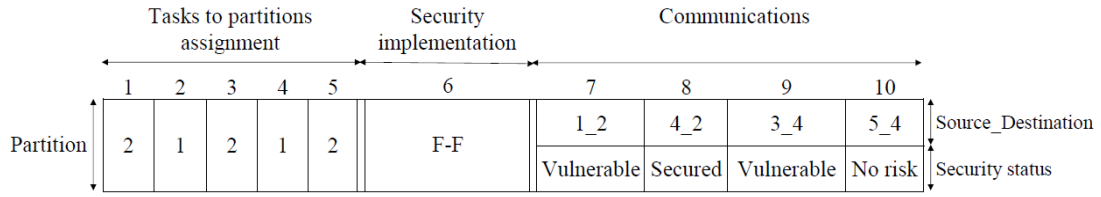
Algorithm *task-grain*

Algorithm *task-grain* is the most intuitive. It consists of choosing a random task τ_i among the task set $(1 \leq i \leq n)$ and a random partition P_j $(1 \leq j \leq r)$ among the set of partitions. If the randomly chosen task is not already assigned to the randomly chosen partition (i.e. $chrom[i] \neq j$), the mutation is operated (i.e. $chrom[i] = Pj$).

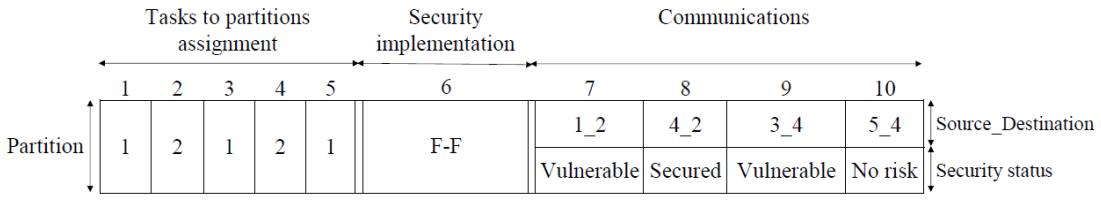
Figure 7.5a illustrates a *task-grain* mutation operation. It shows two models. The first model, at the left, corresponds to the one previously presented in figure 7.3 (Section 7.2.3). The second model, at the right, corresponds to a mutated model



(a) Task-grain mutation model illustration



(b) Task-grain mutation chromosome illustration (Before normalization)



(c) Task-grain mutation chromosome illustration (After normalization)

FIGURE 7.5: Task-grain mutation illustration

obtained after applying a *task-grain* mutation to the first model. This mutation only consider moving the task τ_1 from partition $P1$ to partition $P2$. We remark that the intra-partition communication from task τ_1 to task τ_2 becomes an inter-partition communication after the mutation since both tasks are no more in the same partition. This implies an impact on communications overhead of the model.

We remind that we operate normalization on each mutated chromosome. Then the chromosomal representation of the mutated model is presented in figure 7.5b. After each mutation, the chromosome has to be normalized as defined in section 7.2.3. Then figure 7.5c presents the chromosome of the mutated model after its normalization. We can remark that tasks assigned to partition $P2$ are moved to partition $P1$ because task τ_1 and that tasks embedded with it in the same partition, must always be assigned to the partition $P1$ based on our normalization principle.

Algorithm *app-grain*

Instead of moving only one task to another partition as for the *task-grain* algorithm, we can also move an application constituted of a set of tasks. Then it consists of choosing a random application A_i $(1 \leq i \leq m)$ among the applications of the model and a random partition P_j $(chrom[i] \neq j)$ among the set of partitions. If all the tasks of A_i are not already assigned to the randomly chosen partition, the mutation is operated by assigning all the tasks of A_i to the chosen partition P_j. This algorithm is intended to be compliant with the ARINC653 standard. It also guides the exploration based on the fact that the tasks of an application communicate more with each other, and the communications will therefore be more intra-partition communications. It helps to minimize the overhead of communications because inter-partition communications are more costly than intra-partition communications.

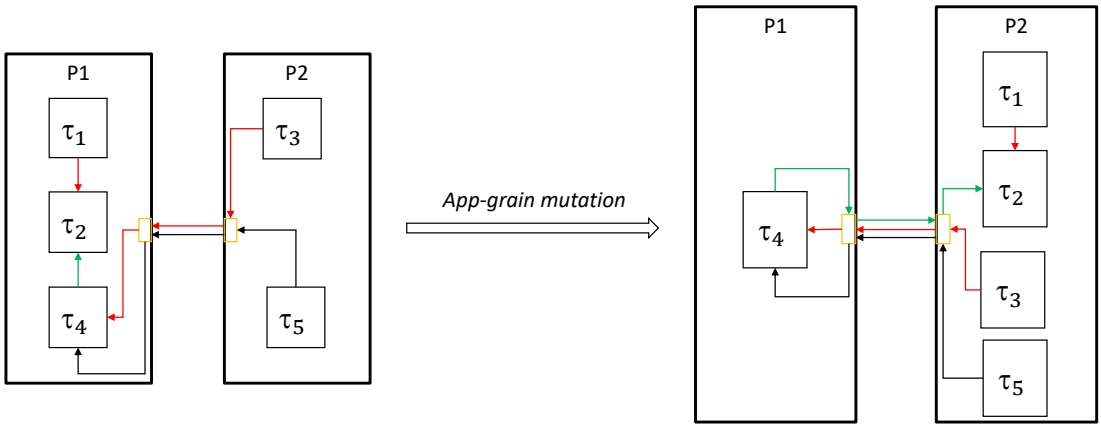
Figure 7.6a illustrates an *app-grain* grain mutation operation. It shows two models. the first model, at the left, corresponds to the one previously presented in figure 7.3 (Section 7.2.3). The second model, at the right, corresponds to a mutated model obtained after applying an *app-grain* mutation to the first model. This mutation only consider moving the application composed of tasks τ_1, and τ_2 from partition $P1$ to partition $P2$.

The chromosomal representation of the mutated model is presented in figure 7.6b. Its normalization leads to the chromosome presented in figure 7.6c. We remark different changes in the communications after the mutation operation. Since tasks τ_1, and τ_2 are moved to the same partition, the communication between them remains an intra-partition communication. The communication from task τ_4 to task τ_2 becomes an inter-partition communication since both tasks are no longer in the same partition after the mutation operation. This has an impact on the communications overheads.

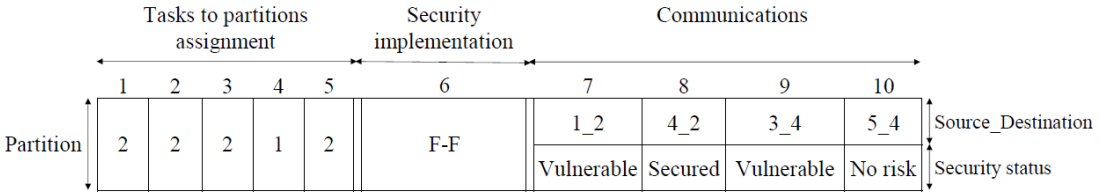
Algorithm *mix-grain*

Algorithm *mix-grain* consists of mixing the *task-grain* and *app-grain* algorithms in the expectation of having a better quality of solutions at the end of the DSE. It consists of starting the DSE with the *app-grain* algorithm for a prefixed number of iterations and then proceeds with a refinement by applying the *task-grain* algorithm till the end of the exploration.

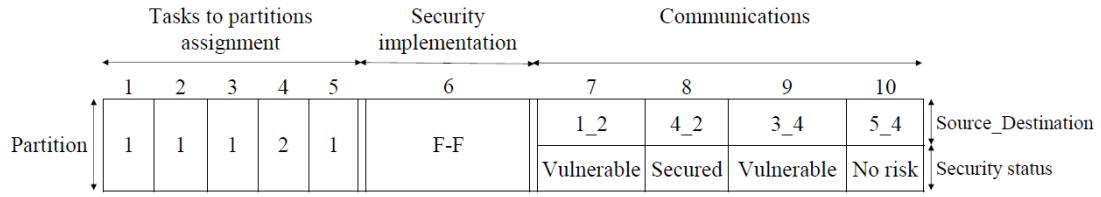
It is then composed of two phases: a first phase with *app-grain* algorithm followed by a second phase with *task-grain* algorithm. The second phase takes advantage of *app-grain*'s guidance. Indeed, instead of exploring directly the whole space, with the risk of an inefficient exploration, it refines the solutions provided in the first phase algorithm which offers more degrees of freedom.



(a) App-grain mutation model illustration



(b) App-grain mutation chromosome illustration (Before normalization)



(c) App-grain mutation chromosome illustration (After normalization)

FIGURE 7.6: App-grain mutation illustration

7.2.4.2 Mutation process: communications

The second step of the mutation operator is dedicated to communications and concerns the second and third slices of the chromosome. For this part, we randomly choose a communication k among the m_u ones that are allowed to be vulnerable. We change its status, marking it unsecured when it was secured, and conversely marking it secure if it was not secured.

The choice of the security implementation depends on the assumptions we took for intra-partition communications decided before starting the DSE. In case an intra-partition communication has to be secured (section 6.2), a security implementation is randomly chosen among the four alternatives (X-F, X-T, X-TM). We highlight that these choices are equiprobable (i.e. each security implementation has the same probability to be chosen).

Otherwise, in the case of the use of proper mechanisms to ensure memory protection for attacks from inside a partition, only the functions calls will be used for all the communications during all the DSE. Then the value F-F is assigned to the chromosome.

The security implementation is set at location $n + 1$ in the chromosome. It will be used for all secured communications. The chosen security implementation will be applied to ensure the security of all the communications marked as secured in the chromosome.

Figure 7.7a illustrates a mutation operation on a communication. It shows two models. the first model, at the left, corresponds to the one previously presented in figure 7.3 (Section 7.2.3). The second model, at the right, corresponds to a mutated model obtained after applying a communication mutation to the first model. The chromosomal representation of the mutated model is presented in figure 7.7b.

This mutation considers the changing of the security status of the communication from task τ_3 to task τ_4 chosen randomly. This communication, previously marked as vulnerable (represented with a red arrow on the left side of figure 7.7a), becomes secured (represented with a green arrow on the right side of figure 7.7a).

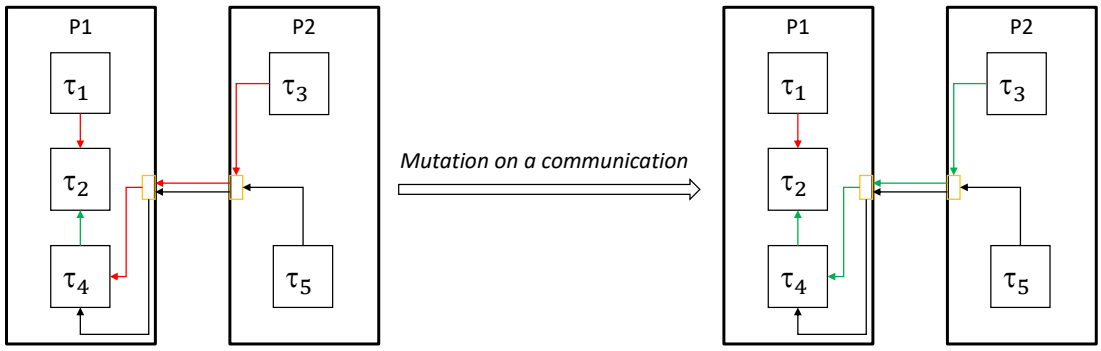
With its new status, this communication needs security features applied according to the chosen security implemented. In this example, we select the F-F value.

We highlight that in case of security through dedicated tasks (X-F, X-T, X-TM), new tasks (encryption, decryption, key set up, or hash tasks) and communications are added as described in section 6.2.2. These tasks and communications are not represented in the chromosome but added in the actual model when evaluating the corresponding solution by Cheddar.

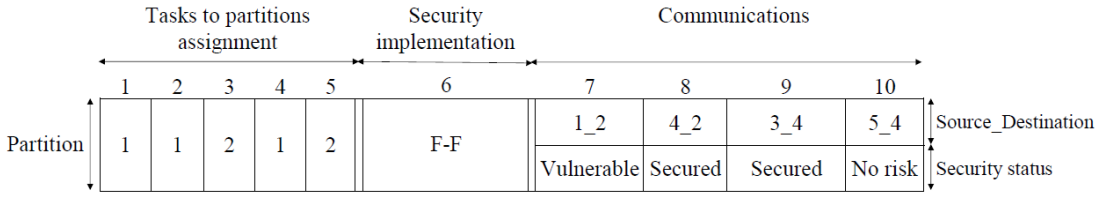
Our mutation operator is sketched in Algorithm 2.

Algorithm 2: Mutation algorithm

```
1  Input:
2  A chromosome that represents a solution
3  A mutationOption specifying which flavor of mutation to apply
4  A number of iterations nb_iter of the exploration
5  The number of the actual iteration actual_iter
6  Output: A mutated solution
7  while solution is not mutated do
8      Choose a random partition P_p
9      if (mutationOption = "task_grain") or ((mutationOption =
         "mix_grain")and (actual_iter <= nb_iter/2) then
10         Choose a random Application A_j
11         if All the tasks of A_j are not assigned to P_p then
12             Assign all the tasks of A_j to P_p
13         end if
14     else
15         if (mutationOption = "app_grain") or ((mutationOption =
             "mix_grain")and (actual_iter > nb_iter/2)) then
16             Choose a random task τ_i
17             if Task τ_i is not assigned to P_p then
18                 Assign task τ_i to P_p
19             end if
20         end if
21     end if
22     Choose a random communication k that is vulnerable, among m_u
         ones
23     if k has no encryption or hashing function then
24         Choose a random security implementation and update
             chrom[n + 1] with it.
25         mark communication n + 1 + k as secured
26     else
27         mark communication n + 1 + k as vulnerable
28     end if
29     Evaluate the new solution
30     if It does not respect the scheduling and security constraints then
31         Reject the mutated solution
32         Proceed with another mutation
33     else
34         Return the mutated solution
35     end if
36 end while
```



(a) Communication mutation model illustration



(b) Communication mutation chromosome illustration

FIGURE 7.7: Communication mutation illustration

During the exploration, it may happen that after multiple consecutive mutations, we fail to provide a feasible solution. Therefore, instead of running an infinite number of consecutive mutations without success, we propose a predefined number of unsuccessful mutations after which the DSE should be terminated and we then inform the designer that the DSE fails to explore more solutions. The solutions already stored in the archive till that event are then proposed to him.

7.2.5 Mutation algorithm improvement

Whatever the considered mutation algorithm (*task-grain*, *app-grain*, *mix-grain*), at the end of the exploration, it may happen that the archive contains only a few solutions. At a given iteration, multiple iterations may fail to find another nondominated solution (i.e. all the feasible solutions find after mutations are dominated by at least one solution already in the archive). At the given iteration, a current solution may be unable to mutate enough to provide more non-dominated solutions. As a solution to those problems, we proposed that after a predefined number of successive mutations on a current solution that fail to provide a nondominated solution, we choose randomly a solution in the archive to become the current solution that should be mutated at the next iteration. It helps to increase the chance to provide a nondominated solution and then increase the diversity and the number of solutions in the archive at the end of the DSE.

7.2.6 Initial solutions and archiving process adaptation

The initial current solution we choose (step 2 of Fig. 7.1) is a solution which resolves all the security vulnerabilities while using one partition for all the tasks. If the scheduling analysis of that solution reveals that there is no missed deadline, then the optimal solution is found (perfect for both schedulability and security aspects) and DSE (step 3 in Fig. 7.1) is not useful.

To make our PAES method faster and to favor diversity of solutions, we also add extra solutions in the archive (step 5 in Fig. 7.1), by combing various strategies based on tasks to partitions assignment and security vulnerabilities while considering the Pareto dominance concept. Indeed, we consider solutions with single partition (i.e. all the tasks assigned to a single partition) or balanced partitions (i.e. tasks are equally distributed to a fixed number of partitions). We also consider Multi Single Level Secure (MSLS) [102] partitioning based on confidentiality (resp. integrity). It implies that each partition can only host tasks of the same confidentiality (resp. integrity) level. We add in the archive two MSLS solutions: an MSLS solution based only on confidentiality level and another one based only on integrity level. For each above-mentioned option of tasks to partitions assignment, we generate two solutions by solving none or all vulnerabilities.

7.3 Conclusion

This chapter presents a DSE approach to provide trade-offs between schedulability and security. The problem raised being a MOOP, we opted for MOEA by applying PAES multi-objective metaheuristic. We adapted the PAES technique by specifying the objectives functions, constraints, encoding of solutions, mutation operators, initial solutions and archiving process according to the addressed problem. Since we consider TSP systems, our customized PAES includes the tasks to partitions assignment and the security of intra and inter-partition communications. Different experiments are conducted in the next chapter to evaluate the proposed approach.

8

Experiments and evaluations

In this chapter, we evaluate the proposed mutation algorithms presented on the previous chapter. We also identify key architecture parameters to build trade-offs between security and schedulability. Schedulability is evaluated by the number of soft deadline misses. Confidentiality and integrity are evaluated through the number of security vulnerabilities.

We perfomed seven experiments based on six benchmarks. Section 8.1 shows an experiment presenting a case where there is no conflict between security and schedulability. Section 8.2 describes an experiment that shows the effectiveness of our approach in providing non-dominated solutions and the impact of data size in the conflict between schedulability and security. Section 8.3 presents a set of experiments that show the impact of the processor utilization, the number of partitions, and the data size. Section 8.4 proposes a comparison with an exhaustive DSE, which allows us to evaluate the quality of the solutions provided by the heuristic. Finally a summary of the experiments and a conclusion of the chapter are given in Section 8.5.

8.1 Experiment 1: illustration with a flight controller application

DSE has to be done when there is a conflict between security and schedulability. This experiment is performed to verify if a conflict between security and schedulability exists. We evaluate the ability of our DSE approach to detect TSP systems for which there is no need to proceed with DSE.

8.1.1 Conditions of experiment

We conduct this experiment with the Research Open-Source Avionics and Control Engineering (ROSACE) [27] benchmark that describes a longitudinal and multi-periodic flight controller. It is composed of 15 periodic tasks, a processor utilization of 29% and on average a small size data of 8 bytes. Figure 8.1 presents in detail the communications between the tasks of ROSACE. Communications vulnerabilities are induced from confidentiality and integrity levels of the tasks.

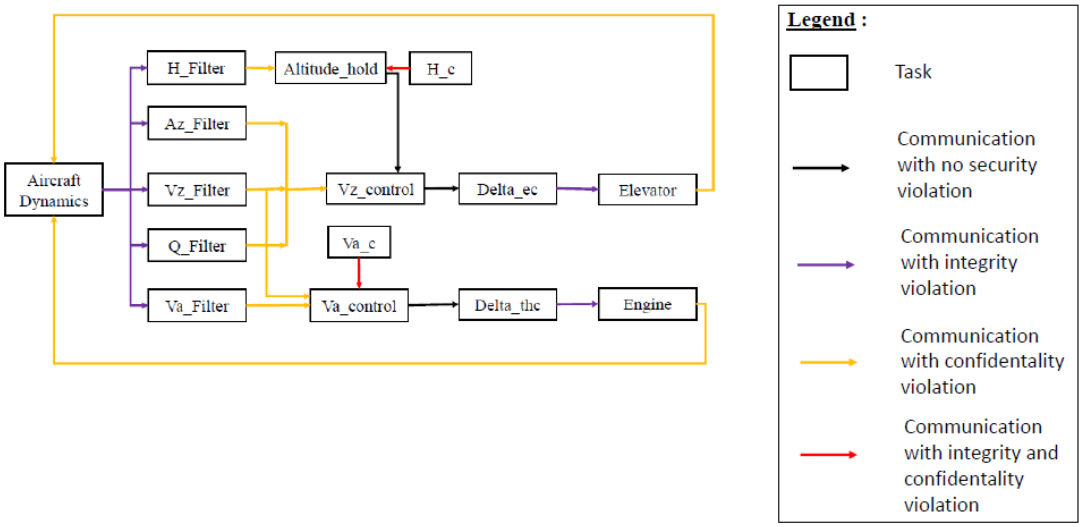


FIGURE 8.1: ROSACE flight controller application

Task parameters, which are summarized in Table 8.1, are taken from [27]. We fixed the security parameter (confidentiality and integrity levels) values to fit with the worst-case: they are set to maximize the number of vulnerabilities in the application.

If we can show that for the ROSACE application with a very high number of security vulnerabilities, resolving all the security vulnerabilities does not impact the schedulability, then we expect that for the same application with few security vulnerabilities, there would be no conflict between schedulability and security.

By assuming that the CPU frequency is 1.2 GHz, values provided by the crypto++ benchmark [163], and the data size of the case study is 8 bytes, encryption execution, refreshment encryption key and hash execution times are respectively 0.166 us, 88.83 us and 0.1 us. Those execution times are added to the C_i parameter of the ROSACE tasks as for this experiment security features are implemented by function calls only (security implementation F-F in table 6.1). For this experiment, we fixed a maximum of 2 partitions.

8.1.2 Results

Our DSE approach starts with an initial solution solving all security vulnerabilities (see Section 7.2.6). The scheduling analysis of the resulting architecture shows that all tasks meet their deadlines. This is due to the fact that initially, ROSACE is characterized by a low processor utilization of 29%. The method we propose therefore returns that it is not necessary to carry out a DSE for such an application. We note that the addition of the security features only increases the processor utilization to 37%. We explained this result by the low overhead introduced by encryption and hash tasks because they are proportional to the data size of the considered application and ROSACE has small data size (8 bytes).

From this experiment, we conclude that the data size and the initial processor utilization of the application are part of the most important criteria that determine the necessity of the DSE. By starting with an initial solution in which the system is fully secured, our method detect such cases.

8.2 Experiment 2: illustration with a flight controller and JPEG applications

We conduct this experiment with two applications: ROSACE and a JPEG application [28] which has a higher processor utilization. The fully security-oriented initial solution is not schedulable. The objective of this experiment is to show the effectiveness of the DSE approach in providing trade-offs between security and schedulability with objective values close to the Pareto front. Further, it consists of comparing our three proposed mutation algorithms *task-grain*, *app-grain* and *mix-grain* in order to determine the most efficient.

8.2.1 Conditions of experiment

The JPEG application is composed of five computation steps: color space conversion, DCT (Discrete Cosine Transformation), quantization, encoding, and memory Read/Write. It is characterized by a processor utilization of 12%. We assume that the image is in 4CIF format (704x576 pixels) for the JPEG. With 2 bytes per pixel, the data size is equal to 792 Kilobytes. Considering a processor frequency of 1.2 GHz and the data size of each application, the execution times of the encryption task, the refreshment encryption key task and the hash task are respectively of 0.166 us, 88.83 us and 0.1 us for ROSACE and 16834 us, 88.83 us and 10173 us for the JPEG application. We computed these values by considering values provided by the crypto++ benchmark. We supposed that the key

for encryption is cyclically refreshed. Then we fixed a period of 1000 s which guarantees that it is set only once during application execution time.

