UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAFAEL ANDRÉAS RAFFI LERM

# A Model-Driven Design-Space Exploration Tool for the HIPAO2 Methodology

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre
2015

**CIP – CATALOGING-IN-PUBLICATION**

Lerm, Rafael Andréas Raffi
A Model-Driven Design-Space Exploration Tool for the HIPAO2 Methodology / Rafael Andréas Raffi Lerm. – 2015.
85 f.

Advisor: Prof. Dr. Carlos Eduardo Pereira

Thesis (Master) – Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015.

1. Design-space exploration. 2. UML. 3. System optimization. 4. Model-driven engineering. 5. Metaheuristic optimization. I. Pereira, Carlos Eduardo, orient. II. Título.

## ACKNOWLEDGEMENTS

First and foremost, I thank God, without whom I would not be. And my family and friends from home, for their support despite the distance. I thank my new found friends as well, even though my time in Porto Alegre was brief.

I would also like to thank my advisor Carlos Eduardo Pereira, for his guidance and support during this course. Furthermore, I thank my colleague Dionisio Doering, for the development of HIPAO2, as well as for the lengthy discussions and many ideas that helped shape this work.

I would also like to mention my fellows from the Embedded Systems Laboratory at UFRGS — not only for being friends and colleagues through all the coursework, but also for making lunch lines much more tolerable.

I would like to thank the Comission and all the staff of the Graduate Program in Computer Science for their help and support, especially for their accommodation and understanding when I required a leave of absence for my internship.

Finally, my thanks to CNPq for the financial support during my course.

*Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head.*

— Donald Knuth

## ABSTRACT

Designers of today's embedded systems are faced with increasing complexity both in the applications being developed and the platforms they run on. The use of complex platforms means that the engineers need to make non-trivial and many times non-intuitive decisions during the design phase. To help developers work with this complexity, model-driven techniques are gaining attention, and in this context, the HIPAO2 model-driven engineering methodology is being developed at UFRGS. Among the problems that designers must solve, the task-to-processor mapping in heterogeneous multiprocessor systems is an NP-complete problem and the design space will quickly become too large to be explored adequately by humans. This work details the extension of the tools that support HIPAO2 to include semiautomatic Design-Space Exploration capabilities for the mapping problem. The proposed tool includes the use of a multiobjective genetic algorithm to make tradeoffs explicit to the designers; it also uses synchronous dataflow analysis algorithms to evaluate potential alternatives with a reasonable computational cost.

**Keywords:** Design-space exploration. UML. System optimization. Model-driven engineering. Metaheuristic optimization.

**Ferramenta de Exploração de Espaço de Projeto Baseada em Modelos para a Metodologia HIPAO2**

## RESUMO

Hoje em dia, desenvolvedores de sistemas embarcados enfrentam uma crescente complexidade de projeto, tanto nas aplicações quanto nas plataformas usadas para executá-las. O uso de plataformas complexas faz com que os engenheiros precisem fazer escolhas não-triviais, e muitas vezes contra-intuitivas durante a fase de projeto. Para permitir que os projetistas gerenciem esta complexidade, o uso de metodologias baseadas em modelos tem atraído atenção, e dentro deste contexto, a metodologia HIPAO2 está sendo desenvolvida dentro da UFRGS. Dentre os problemas que os engenheiros precisam enfrentar, o mapeamento entre tarefas e processadores em sistemas multiprocessados heterogêneos é um problema NP-completo, onde o espaço de projeto rapidamente se torna grande demais para que seja explorado satisfatoriamente de maneira manual. Este trabalho detalha a extensão das ferramentas que suportam a metodologia HIPAO2, de maneira a incluir facilidades de Exploração de Espaço de Projeto semi-automática para a solução deste problema. A ferramenta proposta faz uso de um algoritmo genético multiobjetivo para evidenciar *tradeoffs* existentes no projeto, e algoritmos de análise de aplicações modeladas como *synchronous dataflow* para avaliar possíveis mapeamentos sem um custo computacional proibitivo.

**Palavras-chave:** Exploração de Espaço de Projeto. UML. Otimização de sistemas. Engenharia Baseada em Modelos. Otimização por metaheurísticas.

# LIST OF FIGURES

# LIST OF TABLES

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| **ACO** | Ant Colony Optimization. |
| **AO** | Aspect-Oriented. |
| **API** | Application Programming Interface. |
| **ASIC** | Application-Specific Integrated Circuit. |
| **BDD** | Block Definition Diagram. |
| **CAPS** | Compositional Approximation of Pareto Sets. |
| **CDFG** | Control and Dataflow Graph. |
| **CGP** | Categorical Graph Product. |
| **CMP** | Chip Multiprocessor. |
| **CPACO-MO** | Crowd Population-based Ant Colony Optimization for Multiple Objectives. |
| **CSV** | Comma-Separated Values. |
| **DCT** | Discrete Cosine Transform. |
| **DSE** | Design-Space Exploration. |
| **DSP** | Digital Signal Processor. |
| **EMF** | Eclipse Modeling Framework. |
| **FPGA** | Field-Programmable Gate Array. |
| **GA** | Genetic Algorithm. |
| **GPU** | Graphical Processing Unit. |
| **HIPAO2** | Hardware Image Processing system based on model driven engineering and Aspect Oriented modeling version 2. |
| **IBD** | Internal Block Diagram. |
| **IDE** | Integrated Development Environment. |
| **JSON** | Javascript Object Notation. |
| **KPN** | Kahn Process Network. |
| **M2M** | Model-to-Model. |
| **M2T** | Model-to-Text. |
| **MARTE** | Modeling and Analysis of Real-Time and Embedded Systems. |
| **MDE** | Model-Driven Engineering. |
| **MILP** | Mixed Integer Linear Programming. |

| | |
|---|---|
| **MoC** | Model of Computation. |
| **MOF** | Meta-Object Facility. |
| **MPSoC** | Multi-Processor System-on-Chip. |
| **MTG** | Mapped Task Graph. |
| **NFR** | Non-Functional Requirement. |
| **NoC** | Network-on-Chip. |
| **NSGA-II** | Non-dominated Sorting Genetic Algorithm II. |
| **OMG** | Object Management Group. |
| **OO** | Object-Oriented. |
| **PBD** | Platform-Based Design. |
| **PIM** | Platform Independent Model. |
| **PM** | Platform Model. |
| **PSM** | Platform Specific Model. |
| **PSO** | Particle Swarm Optimization. |
| **PTPO** | Parameterized Timed Partial Order. |
| **PU** | Processing Unit. |
| **ReSPIR** | Response Surface-based Pareto Iterative Refinement. |
| **RSM** | Response Surface Modeling. |
| **SDF3** | Synchronous DataFlow For Free. |
| **SDFG** | Synchronous Dataflow Graph. |
| **SPEA2** | Strength Pareto Evolutionary Algorithm. |
| **SysML** | Systems Modeling Language. |
| **TGFF** | Task Graphs For Free. |
| **UI** | User Interface. |
| **UML** | Unified Modeling Language. |
| **XMI** | XML Metadata Interchange. |
| **XML** | Extensible Markup Language. |

**CONTENTS**

# 1 INTRODUCTION

## 1.1 Motivation

The embedded system market has passed through a great revolution in the past years. These systems now have a ubiquitous presence on everyday lives, whether they are visible or not — be it in consumer products from cellphones to home appliances, or industrial applications that range from PID controllers to complete autonomous inspection systems (QIAN; HARING; CAO, 2009). Also of note are automotive systems, where products that use over 60 different processors are not unheard of (PRETSCHNER et al., 2007).

Advances in processor technology have enabled the development of new applications, allowing the inclusion of increasingly complex features in smart products. Traditionally, these advancements in performance came with architectural and fabrication developments that allowed increasing the single-core performance in a way that was mostly invisible to a programmer. However, the single-core performance can no longer be improved as easily, without greater penalty in other areas such as power consumption — which has also become a leading concern in all types of computing products, from battery-operated embedded systems to large computing warehouses (JERRAYA et al., 2007).

As difficult as it is today to advance in performance without increasing greatly the power consumption (phenomenon also known as the "power wall"), the market has not stopped asking for more and more advanced features. Although designing a new chip with a faster clock rate is no longer an option, engineers still have to create faster and more feature-filled products. The main solution adopted by chip vendors is the offering of more complex hardware systems, especially in the case of the embedded market. Today's processors frequently are multicore and heterogeneous, and very often include non-conventional processing elements such as DSPs and FPGAs.

In other words, today's platforms are still becoming more powerful, but also more complex. This imposes great difficulty in creating designs that use these platforms, at the same time that the engineers have to meet increasing market demands for greater functionality and lower time-to-market.

Beyond functionality, in embedded systems Non-Functional Requirements (NFRs) normally have a much greater importance than in general-purpose systems. These requirements consist in system characteristics that go beyond functionality, i.e., are not concerned with *what* the system performs, but *how* it does so. Namely, the classical, most common NFRs considered in embedded system design are performance, memory usage, power consumption, and the real-time requirements that are present in many projects, especially in the so-called *cyber-physical* systems (LEE, 2008). When programmable hardware such as FPGAs are included in the project, hardware area also becomes a concern.

Great effort has been expended to find better programming paradigms for embedded systems, with special interest in ways to handle the design complexity and bring it to humanlymanageable levels. These proposals have in common the objective of hiding low-level platform details from the developers, allowing them to focus on high-level functional and algorithmic aspects. Hardware abstraction is a desirable feature as it leads to portability, which allows future changes and avoids dependence from specific vendors (the much-dreaded *vendor lock-in*). Concerns about abstraction, code reuse, and maintainability have led the embedded software community to increasingly adopt software engineering practices, which were previously restricted to other, general-purpose software domains.

In this context of design complexity, the use of Model-Driven Engineering (MDE) has been gaining attention. MDE consists in the engineering practice that exploits models for more than just after-the-fact documentation of code written in conventional programming languages, specifying in models all the information necessary to the system's implementation (STAHL; VOELTER, 2006). These models are later used to automatically generate source code in a conventional programming language, using Model-to-Model (M2M) and Model-to-Text (M2T) transformations[1]. This also means that the use of MDE requires specialized tool support in order to be effective (SELIC; MOTUS, 2003).

Just as the standardization of programming languages has been actively sought after, the development of standards for modeling languages is also of great interest to the software development community. The Unified Modeling Language (UML) has emerged as a *de facto* standard for this application. However, it is not feasible that the same set of abstractions contained in standard Unified Modeling Language (UML) be sufficient to fulfill the needs of diverse fields that range from low-power embedded systems to distributed, large-scale corporate systems. To overcome this semantic gap, the development of specific languages to represent the abstractions inherent to each domain has been proposed (STAHL; VOELTER, 2006). In order to do this without losing the benefits of language standardization, the domain-specific modeling languages can be defined using the *metamodeling* framework provided by UML.

In order to accommodate the needs of the image processing domain, the HIPAO2 (DOERING, 2014) methodology is being developed at UFRGS, which stands for Hardware Image Processing system based on model driven engineering and Aspect Oriented modeling version 2. This methodology is an advancement over the previous HIPAO methodology that was presented in Doering et al. (2013). It is based on Object-Oriented (OO) and Aspect-Oriented (AO) analyses, and is supported by a series of tools that implement M2M and M2T transformations, along with metamodels that allow the modeling of image-processing systems.

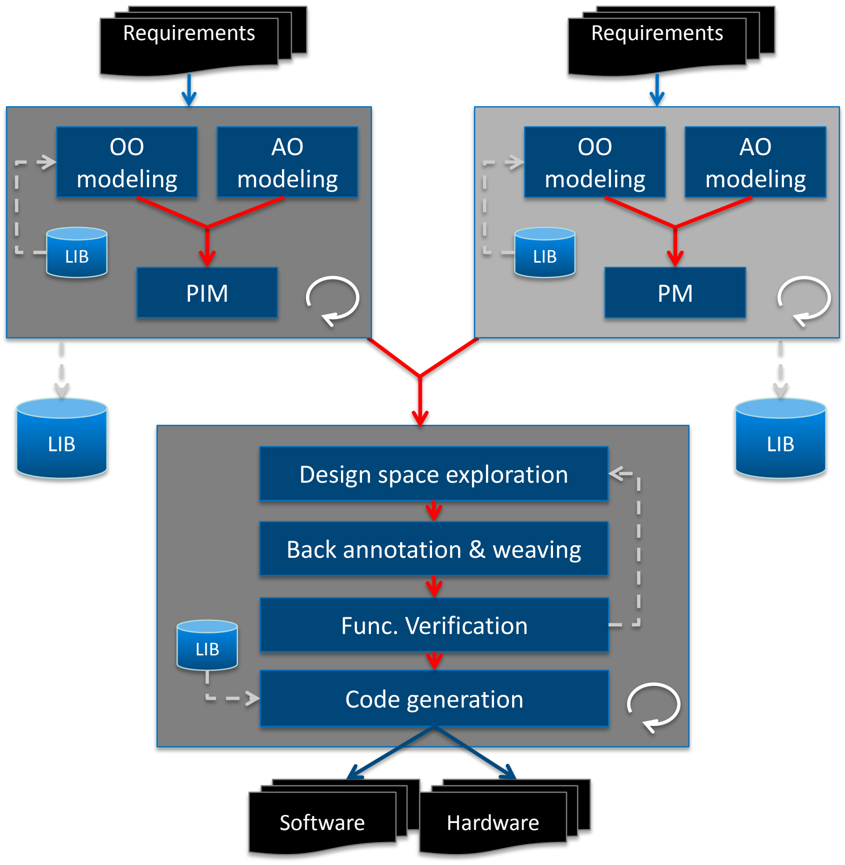
HIPAO2 focuses on supporting the development of Systems-of-Systems, or systems in which the modeled elements in the higher levels are often complete systems in their own right. This methodology should not only enable the construction of new components, but also the use of a library of previously-built ones, favoring component reuse and reducing repeated engineering efforts. The methodology is composed of three phases: *a*) Requirements gathering and modeling; *b*) Platform Independent Model (PIM) and Platform Model (PM) generation, and

[1] MDE concepts are discussed in Section 2.2.2.

*c*) System optimization, validation and code generation.

The workflow proposed for a project using HIPAO2 is shown in Figure 1.1. In the two initial phases represented in that figure, application and platform are developed separately, yielding the Platform Independent Model (PIM) (for the application) and the Platform Model (PM). The beginning of the third step consists of the Design-Space Exploration, where the application is bound to the hardware platform it will run on.

Figure 1.1 – Workflow for the HIPAO2 methodology.



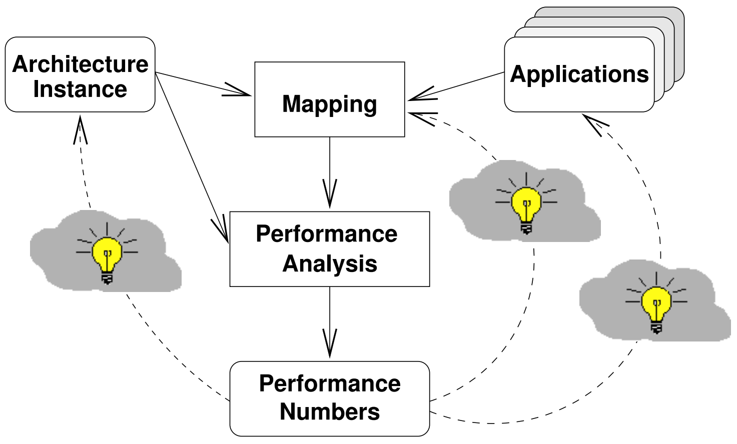
Source: Doering (2014).

Early versions of the methodology (DOERING et al., 2013) had focused on high-performance scientific imaging systems where all the processing was done in programmable hardware (FPGAs), as in Binotto et al. (2013). As a consequence, the transformation from PIM and PM to the Platform Specific Model (PSM) was a trivial process. However, when more complex platforms are considered, such as those that include one or more general-purpose processors, Digital Signal Processors (DSPs), and Graphical Processing Units (GPUs), as well as Field-Programmable Gate Arrays (FPGAs), then the decision of which one to use for each part of the application, known as the *mapping problem*[2], becomes an NP-complete problem (GAREY; JOHNSON, 1990). The development of automated tools to solve this problem presents an opportunity for better end products and decreased engineering effort.

[2] This and other Design-Space Exploration problems are discussed in Chapter 3, p. 29.

Figure 1.1 shows clearly that HIPAO2 follows the Y-chart development approach, defined in Kienhuis et al. (2002). This approach is motivated by both the necessity of abstraction and well-performant results. Figure 1.2 presents an overview of this design approach. It can be seen that platforms and applications are to be defined separately, at a high abstraction level, where there are still many opportunities for changes. These models are called Platform Independent Model and Platform Model, respectively. Later, a specific mapping step exists to bind these two, forming the Platform Specific Model. There are various ways of defining this mapping, forming in many cases a large design space that must be explored. In Figure 1.2, the light bulbs show refinement steps in the development process.

Figure 1.2 – Overview of the Y-chart approach.



Source: Kienhuis et al. (2002).

Modifying iteratively the application and platform as the system evolves is a natural concept represented in the figure, and any methodology that aims to support the development of embedded systems must also support an iterative refinement of these models. However, there is no unique way to implement this application-to-platform mapping, and different mappings may lead to very different results in aspects such as performance, power efficiency, hardware area, and code size. Many times, the mapping decisions that lead to the best-performant implementations are not intuitive, and using simple logic such as "assign each task to the Processing Unit where it is fastest" may fail to find the best solution.

## 1.2 Goals

This work presents the development of a Design-Space Exploration (DSE) tool that was built as one of the components of the last part of the methodology (system optimization), as can be seen in Figure 1.1. Here, we aim to augment HIPAO2's development tools by providing a semi-automatic DSE phase, finding the best possible mappings and revealing tradeoffs to the designer.

The approach is not fully-automated because of the multiobjective nature of the problem — embedded system software design often presents conflicting objectives, such as execution time and power consumption — which makes it almost impossible[3] to point to one specific solution as optimal. Rather than attempting to define a priority weight for each objective in order to find one optimal solution, it is better to support the designer's work by making explicit the different design tradeoffs that are present, and let him choose between the possible solutions. Therefore, human intervention is needed at the end of the exploration process (choosing between the obtained results), while the entire DSE happens automatically.

A secondary goal of this work is to provide techniques that allow evaluation of candidate system implementations as early and as cheaply as possible. This goal arises as a necessary step to fulfill the main ones previously mentioned; it is used to guide the DSE process by providing quantitative information about each possible system implementation. We achieve this by adapting and expanding existing techniques for evaluation in order to make them support the type of platforms and system features that are considered in the HIPAO2 methodology.

## 1.3 Organization of this text

The remaining of this text is organized as follows: Chapter 2 presents relevant theoretical concepts, and Chapter 3 includes a survey of related works in the literature. In Chapter 4 the workflow that was proposed and implemented to solve the DSE problem for HIPAO2 is presented, while in Chapter 5 the application of the proposed framework to case studies as well as the obtained results are discussed. Finally, Chapter 6 draws conclusions and signals possible future work directions.

[3] Impossible in the general case. In specific cases it may be possible to choose a globally optimal solution even while considering all objectives.

# 2 THEORETICAL BACKGROUND

This chapter discusses relevant theoretical concepts that are related to the subjects in this work. It is divided into two main sections: Section 2.1 discusses concepts relevant to optimization algorithms, while Section 2.2 presents subjects related directly to embedded system design.

In no way is this chapter intended to serve as a complete reference on the topics it discusses; the third-party works referenced in each part should be consulted for more in-depth reading.

## 2.1 Optimization concepts

In this work, the development of a tool for automatically solving the Mapping Problem for embedded system design is proposed. By its nature, this is a multiobjective, constrained optimization problem, and for this reason, an explanation of the main concepts from the field of optimization is desirable. Throughout this section, the basic definitions of optimization will be discussed, with special attention given to multiobjective optimization problems and to metaheuristic algorithms, which are employed in this work.

Defining optimization in general is outside the scope of this work. However, it is useful to establish some important concepts of the field, that are used directly in this text, which cannot be done without some definition of what consists optimization and optimization problems.

In a broad (and informal) sense, *optimization* can be understood as "the process of finding the best solution among the possible set for a problem, while obeying to a set of constraints" (EHRGOTT, 2005). This formulation is purposely vague in the definition of "solution", "best", "possible set", and "constraint" — optimization is a vast field, and each of these concepts may vary.

More formally, we will adopt the term *optimization problem* as one that intends to solve a problem of the form

$$\begin{aligned} \text{minimize} \quad & F(x) \\ \text{subject to} \quad & c_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned} \tag{2.1}$$

This definition, as provided by Ehrgott (2005), is intended to be as general as possible, while still being somewhat formal. It allows us to introduce certain key concepts of the field:

$x$ A *solution*, formed by one or more variables, so that $x = (x_1, \dots, x_n)$. The term "solution", although widely used, can be misleading — in practice, it only means that $x$ is a member of the domain of $F(x)$. Solving the optimization problem, however, means finding the *best* solution (or solutions). The domain of $x$ varies with the optimization problem — components $x_i$ can be members of a continuous set, a discrete one, or even values from a finite set.

$F(x)$ The *objective function*, which can be defined, without loss of generality, as $F(x) = (f_1(x), \ldots, f_m(x))$, and represents the metrics that must be minimized.[1] Each individual component is also called an *objective*. Problems that fall into the special and simple case where $m = 1$ are called *single-objective*, in contrast to *multiobjective* problems — the distinction is important because the concept of "minimize" is simple for the single-objective case, whereas defining it can be an issue for multiobjective problems. This work focuses on a multiobjective problem (discussed later on this chapter, on Section 2.1.3).

$c_i(x)$ The *constraint functions*. These functions represent additional rules that acceptable values of $x$ must abide to, effectively restricting its domain.

Additionally, other important concepts are derived from the previous ones:

**Solution space** The domain of $x$.

**Objective space** The image of the solution space, that is, the image $F(x)$.

**Feasible and unfeasible sets** *Feasible* solutions are the solutions for which all constraints are satisfied. Conversely, *unfeasible* solutions are the ones for which at least one of the inequalities in Equation 2.1 does not hold. Infeasible solutions are not acceptable answers of the optimization process, yet they may provide important information for the optimization process (TALBI, 2009).

The details and methods for performing optimization vary greatly according to the details of the given problem — for example, depending on the properties of the objective function, an exact and efficient algorithm may exist that guarantees that the optimal solution is found. Other problems, such as the one of interest in this work, do not even possess analytical formulations for all the objective functions. For these problems where little or no knowledge exists about the form and behavior of the objective functions, it is possible to perform what is called *black box optimization*, often by the use of metaheuristics, which are described in Section 2.1.1.

The remainder of this section first details the definition of metaheuristics, then gives more information about the specific type of metaheuristic that is employed here; afterwards, concepts related to multiobjective optimization are presented.

### 2.1.1 Optimization metaheuristics

For many problems of real-world importance, computing exact solutions is prohibitively expensive. However, in practice, "good enough" solutions can be satisfying in lieu of optimal ones. Heuristic and metaheuristic algorithms are approximate methods that do not guarantee finding optimal solutions, or even how close the discovered solutions are from optimal (as do approximation algorithms) (TALBI, 2009). Still according to the same author,

> Metaheuristic search methods can be defined as upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems. (TALBI, 2009, p. 1)

[1] This formulation assumes a minimization problem. However, this does not result in loss of generality since maximizing a function $f$ is equivalent to minimizing $-f$ (TALBI, 2009).

One major, defining characteristic of metaheuristics is that they to not make assumptions about the objective functions and constraints, unlike exact methods and heuristics. In fact, these functions do not need to have an analytical formulation,[2] and for that reason metaheuristics are said to do *black box optimization* (LUKE, 2013). This means that metaheuristic algorithms work on a separate domain from the problem that is actually being optimized.

Therefore, the first step when using one such method is to define a way of encoding a solution for the problem into the metaheuristic's domain. This encoding's efficiency and completeness can greatly influence the applicability of metaheuristics to a given problem. For a more complete coverage on this subject, see Talbi (2009).

Metaheuristics must balance two conflicting processes in the search for optimal solutions: *diversification* (exploration of previously unvisited areas of the search space) and *intensification* (thorough exploration of promising regions). Different metaheuristics will perform more or less of these activities, with extremes such as random search (diversification) and local search (intensification) (TALBI, 2009). Other methods adopt a middle ground between these, or allow configuration of both aspects of the search, such as Genetic Algorithms (GAs), which allows configuration of the crossover and mutation operators.