Parameters of the two benchmarks are summarized in Table 8.1. Task parameters (period, capacity) are taken from the benchmark in [27] and [28] respectively for ROSACE and JPEG applications.

For this experiment, we fixed a maximum of two partitions. We assumed that intra-partition communications are vulnerable which implies an exploration based only on function calls security implementation (F-F) defined in table 6.1.

For all experiments except experiment 8.3.4, we assumed an overhead of 10 us (resp. 280 us) for an intra-partition (resp. inter-partition) communication. We have chosen overhead values that impact the scheduling results for our test cases, i.e. impacts the search space.

8.2.2 Results

We conduct this experiment for the three mutation algorithms *task-grain*, *app-grain* and *mix-grain*. For each mutation algorithm, the exploration is conducted for a number of iterations fixed to 4000. In this experiment, the first phase (application level phase) of *mix-grain* ends at 3000 iterations. The number of iterations is an input that can be fixed depending on the time available to perform the DSE approach to explore a significant number of candidate solutions. The approach provides archives of 5, 4, and 4 solutions respectively for *app-grain*, *task-grain* and *mix-grain*.

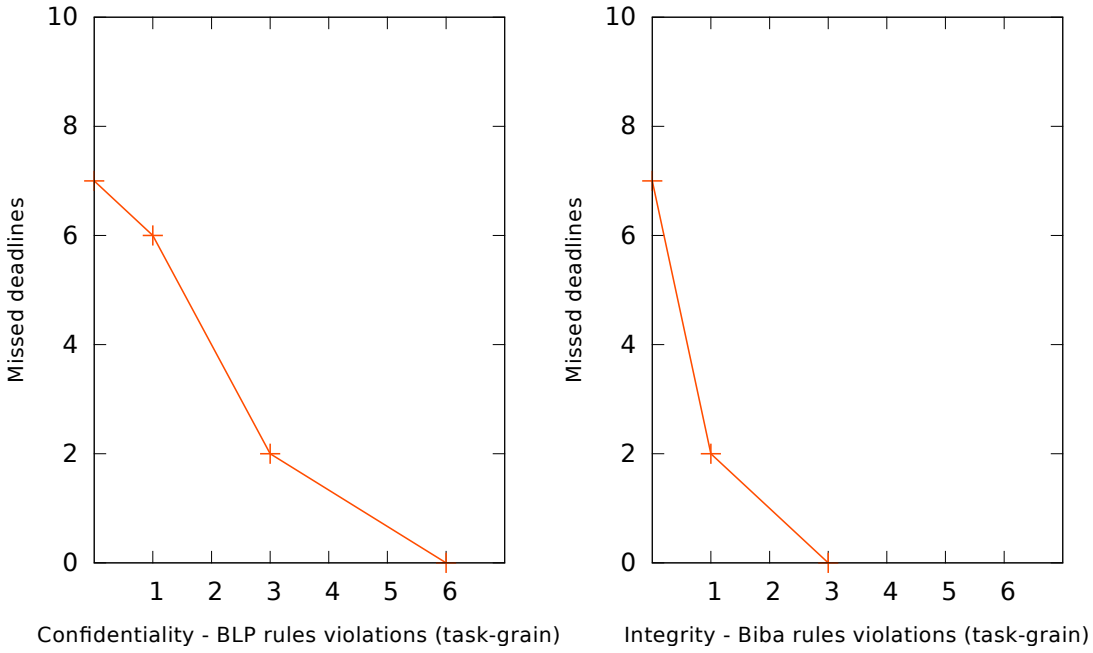
Figure 8.3 shows the set of non-dominated solutions found by couples of objectives for each algorithm.

To keep this case study schedulable, for both *task-grain* and *app-grain*, the DSE proposes a solution that tolerates 6 violations of BLP rules and 3 violations of Biba rules while *mix-grain* proposes a solution with 3 violations of BLP rules and 1 violation of Biba rules. These solutions are based on a single partition and the ones proposed by *task-grain* and *app-grain* are identical and correspond to one of our initial solutions characterized by all tasks assigned to one partition and no security vulnerabilities fixed. We notice that the archive computed by the *mix-grain* contains a schedulable solution different from the initial solution. This solution shows the relevance of proceeding with a DSE to find better solutions than initial solutions which are more intuitive and/or extremes. We underline that this solution is part of the design space of the three mutation algorithms and then could have been found by the *task-grain* and *app-grain* algorithms.

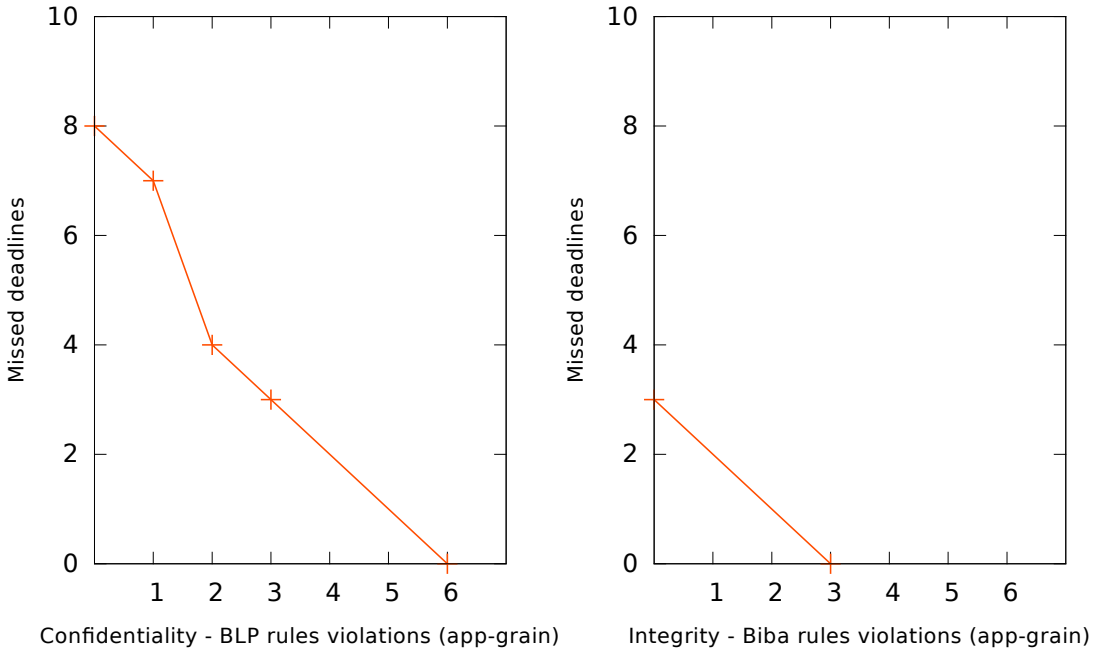
We observe that *mix-grain* proposed better solutions than *app-grain*. As an example, for a fully secured system, *app-grain* proposed a solution with 8 missed deadlines while *mix-grain* proposed a solution with 7 missed deadlines. Both

TABLE 8.1: Case studies task parameters

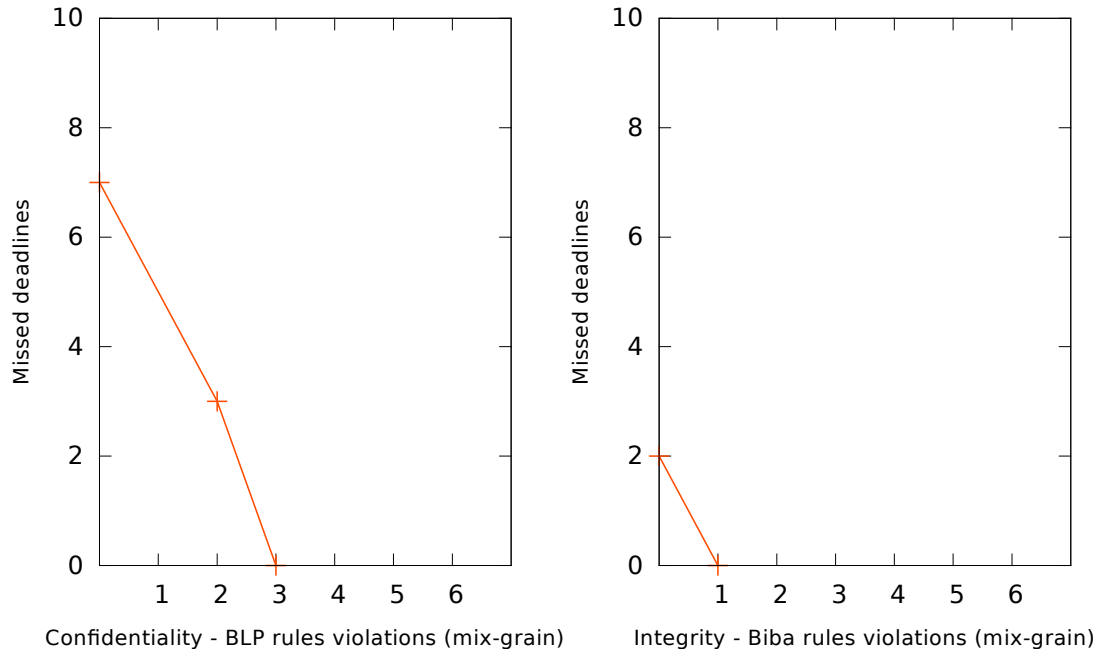
Tasks	C_i **[us]**	T_i **[us]**	CL_i	IL_i
Experiment 1: ROSACE				
Aircraft Dynamics	200	5000	Secret	Medium
Va_c, H_c	500	20000	Top_secret	Medium
H_Filter, Az_Filter, Vz_Filter, Q_Filter, Va_Filter	100	10000	Top_secret	High
Altitude_hold, Vz_control, Va_control	100	20000	Secret	Medium
Delta_ec, Delta_thc	500	20000	Secret	High
Engine, Elevator	100	5000	Top_secret	Medium
Experiment 2: JPEG				
Matrix transpose	41	20000	Top_secret	High
Color space conversion	41	20000	Secret	Medium
Wrapper 1, Wrapper 2	625	20000	Secret	Medium
Quantization	270	20000	Top_secret	Medium
Encoder	760	20000	Secret	High
Memory Read/Write	41	20000	Secret	Medium
Experiment 3: CFAR				
CFAR_complex	10000	90	Top_secret	High
CFAR_square_scale	10000	50	Top_secret	High
CFAR_gather	10000	340	Top_secret	High
CFAR_printer	10000	30	Top_secret	High
Experiment 3: Autopilot				
Data collection	UUnifast	UUnifast	Secret	High
Control law computing	UUnifast	UUnifast	Top_secret	High
Actuator	UUnifast	UUnifast	Secret	Medium
Fault auditor	UUnifast	UUnifast	Secret	Medium
IFBIT	UUnifast	UUnifast	Top_secret	High



(a) Task-grain



(b) App-grain



(c) Mix-grain

FIGURE 8.3: Schedulability vs. security with ROSACE&JPEG

solutions used the same security implementation (F-F) and 2 partitions but the tasks to partitions assignments are different. With *app-grain*, each application is assigned to one different partition, but with *mix-grain*, a task of JPEG application was moved to the partition with the tasks of ROSACE. This solution can never be provided by *app-grain* since this algorithm only explores applications to partitions mapping. This shows the relevance of the algorithm *mix-grain*. These better results could be explained by the fact that the refinement phase of *mix-grain* allows to improve the solutions found at *app-grain* level, while avoiding the difficulty observed with *task-grain* which is not able to converge directly toward those solutions because of a larger search space.

Fully secured solutions are found at the cost of a few missed deadlines for soft deadline tasks. At the opposite, missed deadlines can be reduced if security aspects are partially sacrificed, up to the designer choice.

The difference between this test-case and the previous one in Section 8.1 is the addition of the JPEG application which is characterized by a large data size. We remark that high data size impacts the schedulability. A high data size implies a high extra processor utilization dedicated to securization of data. This experiment shows the relevance of our approach which is able to provide significantly different trade-offs between security and schedulability while considering different tasks/applications to partitions assignments and security implementations. The usage of our DSE approach allows system designers to explore solutions with

trade-offs between schedulability and security.

8.3 Experiments 3-6: illustration with a flight controller, multimedia based application, CFAR and autopilot applications

The objective of experiments 3 to 6 (Sections 8.3.1 to 8.3.4) is to investigate the impact that the variation of some parameters may have on the conflict between security and schedulability. The investigated parameters are: the processor utilization, the number of partitions, security implementation and the data size considering communications overheads.

We perform these experiments with the same conditions except the maximal number of partitions and the security implementations. We constitute a case study based on six applications. We use ROSACE and JPEG aforementioned, CFAR [4], and three instances of an autopilot [164] application.

- CFAR (Constant False Alarm Rate detection) is a digital signal processing application that detects targets based on the variation of background noise [4]. The parameters of the CFAR application are described in Table 8.1. We assume a data size of 8 bytes. The execution times of the encryption task, the refreshment encryption key task and the hash task are respectively 0.166 us, 88.83 us and 0.1 us.

- The autopilot application is an application composed of 5 tasks that collects data from sensors and sends commands via actuators to an aircraft pilot. For the experiments in this section, the tasks parameters of the autopilot are synthetically generated based on the UUnifast algorithm [29]. We adapt the UUnifast algorithm to generate randomly task capacities according to a uniform distribution with a fixed number of tasks and a given processor utilization. Then we generate different models with different values of total processor utilization U from 50% to 100%. We guarantee that the tasks generated are periodically harmonic.

  The security parameters are given in Table 8.1. We assume that the data size is 16 Kilobytes. Therefore the execution time of the encryption, the refreshment of the encryption key, and the hash tasks are respectively 340 us, 88.83 us, and 205.52 us.

- For the ROSACE and JPEG applications, we keep the same parameters fixed in the previous experiment.

8.3.1 Experiment 3: result of PAES when varying processor utilization

8.3.1.1 Conditions of experiment

We initiate this experiment to validate the effectiveness of the DSE in different cases generated by the variation of real-time parameters such as processor utilization. We evaluate the impact of these parameters on the DSE approach.

We evaluate the impact of the processor utilization by performing the DSE approach on different architectures, with a maximum of two partitions, by varying the processor utilization.

We conduct this experiment for the three mutation algorithms on each case study generated by varying the processor utilization U from 50% to 100%. Each test was conducted for a number of iterations fixed to 2000.

For each mutation algorithm, according to security, our PAES approach includes an exploration based on the function calls security implementation (F-F).

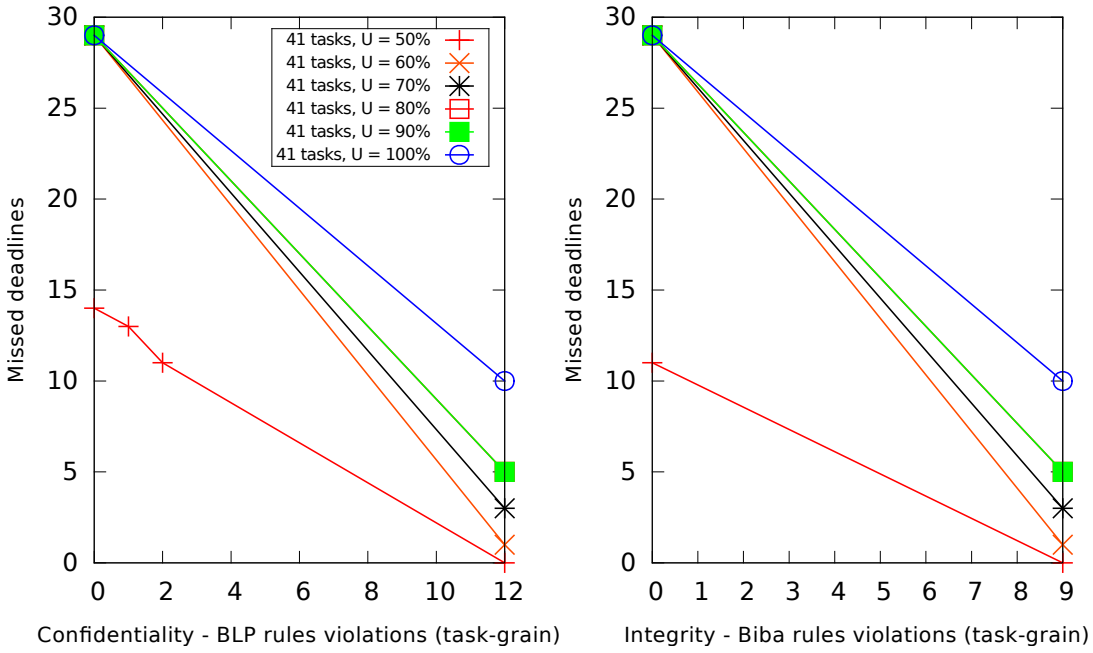
8.3.1.2 Results

Figure 8.5 shows the set of non-dominated solutions by couples of objectives for mutation algorithms *task-grain*, *app-grain* and *mix-grain*.

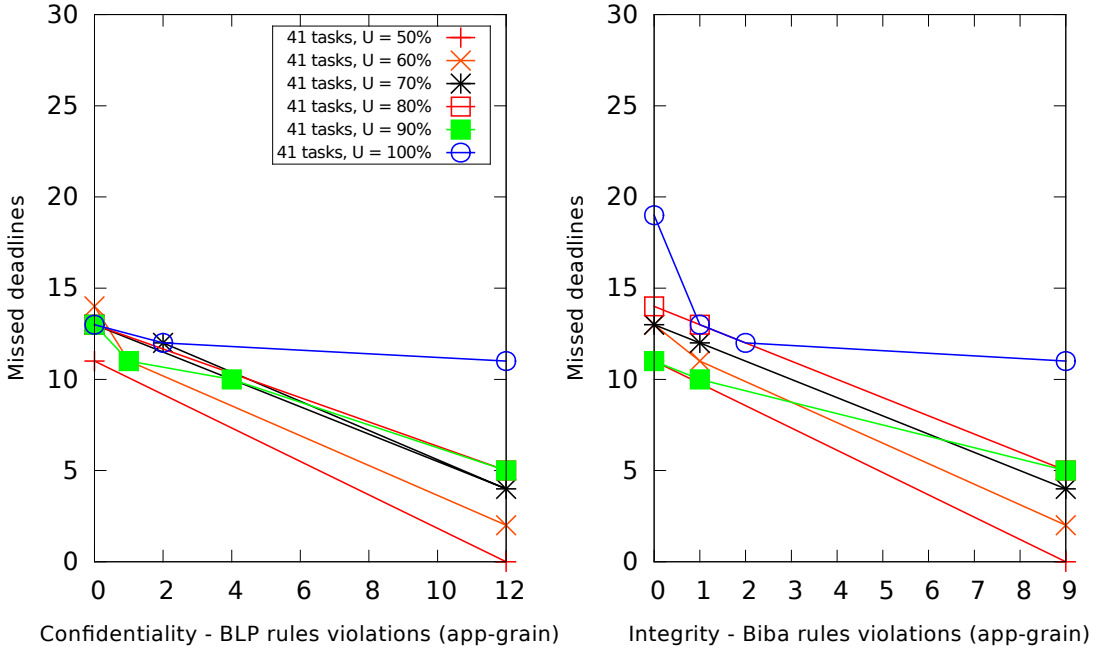
We remark that the more the processor utilization increases, the more difficult it becomes to ensure security while maintaining the system schedulable. In figure 8.5, by tolerating 12 confidentiality and 9 integrity violations, only the DSE with a processor utilization U=50% proposes a schedulable solution. As the processor utilization increases, the more tasks miss their deadline (e. g. with *mix-grain*, 2 missed deadlines for U= 60% and 5 missed deadlines for U= 90%).

For these experiments, in most of the cases, *mix-grain* and *app-grain* propose better solutions than *task-grain*. For example, for all processor utilization values except for 50%, fully secured solutions (for integrity or confidentiality) are obtained with a lower number of missed deadlines with *app-grain* or *mix-grain* as compared to *task-grain*. This result can be explained by the smaller size of solution space (for *app-grain* or for the first phase of *mix-grain*).

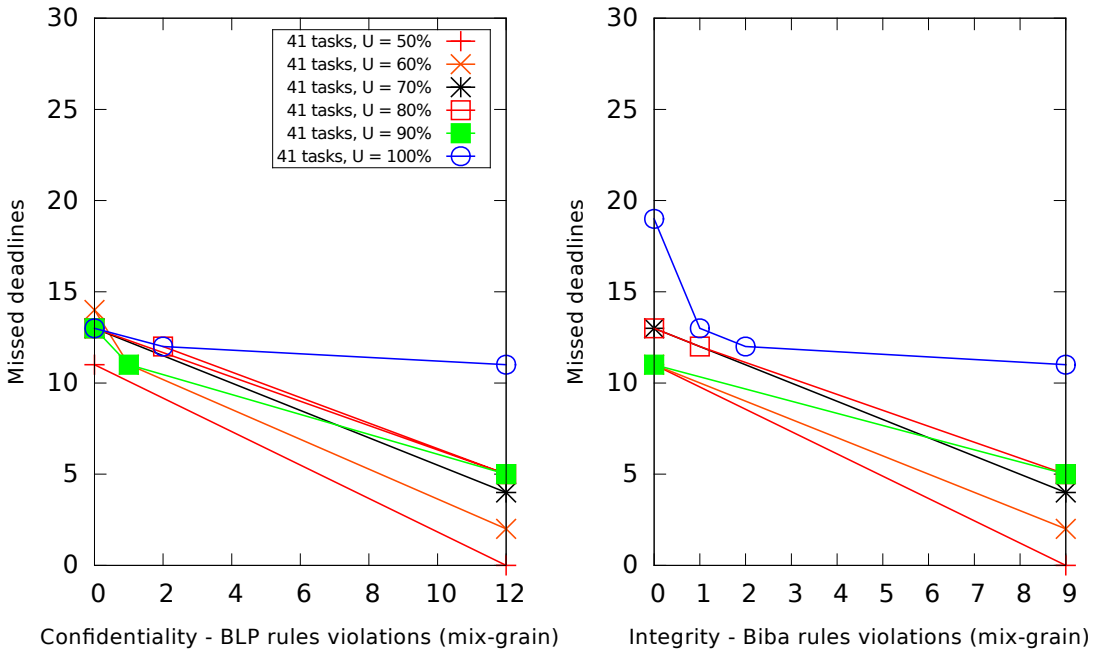
We also remark that *mix-grain* never proposes worse solutions compared to *app-grain*. We only have one solution where *mix-grain* proposes a solution out of the scope of *app-grain* by allowing the assignment of tasks of the same application to different partitions. The rarity of this kind of solution can be explained by the larger size of the search space of the first phase of *mix-grain* (larger than in the previous experiment that only comprises 22 tasks instead of 41 in the present experiment). This rarity can also be explained by the fixed duration of the 2 phases of *mix-grain* which are non-negligible parameters in the exploration. By



(a) *task-grain*



(b) *app-grain*



(c) *mix-grain*

FIGURE 8.5: Schedulability vs. security with processor utilization variation

increasing the duration of phase 2 and reducing slightly the duration of phase 1, more and interesting solutions out of the scope of *app-grain* could have been found. From figure 8.5, we conclude that the more the processor utilization increases, the more the security impacts the schedulability for this experiment.