Many metaheuristic algorithms are inspired in natural processes, such as Simulated Annealing (KIRKPATRICK; GELATT; VECCHI, 1983), Genetic Algorithms (HOLLAND, 1975), Ant Colony Optimization (DORIGO, 1992), and Particle Swarm Optimization (KENNEDY; EBERHART, 1995). Each of these are based on desirable characteristics of certain processes such as metal cooling in metallurgy for Simulated Annealing, and the evolution of populations for Genetic Algorithms. New kinds of metaheuristics are under constant research, although the extent to which new methods inspired in different processes are useful is a matter of debate (SÖRENSEN, 2013).

An important distinction can be made between metaheuristics that work on varying only one solution at a time, and ones that work on sets of solutions. In the latter methods, which are called *population-based metaheuristics*, one solution may influence how others will be modified, created, or discarded (LUKE, 2013). Many such algorithms have understandably been inspired in biological phenomenons — commonly-known examples are Genetic Algorithms, Particle Swarm Optimizations, and Ant Colony Optimization. The term "population-based" itself comes from the terminology commonly used in Genetic Algorithms, where it means the set of candidate solutions that are being considered.

In this work, we apply a Genetic Algorithm to solve our multiobjective optimization problem. For that reason, the next section presents important concepts and terminology that are used with such methods.

[2] Problems with these characteristics are colloquially referred to as *I Know It When I See It* problems — an allusion to the fact that little or nothing is known about the structure of the problem, or properties of the objective or constraint functions — but given a solution, it is still possible to apply these functions in order to evaluate it (LUKE, 2013).

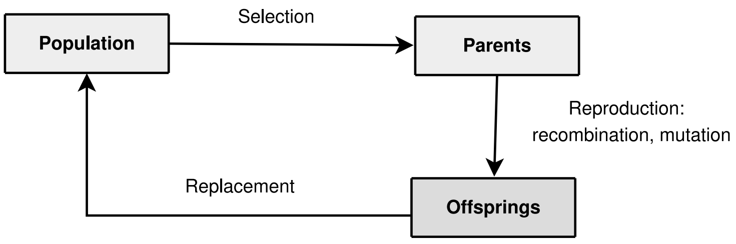
### 2.1.2 Genetic algorithms

Genetic Algorithms (GAs) are a class of population-based metaheuristics that have first been developed in the 1970s, taking its inspiration from the processes of reproduction and intra-species evolution of living beings, and is one of the main members of the class of *Evolutionary Algorithms*. Many of the concepts and terminology related to Genetic Algorithms (GAs) are also borrowed from biology: solutions are commonly referred to as *individuals*, the encoding of solutions is often called *genome* and is composed by several *genes*. Also, producing a new solution from existing ones is called *breeding* or *reproduction*, the quality of a solution is often named *fitness*, and one cycle of fitness evaluation, breeding, and individual selection is called a *generation* (LUKE, 2013).

Unlike the majority of the existing optimization metaheuristics, GAs were first used for solving discrete and combinatorial problems, and later were expanded to continuous problems. They have also been the first place where multiobjective optimization was performed by using only the Pareto-dominance relation (TALBI, 2009), as will be discussed later in this chapter.

In a simple formulation, it can be said that genetic algorithms perform optimization by the use of three distinct operators: *selection*, *mutation*, and *crossover* — each of these inspired by some aspect of nature (MITCHELL, 1996). Figure 2.1 outlines the basic cycle that forms a generation.

Figure 2.1 – A generation in a genetic algorithm.



Source: Talbi (2009).

In the first step, the selection operator is applied to current population, drawing individuals that will be allowed to reproduce. In an analogy to the natural selection phenomenon, it is desirable that individuals with a higher fitness be chosen, but weaker individuals must have *some* possibility of passing their genetic information to the next generation, since they may contain useful material (TALBI, 2009).

Afterwards, the reproduction process takes place, by the application of the two *variation operators*, namely mutation and crossover. In this phase, it is common to refer to *parent* (i.e. the original) and *offspring* solutions (the newly created ones). The mutation operator represents a modification (with a certain probability) on the characteristics (genes) of an individual, being an unary operator. Crossover (or recombination), however, is a binary (and sometimes $n$-ary) operator, and is responsible for creating offspring solutions from parts of its parents.

Closing the cycle comes the *replacement step*, in which an updated version of the population will be created considering the solutions created in the reproduction stage. Originally, GAs used the simplest replacement strategy possible, that is, the newly created solutions will directly replace their parents in the population. Another possible strategy is called *elitism*, in which the new population is chosen from the union of both the parents (the current population) and their offspring, allowing that good solutions be maintained from one generation to the other. Elitism has been shown to improve the convergence speed, and is present not only in most recent variants of GAs, but also of other evolutionary algorithms.

The separation of these operators can be defined in various ways, making GAs a very modular type of heuristic — new research is often made by working on one operator (such as Deb & Jain (2014)). Also, this modularity helps to define new problems and encodings, as creating a new type of encoding requires only to define the mutation and crossover operators, leaving other aspects such as selection and replacement strategies unaffected. This has also made easy the development of multiobjective versions of the metaheuristic, which involves primarily changes to the selection operator.

All the existing variations on the aforementioned operators are outside the scope of this work. For a more in-depth discussion of these subjects (and other important ones, such as different encoding schemes, which are closely related to the variation operators), see Luke (2013) and Talbi (2009). For a discussion on the variation operators used in this work, see Section 4.4.2.

### 2.1.3 Multiobjective optimization

Many real-world optimization problems do not deal with the optimization of only one criterion; they must optimize several metrics, which often come into conflict with each other.[3] Often, a gain in one objective comes at the cost of a loss in another one (and possibly more than one) (PAPPALARDO, 2008).

Example multiobjective problems occur in engineering (such as engine design), where it may be necessary to increase efficiency and reliability, while also improving cost and reducing noise. Also, when planning a route for the transportation of hazardous materials, objectives considered would involve the distance, risk for population, and running costs (CARAMIA; DELL'OLMO, 2008). Yet another case of multiobjective problem is found in the domain studied in this work, embedded system design — where engineers must optimize metrics such as performance, power consumption, and memory usage.

Single-objective optimization problems can be defined as finding the solution $x$ that minimizes the objective function, or

$$\min f(x), \quad x \in S$$

[3] As is common in the literature (EHRGOTT, 2005; TALBI, 2009), this work uses the terms *objective* and *criteria* interchangeably, as it does with *multiobjective* and *multicriteria*.

where $f$ is the (scalar) objective function, and $S$ is the set of possible (feasible) solutions. This is a very simple definition, because in this case the *min* relation is trivially defined — i.e., the usual ordering relation of $\mathbb{R}$. For the multiobjective case, however, there is not a single one objective function, but a series of functions, so that the problem becomes:

$$\text{“min”} F(x), \quad x \in S, \quad F(x) = (f_1(x),\, f_2(x),\, \dots\,,\, f_2(x))$$

However, the ordering of solutions implied by the “min” relation is not trivially defined (EHRGOTT, 2005). On the other hand, a *partial* ordering can be defined in the form of *pareto dominance*, as will be explained in the sequence.

#### *2.1.3.1 Pareto optimality*

The concept of Pareto optimality takes its name from the Italian economist Vilfredo Pareto (1848–1923), whose work reasoned about economic equilibrium and welfare. He came to define that a society is in a point of maximum ophelimity[4] when no individual can have its ophelimity increased without diminishing that of another (LUC, 2008).

The problem studied by Pareto is considered an early work on multiobjective optimization, in which the ophelimity of each individual represents an objective to be optimized. Extending Pareto's concept to general multiobjective problem domains, it is possible to define a relation operator that can compare two solutions. This relation is known as the (Pareto) dominance relation (TALBI, 2009):

**Definition 1** (Pareto dominance)**.** *An objective vector $u = (u_1, \dots, u_n)$* dominates *another objective vector $v = (v_1, \dots, v_n)$ (noted by $u \prec v$) if and only if no component of $u$ is larger than its correspondent component in $v$, and at least one component of $u$ is smaller than its correspondent in $v$, that is,*

$$\forall i \in \{1, \dots, n\} : u_i \leq v_i \quad \wedge \quad \exists i \in \{1, \dots, n\} : u_i < v_i \tag{2.2}$$

However, the most commonly used concept in the multiobjective optimization literature is the one of *Pareto optimality*. In this terminology, a *Pareto-optimal* solution is one where it is impossible to improve the value of one objective without degrading another (TALBI, 2009), that is,

**Definition 2** (Pareto optimality)**.** *A solution $u$ is Pareto-optimal if it is not dominated by any other in the feasible set, that is,*

$$\forall v \in S, \quad F(v) \nprec F(u) \tag{2.3}$$

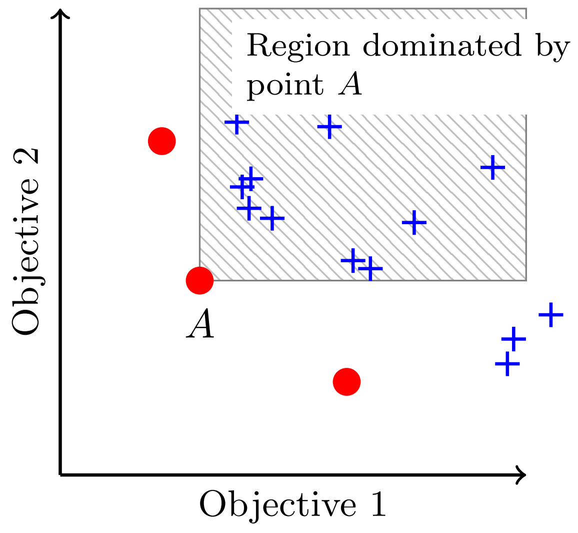
In general, single-objective optimization problems will present only one Pareto-optimal solution. For multiobjective problems, because Pareto dominance is only a partial ordering, it is possible (and common) that there is a *set* of non-dominated solutions, called the *Pareto-optimal set*. The image of these solutions in the objective space is called *Pareto front* (TALBI, 2009).

[4] A measure introduced by Pareto to represent the “economic satisfaction” of an individual.

**Definition 3** (Pareto-optimal set)**.** *For a given multiobjective optimization problem, the Pareto-optimal set is defined as* $\mathscr{P} = \{x \in S | \nexists\, x' \in S,\ F(x') \prec F(x)\}$.

Figure 2.2 shows a visual example of the above concepts by showing the objective space of a hypothetical two-objective problem. In the image, points represent the objective value of solutions, and the type of marker indicates a solution's type: dominated ones are marked by a cross, and solutions in the Pareto front (that is, non-dominated solutions) are marked by a circle. The figure also shows an example of the partial ordering provided by Pareto domination: points in the highlighted region are *dominated* by point $A$. However, between the solutions in the Pareto front no ordering is possible — they all have the same rank in the ordering provided by the dominance relation.

Figure 2.2 – Example of Pareto front.



Source: Author (2015).

### 2.1.3.2 Solving multiobjective optimization problems

In single-objective problems, the total ordering provided by the fact that the objective space is $\mathbb{R}$ leaves no doubt when ranking solutions. This does not happen with multiobjective problems, where solutions in the Pareto front cannot be ranked without additional information.

One very common approach to the problem of ranking solutions is to provide an additional function $g : \mathbb{R}^n \to \mathbb{R}$ that will effectively reduce the original multiobjective problem to a simpler, single-objective one:

$$\min g \circ F(x), \quad x \in S \tag{2.4}$$

Techniques of this type are called *scalarization*. Luc (2008) provides a theoretical reasoning on the properties $g$ must have so that solutions to the above single-objective problem are indeed Pareto-optimal solutions of the original problem.

It also surveys several common scalarization techniques, among which the most simple and widely-used one is the *weighted sum*, where $g(\vec{x}) = \sum_{i=1}^{n} w_i \cdot x_i$. This is a case where the aforementioned "additional information" for the ranking is evident: the decision maker encodes the importance of each objective into the set of weights. Other possible formulations for $g$ (or *aggregation function*) will also include the encoding of preferences of some sort into it — even for parameterless formulations of $g$ such as the minimum of maximum objectives (EHRGOTT, 2005), the formulation of the aggregation function itself contains the preference information.

It is also possible to define a comparison operator over the objective space, such as a lexicographical comparison. Even tough this case is not considered a case of scalarization (since it modifies the comparison operator instead of defining a scalar objective function), it still reduces the optimization to a single-objective one (EHRGOTT, 2005). In this case, the decision maker's preference is expressed into the comparison order of the objectives. For a comprehensive classification of approaches for dealing with multiple objectives, see Talbi (2009).

In the previously mentioned techniques, the decision maker includes his preferences *before* the optimization process runs. However, it is possible to devise an approach which optimizes the problem without any additional information, using only the Pareto-dominance relation to rank solutions. Such approaches have been called *domination-based* in the literature, and do not require converting the optimization problem to a single-objective one. Because Pareto-dominance only provides a partial order, the output of these algorithms is a set of Pareto-optimal solutions — the decision maker must make his choice among the Pareto-optimal solutions after the optimization process.

Since a domination-based technique must inherently work with sets of solutions, it is natural that they be implemented in population-based metaheuristics. Indeed, this method has first appeared in Genetic Algorithms (GAs), and today a large variety of multiobjective GAs exist (TALBI, 2009). The same approach to handling multiple objectives has been used with other types of population-based metaheuristics, such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

## 2.2 Embedded system design concepts

HIPAO2 is a Model-Driven Engineering methodology that is intended to enable the development of embedded image processing systems, making use of dataflow computation models. This section discusses fundamental concepts related to this field.

Namely, the next part presents basic understandings about dataflow Models of Computation, while Section 2.2.2 discusses general concepts related to Model-Driven Engineering (MDE) that are necessary for the understanding of this work.

### 2.2.1 Dataflow Models of Computation

Various domains such as DSP and multimedia applications involve the processing of continuous, infinite *streams* of data, with programs that normally run for long periods of time. For these applications, it makes more sense to adopt a *dataflow* Model of Computation (MoC) rather than more common von Neumann programs that must terminate (CARKCI, 2014).

In contrast to von Neumann formalisms where data is considered to be located statically in a central memory element, in dataflow models the computation is typically divided into small, separate actors with little interaction between each other. Data is transferred from the output of an actor to the inputs of others, forming a data dependency graph. This brings the second great advantage for dataflow Models of Computation (MoCs), the fact that they are naturally parallel (CARKCI, 2014). These formalisms can help make explicit the parallelisms that naturally exist in data processing algorithms (LEE; MESSERSCHMITT, 1987).

Several dataflow MoCs have been defined in the literature, such as Kahn Process Networks (KAHN, 1974), Synchronous Dataflow Graphs (LEE; MESSERSCHMITT, 1987), and Parameterized Timed Partial Order (BASTEN et al., 2010). In this work, we have adopted the Synchronous Dataflow Graph (SDFG) model to express our applications, due to their suitability to the image processing domain and to the availability of good analysis algorithms. Therefore, in the remainder of this section we will present the main concepts pertaining to this formalism.

#### *2.2.1.1 Synchronous Dataflow Graphs (SDFGs)*

Various DSP and multimedia applications are divided into a set of suboperations (or *tasks*) that are to be performed in a defined order, while data is passed between these tasks. This is the type of applications that the SDFG MoC represents very naturally (GHAMARIAN et al., 2006). Tasks are nodes in the dataflow graph, and edges (or *channels*) represent the flow of information between tasks (also called *actors*). Information in channels is represented by discrete amounts of data known as *tokens*.

Actors execute in events called *firings* — an actor will be enabled to fire every time it has the necessary amount of tokens in its input, sending tokens to its outputs. A defining characteristic of SDFGs is that every time an actor fires, it will consume a fixed amount of tokens from its inputs and produce a fixed amount of tokens in its outputs. These amounts are called the *rates* of an actor (BONFIETTI et al., 2009).

In the original formulation for the SDFG formalism (LEE; MESSERSCHMITT, 1987), the time necessary for the execution of an actor is neglected (it is considered to be unitary and actor firings are atomic). However, a simple extension to the formalism, as used in e.g. Sriram & Bhattacharyya (2000), Ghamarian et al. (2006), assigns a fixed execution time for each actor, allowing time analysis of SDFGs, including the analysis of the throughput of the application, which is used in this work. Several research works dealing with the throughput analysis of SDFGs have been published (KARP, 1978; YOUNG; TARJAN; ORLIN, 1991; DASDAN, 2004); the availability of such literature was one of the reasons for the adoption of this MoC in the present work. More concretely, we have used the throughput analysis algorithm presented in Ghamarian et al. (2006) and implemented in the SDF3 library (STUIJK; GEILEN; BASTEN, 2006), due to its shorter execution times when compared to other algorithms.

### 2.2.2 Model-Driven Engineering

Electronic and software systems, especially those in the embedded domain, share similarities with traditional engineering — both present tradeoffs that must balance desired functionality against other non-functional requirements, such as the materials (or hardware) available, the anticipated workload, available tools, etc. In an analogy with the use of models in traditional engineering fields, the use of models has been proposed in software development domains, allowing to understand complex relationships between entities, show relevant aspects while hiding unimportant details, and effectively communicate design intent (SELIC; MOTUS, 2003).

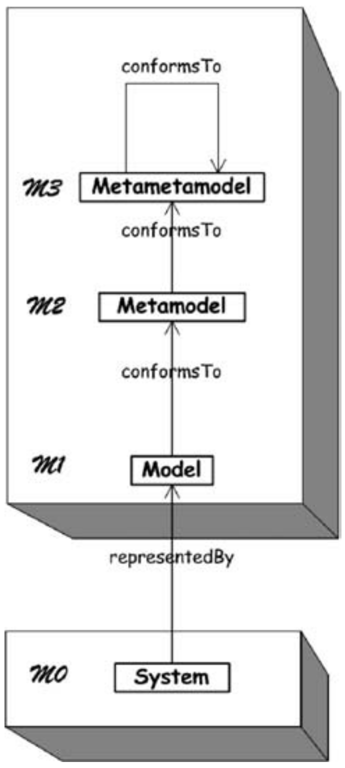
Model-Driven Engineering (MDE) is an approach to (originally) software development in which models are the main artifacts of development instead of code. Different models can be used to represent different aspects of the system and can be automatically manipulated, adding or removing information, or generating other types of artifacts, such as documentation or source code. These transformations applied to models are normally classified between Model-to-Model (M2M) and Model-to-Text (M2T) transformations (BOUQUET et al., 2012).

Models are built conforming to a standard language, in a similar way that code is written in a well-defined programming language. In MDE, this specification is itself a model, and is called a *metamodel*. Analogous to source code and programming languages, models can be checked to conform to a metamodel by automatic means (GAŠEVIC; DJURIC; DEVEDŽIC, 2009). Metamodels themselves are models conforming to a model, often called *meta-metamodels* — this recursion can theoretically extend indefinitely. However, in practice it is possible to define a base model as the beginning of the conformance tree, and it is even possible to define this base model as conforming to itself (BÉZIVIN, 2005), as is shown in Figure 2.3.

For models to be useful, they must be clear and facilitate communication between different designers and stakeholders; they must also be easily interpreted by the various tools used to support the development process. Interoperability is paramount to make this possible, and therefore much effort has been made by the community to create relevant standards for MDE. The competing standards in the early 1990s have led to the creation of the Unified Modeling Language (OMG, 2011b) for software development, and the emergence of the Object Management Group (OMG) organization to oversee the development of related standards.

The necessity of building a modeling infrastructure for different uses has led to the development of a generic modeling infrastructure, which includes the Meta-Object Facility (MOF) (in the M3 level according to Figure 2.3) and *profile* mechanism that can be used for extending Unified Modeling Language (UML). UML itself is an M2 metamodel, defined in terms of

Figure 2.3 – Model hierarchy in OMG.



Source: Bézivin (2005).

MOF. It also defines a standard for model serialization, in order to promote the aforementioned interoperability, which is called XML Metadata Interchange (XMI).

For the embedded systems domain, relevant profiles, also published as standards by the OMG, are MARTE and SysML. The Modeling and Analysis of Real-Time and Embedded Systems (MARTE) profile (OMG, 2011a) defines UML constructs specifically for the needs of the embedded systems domain, such as stereotypes for specifying processors, buses, etc, with relevant attributes. The Systems Modeling Language (SysML) profile (OMG, 2012) defines a language for what are called *Model-Based Systems Engineering* applications, effectively extending UML to less software-centric domains.

# 3 RELATED WORK

Design-Space Exploration is a broad subject and research field, when considered in its broader definition. Therefore, research has produced a large quantity of works that cannot all be compared due to different goals and achievements. Most importantly, different works have concentrated on various aspects of the Design Space, given the range of possible project decisions — hence, it is necessary to sort literature on the subject in order to better understand the state of the art.

In an attempt to categorize the different design decisions, or problems, authors have created different classifications, and even so, these can be limited to one project domain. For example, an early work by Dick & Jha (1997), written in the context of hardware-software codesign, divides design decisions into three categories:

**Allocation** To determine the quantity of each type of Processing Unit (PU) and communication link to use.

**Assignment** To select a PU to execute each task upon. Choose a link to use for each communication event.

**Scheduling** To determine the time at which each task and communication event occurs.

This classification is specific to one domain and cannot be applied to all works on Design-Space Exploration (DSE). For example, it fails to classify works such as Mattos (2007), which proposes to choose between alternate software implementations automatically.

The work by Saxena & Karsai (2011), which aims to define a generic representation to support multiple design decisions, has classified them into six categories:

**A. Configuration Problem** Assignment of values to certain parameters of the system. The term *system* here is used in the broader sense possible — these parameters can pertain both to the platform and to the application. Examples of this kind of DSE problem can be found in Palermo, Silvano & Zaccaria (2009), Oliveira (2013), Yang et al. (2009).

**B. Mapping Problem** Many current design frameworks are based on the *Y-Chart* design philosophy (KIENHUIS et al., 2002), in which the platform and application are modeled separately, with an explicit mapping step. With the market trend toward using heterogeneous platforms, this mapping becomes an NP-hard problem (GAREY; JOHNSON, 1990). Many works have been presented to address this problem in an automatic fashion, and will be alluded to throughout this chapter.

**C. Construction Problem** Consists on the construction of a system from a set of preexisting objects. This is a recurring problem in hardware synthesis, in which these objects can be different models of processor, buses, etc. The construction problem has also been studied in the context of software development processes, handling the selection between alternative implementations of software components (MATTOS, 2007).

**D. Placement Problem** The arrangement of objects while respecting geometric constraints (e.g., objects must stay inside boundaries, and cannot overlap). This kind of problem frequently shows up in VLSI design, many times associated with the Routing Problem.

**E. Routing Problem** This problem consists in connecting nodes in a geographic area in an optimal manner. Often the capacity of links must be respected, and the goal may be to find the shortest paths, or the shortest delay. This problem appears, for example, in VLSI synthesis and in the project of Networks-on-Chip (NoCs).

**F. Scheduling Problem** This problem is characterized as the definition of starting times to a set of tasks, while respecting precedence constraints between them and often guaranteeing that the execution time will remain under a maximum allowed value (deadline).

Due to its more comprehensive nature, we will use the above classification throughout this text when referring to the different types of design decisions. Given the very different nature of these problems, it is important to note which one(s) each work in the literature tries to solve, and special attention will be given to this aspect.

The previous works on automated DSE themselves can be classified from other, different points of view, in addition to the design decisions they attempt to support. For example, every framework that attempts to optimize a project must work in some sort of model — which can exist at any abstraction level possible. This has the potential to affect the kind of optimizations that are possible, the accuracy of the evaluated solutions, and the information available for the process. Therefore, the way DSE frameworks *model* the systems they optimize is a valuable information when studying them.

Also, given the model of the system to be optimized, there are many different ways to perform the actual exploration process. This is another very important aspect that should also be taken into account when analyzing the different works that were found in the literature. In the following sections, these existing works will be studied according to the aforementioned criteria.

## 3.1 Modeling techniques

Whatever the technique used for design space exploration, it is necessary to represent in some way the system and the design decisions in question. In the present section, this is what we refer to *model*, which may or may not (depending on the work in question) be related to Model-Driven Engineering (MDE) aspects.

Furthermore, the representation used is not necessarily unique throughout one proposal. In the most immediate example, the exploration framework may receive the system model in one format, but represent it internally in a different manner. For this reason, we make a distinction between the *external* and *internal* models, or representations, of the design. The *external* representation defines the interface with the outside world, i.e., the manner by which the designers and/or development toolchain communicate with the DSE framework. The *internal* representation defines the way the problems and problem space are represented during the exploration process, and is often closely linked to the actual exploration algorithm that is used.

For the external model, various works employ some form of source code for the definition of the platform; either by defining a domain-specific language, or by parsing actual, executable source code. The use of programming language code eases use of the framework by the designer, by using familiar tools, which helps adoption since it normally meets a lower resistance. On the other hand, this happens at the cost of working at a lower abstraction level, where the number of decisions that are available is reduced.

In addition to the technique used to define the model, many of the works model the applications according to a specific Model of Computation (MoC). This happens because of the bottleneck presented by the evaluation of solutions — full-system synthesis and simulation is normally prohibitively expensive for use in DSE. In contrast, modeling the application according to a specific MoC enables the use of parametric analysis algorithms that are typically orders of magnitude faster, as exemplified by Piscitelli & Pimentel (2012). The choice of the actual Model of Computation is normally influenced by the type of application that is to be modeled, and by the analysis algorithms that are available.

The DSE method in Pimentel, Erbas & Polstra (2006) adopts the Kahn Process Network (KPN) MoC for the application model, which is well applicable to the dataflow-oriented systems it considers, which lie in the media processing domain. It has formal properties (such as determinism in front of different schedulings) which can be advantageous; however it fails to model time-dependent behavior, e.g. systems that employ interrupts. The KPN model is constructed directly from sequential C/C++ or Matlab source code. This has the potential to lower the designer's resistance to adoption of the tools (as it is less disruptive to a traditional workflow) — however, the DSE must be done at a later stage of development. Furthermore, while this method of specification can well be used to choose between different PUs that present source-code compatibility, it will fail when the designers must consider, for example, general-purpose processors, DSPs and FPGAs.

Trčka et al. (2011) uses a model-driven approach to define applications and platforms, in the Octopus framework. However, instead of being based on standards such as UML/MARTE, this work uses a custom-built framework, the Design-Space Intermediate Representation (DSEIR), defined in Basten et al. (2010). This modeling framework defines applications according to the Parameterized Timed Partial Order (PTPO) formalism, and models platforms as providers of services that are consumed by the application. The PTPO is a very expressive formalism, and provides enough information so that models can be converted to other MoCs. This capability is extensively used in the evaluation process, using preexisting tools that work on other formalisms, such as Colored Petri Nets, Synchronous Dataflow Graph (SDFG), and Timed Automata.

Oliveira (2013) also proposes a MDE methodology supporting DSE, however it is based upon standard infrastructure such as Unified Modeling Language (UML)/MARTE and associated standards. Platforms are also modeled as providers of services which are consumed by applications, with the information on the consumption of these services being extracted directly from the UML models. One of the main goals of this work is to provide a generic framework for DSE, not being fixed to any single design decision. This is done by including the modeling of the decisions into the framework itself, with the designer being required to describe the design decisions that should be explored, in terms of mappings between graphs. The author then shows a possible implementation for the mapping and configuration problems. For the internal representation, a Categorical Graph Product (CGP) is used to describe the mapping between different graphs that where defined by the designer.

One of the disadvantages of defining a specific Model of Computation for the modeling of an application are the limitations that these impose. For example, most dataflow MoCs, such as KPN and SDFG fail to represent applications that are based on discrete events such as interrupts. Conversely, models based exclusively on discrete events (e.g. synchronous-reactive MoCs) cannot adequately model a more dataflow-oriented system. Also, a MoC can be too simplistic in its assumptions (e.g., KPN assumes infinite-sized queues), or may be too limited in what information it represents. In this regard, new MoCs have been built to address this issue, such as Resource-Aware SDFGs (YANG et al., 2010) and Scenario-Aware SDFGs (BHATTACHARYYA; DEPRETTERE; THEELEN, 2013).

## 3.2 Design-Space Exploration techniques

Optimizing the aforementioned design problems is not a trivial task, due to the large search spaces and to the multiobjective nature of the tasks, which often introduce complicated tradeoffs. Therefore, it is expected that the algorithms used for these also come to be very complex.

According to Talbi (2009), optimization algorithms can be classified as *exact* and *approximate*. The last ones comprise, among other classifications, the *specific heuristic* and *metaheuristic* families, which are also of interest in the works studied here.

As their name states, exact methods find the optimal solutions to a given problem, and guarantee their optimality. They are problem-specific and struggle with large search spaces, specially in NP-complete problems. While their scalability discourages the use of this kind of method, some examples exist, such as the use of Mixed Integer Linear Programming (MILP) (PRAKASH; PARKER, 1991) and SAT-solvers (SAXENA; KARSAI, 2011). Also, Erbas, Cerav-Erbas & Pimentel (2006) has used the lexicographic Chebyshev method in order to find the exact, full Pareto front to support his comparison of genetic algorithms.

Approximate methods attempt to solve large problems in a reasonable time. However, they also present a tradeoff, in the fact that there is no guarantee of global optimality. Following the classification used by Talbi (2009), *specific heuristics* and *metaheuristics* are all approximate methods.

Specific heuristics are algorithms designed to optimize one specific problem or instance, while metaheuristics are general-purpose and can be applied to almost any kind of problem.

They work in a higher level of abstraction, separate from the problem being optimized.

Several examples can be found in the literature as to the use of specific heuristics for DSE. For one, Palermo, Silvano & Zaccaria (2009) presents Response Surface-based Pareto Iterative Refinement (ReSPIR), a heuristic designed for the configuration problem of hardware parameters in Multi-Processor Systems-on-Chip (MPSoCs) and Chip Multiprocessors (CMPs) systems. This heuristic employs a Response Surface Modeling (RSM) technique in order to reduce the number of solution evaluations that must be made.

One heuristic tailored to solve the construction problem is CAPS, for Compositional Approximation of Pareto Sets (LIU et al., 2011), combining predefined hardware blocks while choosing between the possible synthesis results of each one. Some characteristics of this method resemble ones of metaheuristics, namely the abstraction of the solution evaluator as a black-box component; however, it is still single purpose and cannot scale to more than two optimization objectives (and in fact, that is not its goal).

In order to optimize the buffer sizes in an application modeled as a *Resource-Aware SDFG* (which can be classified as an instance of the configuration problem), Yang et al. (2010) presents a heuristic called *bottleneck-driven DSE*, using information from the evaluator in order to guide the exploration process — thus giving an example of how a specific heuristic can take advantage of additional, problem-specific information in order to obtain a faster or better exploration process.

In an alternative to heuristic methods, the use of metaheuristics has attracted much interest in the field of automated DSE. These work outside the problem domain, and are largely targeted to search large problem spaces in reasonable time. Also, metaheuristics are considered to be a good strategy where the objective functions are very time-consuming (TALBI, 2009), which is often the case in the design of embedded systems.

Given the nature of the DSE problem studied in this work, we are concerned primarily with previous works that have used multiobjective versions of these algorithms. However, there are some other works that have used regular, single-objective Genetic Algorithms (GAs), such as Chockalingam & Arunkumar (1995). Also, Abdelhalim & Habib (2009) uses a single-objective version of a PSO, and Miramond & Delosme (2005) employs a single-objective algorithm based on simulated annealing. In general, these works combine the multiple objectives by mean of a weighted sum. Although simple and straightforward, the calculation of adequate weights can be very difficult and error-prone. Furthermore, the exploration process will not provide some of its most useful information, which is the enumeration of the tradeoffs existent in the design, which can be made evident in the Pareto Front.

One of the most studied and used type of metaheuristic across various fields are Genetic Algorithms (GAs). Dick & Jha (1997) presents the use of MOGAC, a multiobjective genetic algorithm, to the simultaneous solution of the mapping and construction problem. The concept of solution clustering is exploited in this work in order to prevent the crossover operator from generating structurally invalid solutions. In this work, clusters are groups of solutions that are equal with respect to some of the design decisions. Separate crossover operators are used inside clusters and between clusters, in what could be viewed as a hierarchical application of a genetic algorithm. Still, the interdependence between the design decisions explored can cause many invalid solutions to arise, making this methodology rely heavily on constraint handling and repair mechanisms. Also, the use of highly specialized modification operators that must be used alternately at specific moments in the exploration makes it questionable whether the algorithm is in fact a metaheuristic or a heuristic inspired by genetic algorithm concepts.

Much work is currently happening on the optimization community regarding multiobjective metaheuristics, creating several competing algorithms. Of these, Strength Pareto Evolutionary Algorithm (SPEA2) (ZITZLER; LAUMANNS; THIELE, 2001) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) (DEB et al., 2002) are strong competitors and have been considered by several works on DSE. Pimentel, Erbas & Polstra (2006) uses the SPEA2 algorithm in their framework, and Piscitelli & Pimentel (2012), Silvano et al. (2011), Jahr et al. (2011) all use NSGA-II. However, Erbas, Cerav-Erbas & Pimentel (2006) performs a side-by-side comparison between the two, and arrives to the conclusion that the higher computational cost of SPEA2 does not render better results. Another similar work that has compared the two algorithms was done by Calborean et al. (2011).

Although GAs are arguably one of the oldest and more studied type of metaheuristic, much interest is given to other, newly established paradigms. One that has received much attention is Particle Swarm Optimization (PSO), as in Abdelhalim & Habib (2009), Calborean et al. (2011). Also, a multiobjective variant of Ant Colony Optimization has been used by Oliveira (2013).

Oliveira shows the use of a metaheuristic called CPACO-MO,[1] for DSE purposes. This a multiobjective algorithm of the Ant Colony Optimization type, using additional concepts (such as population and crowding) that are commonly found on GAs. Furthermore, this works builds a framework for solving (exploring) multiple design problems at once. This is done by placing the responsibility of defining the DSE problem on the designer (although a set of ready-made definitions are presented). In order to achieve this generality, a special encoding is also used, the Categorical Graph Product (CGP), which can represent mappings between elements of an arbitrary number of graphs at the same time. The use of this encoding, therefore, allows exploring different design decisions at the same time, at the cost of only being able to represent decisions that can be defined as mappings between graphs. The author has shown such definitions for the mapping and configuration problems. Although this work represents a relevant evolution towards exploration of multiple design decisions at the same time, this generality comes at the cost of burdening the constraint evaluation system, as the encoding cannot enforce some simple consistency rules that other encodings naturally guarantee (such as every task being mapped to a valid processor).

[1] CPACO-MO — Crowd Population-based Ant Colony Optimization for Multiple Objectives (ANGUS, 2007).

## 3.3 Discussion

In this section a state-of-the-art analysis in the field of automated DSE for embedded systems was developed. Related works were analyzed in terms of several characteristics, such as the design decisions they aim to support and the methods used in Design-Space Exploration.

From this survey, two trends relevant to this work can be summarized. First, the vast majority of previous contributions use some form of parametric estimation of solution characteristics, since more accurate schemes are prohibitively expensive (Calborean et al. (2011) is a lone exception to this rule). For the same reasons, we have followed this trend, as will be discussed in Chapter 4.

The literature also leans heavily towards the use of metaheuristics, motivated by the large design space of most DSE problems. Some works do use *ad-hoc* heuristics, but exact methods have definitely fallen out of interest. Again, the situation contemplated in our work is similar to the previous ones; having the same problem of a large design space, our work also uses a metaheuristic algorithm to perform its search.

Genetic Algorithms are by far the most widely used metaheuristic in the works that were analyzed, and have comprehensive literature comparing their many variants. Some works such as Oliveira (2013) go into the use of different types of metaheuristics such as Ant Colony Optimization (ACO), but these are a minority. Although the field of optimization has recently seen a lot of interest in the development of new paradigms of metaheuristic algorithms, some papers still have a critical view in relation to the use of such new proposals (SÖRENSEN, 2013). The in-depth study of the metaheuristic algorithm was considered out of scope for this text; therefore we have decided to employ NSGA-II (DEB et al., 2002), a multiobjective GA with a good amount of literature comparing it favourably to other alternatives.

The use of MDE in the context of Design-Space Exploration is still in its early stages; not many works propose performing DSE in the context of an MDE methodology. Some notable works that do are Basten et al. (2010) and Oliveira (2013). This work is inserted within this context; important differences are the focus on the development of Systems-of-Systems in the HIPAO2 methodology, and the proposed use of Systems Modeling Language (SysML) to model these systems.

# 4 MODEL-DRIVEN DESIGN EXPLORATION TOOL

HIPAO2's focus on systems-of-systems means that, in general, the blocks that will be modeled represent fairly large components of the design, often being considered systems in themselves. This is the case for entire processors in the case of the Platform Model, and complete image algorithms in the case of the Platform Independent Model. The act of composing the system by combining high-level blocks, each with a well-defined interface, opens possibilities for reuse and, when necessary, outsourcing of the production of some of these blocks. When considering the modern, heterogeneous platforms that are increasingly becoming available today, it is reasonable to suppose that a same block may have different implementations for distinct target processors, all presenting the same functionality — for example, a block may be available for general-purpose processors, Digital Signal Processors (DSPs), and Field-Programmable Gate Arrays (FPGAs). Therefore, one of the design decisions, considering a heterogeneous multiprocessor platform, is to choose which Processing Unit (PU), between the available ones, to run each task on. The problem of choosing the best PU for each task is known as the *Mapping Problem*, as mentioned in Section 1.1.

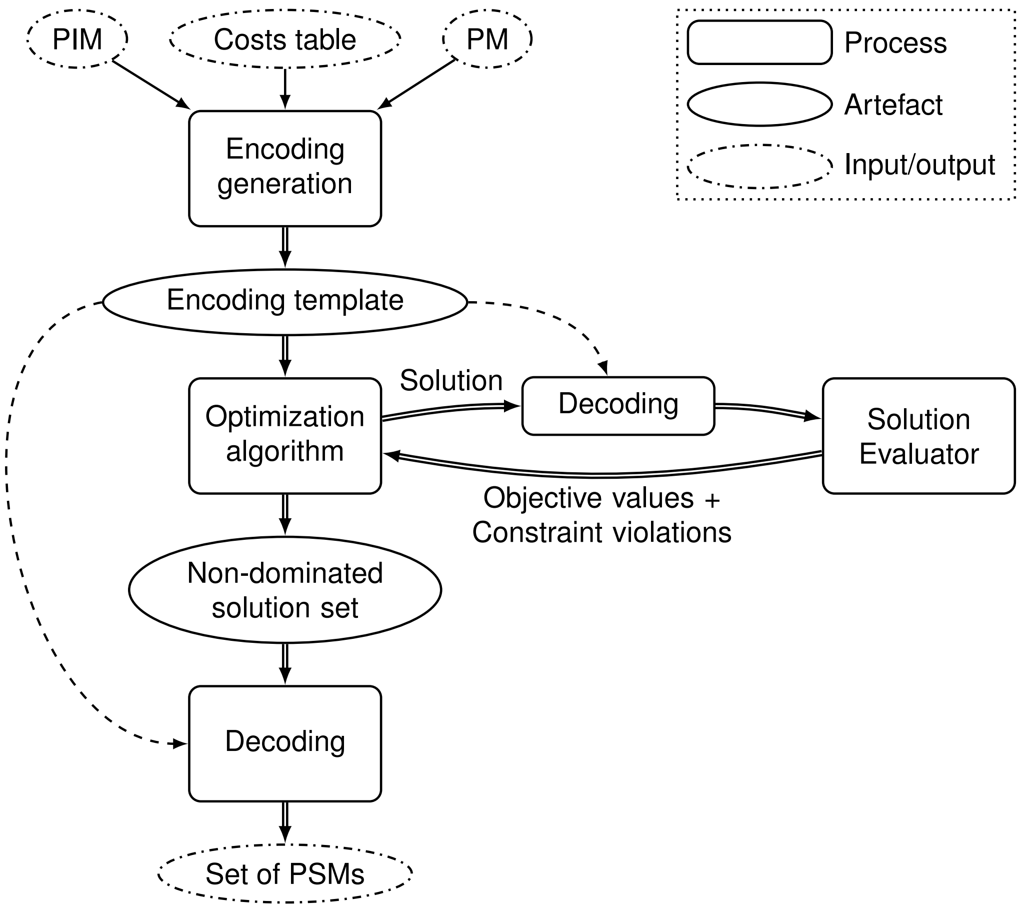
The existence of this choice increases the design space and the engineering effort required from the engineers, but it also presents a possibility for automation. This work proposes to include an automated Design-Space Exploration (DSE) step in HIPAO2's tools, to find the best (even if optimality is not guaranteed) task-to-platform mappings for a given system. That problem increases combinatorially with the amount of tasks and PUs in the system, making the design space very large for even moderate design sizes. Furthermore, the task-to-platform mapping problem has been proven to be an NP-hard problem (GAREY; JOHNSON, 1990), which leads away from the idea of developing a complete, optimal algorithm, and points towards the use of heuristics (or metaheuristics) to solve this problem.

Also of note is the multiobjective nature of this problem. Embedded system designers normally have to consider several, usually conflicting, design goals, making many tradeoffs, such as favoring processing speed over power consumption or vice-versa. Often, there are also constraints that must be met by the system under design, such as an available power budget or limited memory storage.

These multiple design goals provide a motivation for the use of multiobjective optimization metaheuristics. Metaheuristics are general methods that can be used to optimize a variety of problems, since they work outside the problem domain being optimized. Conceptually, they work by mapping the domain of interest to another problem domain (often inspired by natural or man-made phenomenons completely unrelated to optimization), and then solving the latter. Examples of metaheuristics are Genetic Algorithms (which solve the problem of optimizing a given set of genes) and Particle Swarm Optimization (KENNEDY et al., 2001) (which are based on the idea of a population of individuals converging towards a food source, such as schools of fish and bird flocks).[1]

One of the advantages of the use of metaheuristics is the fact that they can be defined separately from the problem domain, in a way that allows changes in the problem's formulation (such as modifying the objective functions). However, this also means that a translation must be made between the problem domain and the metaheuristic's domain. Figure 4.1 shows this work's proposal of a workflow for performing automatic optimization of the task-to-platform mapping for systems built using the HIPAO2 methodology.

Figure 4.1 – Overview of the proposed DSE workflow for HIPAO2.



Source: Author (2015).

This figure shows the DSE process as the first stage in HIPAO2's *System optimization, validation and code generation* phase (as shown in Figure 1.1), and takes as input the separate Platform Independent Model (PIM) and Platform Model (PM) models, as well as a cost table informing the costs of each task when mapped to each possible PU. Since metaheuristic algorithms use their own encoding for candidate solutions, it is necessary to transform from the metaheuristic-specific encoding to the problem-specific encoding. This is done by defining an Encoding Template, which stores all the information necessary to convert between the two encodings, and back. The Encoding Template is used in various steps in the workflow, more

[1] For a more detailed discussion on metaheuristics and Genetic Algorithms, see Section 2.1.

specifically whenever there is an interaction or a flow of information from the optimization algorithm to a problem-specific process. Such interactions include the evaluation of the objective functions and constraint violations of candidate solutions (which is a problem-specific process), and the output of the best solutions as models.

Note that the use of the Encoding Template as shown here implies in a one-step conversion from the representation used in the metaheuristic and the problem-specific representation. As creating a generic framework for the exploration of any aspect of any design decision is outside the scope of this project, such a simple conversion step was considered to be acceptable. Other works such as Basten et al. (2010), Neema et al. (2003), Oliveira (2013) have attempted to create frameworks that allow designers to extend the DSE process to any design decision; they accomplish this by using a two-step conversion between representations. They introduce an intermediate representation in order to build an independence between the design decisions being evaluated and the optimizer-specific encoding. While the genericity obtained by such a workflow is desirable, defining a model that can represent arbitrary design decisions is not trivial and can greatly complicate other aspects of the DSE process such as requiring additional constraint checking that could otherwise be avoided by using knowledge about the problem domain.

Being multiobjective, the optimization algorithm cannot produce a single, "best" solution to the optimization problem, as that would require weighting or otherwise giving priority to some of the objectives.[2] This is not considered desirable because these priorities do not exist in real-world applications. Engineers are prepared to make tradeoffs at every point in their designs, but *how* much they are willing to trade one objective for another is not generally known. While a notion of priority normally exists ("improve execution time, even if it uses more power"), there is always a limit on how far one would be willing to go. Furthermore, assigning priorities to an objective would mean turning the DSE step into a black box, reducing the insight that the process gives on the characteristics and tradeoffs present in the design. The goal of the DSE process is to provide engineers with *more* information so that they can make better-informed decisions, not to take away the whole decision altogether. Therefore, the output of the workflow is a set of Platform Specific Models (PSMs) corresponding to the Pareto-optimal solutions[3] that were found. It is then the designer's responsibility to consider these solutions and the tradeoffs between them, and make the final choice on which one to use.

Because this final step of choosing the actual output that will be used requires human intervention, we have labeled our tool as performing "semi-automatic" DSE. It should be noted however, that the rest of the workflow introduced in Figure 4.1 was implemented in a fully-automated manner. For more details about the concrete implementation of HIPAO2's DSE capabilities, the reader should refer to Appendix A.

The remainder of this chapter is organized as follows: Section 4.1 explains the specifics

[2] In the general case. For a specific system under design, it may be possible that a single optimal solution exists, even when considering all objectives.

[3] For a discussion on the concept of Pareto-optimality, see Section 2.1.3.1.

of the process of evaluating the objective functions and constraint violations of the individual solutions. Sections 4.2 and 4.3 explain the transformations necessary to use this model to express the type of application and platform that are targeted by HIPAO2. Finally, Section 4.4 explains the specific optimization metaheuristic that was used, as well as related concepts such as encoding and variation operators.

## 4.1 Evaluating solutions

Optimization metaheuristics, by definition, do not have any knowledge about the underlying problem. They must be provided with a sense of the quality of potential solutions to the problem being optimized. In the case of this work, there are several objectives that must be optimized, making this a multiobjective optimization — one in which more than one objectives are simultaneously optimized, without weighting or otherwise giving priority to any of them. Furthermore, potential solutions must also adhere to a set of constraints in order to be valid; these constraints may or not be directly related to the objective metrics.

The process of calculating the objective metrics and the constraint violation for a given potential solution is referred to as “Solution Evaluator” in Figure 4.1. In order to maintain separation of concerns, this evaluation subsystem is not dependent of any feature (such as encoding) of the metaheuristic that is being used for optimization — it works entirely on the problem domain, dealing with tasks and processors as opposed to genes and indexes. The information contained in the Encoding Template shown in Figure 4.1 is used to convert from the optimization algorithm’s encoding to a problem-specific solution representation suitable for use by the solution evaluator.

In embedded system design, the objectives that normally must be optimized are the execution time, power consumption, memory usage and hardware area (when programmable hardware is part of the development). This is in line with other works on DSE (GRIES, 2004; BASTEN et al., 2010; SILVANO et al., 2011; NATH; DATTA, 2014).

Not all designs will need to optimize all these objectives. For example, not all projects have programmable hardware components, making the concept of hardware area inapplicable. Or, in a project that does use an FPGA, a designer may not care about the area used, as long as it is not larger than the available on the chosen chip. For that reason, the reference implementation of the optimization workflow allows the selection, at runtime, of the objectives that should be evaluated.

The last example presents a type of constraint that is very common in embedded system design: one in which the engineer does not care about the consumption of a resource, as long as it is not over a specified budget. Since they work over the value of an objective function, this may be considered to be a simple class of constraint, which can be evaluated using the same process as the underlying objective metric. Even if the metric itself is not included in the set of objectives to be optimized, it will be evaluated for use by the constraint evaluator. We take advantage of this in our implementation by defining both objectives and constraints in terms of generic *metrics*.

Metrics are very loosely defined in our tool: they simply represent a value inferred from a candidate solution. Examples of metrics are "power consumption" and "execution time". A possible objective would be to minimize one of these metrics. Constraints are defined in terms of a predicate over a metric, such as "power consumption $\leq 600\,\text{mW}$". Our implementation compiles a list of metric evaluators that must be called for every solution. Using this approach, if an objective and a constraint must use the same metric, that metric will only need to be evaluated once for each solution.

Evaluating solutions can be a very expensive and time-consuming task, and is usually the bottleneck in DSE methodologies (and in fact, in most applications of metaheuristic optimization). This has discouraged the use of more accurate estimations of objectives, such as cycle-accurate or full-system simulations, although some works attempt to do this, such as Calborean et al. (2011). The high cost of these simulations creates the necessity for faster ways to estimate a system's quality, even if this means trading accuracy for speed. One method to save time in any algorithm is to precompute parts of the evaluation, whenever it is possible to do so — in this case, the task-mapped-to-PU is a good granularity for which we can precalculate performance metrics.

For example, if the power consumption of all the tasks when executing in all possible processors can be known, the estimation of the global power consumption of a given task-to-platform mapping becomes trivial. This idea can also be extended to other objectives, such as memory usage and hardware area, without much effort. When execution times for each task are known and a specific Model of Computation (MoC) is used, there are efficient algorithms for estimating the global execution time.