Second, we affirm that *mix-grain* can be interesting in providing some particular solutions that *app-grain* cannot propose if the durations of phase 1 and 2 of *mix-grain* are well defined. That leads us to point out that *task-grain* could be less efficient in many cases because its design space is too large and it becomes difficult for this algorithm to converge.

8.3.2 Experiment 4: results of PAES when considering intra-partition communications non-vulnerable

8.3.2.1 Conditions of experiment

We conduct this experiment on case studies generated in the experiment 8.3.1 for processor utilization from 50% to 100% and a maximum of 2 partitions. We assume that intra-partition communications are not vulnerable. Then we only consider the security of inter-partition communications through the security implementations X-F, X-T, and X-TM.

Case study processor utilization	50%	60%	70%	80%	90%	100%
#Missed deadlines	0	0	1	5	5	11
#BLP rules violations	0	0	0	0	0	0
#Biba rules violations	0	0	0	0	0	0

TABLE 8.2: DSE with intra-partition communications considered as secured with *mix-grain* and a maximum of 2 partitions

We conduct this experiment with only *mix-grain* mutation algorithm on the case studies. We choose *mix-grain* as we remarked during the previous experiments that its solutions are non-dominated by those proposed by *app-grain* and *task-grain*.

8.3.2.2 Results

Table 8.2 presents the fitness values of solutions proposed for each case study. We only have one solution for each case study that we analyze in the remainder of this section. For these case studies, we remark that the DSE proposed better solutions when we suppose that intra-partition communications are secured and non-attackable. It may be explained as it implies fewer communications to secure comparing when all communications including intra-partition communications are vulnerable. The overhead introduced by the security is less significant. When only one partition is used, there is no communication to secure, and the model is considered as fully secured. Then if the model is initially schedulable, our tool declares that there is no need to proceed with DSE since the optimal solution corresponds to one of our initial solutions (all tasks assigned to one partition and no secured communication). For example, the case study with processor utilization of 50% is schedulable with all the tasks assigned to the same partition.

With a processor utilization of 60%, by assigning all the tasks to one partition and considering the communication overhead, the model is considered as fully secured since there is no inter-partition communication but the model is not schedulable (2 missed deadlines). Then at the 39th iteration, the DSE proposes a schedulable solution with 2 partitions where ROSACE is assigned to a first partition and the others applications (CFAR, JPEG, 3 autopilots) are assigned to a second partition (solution in Table 8.2). Since there is no communication between ROSACE and the other applications, there is no inter-partition communication. Then the solution has no security overhead and is considered as fully secured. The exploration stops at the 39th iteration instead of continuing till the end of the fixed number of iterations (2000) since the optimal solution is found.

Another option is revealed when processor utilization is 70%. For this case study, by assigning all the tasks to one partition and considering the communication overhead, the model is considered as fully secured since there is no inter-partition

communication but the system is not schedulable (4 missed deadlines). The DSE proposes a better solution, fully secured, by assigning ROSACE application and 4 among 5 tasks of the second autopilot application to a partition and other tasks (from CFAR, JPEG, first autopilot, third autopilot applications, and the remaining task of the second autopilot) to another partition. This model implies 2 inter-partition communications to secure, and they were secured with the security implementation X-T.

This case study confirms also the good choice and the relevance of *mix-grain* because the proposed solution has tasks of the same application split onto different partitions (solution out of the scope of *app-grain*) and the larger space of solution impacts significantly the convergence of *task-grain*.

For the remaining case studies (80%, 90%, 100%), the DSE does not propose a better solution than the initial solution fully secured with all tasks assigned to only one partition.

8.3.3 Experiment 5: results of PAES with variation of the maximum number of partitions from 2 to 4

8.3.3.1 Conditions of experiment

Here, we conduct the experiment 3 but with a maximum of four partitions, to evaluate how increasing the number of partitions impact the search. Then, during the exploration, we investigate the solutions with one, two, three, and four partitions in order to find those which allow the best trade-off solutions. Increasing the number of partitions enlarges the solution space, offering opportunities to find better solutions but it also induces the difficulty of exploring efficiently this larger solution space. Our different mutation algorithms may or not handle this increasing complexity. Furthermore, because of the impact of inter-partition communications on both security and scheduling, increasing the number of partitions may degrade ours metrics and could be in fact a drawback.

We conduct this experiment for *mix-grain* algorithm on the case studies generated in experiment 3, with processor utilizations of 60% and 90%, with a number of iterations fixed to 2000. We run the PAES tool with a maximum number of 2 partitions and of 4 partitions and by assuming that intra-partition communications are attackable; then only security implementation (F-F) is applied.

8.3.3.2 Results

As shown next, the size of the search space is much larger for 4 partitions than for 2 partitions. Let m applications with a total number of n tasks, assigned

to r partitions. The size of the search space DS corresponds to the number of solutions of the addressed problem.

It is computed with the number of tasks/applications to partitions assignment DT and the number of communication parameter implementations DC. The former corresponds to the Stirling number of the second kind $S(n, r)$ [165], that is the number of possibilities to divide n tasks into r partitions at most.

According to our approach, to represent communications, each of m_u vulnerable communications (see objective functions rows of Table 7.2) has a possibility of 2 values (secured or vulnerable), and the secured option has one possibility to implement security feature, thus $DC = 2^{m_u}$. In the case when intra-partition communications are non-attackable and only inter-partition communications are attackable, the DC becomes $DC = 3 \cdot 2^{m_u}$ since there are 3 possibilities to implement security feature. For the rest of this section, we consider that intra-partition communications are attackable. *task-grain* and *mix-grain* have the same size of search space since they can investigate all the tasks to partitions assignment possibilities.

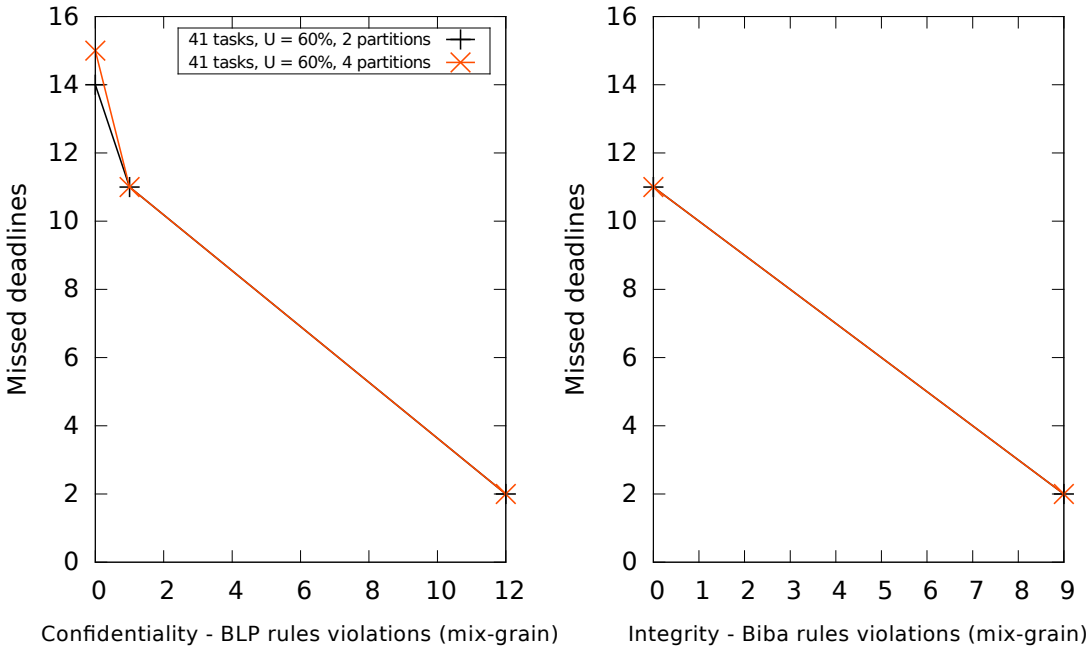
For these methods, with m applications composed of n tasks, and m_u vulnerable communications to be mapped into r partitions, $DS = DT \cdot DC = (\sum_{q=1}^{r} S(n, q)) \cdot 2^{m_u}$. For the same case study, *app-grain* is defined by a size space of $DS = DT \cdot DC = (\sum_{q=1}^{r} S(m, q)) \cdot 2^{m_u}$ since it considers only applications to partitions assignments.

As example, for this experiment with 6 applications, 41 tasks, 10 weakly sensitive communications and a maximum number of 2 partitions (resp. 4 partitions), *task-grain* and *mix-grain* have a search space of 1.12×10^{15} (resp. 2.06×10^{26}) solutions while *app-grain* has a search space of 31,744 (resp. 65,560) solutions.

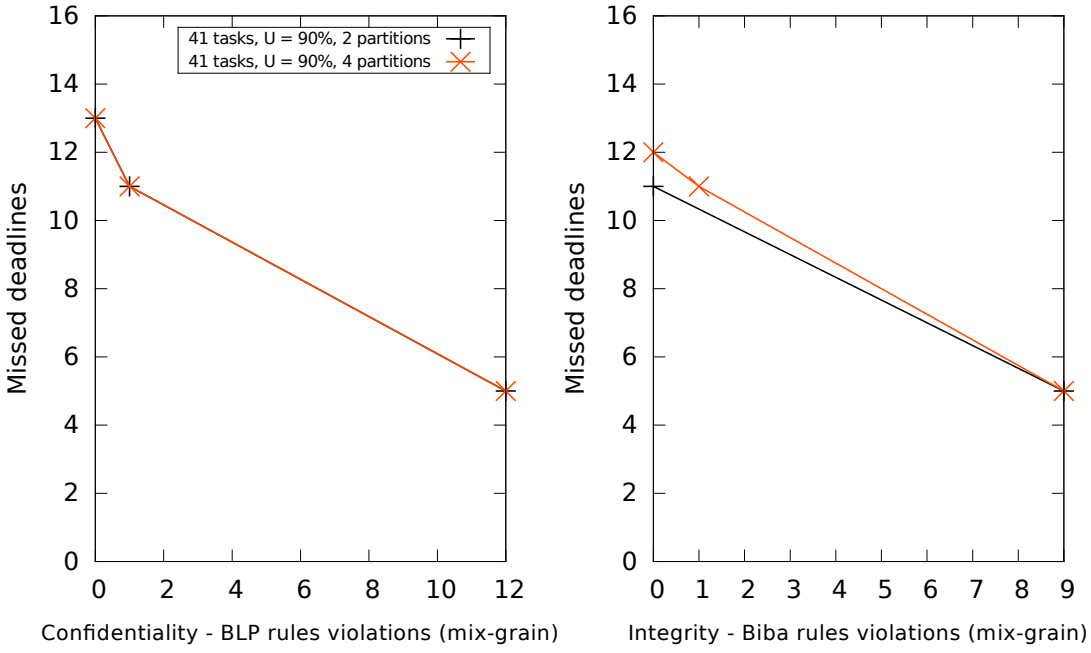
Figure 8.6 shows the set of non-dominated solutions by couples of objectives for *mix-grain*. In this experiment, we run the first phase during 1/2 of the total number of iterations.

For the case study with processor utilization of 90%, the solutions with the maximum number of 2 partitions are better or equal to the solutions with the maximum number of 4 partitions (figure 8.6b). By analyzing the architectures of the solutions, we remark that for the exploration with the maximum number of 4 partitions, no solution in the archive used more than 2 partitions.

For the case study with processor utilization of 60%, the solutions with the maximum number of 2 partitions are better or equal to the solutions with the maximum number of 4 partitions (figure 8.6a). However, the DSE with a maximum of 4 partitions proposed an archive with some solutions with more than 2 partitions. We remark that these solutions are worse than the solutions provided by the exploration with a maximum of 2 partitions. For example, there is a fully secured model with 17 missed deadlines obtained when the tasks are split into



(a) Processor utilization: 60%



(b) Processor utilization: 90%

FIGURE 8.6: Schedulability vs. security with variation from 2 to 4 partitions

4 partitions instead of 14 missed deadlines obtained where tasks are split into 2 partitions during the DSE with a maximum of 2 partitions. This can be explained because the design space with 4 partitions is larger and the algorithms have difficulties to converge to non-dominated solutions in this case. The inter-partition communications overheads can also explain the reject of 4 partitions solutions.

In conclusion, for these specific case studies, the maximum number of partitions used by non-dominated solutions for our PAES exploration is 2 even when 4 partitions are allowed. With 4 partitions, interpartition communication costs reduce the possibilities for such solutions, making them difficult to find in an enlarged search space.

For these case studies and the same number of iterations, we remark that the DSE with 4 partitions has a higher execution time than the DSE with 2 partitions. For example, considering the case study with processor utilization of 60%, the DSE with 2 partitions takes 51 minutes instead of 74 minutes for the DSE with 4 partitions.

8.3.4 Experiment 6: results of PAES while considering APEX calls execution times given in SFPBench [2]

8.3.4.1 Conditions of experiment

This experiment is conducted to evaluate how different overheads of intra-partition and inter-partition communications can impact the results of the DSE. We perform this experiment with the three mutation algorithms on the case study generated in experiment 3 with a processor utilization of 60%.

We assume that intra-partition communications are attackable; then only security implementation (F-F) is applied. The particularity of this experiment is that we consider the overheads of intra-partition and inter-partition communications based on the execution time of the APEX calls given from a benchmark in [2]. Table 8.3 presents the values (in microseconds) taken from this benchmark and the corresponding values used for each application in our case study. We use blackboards (resp. sampling ports) for intra (resp. inter) partitions communications.

8.3.4.2 Results

We compare the results to the ones obtained in figure 8.5 of the section 8.3.1 for the same case study with the processor utilization at 60%, using *mix-grain* and a lower communication overhead of 10 us (resp. 280 us) for intra-partition (resp. inter-partition) communications. As shown in Table 8.4, the archive contains only one solution per mutation algorithm.

	Display_black_board [us]	**Write_black_board [us]**
SFPBench (16 bytes)	1.52	1.96
ROSACE/CFAR (8 bytes)	0.76	0.38
JPEG (792 Kbytes)	77143.04	99473.04
Autopilot (16 Kbytes)	1556.48	2007.04
	Read_sampling_port [us]	**Write_sampling_port [us]**
SFPBench (16 bytes)	8.48	10.09
ROSACE/CFAR (8 bytes)	4.24	5.04
JPEG (792 Kbytes)	430376.96	512087.68
Autopilot (16 Kbytes)	8683.52	10332.16

TABLE 8.3: APEX calls execution times

	#missed deadlines	**#BLP rules violations**	**#Biba rules violations**
Solutions with *task-grain*	37	12	9
	38	0	0
Solutions with *app-grain*	23	0	0
Solutions with *mix-grain*	23	0	0

TABLE 8.4: Schedulability vs. security with SFPBench APEX calls measurements

First, we observe that the DSE proposed worse solutions, as compared to the previous experiment, with much higher missed deadlines. This can be explained by the considerable overheads of the sampling port and the large data size of some applications in our case study such as JPEG. Second, we observe that all the proposed solutions used 2 partitions (except one from *task-grain*) and that all the solutions generated with only one partition were dominated and thus rejected (the one from *task-grain* is in fact dominated by those of *app-grain* and *mix-grain* archives), despite the overheads induced for inter-communications. The communications between partitions are sufficently reduced in the solutions found to avoid too much extra missed deadlines due to overcosts, as compared to the ones induced when packing tasks into a single partition.

We observe that the 3 mutation algorithms proposed fully secured solutions with different numbers of missed deadlines. *mix-grain* and *app-grain* were most efficient with a better number of missed deadlines (23 missed deadlines for *mix-grain* and *app-grain* vs. 38 for *task-grain*). This can be explained by the larger space of solutions for *task-grain* which implies difficulties to converge.

In conclusion, the data size of the application impacts again the conflict between security and schedulability not only through security features (encrypter, decrypter, hashing functions) but also through the inter-partition and intra-

partition communication overheads.

8.4 Experiment 7: comparison of our PAES tool results vs. exact solutions

This experiment consists of validating the accuracy of our DSE approach by comparing an approximate Pareto front obtained with our 3 PAES based approaches to the exact Pareto front for testcase of Section 8.2. The exhaustive method allows to compute the exact Pareto front: all feasible solutions are generated and evaluated and non-dominated ones among them constitute the exact Pareto front. For this purpose, we implemented an exhaustive search tool that works as follows:

- It initializes an empty archive
- It enumerates all the possible solutions. So for each implementation of tasks to partitions assignment, we enumerate all possible values of vulnerable communications.
- It performs feasibility test on each generated solution. It consists of verifying that the generated solution respects the constraints defined in table 7.1. If the constraints are met then the solution is considered as feasible.
- It evaluates each feasible solution and compares it to the solutions in the archive according to the Pareto dominance principle and update the archive if needed. At the end, the archive corresponds to the Pareto set and thus associated Pareto front.

The sections below present the conditions of this experiment and its results.

8.4.1 Conditions of experiment

For this experiment, we use the case study based on the ROSACE and the JPEG application used in the experiment 2 (section 8.2). This case study is composed of 22 tasks and we limited the number of vulnerable communications to 7 over 26 communications. We assume a maximum of two partitions.

This leads to a design space of 268,435,328 solutions computed based on the defined formula in the experiment 5 (section 8.3.3). We start the enumeration of this large search space and after 3 days of computation, found that it will take approximately 152 years to explore all the design space, based on the progression

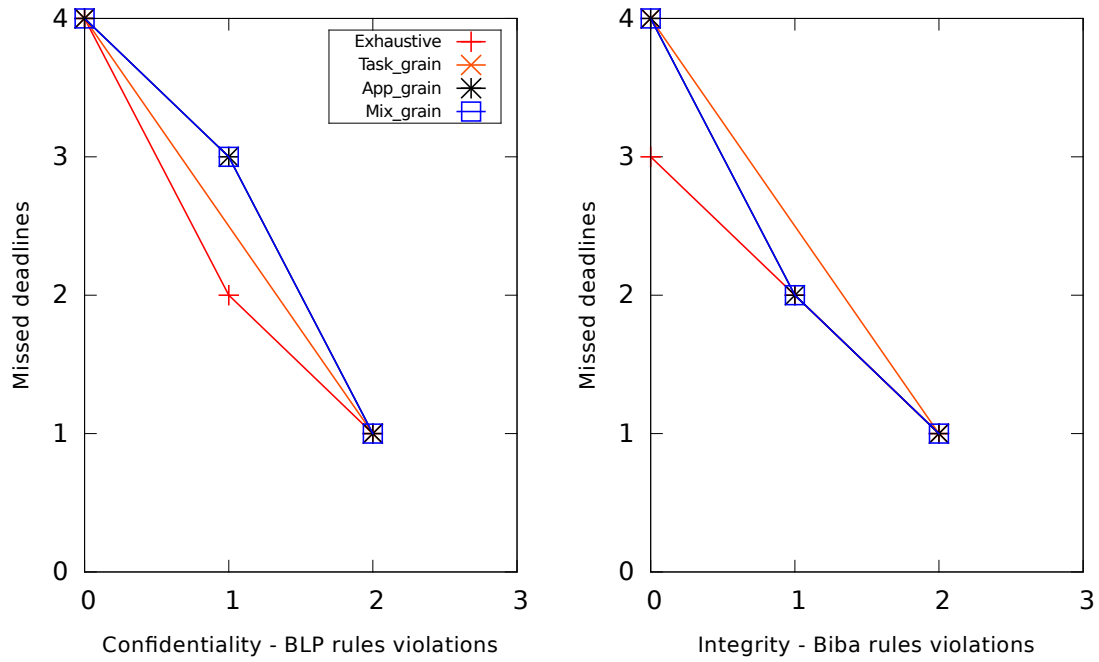


FIGURE 8.7: Exhaustive vs PAES

rate over these 3 computation days. Thus, we choose a smaller test-case in such a way that it can be explored in approximately one computation day. This manageable case study is made of 10 tasks and 3 weakly sensitive communications over 8 communications. It is composed of 2 instances of the autopilot application. Then the design space contains 4088 solutions.

We also assume that the autopilot data size is 16 Kilobytes. Therefore the execution time of the encryption, the refreshment of the encryption key and the hash tasks are respectively 340 us, 88.83 us, and 205.52 us. We assume that intra-partition and inter-partition communications are attackable; then only security implementation (F-F) is applied.

We conducted this experiment, first the exhaustive search and second the PAES approach considering the three mutation algorithms; each mutation for 2000 iterations. The first phase of *mix-grain* ends at 1000 iterations.

8.4.2 Results

Figure 8.7 shows the set of non-dominated solutions by couples of objectives for the *app-grain*, *task-grain*, *mix-grain* and the exhaustive approach which provides the optimal Pareto set.

The exhaustive search proposes the optimal solutions which dominate or are equal to the solutions proposed by the 3 mutation algorithms. These solutions

approximate the optimal solutions with a lower computation effort (an execution time of 37 minutes as compared to the 1 hour and 43 minutes required for the exact set computation).

As shown by the search space size computation and the preliminary aborted experiment described in Section 8.4, this experiment shows the relevance of the PAES approach when the exhaustive method becomes unmanageable. We observe that the PAES approach for the 3 mutation algorithms did not find all the optimal solutions but they find several. Then as expected from a metaheuristic, the proposed frontis close to the optimal front. For example, for the left part of the figure 8.7, over the three exact solutions proposed by the exhaustive method, the *app-grain*, *task-grain*, and *mix-grain* found two exact solutions.

As presented in chapter 5, the hypervolume is a metric that helps to evaluate the quality of a front (i.e. convergence) and then compare two fronts. When considering a MOOP with objective functions to minimize, between two fronts, the front with the larger hypervolume should be preferred. Then we computed the hypervolume of the fronts presented at the left part of figure 8.7 with linear optimization as presented in section 5.3.1.3 (i.e. hypervolume $\in [0, 1]$). The hypervolume of the exhaustive, *mix-grain*, *app-grain*, and *task-grain* fronts are respectively 0.33, 0.16, 0.16, and 0. The hypervolume of *mix-grain*, *app-grain* fronts are equal. Their hypervolume is higher than the hypervolume of the *task-grain* front. It shows that *mix-grain*, and *app-grain* algorithms have proposed better fronts than the *task-grain*. Even if we remark that the three algorithms have found the same two exact solutions over the three proposed by exhaustive algorithms. The difference between their hypervolume resides in the number of solutions. *mix-grain*, and *app-grain* proposed three solutions compared to *task-grain* which proposed only two solutions. This can be explained by the fact that the number of solutions on a front is a criterion that impacts its hypervolume.