Therefore, we have decided to adopt the use of pre-characterized tasks, whose costs are known for each possible processor. Many of the works surveyed in Chapter 3 adopt this level of granularity; exceptions do exist, such as Pimentel, Erbas & Polstra (2006), Oliveira (2013), which use an operation-level granularity.

This approach has its shortcomings; the main one is that this pre-characterization data is not easy to obtain, especially when implementations of the same task in very different architectures must be considered. However, we consider this problem to be outside the scope of this work, and treat it as an input datum to the DSE process, as does Piscitelli & Pimentel (2012).

### 4.1.1 Evaluating objectives

When the characterization of components is available, the evaluation of most objectives becomes very simple. For example, the power consumption evaluation is done as:

$$P = \sum P_i(U(i)) \tag{4.1}$$

Where $P_i(u)$ represents the power consumed by task $i$ when mapped to the PU $u$, and $U(i)$ represents the PU that task $i$ is mapped to.[4] Similarly, memory consumption $M$ and hardware area $A$ can be calculated by:

$$M = \sum M_i(U(i)) \tag{4.2}$$

$$A = \sum A_i(U(i)) \tag{4.3}$$

Where $M_i(u)$ represents the memory requirement of task $i$ when mapped to PU $u$, and $A_i(u)$, the hardware area required to implement task $i$ in PU $u$. For software processors, $A_i(u)$ is considered to be zero (as "hardware area" here is understood only as area occupied in custom hardware such as FPGAs).

The evaluation of the execution time, on the other hand, is much more difficult because of the complex relationships in the task graph, with parallel-running tasks and data dependencies. Given that the simulation of the application, taking into account the platform's characteristics, for each possible task-to-platform mapping, is prohibitively expensive, we choose to model our task graphs using a well-defined Model of Computation known as Synchronous Dataflow Graph (SDFG), for which a good set of analysis algorithms is available, including throughput estimation methods.

Stuijk, Geilen & Basten (2006) describes a library (SDF3 — Synchronous DataFlow For Free) for analyzing Synchronous Dataflow Graphs (SDFGs), which is used in this work. This library is capable of evaluating the throughput of SDFGs when mapped to a platform consisting of processors connected by unidirectional, point-to-point connections. This platform model is not compatible with the original system descriptions (as connections are not necessarily point-to-point or unidirectional, and tasks implemented in programmable hardware execute in parallel). However, by transforming the application and platform graphs prior to the evaluation with SDF3, it is possible to suitably represent the necessary features.

These transformations are applied automatically to the mapped task graph by the DSE tool described by this work, after it is created from the optimization algorithm's encoding by use of the encoding template (as shown in Figure 4.1). This means that the transformations are purely used for the evaluation and do not represent a permanent modification on the application or platform models. Such transformations are presented in sections 4.2 and 4.3.

### 4.1.2 Evaluating constraints

In the design of any system, a number of constraints must be met, such as the cost of the final product, or the set of functionalities that must be implemented. In the DSE phase, be it automated or by hand, possible implementations of the system will have to meet any number of

[4] This is a simplification in that it assumes that a single value for $P_i$ can be obtained — in reality, $P_i$ would be a function of other factors such as the PU's clock frequency and voltage. However, the definition of these parameters is part of the *configuration problem*, which is not being considered here.

constraints — although we are not considering characteristics that will not change during DSE, such as the functional requirements mentioned before.

One can however present a number of conditions that must be met by the system, and still may be modified by the DSE process. Common examples are power consumption (which must stay within the power budget), memory consumption (must stay inside the capacity), or hardware area (must fit in the available chips).

Most optimization metaheuristics are built to handle constraints on solutions and to penalize solutions that violate any of them, as constrained optimization is a very common type of real-world problem. Therefore, it suffices to evaluate the constraint violations into a quantifiable value (where, by definition, the violation of a valid solution is 0), and pass this value back to the optimization algorithm. This was done by defining so-called “constraint evaluators”, that can be defined for each constraint that should be evaluated.

The very simple examples of constraints given before have in common the fact that they are all based on values of metrics that could be chosen as optimization objectives (even if they are not being optimized, as was stated before). Therefore, we solve this simple case in our implementation by allowing a constraint evaluator to access the value of arbitrary “metrics”. The evaluators for these metrics will be executed whenever they are needed (whether they are required by an objective or by a constraint).

This is transparent to the constraint evaluator, in that it does not need to know if these metric values are objectives or not. This enables situations such as when the designer has no wish to optimize the power consumption, and is only interested that it remains below a certain power budget. In this case, the "power consumption" metric will not be required by any optimization objective, but it will be evaluated because of the power constraint.

For the more general case (for more complex objectives that are not simple thresholds over objective values), it is possible to define a new metric that is only used by the constraint evaluator.

There are also some cases where the much desired separation between the optimization algorithm and the problem-domain evaluators becomes weakened. This happens, for example, when defining the structural constraint that each task must be assigned to a PU, and there must exist an implementation for this task and PU type. If the optimization algorithm creates a candidate solution that does not meet this constraint, the Encoding Template’s work in creating the Mapped Task Graph (MTG) will be compromised (it will not be possible to create a valid MTG that may later be evaluated). One alternative to solve this problem would be to abort the conversion to an MTG, including this logic in the Encoding Template. Another would be to include a common constraint evaluator to penalize such solutions, while embedding in all other objective and constraint evaluators the logic to deal with malformed MTGs.

However, in our implementation none of these solutions were used, because of the characteristics of the metaheuristic that was adopted — this is one point where the boundaries between metaheuristic and problem become blurred. By using the chosen optimization algorithm (a multiobjective Genetic Algorithm (GA)) and carefully defining its encoding, it was possible to create a situation in which it is guaranteed that a potential solution will never generate a mapping that violates the previously mentioned structural constraint. This is in line with general literature on the use of metaheuristics for constrained optimization, in which a good general advice is to take special care to match representation and variation operators in order to avoid creating invalid solutions (TALBI, 2009). Both representation and variation operators are discussed later in this chapter in Section 4.4.

It should be noted that the above method prevents the occurrence of only the specific kind of invalid solution where a task is mapped to an impossible PU. Other types of structural constraints may be thought of, such as limiting the number of tasks that can be mapped to a certain PU.[5] In this case, the more general method of constraint handling — to define a new metric that represents the magnitude of the constraint violation — would be used. It is possible to do so in this case because, unlike when a task is mapped to an invalid PU, it is still possible to evaluate the solution's objectives.

## 4.2 Modeling communication delay

One of the costs involved in using multiprocessor and heterogeneous systems, with networks connecting the nodes, is the cost of communication. Therefore, no evaluation of the execution time of an application under such a platform would be complete if it does not take into account the cost of data transfer between the system's PUs.

The application model described previously (SDFG, and more specifically the analysis algorithms provided by SDF3) does not allow such costs to be defined, so it is necessary to either define a new modeling method, or a way to take into account these communication costs using the aforementioned formalism. In this work, we have chosen to do the latter. This can be done by adding new tasks to represent the extra communication time. We propose that the communication cost between two tasks $T_1$ and $T_2$ be represented by three components, as follows:

$$\text{Cost}_{\text{comm}} = \text{Cost}_{\text{pack}} + \text{Cost}_{\text{trans}} + \text{Cost}_{\text{unpack}} \tag{4.4}$$

In the above expression, the first component is the cost associated with packing the output of $T_1$ and sending it to the network interface, which includes work normally done by underlying system primitives in the Processing Unit that is executing task $T_1$. The next cost part ($\text{Cost}_{\text{trans}}$) is the cost of sending the data through the network. The last component of the cost, $\text{Cost}_{\text{unpack}}$, is analogous to $\text{Cost}_{\text{pack}}$ and represents the cost of unpacking the transmitted data and delivering it to $T_2$.

It can be observed that the first component is related to work that is done in the same PU as $T_1$ (even if hidden beneath operating system primitives present in most software-based systems). Similarly, $\text{Cost}_{\text{unpack}}$ represents work done in the same Processing Unit as $T_2$, and the work

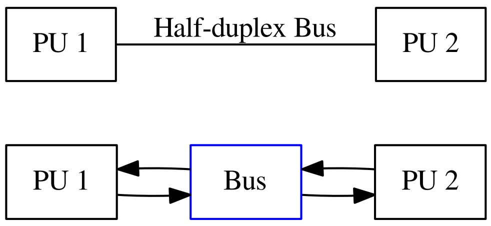
[5] This specific type of constraint has not presently been implemented in our DSE tool.

represented by $\text{Cost}_{\text{trans}}$ happens in the network infrastructure. This property allows us to take these costs into account by only adapting the application and platform models from the original ones, without modifying any of the SDFG analysis algorithms, as will be shown in the sequence.

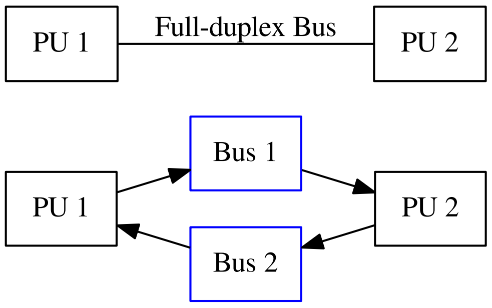
It should also be noted that the cost of communication between tasks that are executed in the same processor is much lower than the cost of moving data between tasks that are mapped to different PUs. Therefore, the above considerations are only taken into account when tasks are assigned to different PUs. In the case that two communicating tasks are mapped to the same Processing Unit, the communication cost is considered to be negligible and is ignored.

In order to represent $\text{Cost}_{\text{trans}}$, we first modify both the application and platform graphs. The platform graph is modified as explained in Figure 4.2, modeling the communication network as a special set of PUs. For half-duplex buses, only one PU is generated; for full-duplex buses two PUs are generated, as shown in Figure 4.2a and Figure 4.2b, respectively. The information of whether a given bus is full or half-duplex must come from the designer, via the platform model. This is done by use of the standard MARTE attribute, `HwBus::transMode`. Using this transformation scheme allows us to represent aspects such as concurrency and contention in the communication.

Figure 4.2 – Modification in the platform to represent communication delay.



(a) Transformations done to a half-duplex bus.



(b) Transformations to a full-duplex bus.

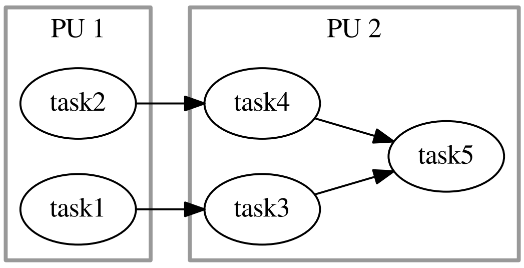
Source: Author (2015).

After this, the application must also be modified to express the work of communicating through the network. This is done by adding a task that represents this communication. The newly created task will be mapped to the respective PU that was created earlier to represent the network, respecting the direction for the case of full-duplex buses.

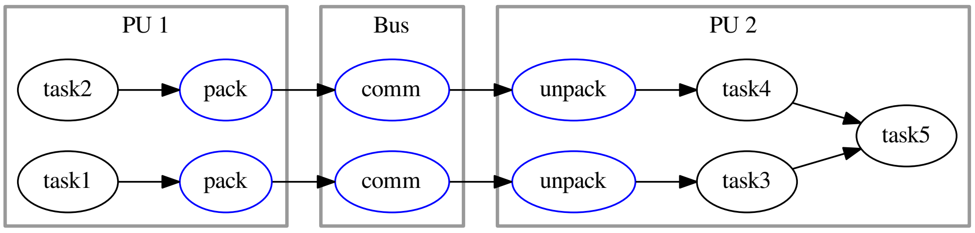
Afterwards, components $\text{Cost}_{\text{pack}}$ and $\text{Cost}_{\text{unpack}}$ from Equation 4.4 must be accounted for. This is done by creating one additional task for each term. It follows from the definition of these terms that they represent work that is done in $T_1$ and $T_2$ respectively; therefore it is logical that the special tasks that represent them should be always mapped to the same PU as these tasks. Entries in the cost table for these specially created tasks reflect the cost of packing and unpacking messages in a particular platform.

Figure 4.3 shows an example of the results obtained by applying the transformations described above to the application in Figure 4.3a and the platform in Figure 4.2a. The containment relation in that figure represents mappings from task to platforms, and the newly created packing/communication/unpacking trio can be seen for the two edges that span different PUs.

Figure 4.3 – Modifications to both the task graph and platform.



(a) Original task graph and mapping



(b) Modified task graph and mapping

Source: Author (2015).

## 4.3 Modeling concurrency in programmable hardware

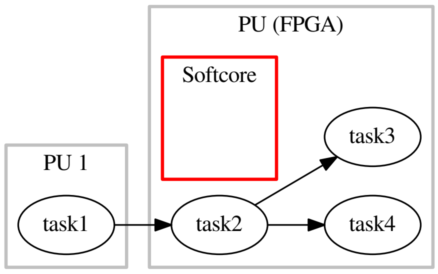
The presence of programmable hardware such as FPGAs can provide further challenges. First, these tasks can be run in parallel, and this is not represented in the model accepted by SDF3, which is suitable for single-thread processors. Second, this parallelism is not complete, as I/O operations use hardware elements (such as network interfaces) that must be shared between tasks. Fortunately, both problems can be solved by using transformations similar to the ones that are used above to represent communication.

Representing parallelism in tasks implemented in programmable hardware is fairly simple. All that is necessary is to create a new PU for each hardware task, as can be seen in the passing from Figure 4.4a to Figure 4.4b. By doing so, the parallelism inherent to FPGA-based tasks is represented in our modeling scheme, and evaluated using the same analysis algorithms.

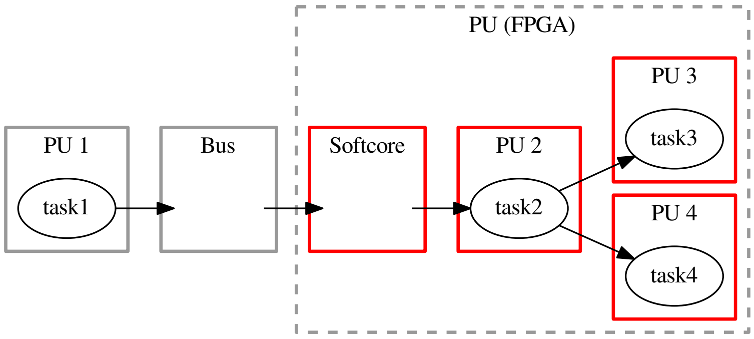
The second problem posed by programmable hardware is where to run to packing and unpacking tasks that were defined for general-purpose software processes in Section 4.2. One valid option for a hardware designer would be to include a new hardware accelerator to deal with the communication protocols in place. This solution may be sometimes desirable and is relatively commonplace, especially when on-chip and same-board buses are considered.

However, in more complicated projects, such as when there are processors in different boards, it may be desirable to use more complex communication protocols such as IP over Ethernet. In these cases, it may be better to implement this communication functionality using

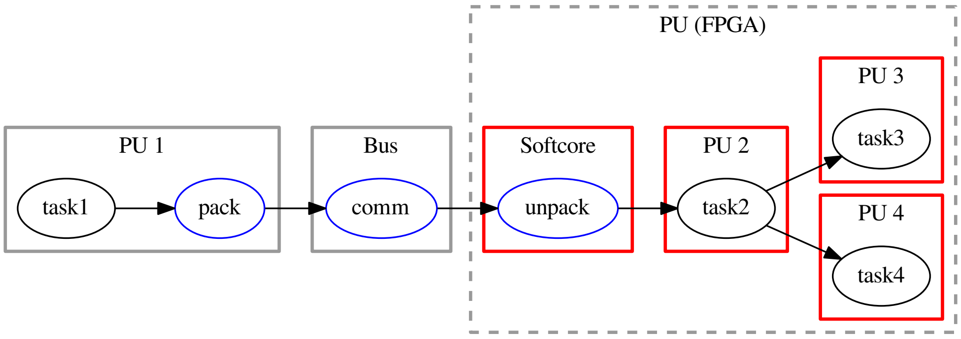
Figure 4.4 – Modification in a task mapped to an FPGA.



(a) Original tasks and mappings.



(b) Creation of additional PUs to represent concurrency.



(c) Addition of packing and unpacking tasks.

Source: Author (2015).

a softcore processor, for example. Eventually, this processor could even execute some tasks in the application — they would probably be slower than a regular, Application-Specific Integrated Circuit (ASIC) processor, but it could be advantageous since the communication cost to other tasks in the same FPGA would be greatly reduced. Being a normal, instruction-based processor, it would also have an advantage in more control-flow tasks. Therefore, we believe this is a valid use case and should be allowed by our tools.

However, the way the communication mechanisms are implemented (whether in a dedicated accelerator in the FPGA, or in a softcore processor) is an integral part of the platform definition. Therefore, in our platform modeling methodology, we require the designer to choose which option should be used, by including an extra PU in the Platform Model with the relevant stereotypes applied.

Figure 4.4c shows the addition of the packing and unpacking tasks, with the unpacking tasks mapped to a softcore processor, thus completing the modifications that must be done in the presence of a programmable hardware element. It must be noted that, although the platform specified in Figure 4.4a shows a softcore processor, a dedicated hardware accelerator (an SPI interface for example) could have been chosen.

## 4.4 Optimization algorithm

One of the main goals of the workflow defined in this work and shown in Figure 4.1 is to promote the separation of the problem-specific sections and whichever metaheuristic optimization algorithm is chosen, thus achieving modularity and allowing each part to be independently changed.

In a practical matter, however, eventually one specific algorithm and associated encoding must be chosen. In this work, we have used the NSGA-II metaheuristic, which is an elitist, multiobjective Genetic Algorithm.[6] GAs have been used extensively to solve complex combinatorial problems with large search spaces. Furthermore, Non-dominated Sorting Genetic Algorithm II (NSGA-II) has been shown to outperform other comparable GAs, such as Strength Pareto Evolutionary Algorithm (SPEA2) (ERBAS; CERAV-ERBAS; PIMENTEL, 2006).

When using a Genetic Algorithm, it is necessary to define two different features, which can be considered independent from the actual GA employed: the encoding for each potential solution, and the two variation operators (mutation and crossover). These two aspects depend on each other, as the operators work directly over the encoding, but there is space for choice between them (the definition of the encoding does not directly determine the operators).

### 4.4.1 Solution representation

Specifying an encoding to be used with a Genetic Algorithm can be defined as determining a way in which to describe all the aspects of a potential solution that should be modified by the optimization process. It is also necessary to define good variation algorithms (mutation and crossover) that operate over this encoding.

Therefore, an adequate encoding should not contain unnecessary information, or aspects that do not vary between different solutions. For example, the task costs for each possible PU should not be included in the encoding, as they do not vary between solutions.

The mapping of tasks to Processing Units can be considered a classic combinatorial problem. In this problem, each task must be mapped to one and only one member of a set of possible PUs for that task. The set of possible PUs for each task can be inferred from the costs table: if there is no entry for a `(task, PU)` pair, the mapping between the two is not possible.

From this abstract definition of the mapping problem, it is simple to write an encoding in a manner more suitable to be used with the GA terminology. We define a genotype in which each

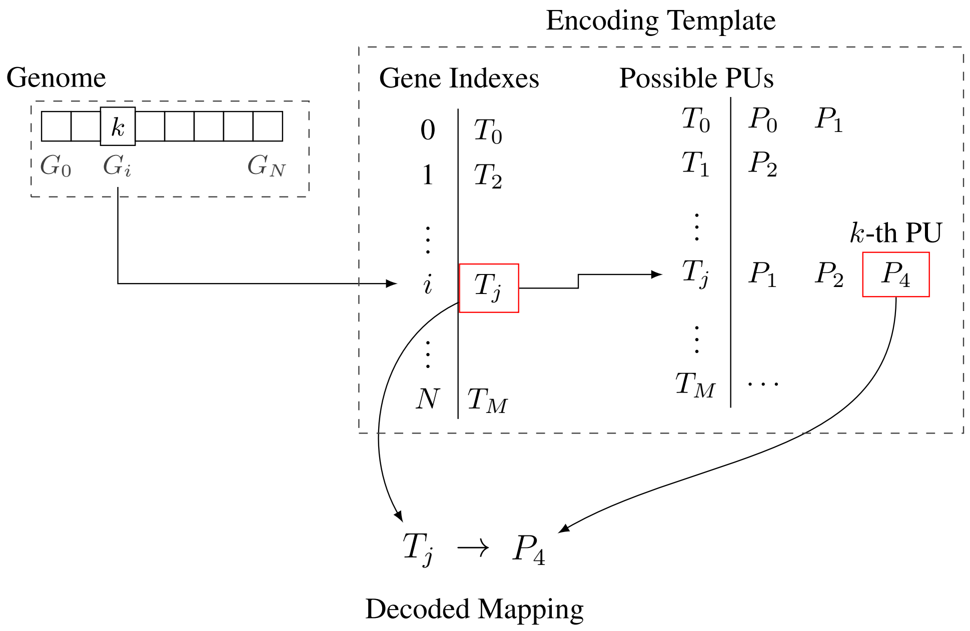
[6] NSGA-II — Non-dominated Sorting Genetic Algorithm II (DEB et al., 2002). For concepts on multiobjective optimization and genetic algorithms, see Section 2.1.

task $T_j$ with more than one possible PU is represented by one gene $G_i$. The value of this gene represents the PU to which that task is mapped to; therefore it is bounded to the range $0 \ldots n-1$, where there are $n$ possible PUs for task $T_j$ to be mapped to.

Since we limit each gene $G_i$ to the range of possible PUs for task $T_j$, it is not possible to create a mapping from a task to an invalid PU. Furthermore, we do not need to check if all tasks are mapped to one and only one Processing Unit, as do other works such as Oliveira (2013).

Using this scheme means that each gene in the encoding will have a different range of values, and a numerical value will potentially mean a different PU in different genes. In order to adapt to that, it is necessary to maintain the interpretation of each gene's possible values elsewhere — this is done in the Encoding Template, which is used to transform the GA encoding to a Mapped Task Graph.

Figure 4.5 – Using the Encoding Template to interpret the mappings in the gene representation.

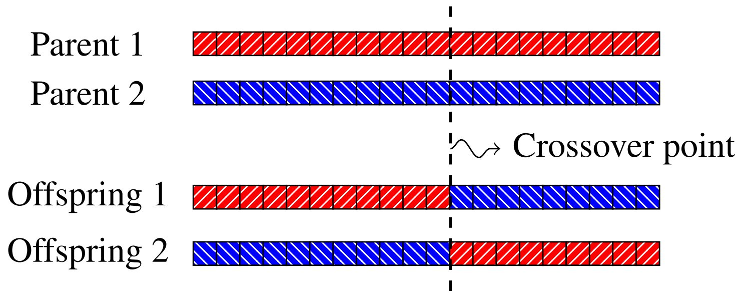


Source: Author (2015).

Figure 4.5 presents an example for the use of the Encoding Template to recover the task-to-platform mapping from the Genetic Algorithm's encoding. That figure shows where each part of the information is kept: the genome representation contains only the index of the PU in the list of possible PUs for the given task. The Encoding Template contains a table relating all the gene indexes in the genome to the task they represent; furthermore, it also contains a second table, that contains all the PUs that a task can be mapped to. This second table is used to retrieve the Processing Unit from the value of the gene $G_i$.

It must also be noted that not all tasks are necessarily represented by a gene in the solution encoding; tasks that can be mapped to only one PU are not included in the genome as there is no mapping decision to be made. This is exemplified in Figure 4.5 by task $T_1$: there is no entry

Figure 4.6 – Single-point crossover operator.



Source: Author (2015).

in the gene index table that points to it as it can only be mapped to $P_2$ in the example. In cases such as these, finding the mapping is trivial as it is constant.

### 4.4.2 Variation operators

The other decision that must be made when employing Genetic Algorithms is the definition of the variation operators that are to be used to create new candidate solutions. GAs have the advantage that they require that two operators be defined, mutation and crossover, with well-defined semantics — or contracts, in the software engineering sense of the term. As in software engineering, the exact implementation of these two operators does not matter, as long as they provide the function defined in the contract. This allows greater modularity and experimentation — the operators can be changed while maintaining the GA, or vice-versa.

Mutation is a unary operator, creating a new candidate solution by a (normally small) modification to an existing one (TALBI, 2009). Due to the nature of the problem that is being optimized, we have defined the mutation operator as a random change in the assignment of one task to another one in its list of possible PUs. This is a very natural mutation operator in mapping and assignment problems, since we have not defined the concept of a “size of a change” of an assignment — every possible PU is considered equal, and therefore it makes no sense to define a “perturbation size” as is common on real-valued problems. Another advantage of this operator is that it makes it simple to constrain the mutation to never produce certain types of invalid mappings (as discussed in Section 4.1.2), thus eliminating a burden on the constraint handling mechanism.