This experiment confirms that the algorithm *task-grain* in the above experiments has problems to converge (to propose better solutions) because its design exploration space is too large even with this small test case of experiment 8.4.

By analyzing the tasks to partitions assignment of the solutions, we remark that there are two optimal solutions out of the scope of the *app-grain* algorithm. These solutions have tasks of the same applications assigned to different partitions. Thus, whatever the number of iterations fixed, the *app-grain* will never be able to reach these solutions; which confirms the relevance of the *mix-grain* algorithm.

8.5 Conclusion

In this chapter, we carry out 7 experiments to evaluate the approach and to identify key parameters that impact the trade-off between security and schedulability.

We show through the experiment 8.1 that securing applications with a low processor utilization and exchanging small messages will not affect schedulability. Therefore, our DSE approach returns that there is no need for DSE in such a situation. This is the case of control-command applications that have a low processor utilization and exchange small data, e.g. 108 bits, 32 bits, and 20 bits with CAN, ARINC 429 or ARINC 629 buses respectively [166, 167].

The experiment 8.2 illustrates an application that requires DSE. This application is characterized by a multimedia part with a large data size, which confirms that the data size has an impact on the conflict between security and schedulability.

The experiment 8.3.1 consists of applying the PAES algorithm to different generated versions on the same initial application with different processor utilization values. It confirms that processor utilization is a key parameter that should be considered to decide if design space exploration has or has not to be performed. With these experiments (experiment 8.2 and experiment 8.3.1), We have compared the mutation algorithms *task-grain*, *app-grain* and *mix-grain*. Results show that *mix-grain* can propose interesting solutions impossible to be generated with *app-grain* because they are out of its design space.

In experiment 8.4, we compare the optimal Pareto front provided by an exhaustive research and the Pareto fronts computed by the 3 mutation algorithms. First, we see that the design space of *task-grain* is too large to converge towards the best solutions. So it often gives solutions dominated by *app-grain* and *mix-grain* ones. Second, even if sometimes *app-grain* and *mix-grain* provide similar results, *mix-grain* can be seen as a solution to the problem of convergence of *task-grain* since both have the same search space and granularity level but *mix-grain* can provide interesting solutions not reachable by *app-grain*. This result has been confirmed by the exhaustive research who proposed solutions with tasks of the same application assigned to different partitions. Whatever the number of iterations, *app-grain* is never able to reach these solutions. Third, *task-grain* and *mix-grain* provided solutions with tasks of the same application split into different partitions. Such solutions were also found by the exhaustive method as part of the Pareto set (optimal solutions). This motivates the need for a mechanism to enforce memory protection between a subset of tasks within the same partition in TSP systems, which does not exist in ARINC 653 for example. Such protection mechanism is considered in [168] where each thread may be protected from others threads. Implementing such a mechanism in TSP systems should reduce the impact of the security on the schedulability.

With the experiment 8.3.2, we compared the 4 security implementations (F-F, X-F, X-T, X-TM). We have noticed that DSE finds solutions that optimize the assignment of tasks to partitions so that there is no inter-partition communication. These solutions do not require any security features. The DSE also proposes a solution with inter-partition communications secured by the task dedicated se-

curity implementation (X-T). This shows that varying security implementation may be relevant depending on the case study.

The experiment 8.3.3 also shows that the number of partitions has a high impact on the size of the search space. For example, for our case studies, there is no need to proceed an exploration with more than 2 partitions because all the solutions proposed by the exploration with a maximum number of 4 partitions are included in the design space of the exploration with a maximum of 2 partitions or dominated by the solutions proposed by this later. Moreover, due to the larger size of the search space with a maximum of 4 partitions, the exploration has difficulties to converge to solutions non-dominated by solutions proposed with the maximum number of 2 partitions.

Finally, we also show in experiment 8.3.4 that values of overheads introduced by intra and inter-partition communication mechanisms such as blackboards and sampling ports have a significant impact on system schedulability. Thus, the number of inter-partition communications and communication overheads are also important parameters that should be considered when addressing both schedulability and security.

In the next chapter, we show that our approach can be extended to deal with other constraints and objective functions. We presents the extension of our approach to multi-core systems and safety through active redundancy on secured real-time TSP systems.

9

Design space exploration for safe and secure Multi-core TSP systems

In chapter 7, we propose a DSE approach based on PAES that provides trade-offs between schedulability and security for TSP systems. This approach only considers unicore platforms and does not consider safety constraints. In this chapter, we show how such DSE can be extended to other similar MOOPs.

We extend our approach to take into account multicore execution platforms and safety constraints. Then, we propose to investigate the impact of multicore platforms on TSP systems while addressing the conflicts between security, and schedulability, and safety constraints. Safety is enforced by both the isolation through partitioning to prevent fault propagation and by active redundancy, i.e. replications of tasks and partitions. We explore the tasks to partitions assignment in TSP systems when communications are secured and tasks are replicated for safety. To validate the approach, we conduct two experiments. We modify the search space by varying the number of cores, and analyze the impact varying the number of cores on the adressed MOOP. A first experiment shows the effectiveness of our approach in providing trade-offs in the context of multicore execution plateforms and safety constraints. It also provides consistent results showing schedulability improvements when the number of cores is increased, which assesses the relevance of our DSE. Further, a second experiment is performed to take into account multicore hardware shared resources overheads, and investigate their impact on schedulability of the systems.

Section 9.1 presents the background of TSP systems on multicore execution platforms, and safety. It also depicts the system model, and the assumptions considered in this chapter. Section 9.2 describes our DSE approach for multicore safe and secure TSP systems. Section 9.3 shows the experiments conducted to

evaluate the approach. Finally, Section 9.4 discusses related work and Section 9.5 concludes the chapter.

9.1 Background and system model

This section presents first the background of multicore TSP systems and safety. Second, we discuss the extensions of our system model and assumptions to take into account the multicore aspect.

9.1.1 Multicore TSP systems

In the multicore section of the ARINC653 avionic standard [169], each task can be assigned to a partition and to one or multiple cores. The tasks to cores assignment is addressed by the core affinity concept that indicates the cores on which each task is allowed to run. Tasks within a partition can be executed concurrently on different cores. A task that has a core affinity with only one core can only be executed on the corresponding core. A task with a core affinity to multiple cores is allowed to migrate from or to one of these cores.

In the sequel, we assume that a task is not allowed to migrate from one core to another and the core affinity defines that each task τ_i is assigned to only one core CO_i.

In ARINC653 multicore TSP systems, offline cyclic scheduling is fixed for the partitions. Partitions are executed cyclically on the major time frame (MAF). Tasks inside partitions are executed concurrently based on a given scheduling policy (i.e. fixed-priority scheduling).

Fig 9.1 shows an example of scheduling of a multicore system with four tasks, assigned to two partitions and two cores. We note that $\{\tau_1, \tau_3, \text{ and } \tau_4\}$ and $\{\tau_2\}$ are respectively assigned to core CO_1 and CO_2. For tasks to partitions assignment, $\{\tau_1, \tau_2, \tau_3\}$, $\{\tau_4\}$ are respectively assigned to partitions P_1, and P_2. The same MAF is assumed for all cores. Then when a partition is activated, only its tasks are executed concurrently on the cores depending on the tasks to cores assignment. Cores that have no task in the activated partition are in idle mode. They are not used till the activation of a partition with tasks assigned to them. In this example, we assumed that there is a communication from τ_1 to τ_2 and another from τ_1 to τ_3. Then τ_2 has to wait for τ_1 completion time before being starting its execution. This explains why even if τ_1 and τ_2 are on different cores, and τ_2 is the only task on CO_2, τ_2 could not start at time 0.

Multicore systems are more and more adopted in real-time systems since they imply high computational capabilities [170]. These systems have cache memory

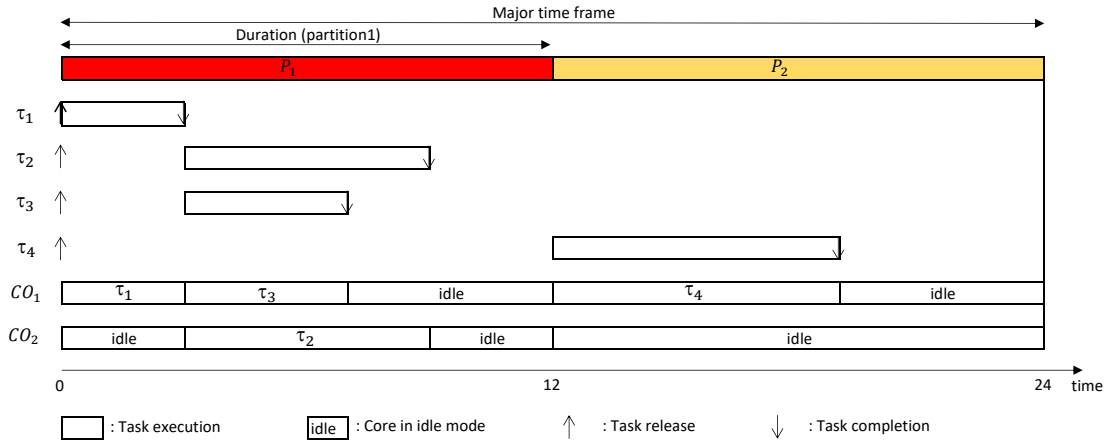


FIGURE 9.1: Example of a multicore TSP system scheduling

that help to reduce the main memory access latencies. In general, they are composed of sets of cores grouped on chips. Each core may have its private Level-1 cache, while cores on the same chip shared the same Level-2 cache [171]. Then tasks on different cores may share the same Level-2 cache resources. When tasks are preempted, the cache is evicted and it has to be reloaded [172]. This leads to overhead on the tasks execution time. Furthermore, tasks on different cores have access to the main memory through the front side bus (FSB) [172]. This implies a contention on the shared bus and then leads to another overhead on the tasks execution time. This overhead is called interconnection overhead in the sequel.

Therefore multiple cores executing simultaneously several tasks, lead to overheads introduced by the hardware shared resources (Level-2 cache, bus, memory) [173]. In [174], it has been proven that they are non-negligible. These overheads may lead some tasks to miss their deadlines and then impact the schedulability of the considered system. They have to be considered when performing the schedulability analysis of a multicore real-time system.

9.1.2 Safety

In this chapter, we address safety constraints during the DSE. Safety is related to failures, the availability and reliability of systems. Two major types of failures exists: halting failures (such as crash failure [175], fail-silent [176] or fail-stop [177, 178]) and arbitrary failures [179].

Halting failures are concerned when a unit remains silent and stops sending or receiving messages, or stops functioning. It assumes that incorrect data cannot be sent by the unit. It considers that while messages are sent, they are correct and the only failure that may occur is their loss. In this case, an extra instance

of the unit can help to guarantee safety. Thus, if one instance of a unit stops sending messages, we rely on a second instance to send them. To resist k failures, it takes $k+1$ unit instances to guarantee safety [178]. If the k units stop sending messages, then we rely on the $(k+1)^{th}$ unit to send them.

Halting failures assume no malicious failure which is the case of arbitrary failures. Arbitrary failures include the detection that some messages are not sent or received, the detection of incorrect messages sent with errors, and the detection of extra sent messages. For this case, at least $2 \cdot k + 1$ units are necessary to ensure safety over k failures [180]. Then if an instance fails, other instances still work to recover the messages. An error is detected by comparing the outputs of all the instances, and voting is made for the majority.

Differents strategies to address different types of crash and arbitrary byzantine failures are proposed in [181]. The authors propose to investigate different distribution of replicas instead of the intuitive balanced replica placement.

9.1.3 System model and assumptions

In this chapter, we consider a multicore TSP system of m applications $(A_1, ..., A_m)$ where each application is a set of tasks. Systems considered are composed of a set n periodic tasks $(\tau_1, ..., \tau_n)$. We assume a multi-core architecture of d identical cores $(CO_1, ..., CO_d)$.

Each task τ_i is defined by a set of parameters $(C_i, T_i, D_i, CI_i, CL_i, A_i, P_i, CO_i)$. As defined in Section 6.1, C_i, T_i, D_i, CI_i, CL_i, A_i, P_i corresponds respectively to the WCET, the period, the deadline, tolerance level, confidentiality level, the application and the partition of the task τ_i. We assume that all the partitions have the same properties and are executed based on a *major time frame* (MAF). A task is assigned to one core CO_i and core migration is not allowed at runtime (affinity of 1). We assume that tasks communicate with each other through intra-partitions or inter-partition communications depending on their assigned partitions.

Considering security, we only address confidentiality issues in this chapter. We only consider the security implementation F-F (see Section 6.2). It assumes that when a communication (intra or inter-partition) from task τ_i to τ_j is vulnerable, functions of a library implementing encryption and decryption are called.

In this chapter, we consider safety problems induced by arbitrary failures. We assume the worst-case situation where the replication is applied to all the software components (tasks and partitions). Then with such safety constraints, each task and partition is triplicated (i.e. implemented by three instances). We impose that two instances of the same task are not allowed to be placed on the same partition.

This work is not an answer to multicore platforms with hardware single point failure; e.g. when cores are interconnected by a bus, the bus is a single point of failure, while it is not the case if cores are interconnected with a crossbar. In this work, we consider safety as a constraint instead of an objective functions to optimize.

Finally, we also consider overheads introduced by the hardware shared resources (level-2 cache, bus, memory, etc) when multiple cores execute simultaneously several tasks [173]. This issue is part of the key point addressed by the CAST-32A for Avionics Multi-Core Processing [182].

9.2 PAES adaptation for safe and secure multicore TSP systems

In this section, we present a DSE approach that computes trade-offs between security and schedulability while considering safety constraints, resources constraints such as the number of cores and partitions, and hardware shared resources overheads. This work is an extension of our PAES-based DSE approach presented in Chapter 7. The general process of our DSE approach presented in figure 7.1 remains the same, but we need to review (1) initial solutions, (2) constraints to perform the feasibility tests, (3) objective functions to optimize, (4) encoding of solutions, (5) mutation operators to generate new solutions based on the new considered context (redundancy, multicore execution platform).

9.2.1 Initial solution

We design the initial solution by resolving all security vulnerabilities in the system, placing all the system tasks in the same partition running on a single core. Then we triplicate the tasks, the communications between tasks, and the partitions to ensure safety. For a communication from task τ_i to τ_j, when both tasks are triplicated (i.e. each task has three instances), each instance of τ_i has a communication to the three instances of task τ_j.

We proceed with a schedulability analysis of this solution. If it is schedulable, there is no need to continue with the exploration: we consider this solution as an optimal solution since it is fully secured, schedulable and safe with the minimal number of cores. Otherwise, we add this initial solution to the archive.

Instead of starting the exploration with an archive containing one solution as specified in the original PAES algorithm, we fill the archive with nine non-dominated solutions. We made this choice to improve solution diversity and exploration of the design space. We fill the archive with solutions modeling various tasks to

cores assignment and communications security. As in chapter 7, we added solutions by combining various strategies. In addition to the solutions proposed in section 7.2.6 (e.g. solutions with single partition, balanced partitions, partitioning based on confidentiality or integrity) which are single core solutions (i.e. all the tasks assigned to a single core), we consider solutions with each task assigned to a different core and, tasks of each application assigned to a different core. For all these solutions, we decided to resolve all or no security vulnerabilities and apply redundancy.

9.2.2 Objective functions and constraints

We defined the constraints and the objective functions based on schedulability, security, safety issues, and number of cores. To the constraints and the objective functions proposed in chapter 7, we added a constraint according to safety (see constraint $C4$ detailed below), and an objective function concerning the number of cores (see objective function $F2$ detailed below).

In our model, tasks can be either hard deadline tasks or soft deadline tasks. As a constraint, a solution is considered invalid and is rejected if a hard deadline task misses its deadline. Missed deadlines are tolerated for soft deadline tasks.

- C1: No missed_deadlines for hard deadline tasks

Our first objective function is defined by the number of soft deadline tasks that missed their deadlines. This number is computed through a scheduling simulation of the solution. This function is noted below:

$$F1 = \#missed\_deadlines$$

Since we decide to investigate tasks to core assignment to evaluate their impact on the considered systems, our second objective function represents the number of cores used in a given solution:

$$F2 = \#cores$$

The problem depicted in this chapter addresses only the confidentiality of communications between tasks on contrary to the previous work that considers both confidentiality and integrity. We defined the constraints below for security vulnerabilities based on BLP rules for strongly sensitive communications:

- C2: No data received by Unclassified task from Secret or Top-secret task

Each model that compromises one of these constraints is rejected. Otherwise, any communication violating the other BLP rules is tolerated. This allows the definition of the security objective function:

$$F3 = \#Bell\_violations$$

The equation F3 represents the number of weakly sensitive communications that violate BLP rules.

Since we address safety issues by applying active redundancy, each task of our model is triplicated. By definition, this redundancy imposes that two instances of the same task should never be placed on the same partition. Then we assumed as safety constraints that every solution with two instances of a task placed on the same partition should be automatically rejected.

- C4: Two instances of the same task cannot be placed in the same partition

In order to find trade-offs for our MOOP, all the defined objective functions have to be minimized. Constraints and objective functions are computed with the Cheddar tool in which our DSE heuristic has been implemented [25].

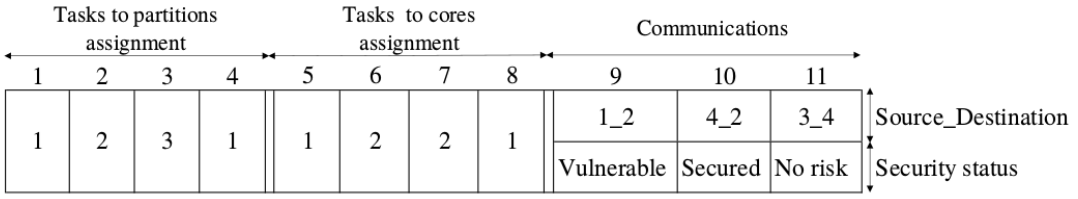
9.2.3 Encoding of solutions

As in chapter 7, we represent our solutions in a chromosomal form. Each solution of n tasks and m communications is represented by a vector of $(2 \cdot n + m)$ positions. The vector representing the chromosome is composed of 3 parts.

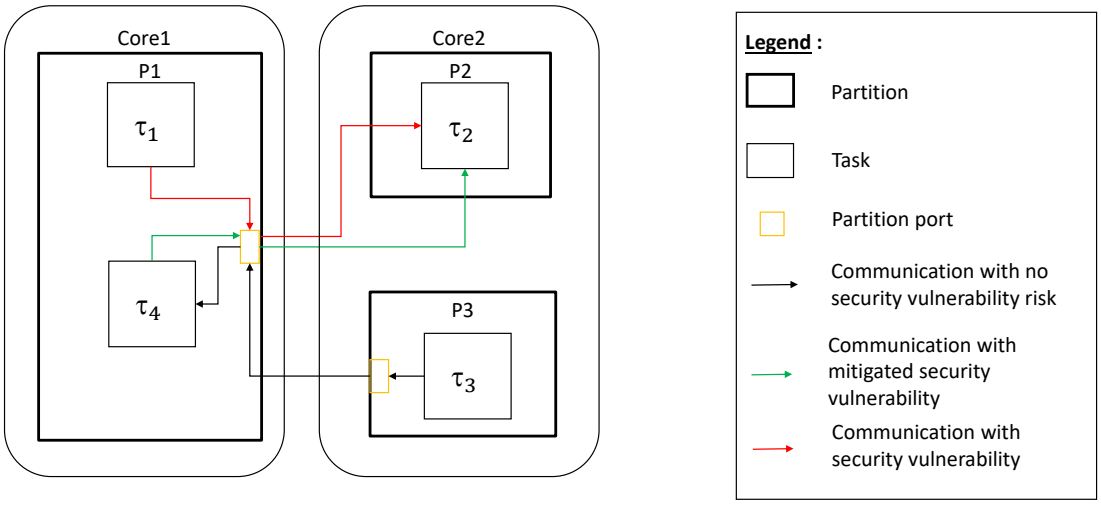
The first part describes the tasks to partitions assignment: for a TSP system of n tasks and p partitions, $chrom[i] = j$ $(1 \leq i \leq n$ and $1 \leq j \leq p)$ indicates that the i^{th} task is assigned to the j^{th} partition.

The second part, similar to the first one, is tasks to cores assignment: considering that the system contains d cores, $chrom[i] = k$ $(n + 1 \leq i \leq 2 \cdot n$ and $1 \leq k \leq d)$ indicates that the i^{th} task is assigned to the k^{th} core.

The last part of the chromosome is dedicated to describing the communications between the tasks of the system. We divided the communications into two categories based on the predefined constraints and objective functions. Then for a TSP system, we distinguish strongly sensitive communications and weakly sensitive communications. Strongly sensitive communications are communications that should be imperatively secured otherwise the solution is invalid. Weakly sensitive communications are allowed to violate some security rules based on the objective functions above mentioned. Thus, we consider that a TSP system of m communications $(m = m_c + m_u)$, is composed of m_c strongly sensitive communications and m_u weakly sentive communications. For each l^{th} communication



(a) Example of a normalized multicore chromosome



(b) Model of the illustrated chromosome

FIGURE 9.2: Illustration of multicore solutions encoding

$(2 \cdot n + 1 \leq l \leq 2 \cdot n + m)$ in the system, the $chrom[l]$ is defined by a set of two values. The first value identifies the communicating tasks and the second value indicates the status of the communications (vulnerable, secured or no risk). It is important to highlight that all the m_c strongly sensitive communications must be secured while vulnerable weakly sentive communications may be tolerated.

Fig. 9.2a shows an illustration of a solution in a chromosomal form. This solution is composed of four tasks assigned to three partitions and two cores.

The first part that represents the tasks to partitions assignment reveals that tasks τ_1 and τ_4 are assigned to the partition P_1 and task τ_2 and τ_3 are respectively assigned to partitions P_2 and P_3. The second part shows the tasks to cores assignment. Tasks τ_1 and τ_4 are assigned to $core_1$ and tasks τ_2 and τ_3 are assigned to $core_2$. The remaining part is reserved to communications. It shows that this solution is composed of 1 vulnerable communication (e.g. communications from τ_1 to τ_2), 1 secured communication (communication from τ_4 to τ_2), and 1 communication with no risk of security vulnerability (communication from τ_3 to τ_4). For normalization purposes as in chapter 7, we assume partitions and cores of solutions are identical. Then we decided that τ_1 should always be assigned to partition P_1 (resp. core $core_1$), and τ_2 should be assigned to P_1 (resp. $core_1$) if

and only if τ_1 and τ_2 are assigned to the same partition. Otherwise, τ_2 should be automatically assigned to P_2 (resp. $core_1$). This normalization is adopted to guarantee the unicity of representation of solutions and reduce the size of the design space.

Figure 9.2b shows an architectural view of the model encoded in the figure 9.2a. It easies the understanding of the presented chromosome by showing the tasks to partitions and cores assignments, and the communications between tasks.

9.2.4 Mutation operator

Since PAES works with a neighborhood-based search, the design space is explored by mutating a solution to another nearby. We are interested in tasks to partitions assignment, tasks to cores assignment, and the security of communications between tasks.

The first operator is based on to the *mix-grain* algorithm proposed in chapter 7. It changes the tasks to partitions assignment of a solution. It is defined with two different steps. The first step consists of moving all tasks of a randomly chosen application to a randomly chosen partition. The second step consists of moving a randomly chosen task to a randomly chosen partition.

The second operator is similar to the first one but changes tasks to cores assignment. Thus, the first step consists of moving all the tasks of a randomly chosen application to be executed on a randomly chosen core. The second step is operated by moving a randomly chosen task to be executed on a randomly chosen core.

Notice that the change of tasks to partitions or tasks to cores assignment has an impact on the schedulability of the solution.

The third operator concerns weakly sensitive communications of the solution on which security vulnerabilities are tolerated. It is realized by a random choice of a communication. If the communication presents security vulnerabilities, then we secure it by adding security functions. Otherwise, we remove the security functions and the communication becomes vulnerable. We highlight that contrary to chapter 7, communications vulnerabilities are mitigated based on only one possibility of security implementation : function calls (F-F in table 6.1, chapter 6).

After each mutation operation, we conduct feasibility tests to check the respect of schedulability and security constraints. If the new solution generated by the mutation does not respect one of the constraints, it is rejected and another mutation operation is performed. Otherwise, if the solution respects all the constraints, then schedulability and security analysis are performed to evaluate the objective functions of the solution.

TABLE 9.1: Case study task parameters

Tasks	C_i **[us]**	T_i **[us]**	CL_i
ROSACE			
Aircraft Dynamics	200	5000	Secret
Va_c	500	20000	Secret
H_c	500	20000	Top_secret
H_Filter, Az_Filter, Vz_Filter, Q_Filter, Va_Filter	100	10000	Secret
Altitude_hold	100	20000	Top_secret
Vz_control	100	20000	Top_secret
Va_control	100	20000	Secret
Delta_ec, Delta_thc	500	20000	Secret
Engine, Elevator	100	5000	Top_secret
CFAR			
CFAR_complex	90	10000	Top_secret
CFAR_square_scale	50	10000	Secret
CFAR_gather	340	10000	Top_secret
CFAR_printer	30	10000	Top_secret

In the next section, we propose to validate and illustrate this new DSE approach through experiments.

9.3 Test cases and Evaluation

The purpose of these experiments is to evaluate our DSE approach with a case study. We conducted these experiments to evaluate the impact of the number of cores, the tasks to cores assignment, and the shared hardware resource overheads on TSP systems schedulability.

We highlight that our choices of tasks model, considered faults, and encryption algorithms are classic and from known benchmarks, but can be changed since they are considered as parameters.

9.3.1 Case study

We use a case study composed of a set of two applications already used in previous experiments in chapter 8: the flight controller application ROSACE (Research Open-Source Avionics and Control Engineering) [27] and the digital signal processing application CFAR (Constant False Alarm Rate detection) [4]. ROSACE is composed of fifteen dependent and periodic tasks. We take the worst-case execution time of tasks and their period from [27]. CFAR is composed of four dependent tasks with the WCETs taken from the StreamIT benchmark profiled in [4]. We also assume for ROSACE and CFAR, an average data size of 8 bytes.

We assumed that cores are identical and have the same predefined MAF. For simplicity, we choose a MAF where each partition has only one partition window and all the partitions have the same partition window duration. The partitions are identical with a duration of 1250 us.

We randomly choose a confidentiality level for each task of the applications since they are considered as inputs from the system designer. The parameters of the applications are resumed in Table 9.1. For simplicity, we assumed that all the tasks of our model are soft deadline tasks.

For securing confidentiality vulnerabilities, we used the blowfish encryption algorithm [31]. With a frequency of 1.2 GHz, we computed the time execution of security functions based on values provided by the crypto++ benchmark [163] and the data size of our applications. Then for both applications, the execution times of encryption, and encryption key refreshment are respectively 0.166 us, and 88.83 us.We consider the decryption execution time equal to the encryption execution time.

We assumed that intra-partition (resp. inter-partition) communications are performed through blackboards (resp. sampling ports). For their cost, we consider the execution times of the APEX calls SFPBench Benchmark proposed in [2]. Considering the data size of our case study, for blackboards (resp. sampling ports), it gives a cost of 0.76 us/0.32 us (resp. 4.24 us/5.04us) for read/write.

About the shared hardware resource overheads, we only consider the interconnection overhead. We conduct the DSE first by considering the best case with negligible interconnection overhead. Second, we conduct another DSE by assuming the overhead percentage provided in [174]. It depends on the number of cores of the considered system. Then for a system with only one core, there is no interconnection overhead. For a system between 2 and 4 cores (resp. between 5 and 8), the interconnection overhead on each task corresponds to 10% (resp. 13%) of its capacity. For systems with more than 8 cores, we assume a 26% overhead.

Each DSE was performed for 20000 iterations which takes 12 hours.

9.3.2 Results of the experiment

Considering the two applications, the initial system model is made of 19 tasks. With our safety assumptions, we triplicated partitions, tasks, and communications. This implies 57 tasks with at least 3 partitions for the DSE. By considering one of the additional initial solutions defined in 9.2.1 that runs each application per partition, we assume a DSE with a maximal number of 6 partitions. Then, we explore multiple solutions with 3, 4, 5, and 6 partitions since the safety imposes a minimum of 3 partitions.

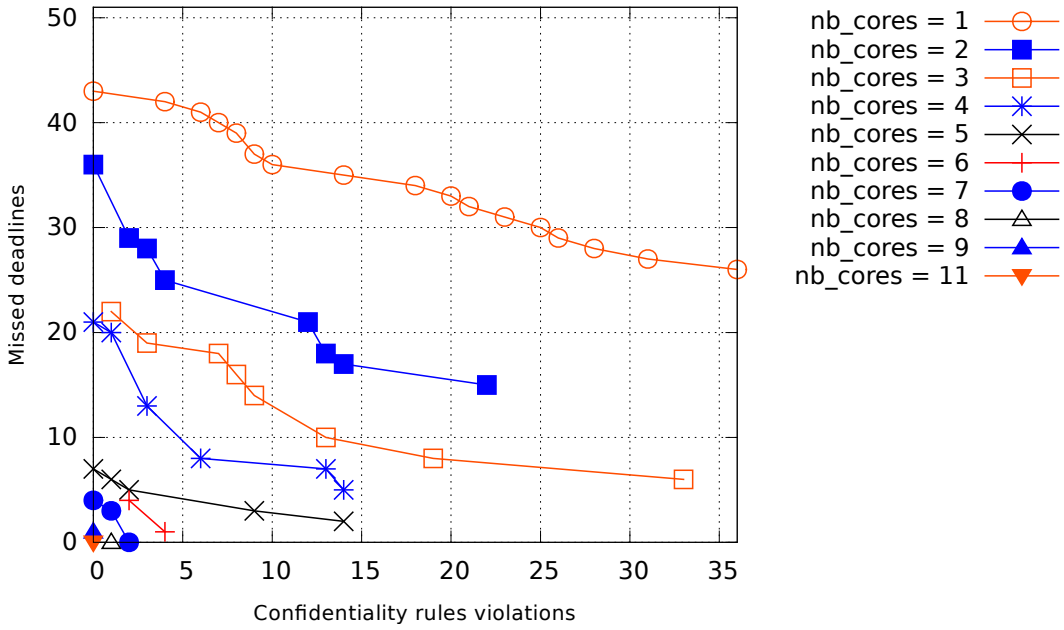
The solution with minimum cores corresponds to the solution with all tasks assigned to a single core. It has a high number of missed deadlines (45 over 57 tasks). By increasing the number of cores to 57 cores (i.e. number of tasks), more tasks are able to meet their deadlines (e.g. from 45 to 0 missed deadlines when interconnection communication is considered negligible). This confirms the impact of multicore platforms. This is explained by the obvious fact that using more cores increases the computation capacity of the system.

Since these solutions are extreme, we propose to investigate the design search space, in order to find interesting trade-offs. The DSE proposes a set of 52 (resp. 40) different trade-offs with no interconnection overhead (resp. with interconnection overhead). Fig. 9.3 shows the set of non-dominated solutions.

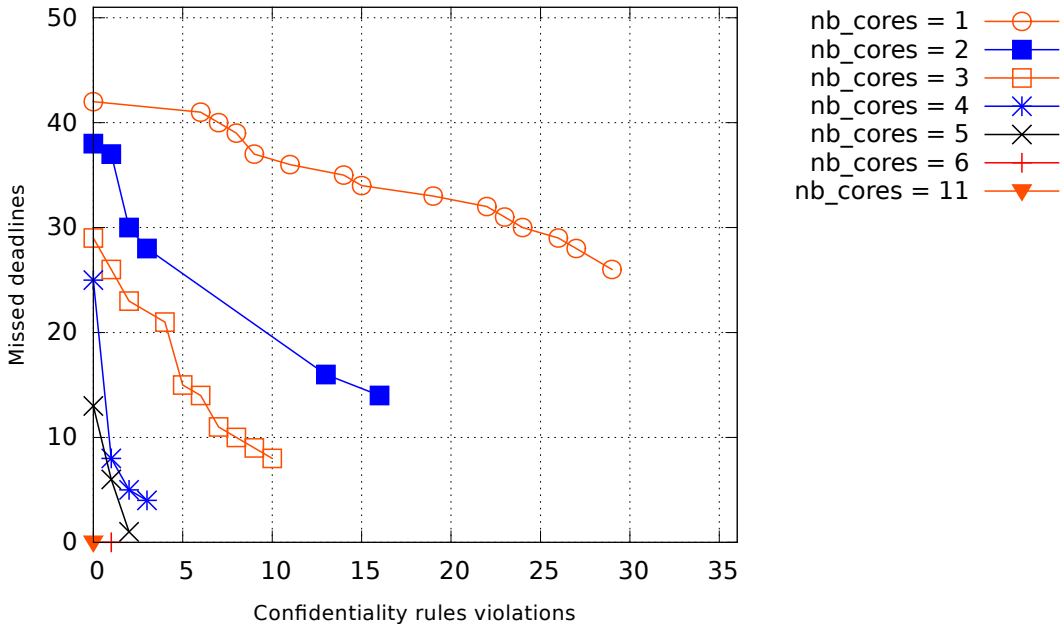
For the system model with negligible interconnection overhead, our DSE tool was able to decrease from 45 to 11 the number of cores required for a fully secured, safe, and no missed deadlines solution (figure 9.3a). Our DSE is then able to detect a minimal number of cores that corresponds to a fully secured, safe, and no missed deadlines solution. This solution considers a better grouping of tasks on the cores in order to propose a reduced number of cores while not allowing any task to miss its deadline. The tasks to cores assignment of this solution is so irregular that it could be difficult and time-consuming to get manually such an assignment considering 57 tasks to assign to 11 cores. This shows the interest of a DSE.

The DSE with interconnection overhead also proposes a solution that reduces to 11 the number of cores for a fully secured, safe, and no missed deadlines solution (figure 9.3b). Contrary to the above-mentioned solution, it has a different tasks to cores assignment and used 4 partitions instead of 3 partitions. This can be explained by the fact that the above-mentioned solution updated with interconnection overhead was not able to keep no missed deadline. Then the DSE was able to explore different tasks to partitions and tasks to cores assignments to avoid missed deadlines without using more cores. Those solutions are not intuitive and motivate again the use of a DSE approach.

As expected, we observe in the figures that the more the number of cores increases, the easier it becomes to ensure the security of our TSP system while considering



(a) No overhead



(b) Variable overhead

FIGURE 9.3: Schedulability vs. confidentiality

safety constraints and minimizing the number of missed deadlines. This confirms the relevance of the proposed DSE.

The increase of the computing capacity related to the increase of the number of cores can be compromised by a high shared hardware resource overhead. As shown on the graphs, trade-offs with no security vulnerabilities proposed by the DSE with interconnection overhead have a number of missed deadlines greater than or equal to the equivalent in the DSE with negligible interconnection overhead. Let us consider the fully secure solutions with 5 cores. With no interconnection overhead, there are 7 missed deadlines (figure 9.3a) while there are 13 missed deadlines when considering interconnection overhead (figure 9.3b). This can also explain that with interconnection overhead, solutions with 6, 7, 8, and 9 cores are dominated by the other solutions and then rejected by the PAES algorithm. This illustrates that overhead related to shared hardware resources is a key parameter in the design of safe and secure multicore TSP systems.

9.4 Related work

In this section, we position the work of this chapter by presenting different approaches that addressed the design of multicore platforms for TSP systems with schedulability, safety, and/or security constraints/objectives.

Many researchers have investigated TSP systems on multicore platforms. In [183], the authors depicted how multicore platforms can intervene in ensuring high-performance requirements. For this purpose, they identified some conditions such as privileging the intra-partition parallelism, which assumes the possibility of running parallel tasks of the same partition on different cores. In [184], the authors proposed the evolution of a TSP unicore system to a TSP multicore system while considering inter and intra-partition parallelism mechanisms. They propose to activate simultaneously many partitions on different cores. The work in [185] explored a similar idea. The authors specifically focused on symmetric multiprocessing (SMP) architectures where each core has access to a common shared memory and I/O resources with a single operating system for all the cores. They defined patterns for SMP/TSP multicore systems with which they extended the Ocarina code generation tool.

Since safety and security are important requirements for TSP systems, several researchers showed interest in these domains. In [186], the authors proposed a survey for validation and certification of TSP multicore systems deployed on the Xtratum hypervisor [1]. For example, it highlights fault tolerance for safety and data protection for security. The authors of [150] addressed multicore platforms not specifically for TSP systems, but for real-time systems in general. They also addressed the systems' security vulnerabilities. Then the authors added

security mechanisms such as a hash algorithm to their systems and then proposed a DSE to optimize their schedulability while exploring the security tasks to cores assignment possibilities.

The potential schedulability benefits of deploying TSP systems on multicore platforms have led to multiple researches on the design and analysis of such systems. Some have addressed their security and safety vulnerabilities. Few have studied the assignment of tasks to cores through a DSE for real-time systems in general. We propose a DSE approach for multicore TSP systems that investigates not only tasks to cores assignment but also tasks to partitions assignment and securing communications in order to find trade-offs. We also integrate safety constraints into our proposal. As far as we know, no work has proposed such a set of combinations.

9.5 Conclusion

The purpose of this chapter is to show that the DSE can be adapted to similar problems in different contexts. In this chapter, we investigate the impact of multicore platforms on safe and secure TSP systems by proposing an adaptation of our DSE. Our adapted DSE approach covers the different possibilities of tasks to partitions assignment, tasks to cores assignment, and securing communications, which is a combinatorial problem. Then, we propose trade-offs between schedulability and security for a safe TSP system while considering different numbers of cores, and redundancy.

As expected, our approach confirms that for a safe and secure TSP system with some missed deadlines, increasing the number of cores effectively helps to optimize the system schedulability. Better solutions can also be obtained by moving some tasks from one partition to another or from a core to another. The DSE can find the required minimal number of core for a safe and secure TSP system. This first result confirms the interest of our DSE.

To illustrate the interest of our approach, we test the DSE by considering shared hardware resources overhead existing in multicore platform. This overhead results from tasks on different cores accessing simultaneously the same hardware resources. It may increase considerably the required number of cores to keep a certain level of schedulability. Our experiments show that the shared hardware resources overhead, the number of cores, the number of partitions are key parameters in the design of multicore safe and secure TSP systems. This work confirms that the proposed DSE is an extensible approach that can be adapted to different contexts.

10

Tool design and implementation

In this chapter, we present the Cheddar framework and the prototype implemented for our work. This implementation is developed in Ada and based on the Cheddar scheduling analysis tool. Our prototype includes security analysis, DSE with PAES and exhaustive search methods. The DSE starts with a Cheddar-ADL model that specifies the architecture of the considered system. We extended Cheddar-ADL with MILS, PAES, and architecture exploration tools libraries.

Section 10.1 describes the Cheddar framework into which is integrated the prototype implemented in the scope of this thesis. Section 10.2 presents the feasibility tests and scheduling simulation provided by the Cheddar analyzer tool to perform scheduling analysis on an RTS. Section 10.3 presents the packages for DSE that we implemented to address the MOOP between security and schedulability in this thesis. Finally, a conclusion of the chapter is given in section 10.4.

10.1 Cheddar framework

Cheddar framework is an open-source scheduling framework designed by the Laboratory Lab-STICC at the University of Western Brittany, and Ellidis Technologies. It uses the Cheddar Architecture Description Language (ADL) which is a language dedicated to the design and the validation of real-time systems. It also includes a scheduling analyzer.

10.1.1 Cheddar Architecture Description Language (ADL)

Cheddar-ADL proposes to specify hardware, software entities, and connections between entities needed to model an RTS and perform its scheduling analysis. It

also supports the modeling of TSP systems.

A RTS can be modeled with Cheddar-ADL through an XML file where each hardware and software component can be instantiated with its specific attributes. Listing 10.1 shows an overview of a TSP system with Cheddar-ADL. It is composed of multiple sections that correspond to the specification of components such as cores, processors, partitions, tasks, and dependencies.

Listing 10.1: Cheddar ADL model in XML format

```
<?xml version="1.0" encoding="utf-8"?>
<cheddar>

  <!--Cores specification section (Listing 10.5) -->
  <core_units>
    ...
  </core_units>

  <!--Processors specification section (Listing 10.7) -->
  <processors>
    ...
  </processors>

  <!--Address spaces specification section (Listing 10.2) -->
  <address_spaces>
    ...
  </address_spaces>

  <!--Tasks specification section (Listing 10.3) -->
  <tasks>
    ...
  </tasks>

  <!--Dependencies specification section (Listing 10.4) -->
  <dependencies>
    ...
  </dependencies>

</cheddar>
```

Listing 10.1 shows an example of a TSP system composed of two partitions deployed on a unicore processor platform. For sake of simplicity, the specification of each of its components has been presented in Listings 10.2, 10.3, 10.4, 10.5, and 10.7.

10.1.1.1 Software entities

The different software components proposed by Cheddar-ADL are described below and depicted in figure 10.1.

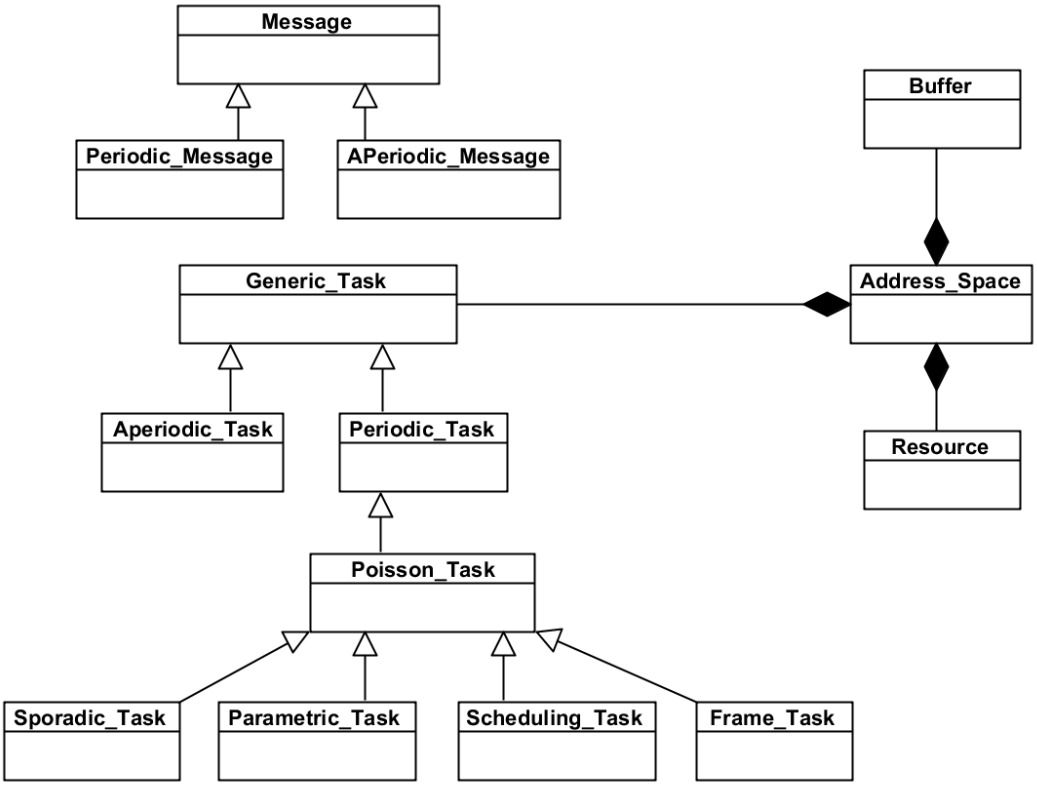


FIGURE 10.1: Software entities in Cheddar ADL [11]

Address spaces are components that group entities such as tasks and resources. The entities inside the same address space have access to the same memory space, and there are mechanisms to protect the memory space between address spaces if needed. For TSP systems an address space can be used to model a partition.

Listing 10.2: Partitions modeling in Cheddar-ADL

```
<address_spaces>
  <address_space id="id_4">
    <object_type>ADDRESS_SPACE_OBJECT_TYPE</object_type>
    <name>addr1</name>
    <cpu_name>processor1</cpu_name>
    <scheduling>
      <scheduling_parameters>
        <scheduler_type>
          POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL
        </scheduler_type>
        ...
      </scheduling_parameters>
    </scheduling>
    ...
  </address_space>
  <address_space id="id_5">
    <object_type>ADDRESS_SPACE_OBJECT_TYPE</object_type>
    <name>addr2</name>
    <cpu_name>processor1</cpu_name>
    ...
  </address_space>
  ...
</address_spaces>
```

Listing 10.2 presents the partitions section of a TSP system modeled in Cheddar ADL. It considers the specification of each partition that composes the model.

Tasks are entities that correspond to control of flow and that are running a given program. Each task is associated to an address space. For TSP systems, tasks are partitioned by being placed on different partitions.

Listing 10.3: Tasks modeling in Cheddar-ADL

```
<tasks>
  <periodic_task id="id_8">
    <object_type>TASK_OBJECT_TYPE</object_type>
    <name>Task1</name>
    <task_type>PERIODIC_TYPE</task_type>
    <cpu_name>processor1</cpu_name>
    <address_space_name>addr1</address_space_name>
    <capacity>20</capacity>
    <deadline>500</deadline>
    <start_time>0</start_time>
    <priority>5</priority>
    <period>500</period>
    <policy>SCHED_FIFO</policy>
    ...
  </periodic_task>
```

```
  <periodic_task id="id_9">
    <object_type>TASK_OBJECT_TYPE</object_type>
    <name>Task2</name>
    <task_type>PERIODIC_TYPE</task_type>
    <cpu_name>processor1</cpu_name>
    <address_space_name>addr2</address_space_name>
    ...
  </periodic_task>
  ...
</tasks>
```

Listing 10.3 presents the tasks section of a system modeled in Cheddar ADL. It considers the specification of each task composing a TSP model with the main task attributes such as capacity (i.e. WCET), period, deadline.