In contrast to mutation, the crossover operator is a binary one, generating two new solutions by recombining two parent solutions (TALBI, 2009). Following the experimental evaluations by Erbas, Cerav-Erbas & Pimentel (2006), we have chosen the single-point crossover over the two-point crossover, given its more favourable evaluation. An example of the use of this operator is given in Figure 4.6. As illustrated, this operator randomly chooses a crossover point, at which the two parent solutions are cut into two portions. These portions are swapped between the two parents, creating two different offspring. It should be noted that this operator does not incur in information loss: every gene from a parent will be present in one of its children.

## 4.5 Integration with modeling tools

It has been said before that, for the use of Model-Driven Engineering (MDE) methodologies to be feasible, specialized tool support is necessary (SELIC; MOTUS, 2003). Therefore, HIPAO2 consists not only of a development methodology, but also of a set of tools that supports the workflow it proposes. It integrates with the model editor GUI from the Eclipse Modeling Framework (STEINBERG et al., 2008), and its Unified Modeling Language (UML)/Systems Modeling Language (SysML) APIs, that allow manipulating models programmatically. The tools use the foundation provided by the EMF, to implement features such as Model-to-Model and Model-to-Text transformations (DOERING et al., 2013).

The DSE workflow proposed here is not an exception in that it needs tool support to be used — it should be simple to start the exploration from a set of SysML models, with as little human intervention as possible. The core of our DSE tools is provided as a Java Application Programming Interface (API), allowing its functionality to be embedded into any program. This library does not work directly on SysML models, but on a few simple Java objects that carry only the information necessary to perform the Design-Space Exploration. This is done in order to separate SysML model handling code from the DSE tool, providing better separation of concerns and making testing and development easier. It also has the effect of not tying the DSE tools to a SysML representation; making it possible to use our tool outside the scope of HIPAO2, as long as the necessary information about platforms and applications can be provided.

There is no technical reason why this API could not be used directly in HIPAO2's Eclipse plugin. In practice, we have instead decided to create a command-line tool that is a thin wrapper over this API. This tool receives the necessary models serialized as Javascript Object Notation (JSON) models. This format was chosen for its simplicity and human-readability. Although JSON is schema-less format, in practice it is only used to directly create the Java objects used by the tools; the Java class definitions effectively represent the expected schema. An excerpt from an application graph serialized as JSON is shown in Figure 4.7.

It can be seen that some of the names directly correspond to certain SysML or Modeling and Analysis of Real-Time and Embedded Systems (MARTE) constructs. For example, `Block` is an SysML stereotype, `HwProcesssor` and `HwBus` are MARTE stereotypes, and the `transmMode` property is a MARTE property that allows defining whether a bus performs full or half-duplex communication.

The existence of this standalone command-line tool makes it much simpler to run and reproduce the different experiments shown in the following chapters — but it does not solve the need for integration with MDE tools. Currently, this is done through an export feature provided in the HIPAO2 Eclipse plugin, generating the necessary JSON files from SysML models. This can be viewed as one more type of Model-to-Text (M2T) transformation provided by HIPAO2's tools.

During this export process, enough information is retained so that entities in this simplified model can be recognized as the original SysML elements. This is important to enable the back-

Figure 4.7 – Example of models in JSON format.

```
{ "connections": [
    { "dst": "pci1",
      "src": "K6-3" },
    { "dst": "pci1",
      "src": "PPC750" },
    ...
  ],
  "elements": [
    { "category": [
        "Block",
        "HwProcessor"
      ],
      "class": "AMD K6-IIIE+",
      "name": "K6-3" },
    ...
    { "category": [
        "Block",
        "HwBus"
      ],
      "class": "busPCI-32-33",
      "name": "pci1",
      "transmMode": "fullDuplex" }
  ]
}
```

Source: Author (2015).

annotation process, that modifies the model to include the mapping information found during the DSE process. However, this step is not yet implemented and constitutes future work.

The main product of the exploration process is the set of task-to-PU mappings that compose the Pareto front which was discovered, and the objective values that correspond to each of these mappings. Secondary outputs are description of the gene encoding that was generated for the input models (which can be used to support the model back-annotation), and log data that supports the experiments that are shown in Chapter 5.

# 5 CASE STUDIES AND RESULTS

This chapter presents three different case studies, so that different reasonings about the proposed workflow can be made. The first one applies our tools to a JPEG encoder that has been previously examined in the literature. Afterwards, variations on this case study are used to evaluate the workflow's behavior under aspects such as different communication costs. The second case study applies the Design-Space Exploration (DSE) workflow to a set of standard benchmarks for Platform-Based Design (PBD), providing an example of what kind of insights a system designer can obtain by the use of HIPAO2's Design-Space Exploration capabilities. The last one uses randomly-generated task graphs in order to verify the scalability of the proposed approach — that means, how it behaves with an increase in the number of elements in the task graph.

## 5.1 JPEG Encoder case study

HIPAO2 focuses on the development of image processing systems. Although cameras and other image acquisition systems normally work with raw, uncompressed images, this is prohibitively expensive for most communication and storage purposes, so compression algorithms are widely used — making such algorithms relevant to this work. One of the most widely used compression algorithms for photographic images is the JPEG algorithm (ITU, 1993), which has been used extensively as case study in the literature.

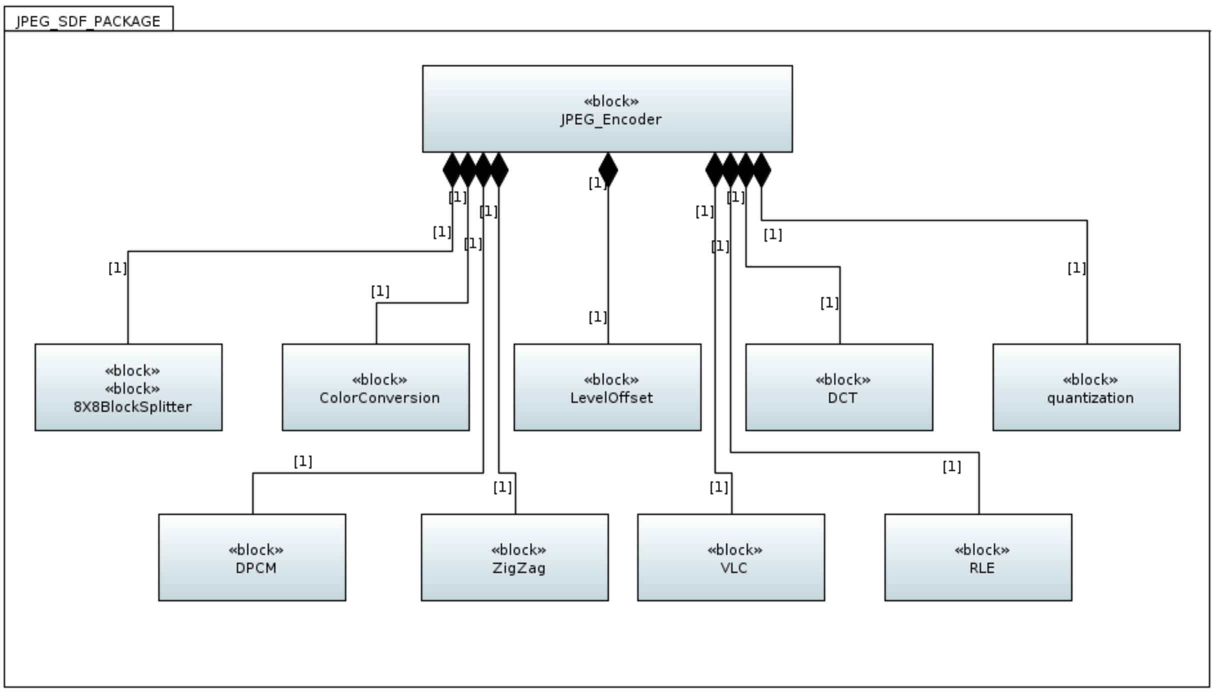
Lee et al. (2007) is one of these works. It studies the hardware-software partitioning of a JPEG encoder composed of 22 tasks, in a system composed of a hardware processor connected to an Field-Programmable Gate Array (FPGA). The same case study has also been used in Nath & Datta (2014) and Abdelhalim & Habib (2009) to validate other proposals for hardware-software partitioning algorithms.

Given that such a partitioning can be seen as a special case of task-to-Processing Unit (PU) mapping, it is a good candidate for use with HIPAO2, although in general use the entities modeled with that methodology would be expected to be larger (e.g. the whole JPEG encoder as a single element). Nevertheless, this presents an interesting use case and allows comparing results with other works.

The 22 tasks of the encoder are divided into nine types. In our SysML-based methodology, this translates to the Block Definition Diagram (BDD) shown in Figure 5.1. They are connected in a pipeline that divides the image into three channels for processing, forming the Internal Block Diagram (IBD) that are presented in Figure 5.2.

Each type of task was characterized by Lee et al. in terms of execution time, power consumption, memory usage (for the processor) and hardware area (for the FPGA), when the task type was allowed to be implemented in either software or hardware — some types of tasks were only allowed to be implemented in software. The costs for each of the task types can be seen in Table 5.1. Unfortunately, the original proponents of this case study did not consider

Figure 5.1 – BDD for the JPEG encoder use case.



Source: Author (2015).

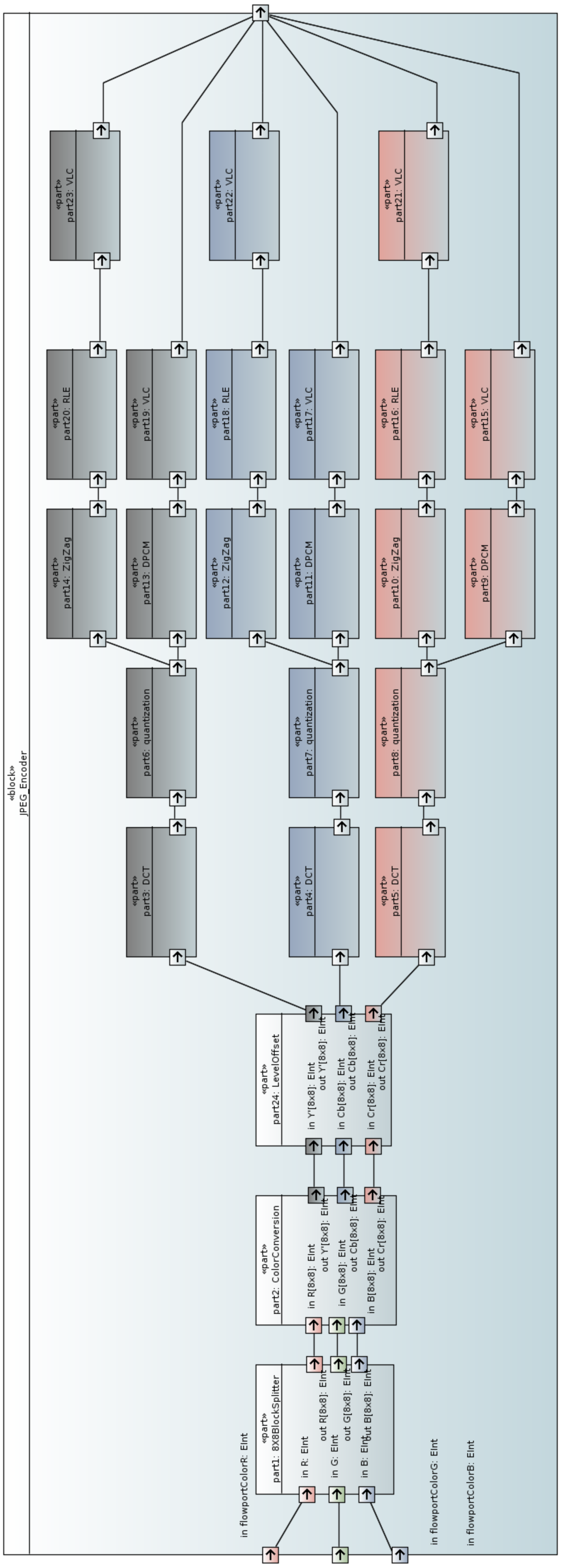
the communication costs between the processor and the FPGA, so there is no data about the communication delay in the actual platform that they used. Therefore, our first experiments will also not consider these costs, and later on the effect of a rising cost of communication will be explored.

Table 5.1 – Task costs for the JPEG case study.

| Task type | PU | Execution time (ns) | Power (mW) | Area (×0.1%) | Memory (×0.1%) |
|---|---|---|---|---|---|
| LevelOffset | CPU | 9380 | 0.096 | 0 | 0.58 |
| | FPGAHwAcc | 155.264 | 4 | 7.31 | 0 |
| DCT | CPU | 20 000 000 | 45 | 0.0 | 2.88 |
| | FPGAHwAcc | 1844.822 | 274 | 378.0 | 0 |
| quantization | CPU | 34 700 | 0.26 | 0.0 | 1.93 |
| | FPGAHwAcc | 3512.32 | 3 | 11.0 | 0 |
| DPCM | CPU | 940 | 0.957 | 0.0 | 0.677 |
| | FPGAHwAcc | 5.334 | 15 | 2.191 | 0 |
| ZigZag | CPU | 13 120 | 0.069 | 0.0 | 0.911 |
| | FPGAHwAcc | 399.104 | 61 | 35.0 | 0 |
| VLC | CPU | 2800 | 0.321 | 0.0 | 14.4 |
| | FPGAHwAcc | 2054.748 | 5 | 7.74 | 0 |
| RLE | CPU | 43 120 | 0.021 | 0.0 | 6.034 |
| | FPGAHwAcc | 1148.538 | 3 | 2.56 | 0 |

Source: Lee et al. (2007).

Figure 5.2 – IBD for the JPEG encoder use case.

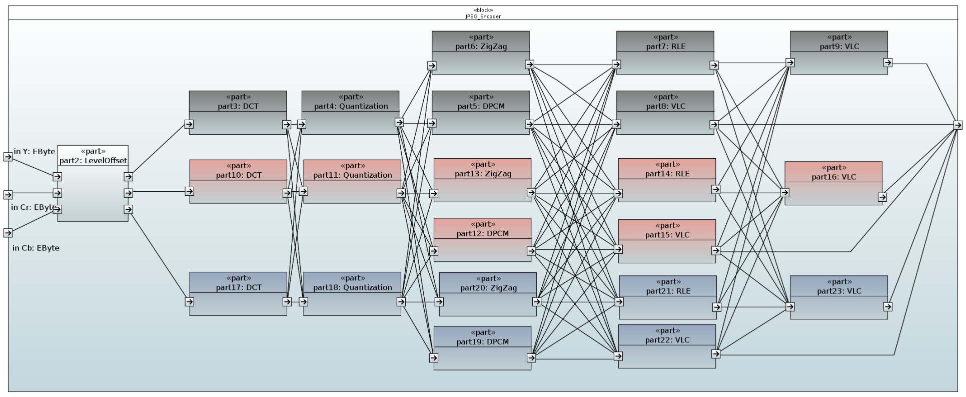


Source: Author (2015), based on Lee et al. (2007).

The goal of this case study's DSE, as defined by its proponents, is to decide on the assignment of each task to hardware or software, minimizing execution time, power consumption, memory requirement, and area usage. Additionally, solutions must also conform to the following constraints: *a*) the slice usage (hardware area) must be less than 100%; *b*) the memory usage must also be smaller than 100%, and *c*) the system as a whole must not require more than 600 mW to run.

The original proponents of this case study had included an additional restriction by modeling the encoder as a Control and Dataflow Graph (CDFG), dividing the processing into separate steps and requiring that each step finishes before the next one starts. This restriction is the conceptual equivalent of modifying the task dependencies of our Synchronous Dataflow Graph (SDFG) model, yielding the task graph seen in Figure 5.3, shown here for comparison. Another difference in the system modeled in this work from the previous ones is the restriction of having only one PU for software while the other works have considered two general-purpose PUs for software.

Figure 5.3 – Overconstrained task graph seen in Lee et al. (2007), Nath & Datta (2014).

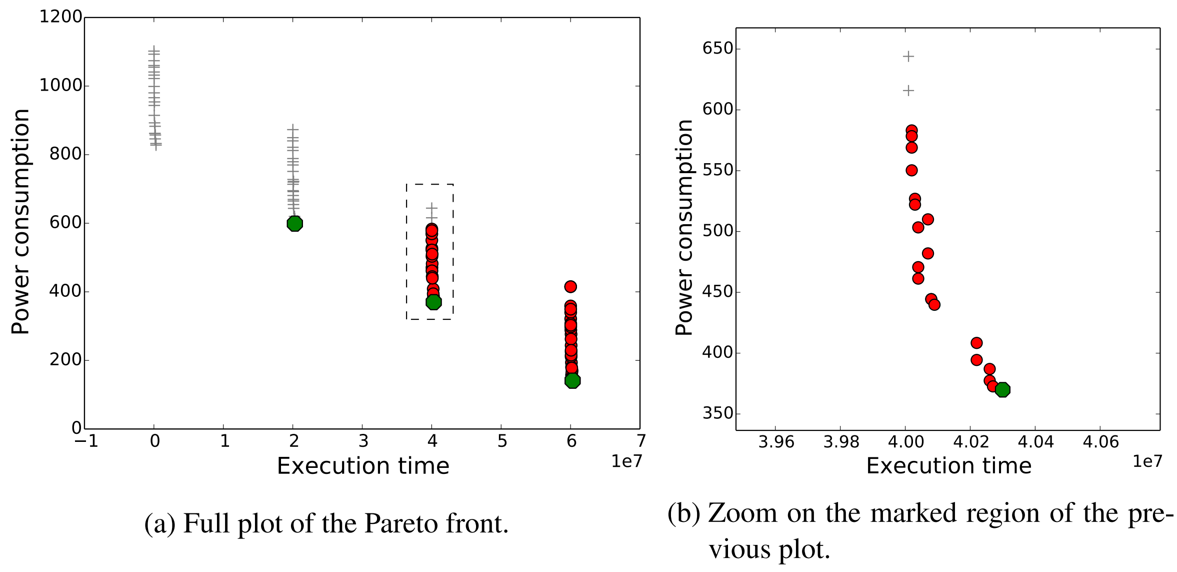


Source: Author (2015), based on Lee et al. (2007).

Furthermore, previous authors also limited the number of tasks that were allowed to be performed in software in each step — they have defined that in each "level", the number of tasks executing in software must be at most the number of software cores present in the system. This restriction of concurrence prevents executing all the required tasks in software (instead of simply executing them in sequence, as would be normal). The model used in the present case study does not implement such restriction, therefore allowing an all-software implementation to be considered as a possible outcome of the DSE process.

The first experiment with the JPEG case study is to simply execute the DSE process with the IBD given in Figure 5.2, optimizing for all possible objectives — i.e. execution time, power consumption, hardware area, and memory usage. This will yield a Pareto front in a four-dimensional objective space. Considering that the most important tradeoff in this design is the one between execution time and power consumption, we show the obtained Pareto front in a two-objective view in Figure 5.4. In the figure, feasible solutions are represented by dots, and alternatives that violate any constraints are shown as gray crosses.

Figure 5.4 – Two-objective view of the Pareto front for the base JPEG encoder case study.



(a) Full plot of the Pareto front.

(b) Zoom on the marked region of the previous plot.

Source: Author (2015).

That figure shows clearly the formation of clusters of solutions, with similar execution times but varying power costs. By looking at the costs table (Table 5.1) and the solutions in the Pareto front, it becomes evident that this is due to the very large cost of one type of task, the Discrete Cosine Transform (DCT), compared to any other. Indeed, examination of the actual mappings (in solution space) represented by each point in objective space shows that each cluster is composed by solutions with the same number of DCTs in hardware — for example, the slowest cluster has all the DCTs in software, but it contains the solutions with the lowest power cost. Also, the leftmost cluster contains solutions with all the DCTs implemented in hardware, but they all violate the power constraint.

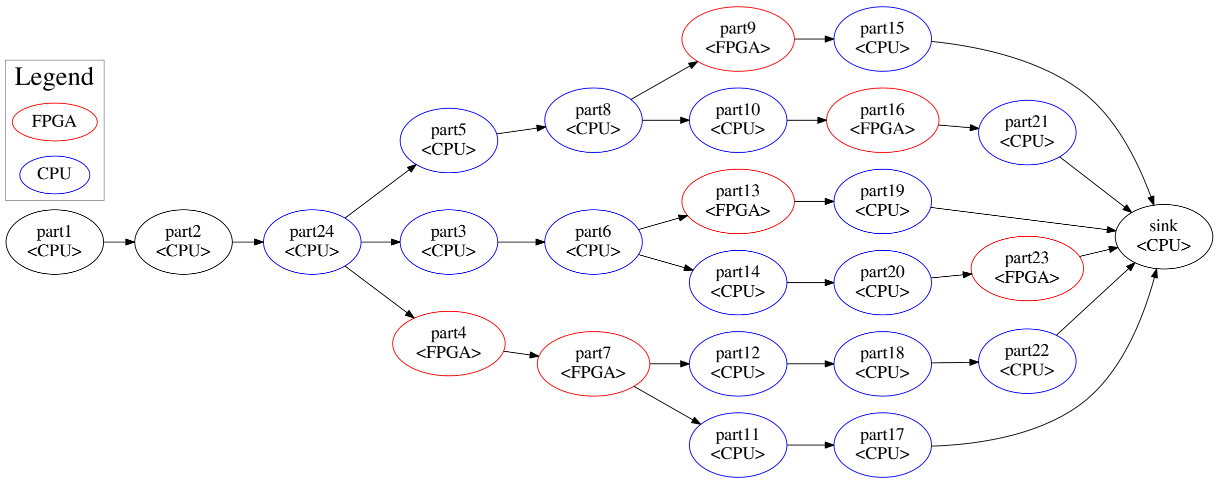
In Figure 5.4, the highlighted points represent the points that contain only DCTs implemented in hardware, with all other tasks being run in software. These always represent the solutions with the lowest power consumption for each cluster. The zoomed version in Figure 5.4b shows that other, alternate mappings in the same cluster (with different tasks implemented in hardware or software) can have a better execution time. However, the difference made by changing any task's assignment from hardware to software is very small when compared to the difference of moving a DCT task from hardware to software. This is again understandable when looking at Table 5.1 and at the difference between the execution time of the different task types when implemented in hardware or software.

One of the problems with this case study is that it does not consider the cost of communication between hardware and software components. To examine the effect that different communication costs would have on the implemented system, and to show that our tools would produce correct insight into the tradeoffs present in the design, the next section shows an analysis of the system under varied communication costs.

### 5.1.1 Effect of communication cost in the JPEG case study

The first and most logical effect of not considering the cost of communication is that there is no incentive whatsoever for the optimization process to minimize the number of communications that must happen from one PU to another. This can be confirmed by examining some of the mappings generated earlier, of which Figure 5.5 is an example. It is possible to observe that there are many connected tasks that are mapped to different PUs. In a more realistic situation where communication costs are being considered, it would be expected that the occurrence of such communications be minimized by the optimization process.

Figure 5.5 – Example of mapping generated without considering communication cost.



Source: Author (2015).

As a consequence, it is important to reason on whether our proposed scheme would work when communication costs are considered. For that, we introduce the communication costs in the JPEG encoder use case, using reasonable values for these costs. In order to do that, we have decided to examine the results of the optimization process with varying communication costs, which are shown in Table 5.2. These values were chosen as different proportions of the task with higher cost when implemented in hardware (i.e. the tasks with the highest execution time and power). Therefore, the “0.5x” configuration represents a system where communication costs half of the time and power of these tasks, and so on. Also, since these configurations will incur in an overall higher power consumption, the power constraint was removed from the following experiments.

An additional configuration was added to represent an extreme case where the communication cost is comparable to that of any task, in hardware or software. By logic, in this situation it should never be advantageous to perform a task in hardware. That configuration was added in order to confirm that our tool would provide this answer.

Table 5.2 – Communication costs considered for the JPEG use case.

| Configuration | Time (ns) | Power (mw) |
|---|---|---|
| 0x | 0 | 0 |
| 0.5x | 1756.16 | 137 |
| 1x | 3512.32 | 274 |
| 10x | 35 123.2 | 2740 |
| Extreme | 10 000 000 | 2740 |

Source: Author (2015).

It is our hypothesis that when non-null communication costs are considered, the optimization process will move towards solutions with fewer instances of communication between PUs. This should happen by choosing solutions which form chains of tasks that are mapped to the same Processing Unit — in Figure 5.5, for example, the maximum chain length of contiguous tasks that are implemented in the FPGA is 2 (formed by `part4` and `part7`). For each mapping in the Pareto front for a configuration, we can generate a list of the lengths for each chain — for example, the mapping in Figure 5.5 has 4 chains of 1 task, 4 of 2 tasks, and one chain of lengths 3, 4, 6, and 7, each.

Examining the chain lengths for all the solutions in each configuration, it is possible to see that the distribution of chain lengths is highly irregular, and therefore we have adopted the median chain length in order to provide a single metric for quantifying chain lengths for a given solution. However, this metric alone is not a good measure of how much communication-optimized a given mapping is, as it does not take into account how many tasks are implemented in different PUs. For example, a solution with most tasks mapped to software would have high chain lengths, but this alone does not mean that the optimization process is preferring to group in the same chain the few tasks that are implemented in hardware. Therefore, we have defined a new metric, the *continuity factor*, to evaluate this aspect of a single candidate solution, expressed as

$$\text{Continuity factor} = \frac{\widetilde{L}}{\max\left(n_{\text{CPU}}, n_{\text{FPGA}}\right)} \tag{5.1}$$

where $L$ represents the chain lengths in the mappings, and $n_{\text{PU}}$ represents the number of tasks mapped to a given Processing Unit. The denominator used here gives a measure of how evenly distributed are the tasks among PUs, as it presents its maximum value when both PUs are used to run an equal number of tasks. It should be noted that this metric will work only for this simple two-PU case — for a more general case, it would be necessary to define another measure for this purpose.

As can be seen in Table 5.3, the continuity factor is considerably higher for all the configurations that have a non-zero communication cost, when compared to the one that does not consider these costs. This indicates a tendency that the Pareto-optimal solutions found by the DSE process involve a lower communication overhead, and that the DSE process is correctly avoiding unnecessary communication. However, no conclusive relation is obtained relating the continuity factor of the configurations that do consider communication cost. This can be seen as an indicator that the actual cost is not determinant to whether the DSE process will avoid communication; it will do so as long as the incentive for reducing communication is not negligible.

Table 5.3 – Average of the continuity factors for each configuration.

| Configuration | Continuity factor | Increase over base |
|---|---|---|
| 0x (base) | 0.122 | – |
| 0.5x | 0.151 | 23.66% |
| 1x | 0.157 | 28.11% |
| 10x | 0.152 | 24.36% |
| Extreme | 0.171 | 39.54% |

As mentioned before, when the communication cost becomes excessively expensive (i.e. greater than any improvement that might be obtained by executing a task in hardware), the tradeoff between execution time and power consumption disappears. In such a configuration, mapping a task to hardware will always incur in a longer execution time as well as a higher power consumption. This is the situation corresponding to the "Extreme" configuration alluded to in Table 5.2 — the time cost of communication is equal to half the greatest time cost of any task.

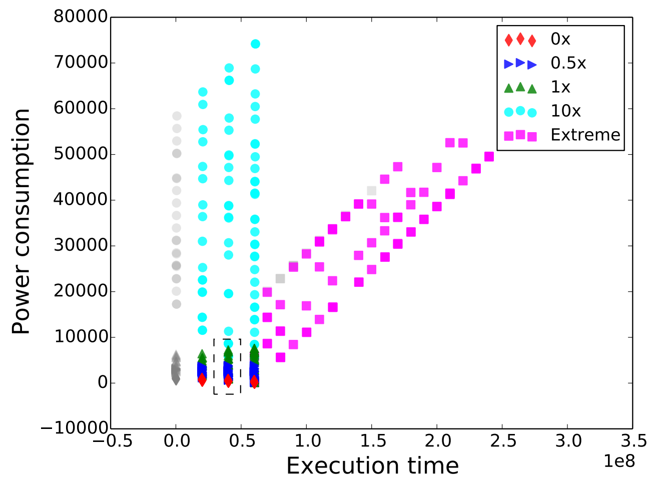
To summarize the effects of the different communication costs shown in Table 5.2, Figure 5.6 shows the time and power objectives of the solutions in the Pareto fronts for each configuration. Since the objective values of the configurations with higher power cost eclipse the other ones, a zoomed version of a region of the plot is also provided.

That figure shows clearly that when the communication is relatively cheap, there is a tradeoff between power consumption and execution time. However, in the last configuration, the cost is prohibitively high so that there will be no gain (in execution time) from mapping a task to hardware (where time costs are smaller, in this case study). This is the expected conclusion from such a situation.

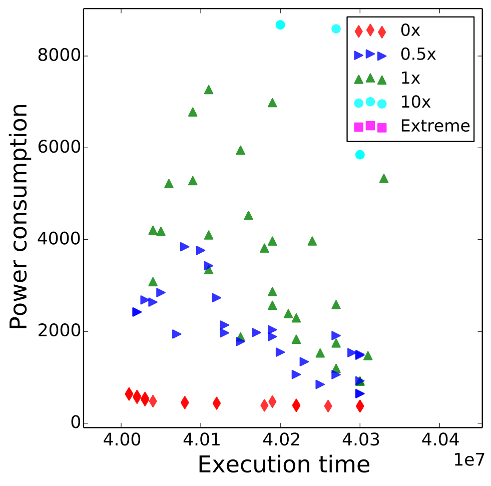
Considering the concept of Pareto-optimality, at a first glance the set of solutions shown in Figure 5.6 does not seem to form a correct front of non-dominated solutions. This happens because in that figure only the execution time and power can be seen, whereas they were generated by optimizing a four-dimensional objective space.

Examining some of the seemingly dominated points in other views of the four-dimensional objective space shows that these points are indeed Pareto-optimal. Figure 5.7 shows the Pareto front for the "Extreme" configuration, shown under two different projections. The first one shows the execution time and power objectives, as previously shown, whereas the second shows power consumption and memory usage. The same point is highlighted in both plots — while it

Figure 5.6 – Pareto front for all communication costs for the JPEG encoder case study.



(a) Pareto front obtained for each communication configuration.

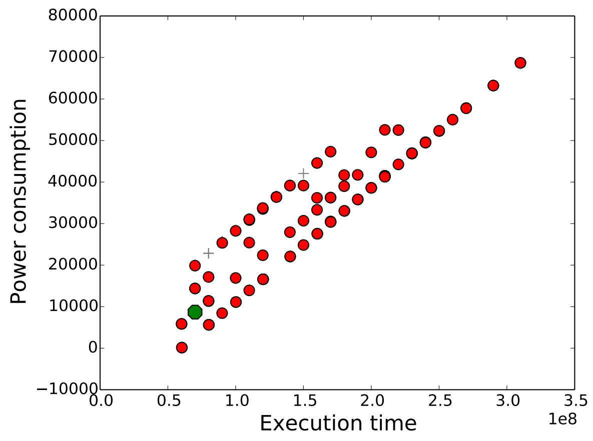


(b) Zoom on the marked region of the previous plot.

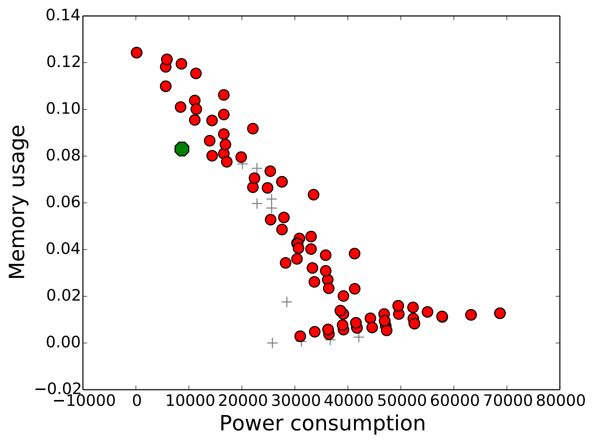
Source: Author (2015).

may seem to be dominated by other solutions in Figure 5.7a, looking at the same point under different projections shows that the solution is not dominated.

Figure 5.7 – View of different objectives for the same Pareto front.



(a)



(b)

Source: Author (2015).

It can be argued that in a realistic design situation, engineers would not be interested in most of the points shown in the Pareto fronts presented up to here — especially in a situation such as in Figure 5.7, where it shows progressively worse solutions in both execution time and power, without a tradeoff between the two. Nevertheless, these points appear in the results because the DSE tools were configured to optimize all available objectives (execution time, power consumption, hardware area and memory usage). It is also questionable if optimizing all these objectives are actually necessary, since not all present tradeoffs between them — for example, in this case study, there is no tradeoff between memory usage and execution time, or between hardware are and power consumption. Nath & Datta (2014) has also disserted over this issue, and now we will present our own findings.

### 5.1.2 Multiobjective *versus* Bi-objective optimization

In Nath & Datta (2014), the authors have studied the same JPEG encoder presented here. However, instead of directly treating the objective space as a four-dimensional one, they have first performed an analysis on which of the objectives conflict with each other. Their finding, which also holds true for the DSE process presented here, was that: 1. Execution time and memory usage do not conflict with each other; 2. Power consumption and hardware also do not present conflicts; 3. There are tradeoffs between objectives of the previously mentioned groups.

They arrived at these conclusions by performing several DSE experiments, each optimizing only two objectives at a time. For the objectives for which there is no conflict, the final Pareto front obtained by Nath & Datta contained only one solution, the one with the best value for both objectives. Performing the same optimization with the DSE proposal presented in this work also arrives at the same one-point Pareto front for these cases.

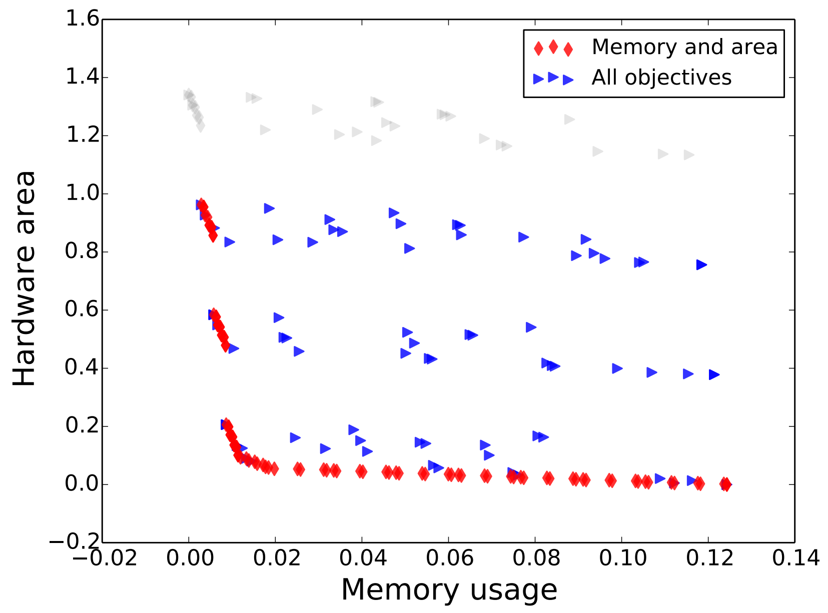
The problem lies with the corollary presented subsequently by the authors: they state that when two objectives do not come into conflict with each other, it is enough to consider only one of them in the optimization process. They then proceed with their analyses performing only a bi-objective optimization of hardware area and memory usage.

However, here we make the statement that this bi-objective analysis is not equivalent to optimizing all four objectives at once. Even if two objectives do not come into conflict with each other, it does not mean that one of them can be disconsidered; their relation is not necessarily linear or even a well-behaved function.

Figure 5.8 compares the effect of optimizing all the objectives (as shown until now), and optimizing only the memory and area objectives, as done in Nath & Datta (2014). It is evident that the Pareto front that is obtained is very different, especially considering the amount of points. Optimizing all the objectives will include many points that are Pareto-optimal only because of the other objectives. Another example is shown in Figure 5.9, which shows the results obtained for two-objective and four-objective optimization, for the “0.5x” configuration. The same aforementioned effect happens in this example — there are much more points when all objectives are considered, since solutions with higher power cost are dominated by others in the dual-objective version, but not when all the objectives are considered, since they have a smaller memory cost.

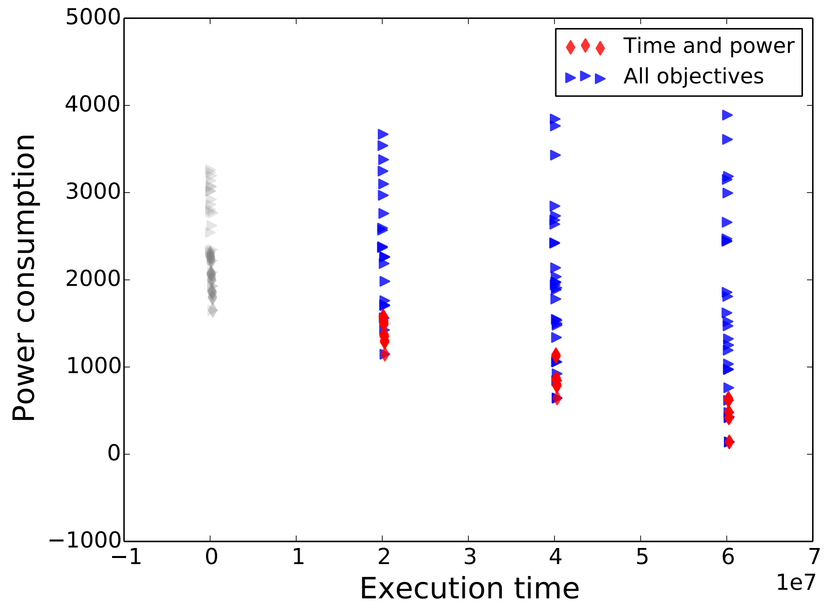
It can be argued, however, that these additional solutions are of little interest to system designers and that the fact that they are not shown is desirable. However, this is so only when designers are not interested in *minimizing* hardware area and memory usage — they could need only that the requirements fit the available resources (in a case where there is little perceived

Figure 5.8 – Comparison between the Pareto front obtained by optimizing all objectives and only two.



Source: Author (2015).

Figure 5.9 – Comparison between optimizing two objectives and all objectives, for the “0.5x” configuration.



Source: Author (2015).

advantage in using less area or memory). In other words, in these situations hardware area and memory usage are viewed only as *constraints*, not as *optimization objectives*.

Nevertheless, our argument here is that using a DSE tool to optimize only metrics that present conflict with each other is not equivalent to using it to optimize all metrics. In the previously mentioned case in which a designer is not interested in optimizing all objectives, that requirement should be used to configure the Design-Space Exploration process to optimize only the wanted metrics. This has no influence over the design *constraints*, which can be arbitrary and not related to the objectives of interest (as in the example where memory and area are only considered as constraints).

## 5.2 E3S Embedded systems benchmark

In order to show the DSE process in work with a realistic system, we have used the E3S benchmark set (DICK, 2002). This is a readily available set of task graphs, each characterized for a number of different processors and for different types of buses. In total, the set of benchmark comprises 17 different processors, and 47 tasks, which form applications divided into four domains: automotive industry, networking, office automation, and telecom.

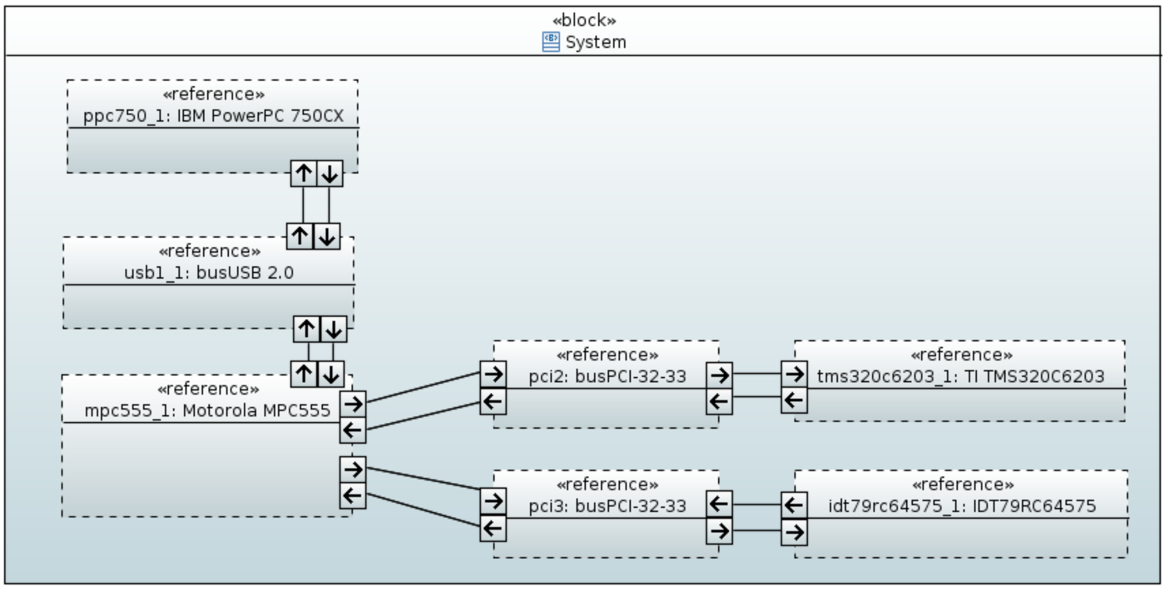
The available processors are of varied types, ranging from general-purpose processors to Digital Signal Processors (DSPs); however this benchmark set does not include programmable hardware PUs. For this reason the hardware area objective does not apply to this case study — this is not a problem since the optimization software is made to be flexible enough to allow defining which set of objectives that are to be optimized.

Since the original benchmarks were provided in the same format used by the TGFF tool (DICK; RHODES; WOLF, 1998), it was first necessary to transform the applications into HIPAO2 Systems Modeling Language (SysML) models. This was done in an automated manner, using the Application Programming Interface (API) exposed by the Eclipse Modeling Framework tools (STEINBERG et al., 2008). A platform model was then built considering the available PU and bus types, and is shown in Figure 5.10 — the objective of the case study was to map the E3S benchmark's task graphs to this platform. Although some of these bus types and processors may be considered general-purpose and not restricted to the embedded market, the same principles apply, allowing the use of these benchmarks as examples.

Of the four benchmarks in the E3S set, we have chosen to show the automotive industry one, as it has a good number of tasks, and can be considered closer to the embedded and signal processing domain that is the main concern of this work. From the TGFF files that represent the automotive industry task graphs, a SysML model was created conforming to HIPAO2's guidelines. The task graph defined by this model is shown in a more compact representation in Figure 5.11.

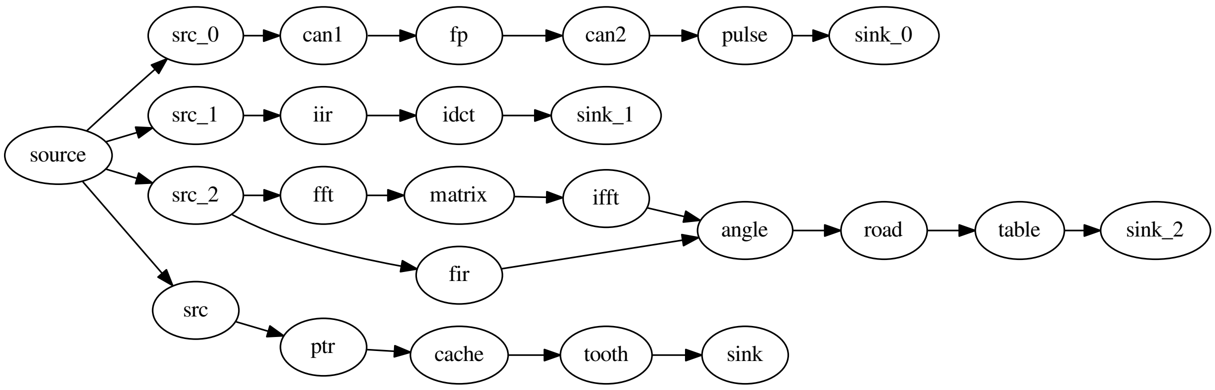
Figure 5.12 shows the Pareto front that was found for this application. Each point represents a possible mapping in the objective space, some of which would be of special interest to an engineer. For example, the point labeled as *A* represents the mapping that was found to have

Figure 5.10 – Platform model used for the E3S benchmark.



Source: Author (2015).

Figure 5.11 – Automotive application task graph extracted from the SysML model.



Source: Author (2015).

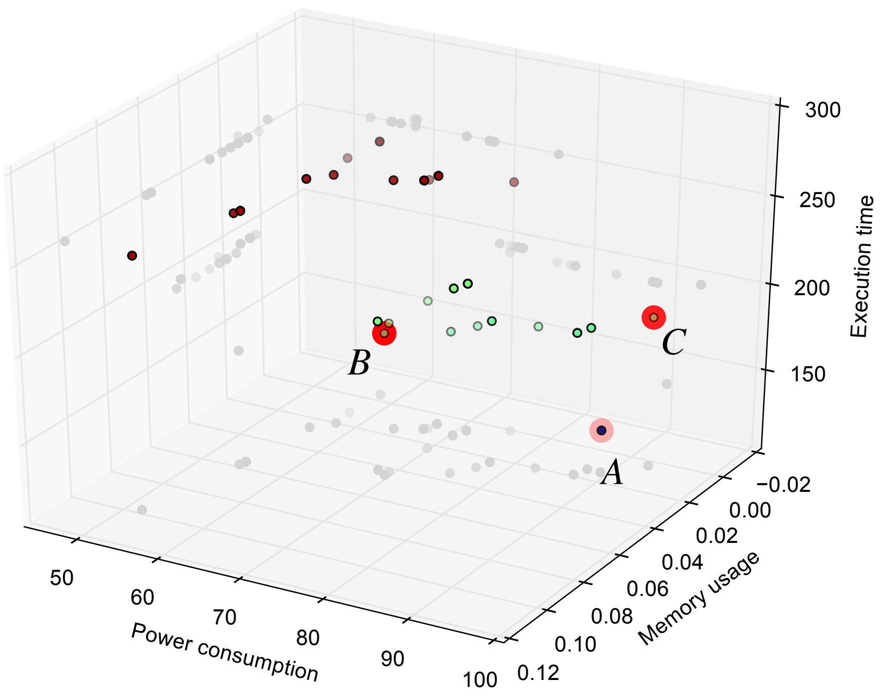
the fastest execution time. However, by comparing this solution with the ones close to it in the objective space, it can be considered to present a very high power consumption and memory usage — it is the designer's responsibility to define if this tradeoff is acceptable.

If it is not, we can observe a set of solutions with a similar execution time, but with varying power and memory requirements. Points labeled *B* and *C* represent the extreme points of this tradeoff. This Pareto front shows the various ways by which solutions with similar execution time can be obtained, and how they vary in memory usage and power consumption.

## 5.3 Randomly-generated task graphs

The third experiment serves to show the scalability of our framework as more complex tasks graphs are used. In order to achieve this, we have employed the Task Graphs For Free (TGFF)

Figure 5.12 – Pareto Front for the automotive application.



Source: Author (2015).

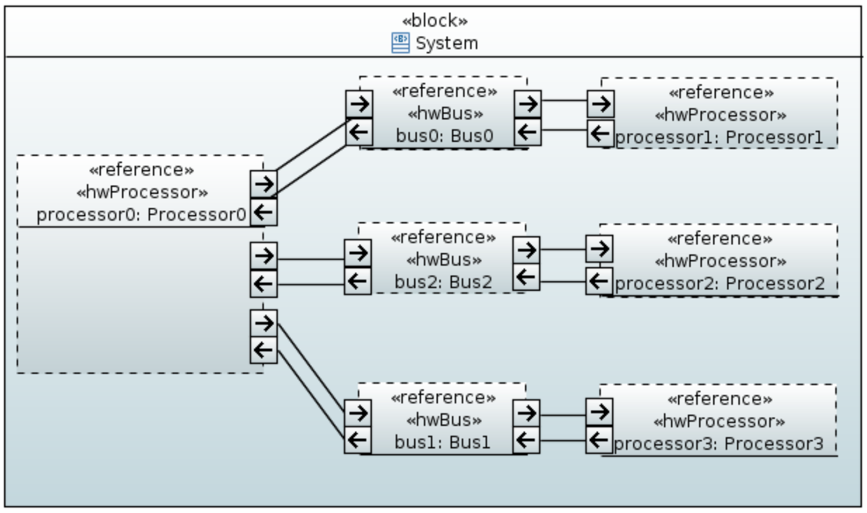
tool (DICK; RHODES; WOLF, 1998), which is capable of generating random task graphs and cost tables of arbitrary size. As with the previous use case, the task graphs were transformed automatically to SysML models, which were then input to the Design-Space Exploration workflow. For all tasks graphs, the platform shown in Figure 5.13 was used, which represents a star-shaped platform, with three satellite PUs connected by point-to-point buses to a central Processing Unit. In total, six distinct task graphs were generated, each containing between 15 and 75 tasks, and 20 to 100 connections.