Dependencies specify the relationships between tasks. They may implicitly define the order of execution of the tasks. They may also specify the relationships between tasks and other entities such as resources.

Listing 10.4: Dependencies modeling in Cheddar-ADL

```
<dependencies>
  <dependency>
    <type_of_dependency>
      PRECEDENCE_DEPENDENCY
    </type_of_dependency>
    <precedence_sink ref="id_9"></precedence_sink>
    <precedence_source ref="id_8"></precedence_source>
  </dependency>
  ...
</dependencies>
```

Listing 10.4 presents the dependencies section of a TSP system modeled in Cheddar-ADL. It considers the specification of each dependency that composes the model. Listing 10.4 shows a dependency from task source ($Task1$ with identification number "id_8" in Listing 10.3) to task sink ($Task2$ with identification numbers "id_9" in Listing 10.3). The execution of $Task2$ can only start after the completion of $Task1$.

There are many other software entities such as resources. A resource can model any data structure assigned to an address space. Tasks can share the same resource and have access to it through different synchronization protocols. It is also possible to specify asynchronous communications between tasks inside the same address space.

Buffers may model queued asynchronous data exchanges between tasks assigned to the same partition (i.e. intra-partition communications).

Messages may be used to model queued asynchronous data exchanges between tasks assigned to different partitions (i.e. inter-partition communications).

10.1.1.2 Hardware entities

The hardware description consists of specifying the platform on which the system will be deployed. It consists of components such as core and processor units which are represented in figure 10.2.

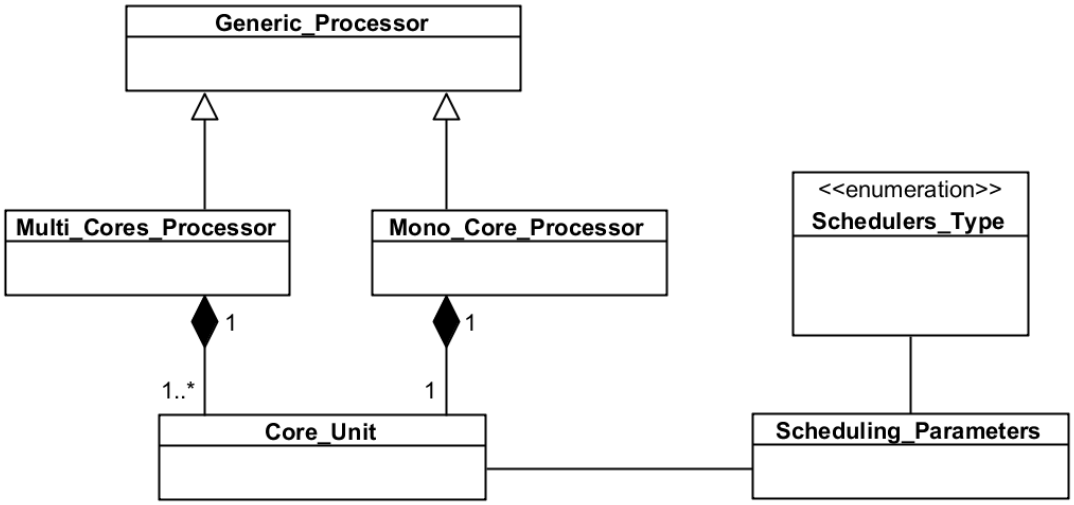


FIGURE 10.2: Hardware entities in Cheddar ADL [11]

A core is a unit that provides a resource to sequentially run tasks. It includes the scheduling parameters specification such as the preemptivity and the scheduling policy.

LISTING 10.5: Core units in Cheddar-ADL

```
<core_units>
  <core_unit id="id_1">
    <object_type>CORE_OBJECT_TYPE</object_type>
    <name>core1</name>
    <scheduling>
      <scheduling_parameters>
        <scheduler_type>
          HIERARCHICAL_OFFLINE_PROTOCOL
        </scheduler_type>
        <preemptive_type>PREEMPTIVE</preemptive_type>
        <user_defined_scheduler_source_file_name>
          partition_scheduling_paes2.xml
        </user_defined_scheduler_source_file_name>
        ...
      </scheduling_parameters>
    </scheduling>
```

```
    ...
  </core_unit>
  ...
</core_units>
```

Listing 10.5 presents the core units section of a TSP system modeled in Cheddar-ADL. It shows the attributes of a core named *core*1 such as the scheduling parameters of the partitions (e.g. offline scheduling). For TSP systems, the scheduling of the partitions is defined with an offline protocol via an XML file that specifies the major time frame on which the partitions should be scheduled. The attribute $user\_defined\_scheduler\_source\_file\_name$ in the core specification corresponds to the name of this file. Listing 10.6 presents an example of partitions scheduling model with two partitions. It corresponds to the MAF of the specified TSP system. Each partition is defined by an event that corresponds to the partition activation time and its duration.

Listing 10.6: partitions scheduling model in Cheddar-ADL

```
<!DOCTYPE Cheddar_Event_Table SYSTEM "event_table.dtd">
<?xml-stylesheet type="text/xsl" href="event_table.xsl"?>
<event_table>
  <time_unit>0</time_unit>
  <time_unit_event>
    <type_of_event>ADDRESS_SPACE_ACTIVATION</type_of_event>
    <activation_address_space>addr1</activation_address_space>
    <duration>250</duration>
  </time_unit_event>
  <time_unit>250</time_unit>
  <time_unit_event>
    <type_of_event>ADDRESS_SPACE_ACTIVATION</type_of_event>
    <activation_address_space>addr2</activation_address_space>
    <duration>250</duration>
  </time_unit_event>
</event_table>
```

Cheddar-ADL supports the modeling of unicore and multicore processors. It also proposes tasks migration during the execution.

Listing 10.7: Processors modeling in Cheddar-ADL

```
<processors>
  <mono_core_processor id="id_3">
    <object_type>PROCESSOR_OBJECT_TYPE</object_type>
    <name>processor1</name>
    <processor_type>MONOCORE_TYPE</processor_type>
    <migration_type>NO_MIGRATION_TYPE</migration_type>
    <core ref="id_1"> </core>
  </mono_core_processor>
```

```
  ...
</processors>
```

Listing 10.7 presents the processors section of a TSP system modeled in Cheddar-ADL. It specifies a unicore processor.

10.2 Cheddar scheduling analyzer

For a given model, the Cheddar tool provides scheduling analysis through feasibility tests or scheduling simulation on the feasibility interval.

10.2.1 Design of a model

The entry point of the Cheddar analyzer is a real-time system that can be modeled with various methods:

- Cheddar-ADL model: A real-time system to analyze with the Cheddar tool can be modeled with Cheddar-ADL as presented in the previous section.
- AADL model: A real-time system to analyze with the Cheddar tool can be modeled through the Architecture Analysis and Design Language (AADL) [187] by editors such as AADL inspector [26].
- Cheddar GUI: Cheddar proposes a graphical editor that helps to model easily a real-time system. The whole system can be designed by selecting and adding one by one any needed component (hardware or software). After the instantiation of all the components needed to build the given architecture, the model can be saved and exported in Cheddar-ADL.
- Ada: A program in Ada language can be created to model and call the Cheddar analyzer. This option can be difficult because users need to have a deeper knowledge about the Cheddar implementation.

10.2.2 Scheduling simulation

Cheddar tool proposes different scheduling policies such as RM, EDF, Round Robin (RR), Posix 1003 Highest Priority First. It also supports preemptive and non-preemptive policies. Tasks are then scheduled based on the specified scheduler. Simulation can be performed for partitioned systems with an offline scheduling of the partitions while considering an online scheduler for tasks inside partitions.

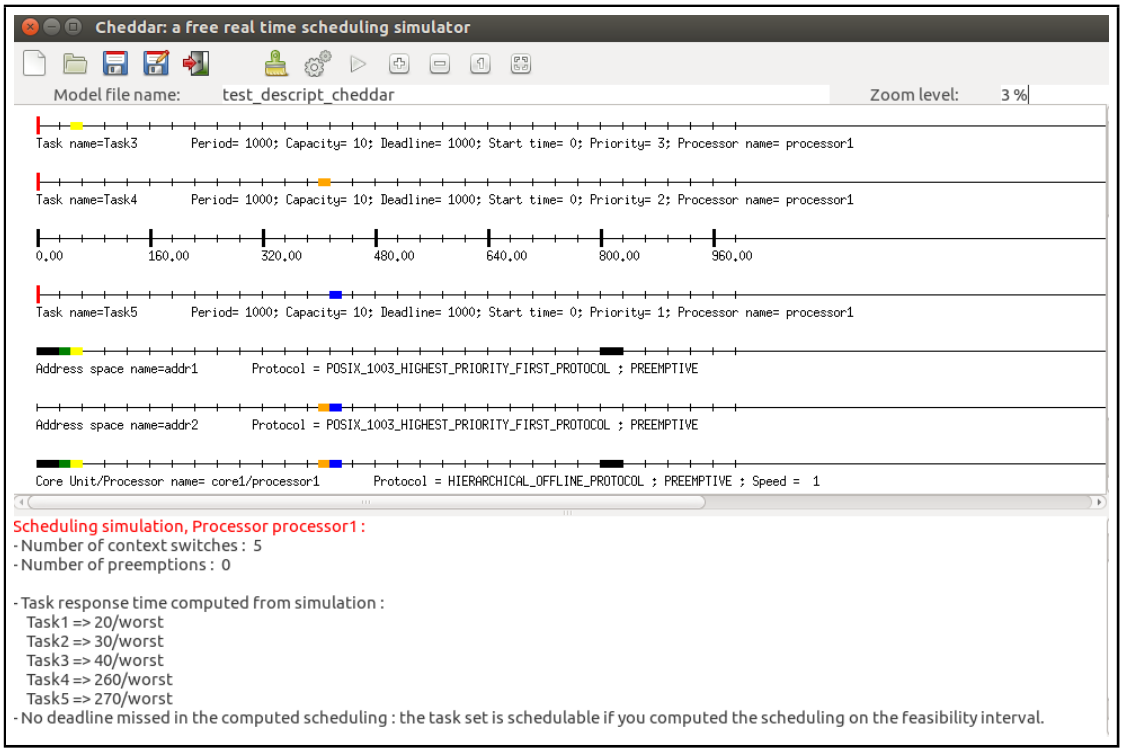


FIGURE 10.3: Cheddar scheduling simulation illustration

The simulation with the Cheddar analyzer considers all the components cited in the Cheddar-ADL sections. Then it managed periodic or aperiodic tasks while considering shared resources, dependencies, messages, buffers depending on the instantiated components.

A designer of a given model can get multiple informations from the scheduling simulation. Figure 10.3 shows the simulation results of the model described at listing 10.1.

The simulation results provide a layout showing the scheduling of the tasks on the assigned cores and address spaces (partitions). It also provides the number of context switches, the number of preemptions, the worst-case execution time (WCRT) of each task. The simulation reveals if the task set is schedulable and if not, it specifies the tasks that have missed their deadline.

The simulation with the Cheddar tool can be launched through Cheddar GUI, or via a terminal with a command line, or inside an Ada program.

10.2.3 Feasibility tests

Feasibility tests are implemented to evaluate the feasibility of a given model and then confirm or not the schedulability of the model. The Cheddar tool integrates many feasibility tests. As an example, it proposes a feasibility test proposed

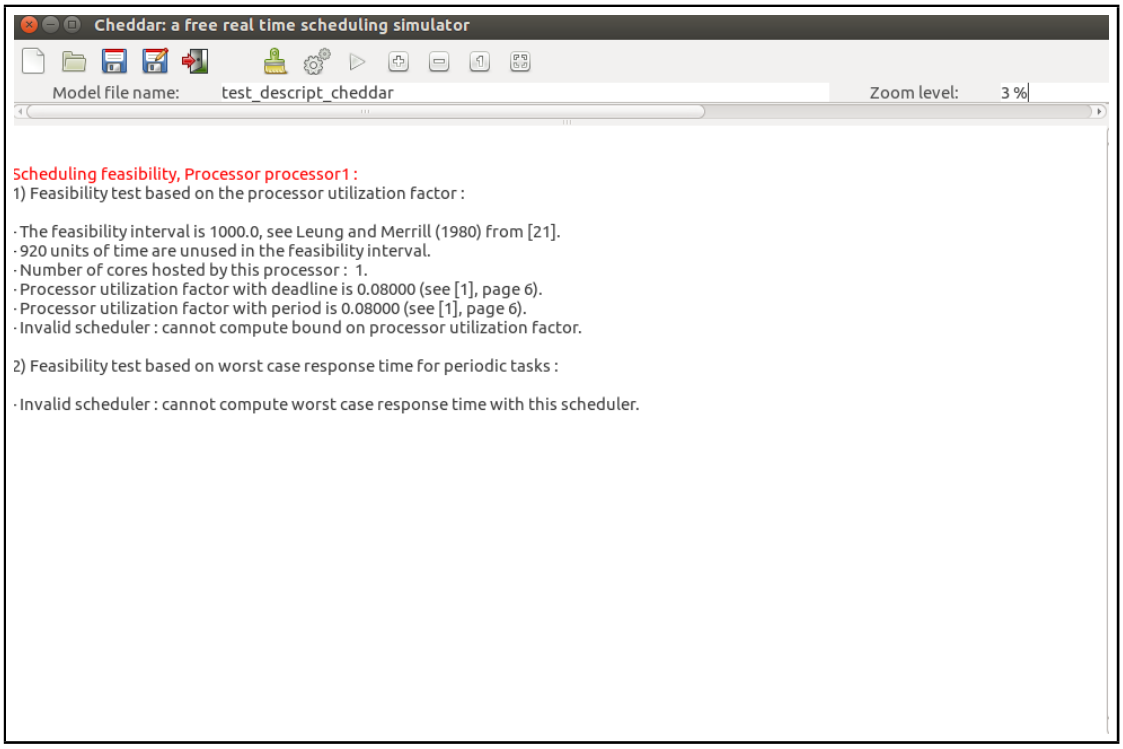


FIGURE 10.4: Cheddar feasibility test illustration

by Liu & Layland [43] based on the processor utilization which is a necessary condition for a schedulable model.

The Cheddar tool proposes also tests based on the WCRT for periodic tasks while including eventual delays implied by components such as shared resources. As for the simulation, feasibility tests with the Cheddar tool can be launched through its graphical editor, or via a terminal with a command line or inside an Ada program.

Figure 10.4 shows the results of the feasibility test applied to the model in listing 10.1 through the Cheddar GUI.

10.3 Implementation

In this section, we present the prototype implemented in the scope of this thesis. Figure 10.5 gives a set of libraries existing in Cheddar (e.g. Cheddar-ADL, graphical editor, scheduling simulator, feasibility tests) and an overview of our prototype. For the sake of lightness, all the packages and relations between them are not represented in the figure.

Our prototype is made of two libraries (PAES library, MILS library) that extend Cheddar framework, and a tool library called architecture exploration tools

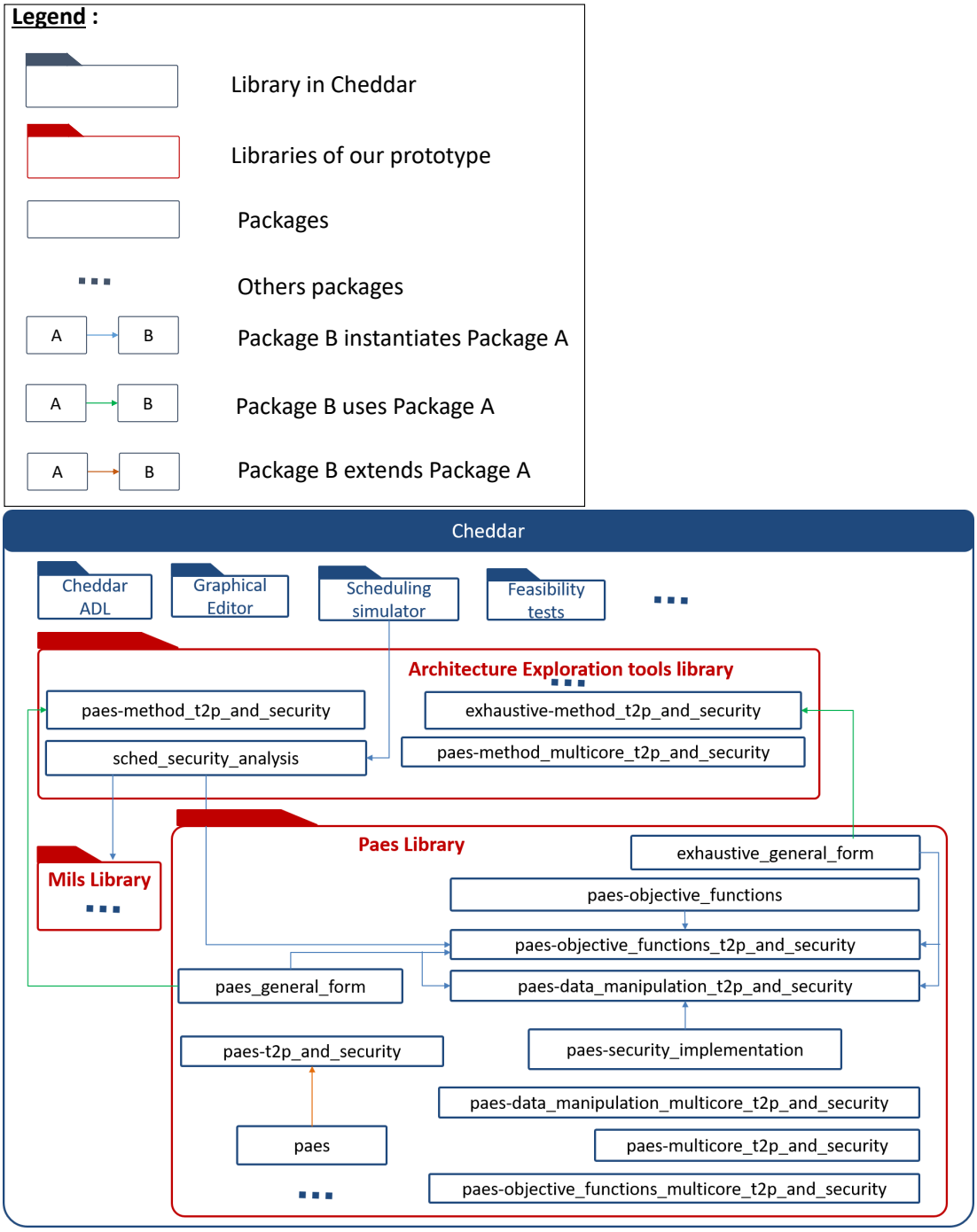


FIGURE 10.5: Prototype overview

library. As figure 10.5 shows, each library is made of packages (represented by blue boxes inside libraries).

10.3.1 MILS library

In order to evaluate our models according to the security properties, first of all, it was necessary to extend Cheddar with a security architecture. We extended Cheddar to model MILS architecture. As presented in chapter 4, a MILS architecture is composed of different entities such as partitions, applications, processes, and objects. Several components in Cheddar-ADL can be already used to model few MILS entities:

- MILS partitions or TSP partitions in general can be modeled by Cheddar-ADL *address space* entities.
- MILS objects can be modeled by Cheddar-ADL shared resources (i.e. buffer or message entities).
- MILS processes can be represented by Cheddar-ADL *tasks*.
- MILS communications have the same semantics as Cheddar-ADL dependencies.
- MILS functions do not need to be represented in Cheddar as they can be modeled by groups of partitions (i.e. groups of Cheddar-ADL *address spaces*).
- Finally, applications (which can be composed of one or more processes) are modeled by Cheddar-ADL *task* entities.

To complete the modeling capabilities of Cheddar for MILS architectures, several Cheddar-ADL entities have to be extended with new attributes modeling MILS properties. As an example, we need to model $buffers$ and *messages* confidentiality and integrity levels and also the right levels of *tasks* and partitions that are using them, i.e. if a partition or a task is allowed to handle a $buffer$ or a *message* according to its authorization levels. We give below the list of properties we actually added to Cheddar-ADL entities:

- An attribute named $confidentiality\_Level$ (Top_secret, $Secret$, $Classified$, $Unclassified$) has to be defined for each Cheddar-ADL *address spaces*, *objects*, and *tasks* entities.
- An attribute named $integrity\_Level$ ($High$, $Medium$, Low) also has to be defined for each Cheddar-ADL *addressspaces*, *objects*, and *tasks* entities.

- *MILS_component_type* (*SLS*, *MLS*, *MSLS*) is an attribute to specify MILS type of security level. Again, such attribute has to be defined in *tasks* and *addressspaces* Cheddar-ADL entities.
- Finally, *MILS_compliant_type* (*Non_Compliant*, *partition*,...) specifies if a Cheddar's entity models a MILS's component or not. Such attribute is defined in any Cheddar-ADL entity.

Thanks to these Cheddar-ADL extensions, we provide the MILS library that consists of the implementation of BLP, and Biba algorithms. For a specified Cheddar-ADL model, MILS library can confirm whether the security rules are respected or not, and enumerate the number of security vulnerabilities.

10.3.2 PAES library

The PAES library contains different packages for the formulation of multi-objective optimization problems. Most of these packages define subprograms needed for both PAES and exhaustive search methods and are described below. This work is an extension and generalization of the work presented by Rahma Bouaziz in [154].

The package *paes* is for any optimization problem, and implements the PAES metaheuristic framework itself. It defines a generic chromosome that can be extended in other packages depending on the optimization problem considered. It also proposes multiple subprograms such as the programs needed for archiving process in PAES and exhaustive search methods.

paes-general_form (resp. *exhaustive_general_form*) is an Ada generic program that could be instantiated by a PAES (resp. exhaustive search) tool for any multi-objective optimization problem.

The package *paes-t2p_and_security* is an extension to *paes* package to feet the schedulability vs security optimization problem in the context of unicore platforms. It contains the complete specification of the needed chromosome.

paes-objective_functions intervenes to set the objectives functions. It can be extended with any additional objective function.

data_manipulation_t2p_and_security provides programs specific to the schedulability vs security optimization problem in the context of unicore platforms. It contains any program that includes chromosome manipulation. There are subprograms such as mutation operator, transformation of a chromosome to a Cheddar-ADL model on which security and schedulability analysis can be performed.

paes-objective_functions-t2p_and_security is implemented to perform the security and schedulability analysis of a solution (chromosome) in the context of unicore platforms. Then with these analysis methods, the feasibility of a solution can

be checked. This feasibility test is based on respect of the safety, security, and schedulability constraints. All solutions that failed the feasibility test are ignored during DSE. This package also provides an evaluation of solutions by giving the objective functions values.

paes-security_implementation package contains subprograms needed to secure intra or/and inter-partitions communications according to the chromosome security configuration parameter. These programs are used during the transformation of a chromosome to the corresponding Cheddar-ADL model.

Each of the PAES library packages has been implemented twice: one version for unicore platforms and another one for multicore platforms.

As we presented in this section the packages that implement the prototype of this thesis, the next section is dedicated to present the tools to perform the DSE for unicore TSP, and multicore TSP systems.

10.3.3 Architecture exploration tools library

This library contains the tools for the unicore PAES, multicore PAES, and the exhaustive search methods. These programs are respectively *paes-method_t2p_and_security*, *paes-method_multicore_t2p_and_security*, and *exhaustive-method_t2p_and_security*. Each tool is a program that instantiates the needed generic programs (e.g. *paes-general_form, exhaustive_general_form*) presented in section 10.3.2.

For these tools, a predefined Cheddar-ADL model (initial system) can be given by the designer as an entry point. But it can also ask the tool to provide a generated initial system based on the Uunifast algorithm. This option is based on the *model_generator* package.

In this section, we presented the implementation of the PAES tool implementation in the scope of our thesis. Table 10.1 gives a specification of the PAES tool for unicore TSP systems.

We extend the method *paes-method_t2p_and_security* to comply with multicore TSP systems while considering safety constraints.

Table 10.2 gives a specification of exhausitive search tool in the context of unicore TSP systems.

In order to perform jointly schedulability and security analysis, we implemented the *sched_security_analysis* package. It can perform these analysis on a solution modeled in Cheddar-ADL. It is used by packages such as to evaluate each solution generated during the PAES or exhaustive methods.

The specification of the security analysis features is presented in Table 10.3.

Method	*method paes-method_t2p_and_security*
Purpose	Provide a set of trade-offs between schedulability and security for unicore TSP systems through a PAES based DSE. It implements the DSE proposed to address the conflict between schedulability and security for unicore TSP systems
Input	- A XML file containing a Cheddar-ADL model (set of processors, tasks, dependencies ...) which is the DSE initial system - Number of tasks - Number of applications - Maximum number of partitions - A set of XML files that describes the partitions scheduling considering different number of partitions (from 1 to the maximum number of partitions) - Scheduling policy - Number of iterations that determines the end of the exploration - List of objective functions (fitness functions) - The mutation algorithm (*app-grain*, *mix-grain* or *task-grain*) - Security implementation (e.g. all the intra partition communications are vulnerable or not)
Output at the end of the DSE	- A file that stores the list of non-dominated solutions (chromosomal form) - A file that stores the values of fitness functions of each non-dominated solutions in the archive - A set of XML files that correspond each to a non-dominated solutions in the archive - A file with the following information: • The values of fitness functions of each feasible candidate solution generated during the exploration • The number of rejected solutions during mutations (i.e. non feasible solutions) • The number of rejected solutions during archiving process - A file with the runtime of the PAES method

Table 10.1: PAES tool implementation

Method	*exhaustive-method_t2p_and_security*
Purpose	Provide optimal trade-offs between schedulability and security for unicore TSP systems. It implements the exhaustive to address the conflict between schedulability and security for unicore TSP systems
Input	- A XML file containing a Cheddar-ADL model (set of processors, tasks, dependencies ...) which is the DSE initial system - Number of tasks - Maximum number of partitions - XML files that described the partitions scheduling considering different number of partitions (from 1 to the maximum number of partitions) - Scheduling policy - List of objective functions (fitness functions)
Output at the end of the exhaustive search	- A file that shows the list of the optimal solutions (chromosomal form) - A file with the values of objectives fitness of each optimal solution in the archive at the end of the exhaustive search - A set of XML files that correspond each to an optimal solution - A file with the following information: • The values of fitness functions of each solution generated during the exhaustive method • The number of rejected solutions during mutations (i.e. non feasible solutions) • The number of rejected solution during archiving process - A file with the runtime of the exhaustive method

Table 10.2: Exhaustive tool implementation

Method	callCheddar_securityAnalysis
Purpose	Perform schedulability and security analysis. It includes: • A scheduling analysis based on a call of Cheddar that computes the scheduling simulation of the input (Cheddar-ADL model). We extract from the simulation, the WCRT of each task and then compute the number of tasks that missed their deadlines. • A security analysis that computes the number of confidentiality vulnerabilities based on BLP security model • A security analysis that computes the number of integrity vulnerabilities based on Biba security model
Input	- An XML file that describe a Cheddar-ADL model with information such as • set of cores with their parameters including the reference to an XML file that described the partitions scheduling in case of TSP systems • Set of processors • Set of tasks with parameters of each task (e.g WCET, period, deadline, confidentiality level) • Set of dependencies or messages between the tasks - An XML file that described the partitions scheduling - The hyperperiod of the set of tasks
Output	- A file with the following information • Schedulability of the taskset • Number of missed deadlines • Number of confidentiality vulnerabilities • Number of integrity vulnerabilities - A file with the Cheddar scheduling simulation information (e.g. number of preemption, WCRT of each task)

Table 10.3: Scheduing and security analysis

10.4 Conclusion

This chapter presents the prototype implemented during the thesis. This prototype is integrated into the Cheddar framework. Therefore, this chapter starts with an overview presentation of the Cheddar framework. Then it describes the packages that we implemented for the different considered DSE. First, it describes the packages to address the conflict between security and schedulability in the context of unicore TSP systems with PAES and an exhaustive exploration. Finally, the libraries can be reused and extended to different MOOPs. We illustrate such extensibility in the context of multicore TSP systems while considering safety constraints.

Part IV

Conclusion

11 Conclusion

The work presented in this thesis addresses the conflict between schedulability and security in real-time TSP systems. Securing real-time systems (RTS) implies extra features such as encryption and hashing algorithms. These features imply overheads. These overheads impact the schedulability of the systems and then may lead some tasks to miss their deadlines. It is then fruitfull to investigate how to mitigate the impact of the security related overheads on the schedulability of RTS.

In the scope of this work, an RTS is made of hard and/or soft deadline tasks. RTS cannot allow a hard deadline task to miss its deadline. On contrary, missed deadlines can be tolerated for soft deadline tasks. Thus soft deadline tasks can be allowed to miss their deadlines in order to optimize security.

Furthermore, we address specifically TSP systems that integrate different applications made of multiple tasks assigned to multiple partitions.

TSP systems host applications of different stakeholders with a potential high level of legacy. Historically, in order to limit fault propagation in integrated modular avionics (IMA) architecture, each partition host tasks of the same application. It is important to highlight that the tasks to partitions assignment has an impact on the schedulability of a system. Then we investigate different tasks to partitions assignment policies in order to find assignments that could favor schedulability. With multiple tasks assigned to multiple partitions, a combinatorial explosion problem is raised. The number of assignment possibilities grows exponentially with the number of tasks. Investigating all the possibilities can become humanly unmanageable and high time-consuming.

Thus the problem statement addressed in this thesis is multi-objective optimization problem (MOOP) between schedulability and security in TSP systems and the combinatorial problem it raised.

11.1 Contribution summary

In this section, we present a summary of the contributions of this thesis.

11.1.1 PAES adaptation to the MOOP between schedulability and security

The conflict between schedulability and security can be addressed as MOOP since for some systems, both objectives cannot be optimized at 100% simultaneously. Indeed, for some architectures, it can be impossible to design a schedulable (i.e. all tasks meet their deadlines) and fully secured system (i.e. security vulnerabilities are all fixed). Then we propose a DSE to explore the design space and compute a set of solutions that realize trade-offs between schedulability and security. A DSE approach fits well the combinatorial problem raised by the interest of investigating all the possibilities of assigning a numerous number of tasks on a large number of partitions. In that case, a DSE approach based on an exact method such as an exhaustive method that guaranties optimal solutions is not envisageable. Then we propose an adaptation of the PAES metaheuristic. This PAES adaptation proposes a set of near-optimal solutions in a reasonable time compared to an exhaustive method that can require several days, months, or even years to provide optimal solutions.

First, according to our addressed objective functions, we define the fitness functions to optimize during the DSE. We choose number of missed soft deadlines for schedulability and number of confidentiality and integrity vulnerabilities of weakly sensitive communications for security. Second, we proposed means to perform evaluations needed to compare solutions in order to find the best solutions among the explored solutions. For this purpose, we performed methods based on schedulability and security analysis to evaluate each solution. This contribution is presented in [151].

11.1.2 Mutation algorithms

The DSE is based on the exploration of the solutions space. These solutions are generated during the exploration. In PAES, solutions are generated based on mutation operations. Then we propose three algorithms that consider different granularity of mutations, correspond to different solution spaces.

First, we propose the *task-grain* mutation algorithm that considers all the tasks to partition assignment possibilities. Then at each iteration, a solution is generated from the previous solution by changing the location of a random task to a random partition. For a large-scale problem, this approach implies a large design space

that can impact the quality of the resulting trade-off solutions. As an alternative, we propose the *app-grain* mutation algorithm to reduce the design space. It proposes at each iteration to generate a solution from the previous solution by changing randomly the location of all the tasks of an application to a different partition. This design space is a subset of the previous one that is compliant with IMA architectures especially ARINC 653 standard.

The *app-grain* mutation algorithm reduces the degree of freedom. We propose a finer granularity approach while optimizing the exploration of a large design space. For this purpose, we propose the *mix-grain* algorithm that starts the DSE with *app-grain* mutation for a predefined number of iterations (over the DSE total number of iterations) and finishes with refining with *task-grain* mutation algorithm for the remaining iterations. The first phase is expected to find interesting solutions in the reduced design space, and the second phase to improve solution quality by enlarging the design space explored. This contribution is presented in [188].

11.1.3 Mutation algorithm improvement

Furthermore, in order to favor the diversity of explored solutions, we propose another algorithm that considers a better choice of the solution to mutate instead of only considering the solution generated at the previous iteration. Indeed, it can happen that during the DSE, after a certain number of iterations, multiple mutation operations fail to propose another non-dominated solution. The generated solutions are dominated at least by one solution in the archive. Considering our initial DSE approach, the current solution remains the same till a mutation operation finds a non-dominated solution. Then we propose to change the current solution after a certain number of mutation operations that failed to provide another non-dominated solution. It helps to operate mutation on another solution of the archive and then increase the chance to generate another non-dominated solution.

The above algorithms consider the tasks to partitions assignment. It is important to highlight that each of these algorithms includes a mutation operator on vulnerable communications between tasks. At each generation, a random communication that is vulnerable is secured based on four possibilities of security implementation.

11.1.4 Identification of the key parameters during DSE

From seven experiments, we identify key parameters impacting the trade-off between schedulability and security in TSP systems. First, we notice that for the

systems with low processor utilization and small data size, the overheads introduced by the security do not impact the schedulability of the systems. Then data size and CPU utilization are key parameters. Second, we also show that the number of partitions has a high impact on the size of the search space. The latter increases with the number of partitions. With a larger search space, it can be difficult for the DSE to converge to non-dominated solutions. A lower search space can reduce the freedom degree and limit the optimization of the objective functions. It is also important to highlight a high number of iterations may increase the chance to converge to better solutions. Furthermore, we confirm that the overheads introduced by the intra and inter-partition communications mechanisms (e.g. blackboards, sampling ports) have a significant impact on system schedulability. The DSE is able to optimize the tasks to partitions assignment in order to minimize inter-partition communications since they have higher overheads than intra-partition communications. The relevance of using different security implementations is confirmed by the diversity of the proposed solutions (i.e solutions with functions calls and solutions with dedicated tasks).

11.1.5 Extensibility of the DSE approach: safe and secure TSP systems on multicore execution platforms

Our DSE approach is an extensible approach that can be adapted to different contexts and/or MOOP. In order to investigate the extensibility of our approach, we propose to extend our approach to TSP systems with multicore execution platforms. We also considering safety requirements based on active redundancy. Safety, security, and shared hardware resources impact schedulability through the overheads they generate. In this approach, there are not only tasks to partitions assignment to consider but also tasks to cores assignment. This contribution is presented in [189].

11.1.6 Security architecture modeling and security analysis implementation

In this thesis, as security properties, we address the confidentiality and integrity of architecture models. In order to evaluate those properties, we implement Bell-La Padula (BLP) and Biba security models in Cheddar. This implementation requires first the modeling of TSP systems based on a security architecture (e.g. definition of confidentiality and integrity levels of tasks of a given model). Therefore we proposed to integrate the Multiple Independent Levels of Security (MILS) architecture [105] into the Cheddar tool that already proposes RTS modeling and their scheduling simulation. This work has been considered in the security analy-

sis proposed in the AADL Inspector commercial tool of Ellidiss Technologies and is presented in [190], and [191].

11.2 Future work

The contributions of this thesis raise some questions that could be considered for future work.

11.2.1 Memory protection mechanism

Considering our experiments, the *mix-grain* mutation algorithm that considers a finer granularity proposes good solutions out of the scope of *app-grain* mutation algorithm. This finer granularity may assume the existence of a memory protection mechanism between tasks within a partition in a system as ARINC 653. Then it could be interesting to study this mechanism since they already exist with threads and processes [168].

11.2.2 Security: investigation of different security models

In this thesis, we considered security vulnerabilities related to confidentiality and integrity. For this purpose, we consider BLP and Biba security models to evaluate security vulnerabilities. It can be interesting to integrate different security models. Since our approach is extensible, the integration of other security models [192] can be investigated. It can also be interesting to consider other security requirements such as availability [83] and then to integrate their related security models into our approach.

11.2.3 Schedulability: investigation of different possibilities of major time frame (MAF)

As presented in assumptions, partitions in TSP systems are scheduled based on offline scheduling executed during a cyclic interval called MAF. In this thesis, for simplicity, the MAFs considered in all our experiments are made with only one slot of each partition. However, as shown in [193] multiple time slots can be allocated to a partition during the MAF. It may be interesting to investigate the impact of MAF when optimizing schedulability and security since MAF is an important element to consider in the TSP systems configuration.

11.2.4 Conflict between schedulability and security: consideration of the possible overheads

For TSP on multicore platforms, we consider to introduce overheads due to interconnection. It is important to highlight that interconnection overheads are not the only overhead due to shared hardware resources in multicore platforms. There are other shared hardware resources such as cache and memory that implies overheads that can impact the schedulability of the systems. It may be interesting to evaluate how such overheads impact the DSE.

11.2.5 Extension to distributed network platforms

As we propose to extend the work to multicore execution platforms, it may be interesting to investigate safe and secure TSP systems on distributed network platforms. This implies other overhead such as network communications overhead [194] known to be bounded but variables.

11.2.6 Finer granularity: functions

In this work, we only investigate tasks, partitions, and cores level. Since tasks can be considered as sets of functions, we may investigate a finer granularity by considering functions level as future work.

Bibliography

[1] Miguel Masmano, Ismael Ripoll, Alfons Crespo, and J Metge. Xtratum: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop*, pages 263–272. Citeseer, 2009.

[2] Felipe Gohring de Magalhaes, Alexy Torres Aurora Dugo, Jean-Baptiste Lefoul, and Gabriela Nicolescu. On the benchmarking of partitioned real-time systems. *arXiv e-prints*, pages arXiv–2007, 2020.

[3] Insup Lee, Joseph YT Leung, and Sang H Son. *Handbook of real-time and embedded systems*. CRC Press, 2007.

[4] Benjamin Rouxel and Isabelle Puaut. Str2rts: Refactored streamit benchmarks into statically analyzable parallel benchmarks for wcet estimation & real-time scheduling. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[5] Airlines Electronic Engineering Committee. Arinc 653: Avionics application software standard interface, supplement 1. 2003.

[6] Artur Oliveira Gomes. *Formal specification of the ARINC 653 architecture using circus*. PhD thesis, University of York, 2012.

[7] Timothy Shimeall and Jonathan Spring. *Introduction to information security: a strategic-based approach*. Newnes, 2013.

[8] Yongwang Zhao. Formal specification and verification of separation kernels: An overview. *ArXiv e-prints, no*, 2015.

[9] S Nguyen and Voratas Kachitvichyanukul. Movement strategies for multi-objective particle swarm optimization. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(3):59–79, 2010.

[10] Lingling Xue, Peng Zeng, and Haibin Yu. Setnds: A set-based non-dominated sorting algorithm for multi-objective optimization problems. *Applied Sciences*, 10(19):6858, 2020.

[11] Hai Nam Tran. *Cache memory aware priority assignment and scheduling simulation of real-time embedded systems*. PhD thesis, Brest, 2017.

[12] John A Stankovic. A serious problem for next-generation systems. *IEEE computer*, 21(10):10–19, 1988.

[13] Neil Audsley, Alan Burns, Rob Davis, Ken Tindell, and Andy Wellings. Real-time system scheduling. In *Predictably dependable computing systems*, pages 41–52. Springer, 1995.

[14] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency: Predictability Vs. Efficiency*. Springer Science & Business Media, 2005.

[15] Pierre Bieber, Frédéric Boniol, Marc Boyer, Eric Noulard, and Claire Pagetti. New challenges for future avionic architectures. *AerospaceLab*, (4):p–1, 2012.

[16] Richard Garside and F Joe Pighetti. Integrating modular avionics: A new role emerges. *IEEE Aerospace and Electronic Systems Magazine*, 24(3):31–34, 2009.

[17] HELTON Steven et FEILER Peter HANSSON, Jorgen. Roi analysis of the system architecture virtual integration initiative. In *Carnegie-Mellon Univerity Software Engineering Institute Pittsburgh United States*, 2018.

[18] Christopher B Watkins and Randy Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2–A. IEEE, 2007.

[19] James Windsor and Kjeld Hjortnaes. Time and space partitioning in spacecraft avionics. In *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*, pages 13–20. IEEE, 2009.

[20] Andoni Amurrio González and Mario Aldea Rivas. Schedulability analysis and optimization of time-partitioned distributed real-time systems.

[21] Alvaro Cardenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, Shankar Sastry, et al. Challenges for securing cyber physical systems. In *Workshop on future directions in cyber-physical systems security*, volume 5. Citeseer, 2009.

[22] Vuk Lesi, Ilija Jovanov, and Miroslav Pajic. Security-aware scheduling of embedded control tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–21, 2017.

[23] Jawahar Thakur and Nagesh Kumar. Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2):6–12, 2011.

[24] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 98–105. IEEE, 1999.

[25] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.

[26] Ellidiss Technologies. Aadl inspector.

[27] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The rosace case study: From simulink specification to multi/many-core execution. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 309–318. IEEE, 2014.

[28] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[29] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[30]

[31] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *International Workshop on Fast Software Encryption*, pages 191–204. Springer, 1993.

[32] Wanzhong Sun, Hongpeng Guo, Huilei He, and Zibin Dai. Design and optimized implementation of the sha-2 (256, 384, 512) hash algorithms. In *2007 7th International Conference on ASIC*, pages 858–861. IEEE, 2007.

[33] Santosh D Chede and Kishore D Kulat. Design overview of processor based implantable pacemaker. *J. Comput.*, 3(8):49–57, 2008.

[34] James Martin. Programming real-time computer systems. Technical report, 1965.

[35] 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.

[36] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.

[37] RTCA (Firme). *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*. RTCA, 2005.

[38] Rick Grehan, Ingo Cyliax, and Robert Moote. *Real-Time Programming: A Guide to 32-Bit Embedded Development with Cdrom*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[39] Chuanpeng Li, Chen Ding, and Kai Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 workshop on Experimental computer science*, pages 2–es, 2007.

[40] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991.

[41] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008.

[42] José Carlos Palencia and M González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pages 26–37. IEEE, 1998.

[43] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[44] Kang G Shin and Parameswaran Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, 1994.

[45] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):1–44, 2011.

[46] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 301–310. IEEE, 2010.

[47] Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, Seyed MH Ashjaei, and Sara Afshar. Support for hierarchical scheduling in freertos. In *ETFA2011*, pages 1–10. IEEE, 2011.

[48] José Rufino, Sergio Filipe, Manuel Coutinho, Sérgio Santos, and James Windsor. Arinc 653 interface in rtems. In *Proc. DASIA*, 2007.

[49] Victor Yodaiken et al. The rtlinux manifesto. In *Proc. of the 5th Linux Expo*, 1999.

[50] Robert Kaiser and Stephan Wagner. Evolution of the pikeos microkernel. In *First International Workshop on Microkernels for Embedded Systems*, volume 50, 2007.

[51] 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.

[52] Neil C Audsley, Alan Burns, Mike F Richardson, and Andy J Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991.

[53] Giorgio C Buttazzo. Rate monotonic vs. edf: Judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

[54] Sanjoy K Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[55] Alan Burns and Sanjoy Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[56] Gilles Lasnier. *Une approche intégrée pour la validation et la génération de systèmes critiques par raffinement incrémental de modèles architecturaux*. PhD thesis, Paris, ENST, 2012.

[57] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software engineering journal*, 8(5):284–292, 1993.

[58] Joël Goossens and Raymond Devillers. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 13(2):107–126, 1997.

[59] 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.

[60] Richard Urunuela, A Deplanche, and Yvon Trinquet. Storm, a simulation tool for real-time multiprocessor scheduling evaluation. In *Proceeding of the 15th Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2010.

[61] Hai Nam Tran, Stéphane Rubini, Jalil Boukhobza, and Frank Singhoff. Feasibility interval and sustainable scheduling simulation with crpd on uniprocessor platform. *Journal of Systems Architecture*, 115:102007, 2021.

[62] Gernot Heiser. Virtualization for embedded systems. *Open Kernel Labs Technology White Paper*, 2007.

[63] Sanghyun Han and Hyun-Wook Jin. Full virtualization based arinc 653 partitioning. In *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, pages 7E1–1. IEEE, 2011.

[64] Moris Behnam, Thomas Nolte, Insik Shin, Mikael Åsberg, and Reinder Bril. Towards hierarchical scheduling in vxworks. In *OSPERT 2008, Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Prague, Czech Republic, July 1, 2008*, pages 63–72, 2008.

[65] Moris Behnam, Insik Shin, Thomas Nolte, and Mikael Nolin. Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 279–288, 2007.

[66] Moris Behnam. *Hierarchical Real Time Scheduling and Synchronization*. PhD thesis, Mälardalens högskola, 2008.

[67] Aloysius K Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, pages 75–84. IEEE, 2001.

[68] Arvind Easwaran. *Advances in hierarchical real-time systems: Incrementality, optimality, and multiprocessor clustering*. PhD thesis, University of Pennsylvania, 2008.

[69] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 2–13. IEEE, 2003.

[70] EASA. Amc 20-170'integrated modular avionics (ima).