The analysis of this case study is motivated by the mapping evaluation process that is explained in sections 4.1, 4.2 and 4.3, and their consequence on the size of the task graphs that are evaluated. Those sections describe a workflow to transform Mapped Task Graphs (MTGs) in order to represent communication and concurrency without departing from the SDFG Model of Computation, and still allowing the use of the analysis algorithms implemented in the SDF3 toolset.

These transformations have the consequence that not only the actual task graphs that are evaluated by the SDFG analysis algorithms are larger than the original, but the exact size also varies with each mapping. Namely, tasks that communicate with each other and are located in different PUs will create additional tasks in order to represent the effort related to communication, whereas if the same tasks were mapped to the same Processing Unit no additional tasks would be created.

In order to quantify the penalty incurred by using the proposed task graph transformation scheme, we have measured the average number of tasks in the transformed task graph for each

Figure 5.13 – Platform used with the randomly-generated SDFGs.



Source: Author (2015).

of the random-generated applications that were evaluated. The result is presented in Table 5.4. In this table, the measure of most interest is the ratio between the number of tasks in the original application task graph and the average number of tasks in the transformed task graph.

Table 5.4 – Comparison between number of tasks in the original and the transformed SDF Graphs.

| Original | Transformed | Ratio |
|---|---|---|
| 15 | 81.45 | 5.4 |
| 24 | 99.04 | 4.1 |
| 39 | 179.46 | 4.6 |
| 50 | 254.14 | 5.1 |
| 67 | 348.33 | 5.2 |
| 75 | 375.13 | 5.0 |

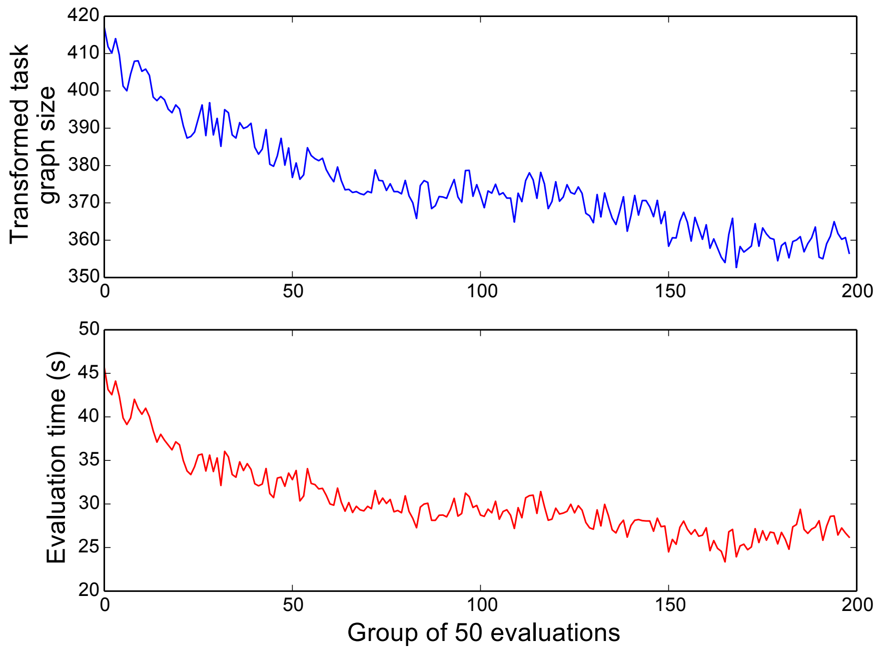
Source: Author (2015).

As the table shows, our workflow incurs in a penalty very close to 5 times in the task graph size. Furthermore, variations in the size of the application task graph do not seem to have an influence on this penalty (i.e. the penalty has a linear behavior), which indicates that this mode of evaluation should scale well even for applications with a large number of tasks.

However, these average task counts by themselves do not provide information about the quality of these mappings. Since the population of solutions is initialized randomly, it stands to reason that initially many candidate mappings will present an excessive amount of communication. Because the communication cost is not negligible, as the optimization process progresses it is expected that the population will converge towards solutions with less overall communication. Even if we consider that more communication may help in some cases, logic dictates that it is not necessarily true that an increase in communication will always improve at least one objective. Together with the transformations in the task graphs that were previously discussed, this means that as the population evolves, the task graphs that are evaluated should become smaller, and be evaluated more quickly.

In order to test this hypothesis, we have examined the time spent to evaluate solutions in the 75-task application, and the results are presented in Figure 5.14. In that figure, each point in the graph represents the average time taken to evaluate a group of 50 solutions. As the system optimization progresses, it can be seen that the size of the transformed task graphs diminishes accordingly. This not only indicates a relation between graph size and evaluation time (which is to be expected), but also shows that the exploration is converging towards a set of solutions with an overall smaller communication cost, as should be expected considering the population's random initialization.

Figure 5.14 – Progress of the task count and evaluation time for the 75-task SDFG.

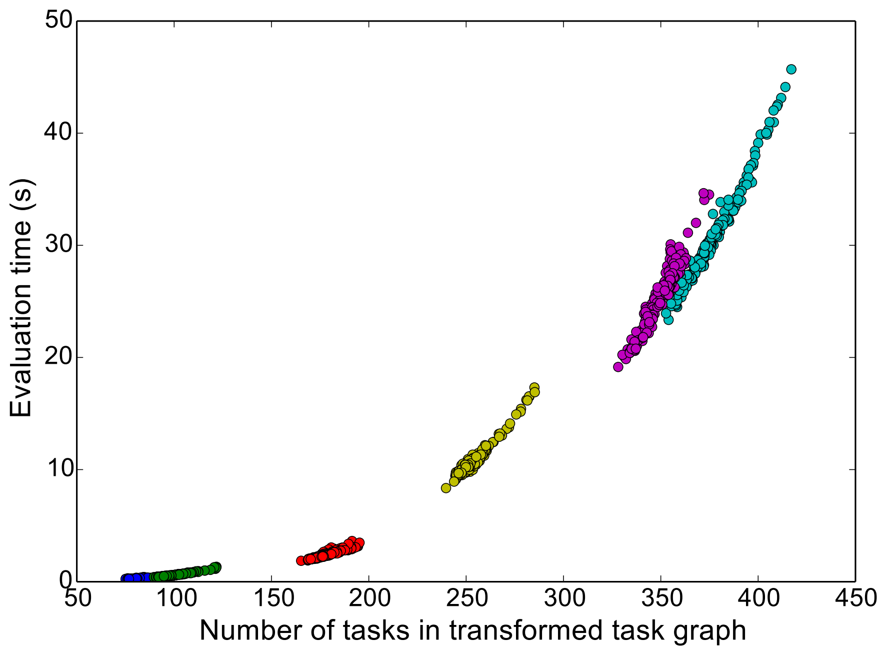


Source: Author (2015).

However, this improvement does not happen monotonically as one might expect; this is due to the randomizing nature of the optimization metaheuristic, which considers solutions in a wider search space around solutions already present in the population. Most of the time, the solutions obtained in this manner will not be better than the existing ones and will be discarded, but this procedure is important in order to escape local minimas in the search space.

Another point of interest is to directly examine the influence of the transformed task graph size on the evaluation time. For this, it is best to discard the information regarding the evaluation progress, viewing the evaluation time as a function of the task graph size, as the scatter plot presented in Figure 5.15. Each point in this figure represents the average time to evaluate 50 consecutive solutions, for all the task graphs. The points from each task graph form clusters with different tendencies, even when the task sizes are similar — this is seen especially in the larger examples. This is an indication that the SDFG evaluation time is influenced by the task graph structure in more complex ways than simply its size.

Figure 5.15 – SDFG evaluation time as a function of the task count.



Source: Author (2015).

Also, although it is possible to observe that the number of tasks has a very steep influence on the time it takes to evaluate each task graph, regression experiments with the information available have failed to conclusively indicate whether this represents a polynomial (possibly cubic) or an exponential relation. The total execution time for each DSE experiment in this chapter is shown in Table 5.5, confirming the rapid increase of the time required to run the DSE process when the task graph increases in size, even though for the more realistically-sized problems (JPEG encoder and E3S benchmark) the cost remains reasonable.

## 5.4 Discussion

Given the results gathered in this chapter, it is possible to say that the use of a framework that contains DSE capabilities can provide engineers with valuable insight into the systems they are designing.

It can also be said that the use of a Design-Space Exploration framework in no manner takes responsibility away from engineers, but focuses on providing them with the necessary insight to make the best-informed decision possible. As we have shown, the designer is still responsible for configuring the DSE tools — in other words, encoding the project's goals into the tools. As

Table 5.5 – Total execution time for the DSE experiments.

| Number of tasks | DSE time |
|---|---|
| Random (15 tasks) | 0:09:40 |
| JPEG encoder | 0:12:16 |
| E3S | 0:15:34 |
| Random (24 tasks) | 0:23:06 |
| Random (39 tasks) | 1:44:26 |
| Random (50 tasks) | 7:41:03 |
| Random (67 tasks) | 17:34:15 |
| Random (75 tasks) | 21:20:47 |

was discussed, a direct, "optimize everything" approach is often not desirable, and the designer must reason about what are the metrics that must be optimized, define constraints for the system under development, and so on.

We have also evaluated the time necessary for the evaluation of solutions using our approach. Although it presents a good execution time for the smaller task graphs, evaluation time remains considerable when working with very large and complex task graphs.

Oliveira (2013) presented performance experiments with both E3S benchmarks and Task Graphs For Free (TGFF)-generated synthetic task graphs. The exact parameters of the experiments differ, especially the platform definitions used, preventing a direct comparison of the results. Furthermore, that work considered only the time used to *generate* the solutions before they were evaluated, while here we have considered the entire generation and evaluation process.

The type of platform and task graph that were considered here in the TGFF experiments are consistent with the worst-case scenario in Oliveira's work. In their experiments, they have shown that the time for solution generation seems to grow exponentially, which is consistent with our results for the entire evaluation process. Their experiments have also shown that this type of worst-case synthetic task graphs are very different from real-life ones, which tend to be much simpler. In those experiments, the computational cost for the task graphs from the E3S benchmark set was shown to be much smaller when compared to the complex randomly-generated ones. This, together with the similarity of their results and the ones presented here, increases the confidence that our approach is feasible for real-life design problems.

# 6 CONCLUSIONS AND FUTURE WORK

In this work, an automated Design-Space Exploration workflow that was implemented as part of the toolset that supports the HIPAO2 development methodology for image processing systems was presented. These tools use Synchronous Dataflow Graph (SDFG) analysis algorithms to feed a multiobjective genetic algorithm (NSGA-II), in order to help systems engineers solve what is called the Mapping Problem for heterogeneous multiprocessor systems. In order to adapt the analysis algorithms to the type of platform that is targeted by HIPAO2, transformations to the original task and platform graphs were proposed.

To validate the proposed toolset, the Design-Space Exploration (DSE) process was applied to three different case studies. The results obtained show that the developed tool is able to take into account communication costs in the systems under design, minimizing superfluous communication between Processing Units (PUs). It was also shown that, contrary to what was argued in other works in the literature, it is not possible to reduce the complexity of the multiobjective nature of the Mapping Problem, by ignoring certain optimization objectives. Our results indicate that, although two objectives might not present tradeoffs among themselves, not taking these into account yields a very different Pareto front.

The results obtained with these case studies also show that our workflow can give engineers the insight needed to make informed decisions pertaining to the mapping problem. They have also shown that, although the exploration times are reasonable for applications with a small to medium number of components, the time needed to evaluate solutions can still be a problem for very large task graphs (although it is still much faster than a full-system simulation).

Possible future work directions may include the support of automated DSE for other design decisions, since the tool that was presented here only supports Design-Space Exploration for the Mapping Problem. The workflow and methods employed here do not scale to any design decision, unlike other prior art. The factors that limit the extension of our proposal are the solution encoding and evaluation algorithms, which are not easily modified to support decisions such as the Construction Problem. Nevertheless, other DSE activities, such as the Configuration Problem, can be represented in ways similar to the Mapping Problem, and therefore a future extension of this tool to support it might be feasible, as long as an evaluation method can be provided. The fact that the evaluation algorithm and the metaheuristic encoding are highly decoupled makes this possible.

Another point of improvement would be to support additional Models of Computation, in order to support more applications. Accomplishing this would require defining algorithms to evaluate different solutions according to other Models of Computation (MoCs). Again, this would require little or no changes to the encoding used by the metaheuristic, making this work feasible.

Similarly, another line of investigation would be to extend this tool to support more complex kinds of inter-PU communication, such as Networks-on-Chip (NoCs). In this work we have shown a way to represent platforms that are formed by PUs that are connected by certain types of buses (which include point-to-point connections as well as bidirectional, multipoint buses). It is not clear at this point whether it is possible to support these more complex hardware platforms by extending the transformations to the task and platform graph that were proposed here, or if the entire evaluation process would need to be changed.

Furthermore, more studies can be made on aspects directly related to the optimization metaheuristics that are used in the proposed workflow. One interesting aspect with potential for practical improvements is the study of stopping conditions for the optimization process. This can help avoid performing incomplete exploration of the solution space (stopping before better solutions can be found) and preventing unnecessary or redundant exploration effort (wasting time and computing resources). Furthermore, other types of metaheuristics, or different parameters for the metaheuristics, can be experimented with.

# BIBLIOGRAPHY

ABDELHALIM, M.; HABIB, S. E.-D. Particle swarm optimization for HW/SW partitioning. In: LAZINICA, A. (Ed.). **Particle Swarm Optimization**. [s.n.], 2009. p. 49–76. ISBN 978-953-7619-48-0. Disponível em: <http://dx.doi.org/10.5772/6740>. Acesso em: 31 mar. 2015.

ANGUS, D. Crowding population-based ant colony optimisation for the multi-objective travelling salesman problem. In: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE IN MULTICRITERIA DECISION MAKING, 2007, Honolulu, HI. **Proceedings...** [S.l.]: IEEE, 2007. p. 333–340. ISBN 1-4244-0702-8.

BASTEN, T. et al. Model-driven design-space exploration for embedded systems: The octopus toolset. In: MARGARIA, T.; STEFFEN, B. (Ed.). **Leveraging Applications of Formal Methods, Verification, and Validation**. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6415). p. 90–105. ISBN 978-3-642-16557-3.

BÉZIVIN, J. On the unification power of models. **Software & Systems Modeling**, Springer-Verlag, v. 4, n. 2, p. 171–188, 2005. ISSN 1619-1366.

BHATTACHARYYA, S. S.; DEPRETTERE, E. F.; THEELEN, B. D. Dynamic dataflow graphs. In: **Handbook of Signal Processing Systems**. [S.l.]: Springer, 2013. p. 905–944.

BINOTTO, A. P. D. et al. A CPU, GPU, FPGA system for X-ray image processing using high-speed scientific cameras. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH-PERFORMANCE COMPUTING, 25., 2013, Porto de Galinhas, Brazil. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2013. p. 113–119.

BONFIETTI, A. et al. Throughput constraint for synchronous data flow graphs. In: INTERNATIONAL CONFERENCE ON INTEGRATION OF AI AND OR TECHNIQUES IN CONSTRAINT PROGRAMMING FOR COMBINATORIAL OPTIMIZATION PROBLEMS, 6., 2009, Pittsburgh, PA, USA. **Proceedings...** [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5547). p. 26–40. ISBN 978-3-642-01928-9.

BOUQUET, F. et al. Transformation of SysML structure diagrams to VHDL-AMS. In: SECOND WORKSHOP ON DESIGN, CONTROL AND SOFTWARE IMPLEMENTATION FOR DISTRIBUTED MEMS, 2., 2002, Besançon, France. **Proceedings...** [S.l.]: IEEE, 2012. p. 74–81.

CALBOREAN, H. et al. Optimizing a superscalar system using multi-objective design space exploration. In: INTERNATIONAL CONFERENCE ON CONTROL SYSTEMS AND COMPUTER SCIENCE, 18., 2011, Bucharest, Romania. **Proceedings...** Bucharest, Romania: Politehnica Press, 2011. v. 1, p. 339–346.

CARAMIA, M.; DELL'OLMO, P. Multi-objective optimization. In: **Multi-objective Management in Freight Logistics**. [S.l.]: Springer London, 2008. p. 11–36. ISBN 978-1-84800-381-1.

CARKCI, M. **Dataflow and Reactive Programming Systems: A Practical Guide**. [S.l.]: CreateSpace Independent Publishing Platform, 2014. ISBN 978-1497422445.

CHOCKALINGAM, T.; ARUNKUMAR, S. Genetic algorithm based heuristics for the mapping problem. **Computers & Operations Research**, v. 22, n. 1, p. 55–64, 1995. ISSN 0305-0548.

DASDAN, A. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. **ACM Transactions on Design Automation of Electronic Systems**, ACM, New York, NY, USA, v. 9, n. 4, p. 385–418, Oct. 2004. ISSN 1084-4309.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. **Evolutionary Computation, IEEE Transactions on**, v. 18, n. 4, p. 577–601, Aug. 2014. ISSN 1089-778X.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. **IEEE Transactions on Evolutionary Computation**, v. 6, n. 2, p. 182–197, 2002. ISSN 1089-778X.

DICK, R. **Embedded System Synthesis Benchmarks Suite**. 2002. Disponível em: <http://ziyang.eecs.umich.edu/~dickrp/e3s/>.

DICK, R.; RHODES, D.; WOLF, W. TGFF: task graphs for free. In: INTERNATIONAL WORKSHOP ON HARDWARE/SOFTWARE CODESIGN, 6., 1998, Seattle, WA, USA. **Proceedings...** [S.l.]: IEEE, 1998. p. 97–101. ISBN 0-8186-8442-9. ISSN 1092-6100.

DICK, R. P.; JHA, N. K. MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1997, San Jose, California, USA. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 1997. p. 522–529. ISBN 0-8186-8200-0.

DOERING, D. A model driven engineering methodology for embedded system designs - HIPAO2. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS, 12., 2014, Porto Alegre, Brazil. **Proceedings...** [S.l.]: IEEE, 2014. p. 787–790. ISBN 978-1-4799-4905-2.

DOERING, D. **Um método para projetar sistemas embarcados baseado na metodologia de engenharia dirigida por modelos aplicado à sistemas de processamento de imagens**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Set 2015.

DOERING, D. et al. A model driven engineering approach based on aspects for high speed scientific x-rays cameras. In: IEEE INTERNATIONAL SYMPOSIUM ON OBJECT/COMPONENT/SERVICE-ORIENTED REALTIME DISTRIBUTED COMPUTING, 16., 2013, Paderborn, Germany. **Proceedings...** [S.l.]: IEEE, 2013.

DORIGO, M. **Optimization, Learning and Natural Algorithms (in Italian)**. Tese (Doutorado) — Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

EHRGOTT, M. **Multicriteria Optimization**. [S.l.]: Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27659-3.

ERBAS, C.; CERAV-ERBAS, S.; PIMENTEL, A. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. **IEEE Transactions on Evolutionary Computation**, v. 10, n. 3, p. 358–374, June 2006. ISSN 1089-778X.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

GAŠEVIC, D.; DJURIC, D.; DEVEDŽIC, V. **Model Driven Engineering and Ontology Development**. [S.l.]: Springer Berlin Heidelberg, 2009.

GHAMARIAN, A.-H. et al. Throughput analysis of synchronous data flow graphs. In: INTERNATIONAL CONFERENCE ON APPLICATION OF CONCURRENCY TO SYSTEM DESIGN, 6., 2006, Turku, Finland. **Proceedings...** [S.l.]: IEEE, 2006. p. 25–36. ISBN 0-7695-2556-3. ISSN 1550-4808.

GRIES, M. Methods for evaluating and covering the design space during early design development. **Integration, the VLSI Journal**, Elsevier, v. 38, n. 2, p. 131–183, Dec. 2004.

HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.** [S.l.]: University of Michigan Press, 1975.

ITU. **ISO/IEC 10918-1 : 1993(E) CCIT Recommendation T.81**. 1993. Disponível em: <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>.

JAHR, R. et al. Automatic multi-objective optimization of parameters for hardware and code optimizations. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND SIMULATION, 2011, Istanbul. **Proceedings...** [S.l.]: IEEE, 2011. p. 308–316.

JERRAYA, A. A. et al. Roundtable: Envisioning the future for multiprocessor soc. **IEEE Design & Test of Computers**, IEEE Computer Society, Los Alamitos, CA, USA, v. 24, n. 2, p. 174–183, Mar.–Apr. 2007. ISSN 0740-7475.

KAHN, G. The semantics of a simple language for parallel programming. In: INFORMATION PROCESSING, 1974, Stockholm, Sweden. **Proceedings of the IFIP Congress**. Amsterdam: North-Holland, 1974. p. 471–475. ISBN 0-7204-2803-3.

KARP, R. A characterization of the minimum cycle mean in a digraph. **Discrete Mathematics**, Elsevier BV, v. 23, n. 1, p. 309–311, 1978.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1995, Perth, Australia. **Proceedings...** [S.l.], 1995. v. 4, p. 1942–1948 vol.4.

KENNEDY, J. et al. **Swarm intelligence**. San Francisco: Morgan Kaufmann Publishers, 2001. (Evolutionary Computation Series). ISBN 9781558605954.

KIENHUIS, B. et al. A methodology to design programmable embedded systems - the Y-chart approach. In: DEPRETTERE, E.; TEICH, J.; VASSILIADIS, S. (Ed.). **Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation — SAMOS**. London, UK: Springer Berlin Heidelberg, 2002, (Lecture Notes in Computer Science, v. 2268). p. 18–37. ISBN 978-3-540-43322-4.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 671–680, 1983.

LEE, E.; MESSERSCHMITT, D. Synchronous data flow. **Proceedings of the IEEE**, v. 75, n. 9, p. 1235–1245, Sept. 1987. ISSN 0018-9219.

LEE, E. A. Cyber physical systems: Design challenges. In: IEEE INTERNATIONAL SYMPOSIUM ON OBJECT ORIENTED REAL-TIME DISTRIBUTED COMPUTING, 11., 2008, Orlando, FL, USA. **Proceedings...** Los Alamitos, CA, USA: IEEE, 2008. p. 363–369. ISBN 978-0-7695-3132-8.

LEE, T.-Y. et al. Enhancement of hardware-software partition for embedded multiprocessor FPGA systems. In: INTERNATIONAL CONFERENCE ON INTELLIGENT INFORMATION HIDING AND MULTIMEDIA SIGNAL PROCESSING, 3., 2007, Kaohsiung, Taiwan. **Proceedings...** Los Alamitos, CA, USA: IEEE, 2007. v. 1, p. 19–22. ISBN 978-0-7695-2994-1.

LIU, H.-Y. et al. Supervised design space exploration by compositional approximation of pareto sets. In: DESIGN AUTOMATION CONFERENCE, 48., 2011, San Diego, CA, USA. **Proceedings...** New York, NY, USA: ACM, 2011. p. 399–404. ISBN 978-1-4503-0636-2.

LUC, D. T. Pareto optimality. In: CHINCHULUUN, A. et al. (Ed.). **Pareto Optimality, Game Theory And Equilibria**. [S.l.]: Springer New York, 2008, (Springer Optimization and Its Applications, v. 17). p. 481–515. ISBN 978-0-387-77246-2.

LUKE, S. **Essentials of Metaheuristics**. second ed. Lulu, 2013. Disponível em: <http://cs-.gmu.edu/~sean/book/metaheuristics/>.

MATTOS, J. C. B. d. **Design space exploration of SW and HW IP based on object oriented methodology for embedded system applications**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2007.

MIRAMOND, B.; DELOSME, J.-M. Design space exploration for dynamically reconfigurable architectures. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION, 2005, Munich, Germany. **Proceedings...** Los Alamitos, CA, USA: IEEE, 2005. v. 1, p. 366–371. ISBN 0-7695-2288-2. ISSN 1530-1591.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: MIT Press, 1996. ISBN 0-262-13316-4.

NATH, P. K.; DATTA, D. Multi-objective hardware-software partitioning of embedded systems: A case study of JPEG encoder. **Applied Soft Computing**, v. 15, n. 0, p. 30–41, Feb. 2014. ISSN 1568-4946.

NEEMA, S. et al. Constraint-based design-space exploration and model synthesis. In: ALUR, R.; LEE, I. (Ed.). **Embedded Software**. [S.l.]: Springer Berlin Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2855). p. 290–305. ISBN 978-3-540-20223-3.