[71] James W Ramsey. Integrated modular avionics: Less is more-fresh approaches to integrated modular avionic architectures will save weight, improve reliability of a380 and b787 systems. *Avionics Magazine*, 31(2):24, 2007.

[72] Christian M Fuchs et al. The evolution of avionics networks from arinc 429 to afdx. *Innovative Internet Technologies and Mobile Communications (IITM), and Aerospace Networks (AN)*, 65:1551–3203, 2012.

[73] Airlines Electronic Engineering Committee et al. Aircraft data network part 7, avionics full duplex switched ethernet (afdx) network, arinc specification 664. *Aeronautical Radio*, 2002.

[74] Airlines Electronic Engineering Committee et al. Arinc: 653p1· 3-2006 avionics application software standard interface pan1—equired services.

[75] Yann-Hang Lee, Daeyoung Kim, Mohamed Younis, and Jeff Zhou. Partition scheduling in apex runtime environment for embedded avionics software. In *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No. 98EX236)*, pages 103–109. IEEE, 1998.

[76] Franck Wartel, Leonidas Kosmidis, Adriana Gogonel, Andrea Baldovino, Zoe Stephenson, Benoit Triquet, Eduardo Quinones, Code Lo, Enrico Mezzetta, Ian Broster, et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 397–402. IEEE, 2015.

[77] Franck Wartel, Leonidas Kosmidis, Code Lo, Benoit Triquet, Eduardo Quinones, Jaume Abella, Adriana Gogonel, Andrea Baldovin, Enrico Mezzetti, Liliana Cucu, et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248. IEEE, 2013.

[78] SYSGO Embedded Innovations. Pikeos hypervisor eclipse based codeo. *En ligne], disponible: http://www. sysgo. com/en/products/pikeos-rtos-and-virtualization-concept/eclipsebased-codeo/(Consulté: 27 Juillet 2014)*.

[79] Julien Delange and Laurent Lec. Pok, an arinc653-compliant operating system released under the bsd license. In *13th Real-Time Linux Workshop*, volume 10, pages 181–192, 2011.

[80] LYNX Software Technologies. Lynxsecure: software security driven by an embedded hypervisor.

[81] Jörgen Hansson, Lutz Wrage, Peter H Feiler, John Morley, Bruce Lewis, and Jerome Hugues. Architectural modeling to verify security and nonfunctional behavior. *IEEE Security & Privacy*, 8(1):43–49, 2010.

[82] Jörgen Hansson, Peter H Feiler, and John Morley. Building secure systems using model-based engineering and architectural models. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2008.

[83] Manuel Cheminod, Luca Durante, and Adriano Valenzano. Review of security issues in industrial networks. *IEEE transactions on industrial informatics*, 9(1):277–293, 2012.

[84] Prerna Mahajan and Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 2013.

[85] Sourabh Chandra, Smita Paira, Sk Safikul Alam, and Goutam Sanyal. A comparative survey of symmetric and asymmetric key cryptography. In *2014 international conference on electronics, communication and computational engineering (ICECCE)*, pages 83–93. IEEE, 2014.

[86] Nan Li. Research on diffie-hellman key exchange protocol. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–634. IEEE, 2010.

[87] D Elliott Bell and Leonard J La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, MITRE CORP BEDFORD MA, 1976.

[88] L Arockiam and S Monikandan. Efficient cloud storage confidentiality to ensure data security. In *2014 International Conference on Computer Communication and Informatics*, pages 1–5. IEEE, 2014.

[89] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. Haval—a one-way hashing algorithm with variable length of output. In *International workshop on the theory and application of cryptographic techniques*, pages 81–104. Springer, 1992.

[90] Ronald Rivest. Rfc1321: The md5 message-digest algorithm, 1992.

[91] James H Burrows. Secure hash standard. Technical report, Department of Commerce Washington DC, 1995.

[92] Patrick Gallagher and Acting Director. Secure hash standard (shs). *FIPS PUB*, 180:183, 1995.

[93] Robert P McEvoy, Francis M Crowe, Colin C Murphy, and William P Marnane. Optimisation of the sha-2 family of hash functions on fpgas. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, pages 6–pp. IEEE, 2006.

[94] Xinxin Fan and Guang Gong. Accelerating signature-based broadcast authentication for wireless sensor networks. *Ad Hoc Networks*, 10(4):723–736, 2012.

[95] James M Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 198(1), 2008.

[96] Ming-Xin Yang, Li-Na Yuan, and Zhi-Xia Yang. A discuss of computer security strategy models. In *2010 Int. Conf. on Machine Learning and Cybernetics*, volume 2, pages 839–842. IEEE, 2010.

[97] Barack Obama. Executive order 13526: Classified national security information. In *United States. Office of the Federal Register*, number Executive order 13526; EO 13526. United States. Office of the Federal Register, 2009.

[98] G Scott Graham and Peter J Denning. Protection: principles and practice. In *Proceedings of Spring Joint Computer conference*, pages 417–429. ACM, 1972.

[99] Francois Mouton, Alastair Nottingham, Louise Leenen, and HS Venter. Underlying finite state machine for the social engineering attack detection model. In *2017 Information Security for South Africa (ISSA)*, pages 98–105. IEEE, 2017.

[100] Lei Gong, Lu Tian, and Fulian Zhang. Application information flow non-interference transmission model. In *Proceedings of 2011 Int. Conf. on Electronic & Mechanical Engineering and Information Technology*, volume 5, pages 2306–2309. IEEE, 2011.

[101] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, MITRE CORP BEDFORD MA, 1977.

[102] Jim Alves-Foss, Paul W Oman, Carol Taylor, and W Scott Harrison. The mils architecture for high-assurance embedded systems. *International journal of embedded systems*, 2(3-4):239–247, 2006.

[103] EURO-MILS Consortium et al. Euro-mils: Secure european virtualisation for trustworthy applications in critical domains. 2012-2016. 7th framework programme. Technical report, FP7-ICT-2011-8.

[104] John M Rushby. Design and verification of secure systems. *ACM SIGOPS Operating Systems Review*, 15(5):12–21, 1981.

[105] W Mark Vanfleet, Jahn A Luke, R William Beckwith, Carol Taylor, Ben Calloni, and Gordon Uchenick. Mils: Architecture for high-assurance embedded computing. *CrossTalk*, 18(8):12–16, 2005.

[106] Sysgo-Embedding Innovations. Pikeos hypervisor, 2015.

[107] R William Beckwith, W Mark Vanfleet, and Lee MacLaren. High assurance security/safety for deeply embedded, real-time systems. In *Proceedings of the Embedded Systems Conference*, 2004.

[108] Rance J. DeLong. Mils: An architecture for security, safety, and real time, 2006.

[109] W Scott Harrison, Nadine Hanebutte, P Oman, and Jim Alves-Foss. The mils architecture for a secure global information grid. *Crosstalk: The Journal of Defense Software Engineering*, 18(10):20–24, 2005.

[110] Jim Alves-Foss, Carol Taylor, and Paul Oman. A multi-layered approach to security in high assurance systems. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10–pp. IEEE, 2004.

[111] Bryan Rossebo, Paul Oman, Jim Alves-Foss, Ryan Blue, and Paul Jaszkowiak. Using spark-ada to model and verify a mils message router. Technical report, IDAHO UNIV MOSCOW CENTER FOR SECURE AND DEPENDABLE SYSTEMS, 2006.

[112] Stephen Chong and Ron Van Der Meyden. Using architecture to reason about information security. *ACM Transactions on Information and System Security (TISSEC)*, 18(2):1–30, 2015.

[113] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. Pareto multi objective optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*, pages 84–91. IEEE, 2005.

[114] Carlos A Coello Coello, Silvia González Brambila, Josué Figueroa Gamboa, Ma Guadalupe Castillo Tapia, and Raquel Hernández Gómez. Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. *Complex & Intelligent Systems*, 6(2):221–236, 2020.

[115] Zhenan He, Gary G Yen, and Jun Zhang. Fuzzy-based pareto optimality for many-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 18(2):269–285, 2013.

[116] 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.

[117] Abdullah Konak, David W Coit, and Alice E Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety*, 91(9):992–1007, 2006.

[118] 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*, 19(3):1867–1901, 2017.

[119] Il Yong Kim and Oliver L De Weck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and multidisciplinary optimization*, 29(2):149–158, 2005.

[120] Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-spektrum*, 22(4):425–460, 2000.

[121] Yezid Donoso and Ramon Fabregat. *Multi-objective optimization in computer networks using metaheuristics*. CRC Press, 2016.

[122] Abraham Charnes and William Wager Cooper. Goal programming and multiple objective optimizations: Part 1. *European journal of operational research*, 1(1):39–54, 1977.

[123] James P Ignizio. Generalized goal programming an overview. *Computers & Operations Research*, 10(4):277–289, 1983.

[124] Carlos Romero. A survey of generalized goal programming (1970–1982). *European Journal of Operational Research*, 25(2):183–191, 1986.

[125] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

[126] Carlos Romero. Extended lexicographic goal programming: a unifying approach. *Omega*, 29(1):63–71, 2001.

[127] UC Orumie and DW Ebong. An efficient method of solving lexicographic linear goal programming problem. *International journal of scientific and research publications*, 3(10):1–8, 2013.

[128] Carlos A Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. Multiobjective combinatorial optimization: Problematic and context. *Advances in multi-objective nature inspired computing*, 272:1–21, 2010.

[129] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

[130] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[131] 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.

[132] Margarita Reyes-Sierra, CA Coello Coello, et al. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

[133] 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.

[134] 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.

[135] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[136] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[137] Carlo R Raquel and Prospero C Naval Jr. An effective use of crowding distance in multiobjective particle swarm optimization. In *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*, pages 257–264, 2005.

[138] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer, 2000.

[139] Joshua D Knowles and David W Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation*, 8(2):149–172, 2000.

[140] David A Van Veldhuizen, Gary B Lamont, et al. Evolutionary computation and convergence to a pareto front. In *Late breaking papers at the genetic programming 1998 conference*, pages 221–228. Citeseer, 1998.

[141] Leonardo CT Bezerra, Manuel López-Ibánez, and Thomas Stützle. An empirical assessment of the properties of inverted generational distance on multi-and many-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 31–45. Springer, 2017.

[142] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[143] Lucas S Batista, Felipe Campelo, Frederico G Guimarães, and Jaime A Ramírez. The cone epsilon-dominance: an approach for evolutionary multiobjective optimization. *arXiv preprint arXiv:2008.04224*, 2020.

[144] Günter Rudolph. *Convergence properties of evolutionary algorithms*. Verlag Dr. Kovač, 1997.

[145] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[146] Yilin Mo, Emanuele Garone, Alessandro Casavola, and Bruno Sinopoli. False data injection attacks against state estimation in wireless sensor networks. In *49th IEEE Conference on Decision and Control (CDC)*, pages 5967–5972. IEEE, 2010.

[147] Yulong Zou and Gongpu Wang. Intercept behavior analysis of industrial wireless sensor networks in the presence of eavesdropping attack. *IEEE Transactions on Industrial Informatics*, 12(2):780–787, 2015.

[148] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned aircraft capture and control via gps spoofing. *Journal of Field Robotics*, 31(4):617–636, 2014.

[149] Tao Xie and Xiao Qin. Scheduling security-critical real-time applications on clusters. *IEEE transactions on computers*, 55(7):864–879, 2006.

[150] Monowar Hasan, Sibin Mohan, Rodolfo Pellizzoni, and Rakesh B Bobba. A design-space exploration for allocating security tasks in multicore real-time systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 225–230. IEEE, 2018.

[151] Ill-ham Atchadam, Laurent Lemarchand, Hai Nam Tran, Frank Singhoff, and Karim Bigou. When security affects schedulability of tsp systems: trade-offs observed by design space exploration. In *25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria - Hybrid*, volume 1, pages 369–376. IEEE, 2020.

[152] Tao Xie and Xiao Qin. Improving security for periodic tasks in embedded systems through scheduling. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(3):20, 2007.

[153] Quazi N Ahmed and Susan V Vrbsky. Maintaining security in firm real-time database systems. In *Proceedings 14th Annual Computer Security Applications Conference (Cat. No. 98EX217)*, pages 83–90. IEEE, 1998.

[154] Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, and Mohamed Jmaiel. Multi-objective design exploration approach for ravenscar real-time systems. *Real-Time Systems*, 54(2):424–483, 2018.

[155] Binto George and Jayant Haritsa. Secure transaction processing in firm real-time database systems. In *ACM SIGMOD Record*, volume 26, pages 462–473. ACM, 1997.

[156] Sang Hyuk Son, Ravi Mukkamala, and Rasikan David. Integrating security and real-time requirements using covert channel capacity. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):865–879, 2000.

[157] Qixuan Xue, Yongxin Zhu, Yajie Wang, Kedun Mao, Han Wu, Mengjun Li, Yishu Mao, and Junjie Hou. A scheduling scheme of task allocation in real time multiple-partition embedded avionic. In *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 41–46. IEEE, 2017.

[158] Wei Jiang, Paul Pop, and Ke Jiang. Design optimization for security- and safety-critical distributed real-time applications. *Microprocessors and Microsystems*, 52:401–415, 2017.

[159] Olivier Gilles. *Vers une prise en compte fine de la plate-forme cible dans la construction des systemes temps réel embarqués critiques par ingénierie des modeles*. PhD thesis, Telecom ParisTech, 2010.

[160] Herbert Dawid and Michael Kopel. On economic applications of the genetic algorithm: a model of the cobweb type. *Journal of Evolutionary Economics*, 8(3):297–315, 1998.

[161] Ludo Waltman, Nees Jan van Eck, Rommert Dekker, and Uzay Kaymak. Economic modeling using evolutionary algorithms: the effect of a binary encoding of strategies. *Journal of evolutionary economics*, 21(5):737–756, 2011.

[162] Eduardo Raul Hruschka, Ricardo JGB Campello, Alex A Freitas, et al. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(2):133–155, 2009.

[163] Wei Dai. Crypto++ 5.6. 0 benchmarks. *http://www. cryptopp. com/benchmarks. html*, 2009.

[164] Kui Zhang, Ji Wu, Chao Liu, Syed Sarmad Ali, and Jian Ren. Behavior modeling on arinc653 to support the temporal verification of conformed application design. *IEEE Access*, 7:23852–23863, 2019.

[165] Basil Cameron Rennie and Annette Jane Dobson. On stirling numbers of the second kind. *Journal of Combinatorial Theory*, 7(2):116–121, 1969.

[166] Steve Corrigan HPL. Introduction to the controller area network (can). *Appl. Rep. SLOA101*, pages 1–17, 2002.

[167] Andrew Kornecki, Janusz Zalewski, Janusz Sosnowski, and D Trawczynski. A study on avionics and automotive databus safety evaluation. *Archives of Transport*, 17(3/4):107–131, 2005.

[168] Juan Carlos Martínez Santos and Yunsi Fei. Hati: Hardware assisted thread isolation for concurrent c/c++ programs. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 322–331. IEEE, 2014.

[169] Airlines Electronic Engineering Committee. "avionics application software interface part 1 - required services", 2010.

[170] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 3–11. IEEE, 2020.

[171] Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan, and Stefan M Petters. Identifying the sources of unpredictability in cots-based multicore systems. In *2013 8th IEEE international symposium on industrial embedded systems (SIES)*, pages 39–48. IEEE, 2013.

[172] José V Busquets-Mataix, Juan José Serrano, Rafael Ors, Pedro Gil, and Andy Wellings. *Adding instruction cache effect to schedulability analysis of preemptive real-time systems*. IEEE, 1996.

[173] Lei Chai, Qi Gao, and Dhabaleswar K Panda. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. In *Seventh IEEE international symposium on cluster computing and the grid (CCGrid'07)*, pages 471–478. IEEE, 2007.

[174] Rakesh Kumar, Victor Zyuban, and Dean M Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 408–419. IEEE, 2005.

[175] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158–179, 1995.

[176] Steven P Miller, Darren D Cofer, Lui Sha, Jose Meseguer, and Abdullah Al-Nayeem. Implementing logical synchrony in integrated modular avionics. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pages 1–A. IEEE, 2009.

[177] Elisabeth A Strunk, John C Knight, and M Anthony Aiello. Distributed reconfigurable avionics architectures. In *The 23rd Digital Avionics Systems Conference (IEEE Cat. No. 04CH37576)*, volume 2, pages 10–B. IEEE, 2004.

[178] Richard D Schlichting and Fred B Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems (TOCS)*, 1(3):222–238, 1983.

[179] Jaynarayan H Lala and Richard E Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[180] WR Moore and NA Haynes. A review of synchronisation and matching in fault-tolerant systems. *IEE Proceedings E (Computers and Digital Techniques)*, 131(4):119–124, 1984.

[181] Filipe Araujo, Serhiy Boychenko, Raul Barbosa, and António Casimiro. Replica placement to mitigate attacks on clouds. *Journal of Internet Services and Applications*, 5(1):1–13, 2014.

[182] Certification Authorities Software Team. Multi-core processors - position paper. technical report cast 32-a, 2016.

[183] Mathieu Patte, Vincent Lefftz, Marco Zulianello, A Crespo, Miguel Masmano, and Javier Coronel. System impact of distributed multi core systems. *Technical Report ESTEC Contract 4200023100*, 2011.

[184] Joao Craveiro, José Rufino, and Frank Singhoff. Architecture, mechanisms and scheduling analysis tool for multicore time-and space-partitioned systems. *ACM SIGBED Review*, 8(3):23–27, 2011.

[185] Jérôme Hugues, Christophe Honvault, and Claire Pagetti. Model-based design, analysis and synthesis for multi-core and tsp avionics targets. 2018.

[186] Javier Coronel, M Tsagkaropoulos, Dimitrios Mylonas, Patricia Balbastre, Vangelis Kollias, and Alfons Crespo. Validation of securely partitioned

systems over multicore architectures based on xtratum. In *Data systems in aerospace (DASIA), Proceedings on*, 2013.

[187] Peter H Feiler and David P Gluch. *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*. Addison-Wesley, 2012.

[188] Ill-Ham Atchadam, Frank Singhoff, Hai Nam Tran, and Laurent Lemarchand. A design space exploration approach to jointly optimize security and schedulability in tsp systems. In *Poster presented in Colloque du GDR SOC2, Strasbourg, France*, 2022.

[189] Ill-ham Atchadam, Laurent Lemarchand, Frank Singhoff, and Hai Nam Tran. Observing the impact of multicore execution platform for tsp systems under schedulability, security and safety constraints. In *17th International Workshop on "Dependable Smart Embedded and Cyber-Physical Systems and Systems-of-Systems (DECSoS22), Munich, Germany, 2022*, volume 13415, pages 83–96. Springer, 2022.

[190] Ill-ham Atchadam, Frank Singhoff, Hai Nam Tran, Noura Bouzid, and Laurent Lemarchand. Combined security and schedulability analysis for mils real-time critical architectures. In *4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2019), Stuttgart, Germany*, pages 1:1–1:12.

[191] P Dissaux, Frank Singhoff, L Lemarchand, Hai Nam Tran, and Ill-Ham Atchadam. Combined real-time, safety and security model analysis. In *9th European Congress ERTSS Embedded Real Time Software and System, Toulouse, France*, 2020.

[192] John McLean. Security models. *Encyclopedia of software engineering*, 2:1136–1145, 1994.

[193] Andoni Amurrio, J Javier Gutiérrez, Mario Aldea, and Ekain Azketa. Priority assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows. *Journal of Systems Architecture*, 122:102339, 2022.

[194] Dar-Tzen Peng, Kang G. Shin, and Tarek F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, 1997.

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

UBO
Université de Bretagne Occidentale

Titre : Exploration d'architectures logicielles pour les systèmes critiques partitionnés sécurisés

Mots clés : systèmes partitionnés, ordonnançabilité, sécurité, optimisation multi-objective

Résumé : Les systèmes temps réel modernes intègrent de plus en plus de fonctions. Face à cette complexité, des mécanismes d'isolation sont employés afin qu'une défaillance survenant dans une fonction ne puisse pas affecter les autres. Cette thèse porte sur les architectures TSP (Time and Space Isolation). Elles introduisent le concept de partition afin d'assurer l'isolation spatiale et temporelle des applications. Les applications peuvent être assignées à des partitions en fonction de diverses fonctions objectives ou contraintes liées aux fonctions à implanter (e.g. sûreté, performances, sécurité). Certaines de ces fonctions objectives peuvent être conflictuelles. Ainsi, l'amélioration de la sécurité d'un système par ajout de fonctions dédiées à la sécurité (e.g. chiffrements) peut avoir un impact sur son ordonnançabilité.
C'est dans ce contexte que nous étudions dans cette thèse, le caractère conflictuel entre l'ordonnançabilité et la sécurité (confidentialité et intégrité) dans les systèmes temps réel TSP. Nous proposons l'exploration de l'espace de solutions (DSE) en utilisant une métaheuristique multi objective, qui fournit des compromis entre l'ordonnançabilité et la sécurité pour ces systèmes. Nous proposons trois algorithmes de DSE pour des systèmes TSP monoprocesseur basés sur la métaheuristique Pareto archived evolutionary Strategy (PAES). Nous proposons également une méthode afin de favoriser la diversité des compromis proposés à l'issue d'une exploration. Ces algorithmes sont implantés dans Cheddar, un outil d'analyse de l'ordonnançabilité auquel nous avons intégré l'analyse de la sécurité. Les algorithmes sont validés avec sept benchmarks. Enfin, nous illustrons l'extensibilité de notre approche en proposant une approche DSE en considérant la sûreté et les plateformes d'exécution multi-coeurs.

Title: A design space exploration approach to jointly optimize security and schedulability in TSP systems

Keywords: time and space partitioning, schedulability, security, multi-objective optimization

Abstract: Modern real-time systems integrate more and more functions. Faced with this complexity, isolation mechanisms are employed so that a failure occurring in one function cannot affect the others. This thesis focuses on TSP (Time and Space Isolation) architectures. They introduce the concept of partition to provide application isolation. Applications can be assigned to partitions according to various objective functions or constraints related to the functions to implement (e.g. safety, performance, security). Some of these objective functions can be conflicting. Thus, improving the security of a system by adding functions dedicated to security (e.g. ciphers) can have a negative impact on its schedulability. In this thesis, we investigate the conflicting aspect between schedulability and security (confidentiality and integrity) in real-time TSP systems. We propose a design space exploration (DSE) based on a multi-objective metaheuristic, which provides trade-offs between schedulability and security for these systems. We propose three DSE algorithms for uniprocessor TSP systems based on the Pareto archived evolutionary Strategy (PAES) metaheuristic. We also propose a method to promote the diversity of the compromises proposed at the end of an exploration. These algorithms are implemented in Cheddar, a schedulability analysis tool extended with security analysis features. The algorithms are validated with seven benchmarks. We also investigate the impact of different security implementations for confidentiality and integrity in TSP systems. Finally, we illustrate the extensibility of our approach by proposing a DSE approach while considering safety and multicore execution platforms.