OLIVEIRA, M. F. d. S. **Model Driven Engineering Methodology for Design Space Exploration of Embedded Systems**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, Dec. 2013.

OMG. **UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems**. [S.l.], 2011.

OMG. **Unified Modeling Language (UML)**. [S.l.], 2011. Disponível em: <http://www.omg-.org/spec/UML/2.4.1>. Acesso em: 23 August 2015.

OMG. **OMG Systems Modeling Language (SysML$^{tm}$)**. [S.l.], 2012. Disponível em: <http://www.omg.org/spec/SysML/1.3>. Acesso em: 23 August 2015.

PALERMO, G.; SILVANO, C.; ZACCARIA, V. ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 28, n. 12, p. 1816–1829, 2009. ISSN 0278-0070.

PAPPALARDO, M. Multiobjective optimization: A brief overview. In: CHINCHULUUN, A. et al. (Ed.). **Pareto Optimality, Game Theory And Equilibria**. [S.l.]: Springer New York, 2008, (Springer Optimization and Its Applications, v. 17). p. 517–528. ISBN 978-0-387-77246-2.

PAPYRUS UML website. Disponível em: <https://www.eclipse.org/papyrus/>.

PIMENTEL, A.; ERBAS, C.; POLSTRA, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. **IEEE Transactions on Computers**, v. 55, n. 2, p. 99–112, Feb. 2006. ISSN 0018-9340.

PISCITELLI, R.; PIMENTEL, A. D. Design space pruning through hybrid analysis in system-level design space exploration. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2012, Dresden, Germany. **Proceedings...** San Jose, CA, USA: EDA Consortium, 2012. p. 781–786. ISBN 978-3-9810801-8-6. ISSN 1530-1591.

PRAKASH, S.; PARKER, A. C. Synthesis of application-specific multiprocessor architectures. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 28., 1991, San Francisco, California, USA. **Proceedings...** New York, NY, USA: ACM, 1991. p. 8–13. ISBN 0-89791-395-7.

PRETSCHNER, A. et al. Software engineering for automotive systems: A roadmap. In: FUTURE OF SOFTWARE ENGINEERING, 2007, Minneapolis, MN, USA. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2007. p. 55–71. ISBN 0-7695-2829-5.

QIAN, K.; HARING, D.; CAO, L. **Embedded Software Development with C**. 1. ed. [S.l.]: Springer US, 2009. ISBN 978-1-4419-0606-9.

SAXENA, T.; KARSAI, G. A meta-framework for design space exploration. In: IEEE INTERNATIONAL CONFERENCE AND WORKSHOPS ON ENGINEERING OF COMPUTER BASED SYSTEMS, 18., 2011, Las Vegas, NV, USA. **Proceedings...** Piscataway, NJ, USA: IEEE, 2011. p. 71–80. ISBN 978-1-4577-0065-1.

SELIC, B.; MOTUS, L. Using models in real-time software design. **IEEE Control Systems Magazine**, v. 23, n. 3, p. 31–42, June 2003. ISSN 1066-033X.

SILVANO, C. et al. MULTICUBE: Multi-objective design space exploration of multi-core architectures. In: VOROS, N. et al. (Ed.). **VLSI 2010 Annual Symposium**. [S.l.]: Springer Netherlands, 2011, (Lecture Notes in Electrical Engineering, v. 105). p. 47–63. ISBN 978-94-007-1487-8.

SÖRENSEN, K. Metaheuristics-the metaphor exposed. **International Transactions in Operational Research**, Wiley-Blackwell, v. 22, n. 1, p. 3–18, Fev. 2013. ISSN 0969-6016.

SRIRAM, S.; BHATTACHARYYA, S. S. **Embedded Multiprocessors: Scheduling and Synchronization**. 1st. ed. New York, NY, USA: Marcel Dekker, Inc., 2000. ISBN 0824793188.

STAHL, T.; VOELTER, M. **Model-Driven Software Development: Technology, Engineering, Management**. [S.l.]: Wiley, 2006. (Wiley Software Patterns Series). ISBN 0470025700.

STEINBERG, D. et al. **EMF: Eclipse Modeling Framework**. 2nd revised. ed. [S.l.]: Addison-Wesley, 2008. ISBN 9780321331885.

STUIJK, S.; GEILEN, M.; BASTEN, T. SDF$^3$: SDF For Free. In: INTERNATIONAL CONFERENCE ON APPLICATION OF CONCURRENCY TO SYSTEM DESIGN, 6., 2006, Turku, Finland. **Proceedings...** Los Alamitos, CA, USA: IEEE Computer Society Press, 2006. p. 276–278.

TALBI, E. **Metaheuristics: From Design to Implementation**. [S.l.]: Wiley, 2009. (Wiley Series on Parallel and Distributed Computing). ISBN 9780470496909.

TRČKA, N. et al. Integrated model-driven design-space exploration for embedded systems. In: INTERNATIONAL CONFERENCE ON EMBEDDED COMPUTER SYSTEMS, 2011, Samos, Greece. **Proceedings...** [S.l.]: IEEE, 2011. p. 339–346.

YANG, Y. et al. Exploring trade-offs between performance and resource requirements for synchronous dataflow graphs. In: IEEE/ACM/IFIP WORKSHOP ON EMBEDDED SYSTEMS FOR REAL-TIME MULTIMEDIA, 7., 2009, Grenoble, France. **Proceedings...** [S.l.]: IEEE, 2009. p. 96–105.

YANG, Y. et al. Automated bottleneck-driven design-space exploration of media processing systems. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION, 2010, Dresden, Germany. **Proceedings...** Leuven, Belgium: European Design and Automation Association, 2010. p. 1041–1046. ISBN 978-3-9810801-6-2.

YOUNG, N. E.; TARJAN, R. E.; ORLIN, J. B. Faster parametric shortest path and minimum balance algorithms. **Networks**, v. 2, n. 21, p. 205–221, 1991.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. **SPEA2: Improving the Strength Pareto Evolutionary Algorithm**. Zurich, Switzerland, 2001.

# APPENDIX A – IMPLEMENTATION OF THE DSE TOOL

This work in its entirety presents the development of an automated Design-Space Exploration (DSE) tool supporting the HIPAO2 development methodology, which has been discussed in a high-level manner in the main body of text. This appendix discusses the concrete implementation of the tool that is described in this work, as well as the rationale for some of the engineering decisions that were made during its development.

The HIPAO2 Model-Driven Engineering (MDE) methodology (DOERING, 2015) is heavily based on Model-to-Model (M2M) and Model-to-Text (M2T) transformations, and as such cannot be feasibly employed without tool support. For this reason, along with its methodology proposal, a plugin was developed for the open-source Eclipse Integrated Development Environment (IDE). This plugin builds upon the well-developed MDE support infrastructure provided by this software via its Eclipse Modeling Framework (EMF) (STEINBERG et al., 2008) and Papyrus (PAPYRUS. . . , ) subprojects which, among other things, implement Application Programming Interfaces (APIs) for working with the Unified Modeling Language (UML) and Systems Modeling Language (SysML) metamodels, as well as a comprehensive graphical editor for such models. Without this infrastructure, it would not have been possible to implement the HIPAO2 support tools in a reasonable timeframe.

However, the DSE tool described in this work is not provided as a part of this plugin. Several reasons exist for this, most important among them is that the DSE tool performs a task much different than the rest of the plugin, so that little would be gained in terms of code reuse by bundling them together. Also, in terms of lines of Java code, the DSE tool (4854 lines) is bigger than the HIPAO2 plugin itself (4347 lines of code).[1]

Ease of development is also a motivating factor. Both of these tools require a non-trivial set of third-party dependencies such as the *jMetal* and Synchronous DataFlow For Free (SDF3) libraries for the DSE tool, and the Eclipse platform itself for the plugin.

The remainder of this appendix discusses both sides of the implementation (HIPAO2 plugin and standalone DSE tool). However, the Eclipse plugin is shown only in passing, presenting only the features that are directly related to the integration with the DSE tool which is the subject of this text. Readers interested in the HIPAO2 methodology and its tools as a whole should refer to Doering (2015).

## A.1 HIPAO2 Eclipse plugin

Models in HIPAO2 are standard SysML models, stored in disk in the industry-standard XML Metadata Interchange (XMI) format, as supported by the Eclipse EMF framework and editors. However, these models include more information than is needed to perform DSE, and this complexity is reflected in the APIs used to programmatically process them. Therefore, the

[1] The latter, however, also includes code in other languages that were not counted here.

decision was made to make the DSE tool not have full SysML models as its input, but rather a simplified model with only the information that is needed, serialized in a simple Javascript Object Notation (JSON) schema. The information that these models contain, as well as the format they are encoded in, are presented in Section A.2.1.

What this means in relation to the HIPAO2 Eclipse plugin is that some conversion from the SysML models prepared by the designer is needed. For this, an export functionality is provided in the tools for both Platform Independent Models (PIMs) and Platform Models (PMs). These are surfaced in the Eclipse User Interface (UI) in more than one way, as can be seen in Figure A.1. Additionally, the Platform Independent Model (PIM) export functionality is also provided via a commandline interface, allowing the batch export of SysML PIMs (this functionality was used primarily to ease working with the various, often automatically generated, models used in Chapter 5.

## A.2 DSE tool

The DSE tool itself is provided as a standalone Java commandline program. Its input data (PIM, PM, and costs table, as shown in Figure 4.1) are received as JSON and Comma-Separated Values (CSV) files. Another JSON input file informs the configuration for the DSE process, defining the objectives that should be optimized as well as the constraints that must be respected. JSON files are human-readable at the same time that they are easy to parse and validate, providing an easy way to perform input to the program. In addition to the aforementioned advantages that separating this tool from the HIPAO2 Eclipse plugin brings, this scheme of having all inputs be human-readable text files lends itself well to experimentation, allowing the inclusion of these files in version control, among other conveniences. The following section details the input and output formats used by our tool.

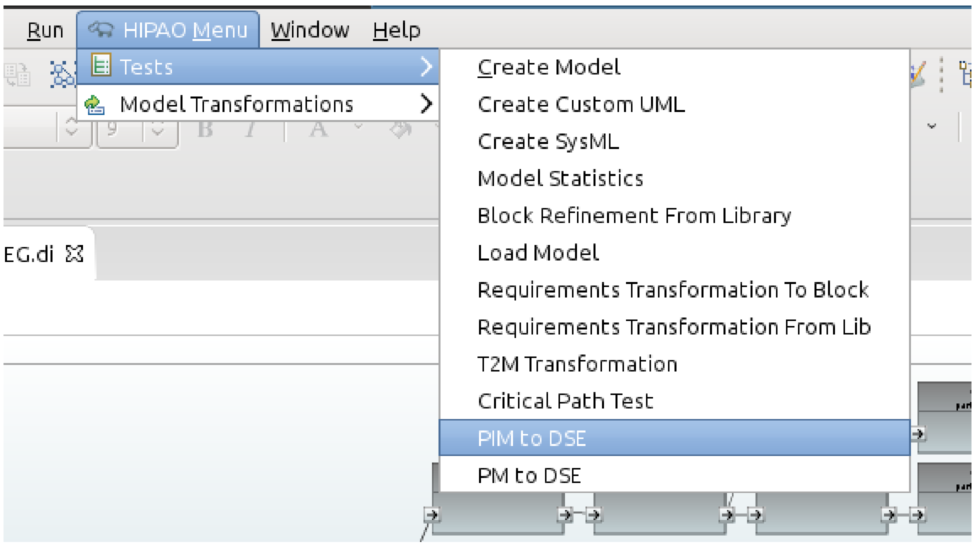
### A.2.1 Inputs and outputs

The three data inputs to the normalization pipeline are the Platform Independent Model (PIM), Platform Model (PM), and the task cost table. The two models consist on a graph structure, plus information about the nodes (tasks in the case of the PIM, Processing Units and buses for the PM). These are serialized as JSON files, a simple text-based format that is easily parsed by code and is also human-readable.[2]

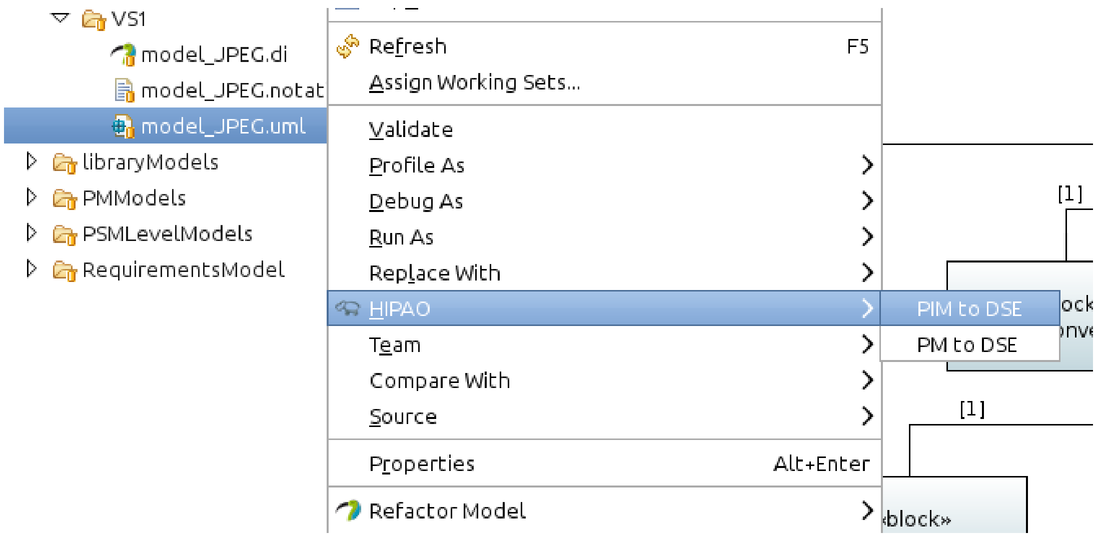
The main rationale of using this format for input to the DSE tool and not SysML models was code complexity, as well as eliminating a large dependency on Eclipse libraries. Aiming for simplicity, this encoding includes only the information that is needed for the DSE process, which is exported automatically by the HIPAO2 plugin as explained previously. The information that is extracted for PIMs is shown in Table A.1, and the data exported for a PM, Table A.2.

[2] The JSON grammar can be found at <http://www.json.org>.

Figure A.1 – Export functionality from SysML models to the DSE tool format.



(a) Export functionality available as a context menu in the Eclipse *Package Explorer*.



(b) Export functionality available in the window menu.

Source: Author (2015).

Table A.1 – Information extracted from the SysML for the Platform Independent Model

| PIM element | Extracted information |
|---|---|
| Task | Name<br>Class |
| Connection | Name of endpoints |

Source: Author (2015).

Table A.2 – Information extracted from the SysML for the Platform Independent Model

| PM element | Extracted information |
|---|---|
| Block | Name |
| | Class |
| | Stereotype list |
| | Stereotype attributes |
| Connection | Name of endpoints |

Source: Author (2015).

An important disadvantage of the JSON format, when compared to other competitor standards such as Extensible Markup Language (XML) and Protocol Buffers[3], is that it is schemaless; for any real-life work, some sort of format definition is needed. To fill this need, in this work we have employed the *Gson*[4] library, which allows serializing and deserializing plain Java objects as JSON. This means that the schema is *de facto* defined by the Java objects that were defined for this interface.

The third input, the cost table, is provided as a Comma-Separated Values (CSV) file, with predefined meaning for each column, as seen in Table A.3. The two first columns are used as keys; they indicate the name of the task and Processing Unit (PU) types that the entry refers to. These names must match the names used in the SysML models. Figure A.2 shows an excerpt from the costs file used in the JPEG encoder used in Section 5.1.

Table A.3 – Meaning of the columns in the cost table CSV file.

| Column | Meaning |
|---|---|
| 1 | Task type name |
| 2 | PU type name |
| 3 | Execution time |
| 4 | Power |
| 5 | Hardware Area |
| 6 | Memory requirement |

Source: Author (2015).

Apart from the three aforementioned data inputs, it is also necessary to provide important configurations for the DSE process, such as what objectives will be optimized, and which (if any) constraints must be met. This is also provided by a JSON file, parsed in the same way (into a Java object) as the PIM and PM. The full configuration file used in the first JPEG encoder case study in Section 5.1 can be seen in Figure A.3. That figure shows that constraints are defined in terms of a metric (which are not necessarily an optimization objective) and predicates; these

[3] A strongly-typed, binary format for data exchange and serialization, available at <http://developers.google.com/protocol-buffers>.

[4] This library may be found at <http://www.github.com/google/gson>.

Figure A.2 – Part of the costs table file used for the JPEG encoder case study.

```
LevelOffset  ; CPU       ; 9380     ; 0.096 ; 0        ; 0.00058
LevelOffset  ; FPGAHwAcc ; 155.264  ; 4     ; 0.00731  ; 0
DCT          ; CPU       ; 20000000 ; 45    ; 0        ; 0.00288
DCT          ; FPGAHwAcc ; 1844.822 ; 274   ; 0.378    ; 0
quantization ; CPU       ; 34700    ; 0.26  ; 0        ; 0.00193
quantization ; FPGAHwAcc ; 3512.32  ; 3     ; 0.011    ; 0
DPCM         ; CPU       ; 940      ; 0.957 ; 0        ; 0.000677
DPCM         ; FPGAHwAcc ; 5.334    ; 15    ; 0.002191 ; 0
ZigZag       ; CPU       ; 13120    ; 0.069 ; 0        ; 0.000911
```

Source: Author (2015).

can be composed flexibly by the user. The concepts of metrics, objectives and constraints as they are used in this tool are discussed in Section A.2.2.

Figure A.3 – JSON configuration file used for the JPEG encoder experiment.

```
{
  "objectives": [
    "EXEC_TIME",
    "POWER"
  ],
  "constraints": [
    { "requiredMetric": "MEMORY_OVERUSE",
      "predicates": [{"type": "EQUALS_ZERO"}]
    },
    { "requiredMetric": "HARDWARE_AREA_OVERUSE",
      "predicates": [{"type": "EQUALS_ZERO"}]
    },
    { "requiredMetric": "POWER",
      "predicates": [{
        "type": "LESS_THAN",
        "value": 600.0
      }]
    }
  ]
}
```

Source: Author (2015).

The product of the DSE tool explained here consists in the Pareto front that is found for the input design. Concretely, our tool outputs text files that contain the encoding and the objective values of the Pareto-optimal solutions, along with the interpretation of each field in the Genetic Algorithm (GA) encoding. Although the information contained in these files is sufficient to correctly interpret the output solutions and generate a SysML model containing the Platform Specific Model (PSM), this has not been implemented.

In addition to that output, a series of utility information is also emitted, such as a graph visualization of the input PIM and PM (in *Graphviz*[5] format), and logs for key components of the tool (such as each metric evaluator). These logs are used for debugging and for generating secondary information about the execution, such as measuring the evaluation time for the experiments in Section 5.3.

### A.2.2 Metrics, Objectives and Constraints

Even though the optimization metaheuristic may treat them differently, in the perspective of a solution evaluator there is not much difference between an objective value and a constraint (or rather a "constraint violation" value, which is what must be calculated). Both simply represent a numeric value that should be derived from a solution.

According to the DSE configuration desired by the user, one of these values may even be used as both a constraint and a solution, as we have shown in the case studies of Chapter 5. In the JPEG encoder case study shown in that chapter, for example, the power consumption value was sometimes used both as an objective to be minimized and as a constraint, and sometimes configured as only an objective. It is also conceivable that a user may want to regard power consumption as only a constraint, not trying to minimize the power consumption as long as it remains under a certain threshold. However, the user's intent does not alter *how* the power consumption value must be estimated, regardless of what will be the use for this value. This fact motivates us to find a common infrastructure that may be used to abstract the values from their usage — this was achieved in our tool by defining both constraints and objectives in terms of *metrics*.

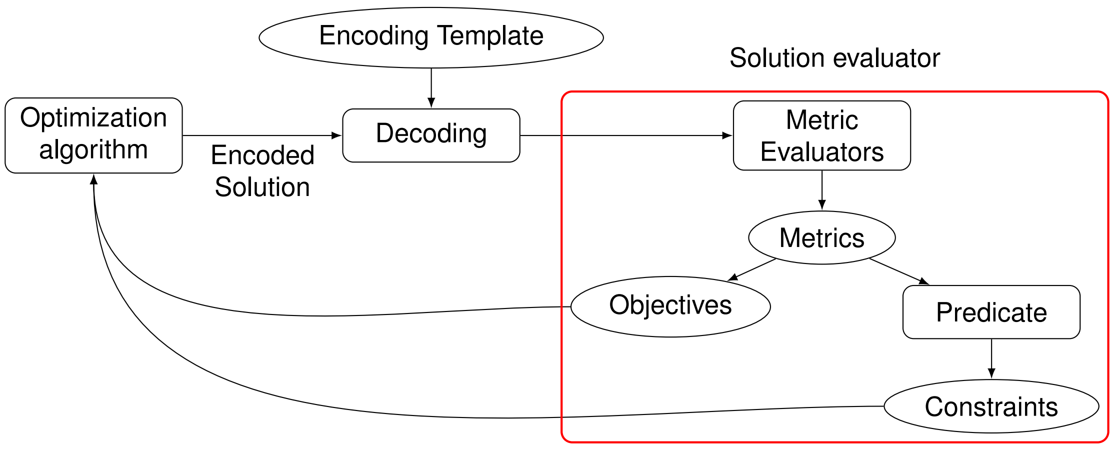
Metrics are simply abstract values that may be derived algorithmically from a candidate solution. These metrics can then be used both in objectives and in constraints. Each metric that was implemented (such as power consumption and execution time) is associated with an implementation of a metric evaluator (a concrete implementation of the `MetricEvaluator` interface). This interface provides a method that receives the Mapped Task Graph (MTG), and returns a single numerical value. The MTG includes not only the task-to-PU mapping, but also the task and platform graphs and the cost table.

The configuration file mentioned before (of which Figure A.3 is an example) specifies a list of metric names that should be the optimization objectives. Similarly, constraints are defined as a predicate over a metric, such as "Less than *x*" and "Equal to zero". Figure A.4 shows the internal workflow of the "Evaluator Algorithm" block that was mentioned in Figure 4.1, making explicit the relation between metrics, objectives and constraints.

Upon reading the configuration file during initialization, the DSE tool reads the list of objectives and constraints that were requested, determines the set of metrics that will be needed for this configuration, and instantiates the appropriate metric evaluators. This scheme has the advantage of separating the implementation of the metric evaluators from the use of the met-

[5] An open-source graph visualization package, available at <http://www.graphviz.org>.

Figure A.4 – Internal workflow of the Solution Evaluator.



Source: Author (2015).

ric (since they do not need to know how they will be used), and prevents that the same metric evaluator be called more than once (if an objective and a metric use the same metric, the metric evaluator will be called only once).

## A.3 Secondary tools and utilities

To ease developing the tool and performing the validation experiments that were presented in the main body of text, a series of secondary utilities were also created, mostly small scripts written in the Python language (a total of 23 different Python scripts, amounting to 3167 lines of code). Several of these work on the output of the DSE tool, performing functions such as:

- Allowing better visualization, such as the one used to graphically visualize a mapping (as seen in Figure 5.5);
- Calculating the Continuity Factor defined in Equation 5.1;
- Converting the JSON PIM and PM representation to a *Graphviz* figure;
- Plotting the objective space of the Pareto Front, as seen in graphs such as Figures 5.4, 5.6, and 5.12.

### A.3.1 TGFF to SysML tool

In addition to these small complimentary tools, the use of the Task Graphs For Free (TGFF) random task graph generator also necessitated the development of another standalone tool. TGFF (DICK; RHODES; WOLF, 1998) generates random task graphs and cost tables in its own text-based format, which is also the format used to provide the E3S benchmark set analyzed in Section 5.2. Since this format is very different from the one used by our tool, it was necessary to convert these, preferably in an automated manner. Also, although it would be simpler to convert directly from the TGFF format to our tool's JSON-based format, it was decided to convert the task graphs to SysML models, so that in every case study we would use the entire toolset, starting by the export from the HIPAO2 Eclipse plugin.

This tool was built as a standalone commandline Java application. In order to create and manipulate SysML models, it uses the Eclipse EMF libraries, which can imported in a standalone way, without using the entire Eclipse framework. For parsing the TGFF input files, this tool uses a parser built using the ANTLR parser generator[6] from the definition of the format's grammar. Since the SysML models are generated programmatically, they do not have the visual SysML diagrams associated with them. In total, this converter tool amounts to 1231 lines of Java code and 85 lines of ANTLR grammar definition.

[6] *Another Tool For Language Recognition.* Hosted at <http://www.antlr.org>.