UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

# Simulation-based Optimization of System-of-Systems Software Architectures

**Wallace Alves Esteves Manzano**

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

**Wallace Alves Esteves Manzano**

# Simulation-based Optimization of System-of-Systems Software Architectures

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Elisa Yumi Nakagawa

**USP – São Carlos**
**June 2023**

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

A474o

Alves Esteves Manzano, Wallace
Otimização baseada em Simulação de Arquiteturas de Sofware de Sistemas-de-Sistemas / Wallace Alves Esteves Manzano; orientadora Elisa Yumi Nakagawa. -- São Carlos, 2023.
76 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Sistemas-de-Sistemas. 2. Arquitetura de Software. 3. Otimização. 4. Aprendizado de Máquina. I. Yumi Nakagawa, Elisa, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

**Wallace Alves Esteves Manzano**

# Otimização baseada em Simulação de Arquiteturas de Sofware de Sistemas-de-Sistemas

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Prof. Dr. Elisa Yumi Nakagawa

**USP – São Carlos**
**Junho de 2023**

*To my dear grandmother who raised me and founded my path until here.*

# ACKNOWLEDGEMENTS

I first thank my family who always supported me no matter what, especially my grandmother Renailda, my aunt Ingrid, my mother Juliana, my uncle Dennis.

I also thank my advisor, Professor Elisa Yumi Nakagawa who welcomed me in my first year of graduation and has given me immense support for six years always helping and encouraging me in my academic life. I also thank Professor Valdemar Vicente Graciano Neto who was my co-advisor during my scientific initiation and since then has given me immense support. I would also like to thank Dr Thiago Bianchi who gave me immense help during the systematic mapping study carried out in my master's degree, also helping with the initial direction.

Finally, I would like to thank all the incredible professors I've had so far at ICMC and all administrative and technical servers of ICMC that give immense support to students.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 88887.700968/2022-00. This study was also financed in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Finance Code 133436/2020-9.

*“Once we’ve learned enough about the universe*
*we will admit to ourselves that we will never know everything.”*
*(Jack Kirby)*

# RESUMO

MANZANO, W. **Otimização baseada em Simulação de Arquiteturas de Sofware de Sistemas-de-Sistemas**. 2023. 76 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

A sociedade está cada vez mais dependente de serviços prestados por sistemas de software complexos e integrados, de modo que um sistema isolado não tem conseguido atender a essas demandas. Nesse cenário, os Sistemas-de-Sistemas (SoS) surgiram como resultado da interoperabilidade de sistemas independentes tanto operacional quanto gerencialmente para fornecer soluções mais complexas que nenhum sistema seria capaz de fornecer isoladamente. Devido à independência dos sistemas constituintes do SoS, esses constituintes podem sair e entrar no SoS a qualquer momento, resultando em uma arquitetura altamente dinâmica. O SoS costuma estar vinculado a tarefas críticas, ou seja, que podem representar ameaças à integridade humana. Assim, o sistema deve garantir que a arquitetura de software tenha um alto nível de qualidade para que o sistema possa oferecer um serviço confiável e sem falhas. Porém, devido ao não determinismo gerado pela arquitetura dinâmica, é necessário verificar a arquitetura em tempo de projeto. Considerando o alto custo e as ameaças de implementar sistemas críticos sem a devida verificação de sua arquitetura, este projeto de mestrado propõe um framework de otimização baseado em simulação usando meta-heurísticas para permitir a otimização de atributos de qualidade, incluindo desempenho e disponibilidade, de arquiteturas de software SoS. Este framework consiste em dois elementos principais, um processo que define como a otimização baseada em simulação é realizada no contexto da arquitetura de software SoS, e uma infraestrutura que define os requisitos de software para realizar o processo. Este framework foi avaliado por meio de um estudo de caso de um edifício inteligente, e os resultados mostram que ele é bem-sucedido em realizar com bom desempenho a tarefa de realizar a otimização baseada em simulação de uma arquitetura de software SoS.

**Palavras-chave:** Sistemas-de-Sistemas, Arquitetura de Software, Otimização, Aprendizado de Máquina.

# ABSTRACT

MANZANO, W. **Simulation-based Optimization of System-of-Systems Software Architectures**. 2023. 76 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Society is increasingly dependent on services provided by complex, integrated software systems so that an isolated system has not been successful in meeting these demands. In this scenario, Systems-of-Systems (SoS) have emerged as a result of the interoperability of independent systems both operationally and managerially to provide more complex solutions that no system would be able to provide in isolation. Due to the independence of the SoS constituent systems, these constituents can exit and enter the SoS at any time, resulting in a highly dynamic architecture. SoS is often linked to critical tasks, that is, which can pose threats to human integrity. Thus, the system must ensure that the software architecture has a high level of quality so that the system can offer a reliable and flawless service. However, due to the non-determinism generated by dynamic architecture, it is necessary to check the architecture at design time. Considering the high cost and threats of implementing critical systems without due verification of their architecture, this master's project proposes a simulation-based optimization framework using meta-heuristics to enable the optimization of quality attributes, including performance and availability, of SoS software architectures. This framework consists in two main elements, a process that defines how the simulation-based optimization is performed in the context of SoS software architecture, and an infrastructure that defines the software requirements to perform the process. This framework was evaluated using a case study of a smart building, and results shows that it is successful in carrying out with a good performance the task of performing simulation-based optimization of a SoS software architecture.

**Keywords:** Systems-of-Systems, Software Architecture, Optimization, Machine Learning.

# LIST OF FIGURES

# LIST OF TABLES

# CONTENTS

CHAPTER

# 1

# INTRODUCTION

## 1.1 Contextualization

The society has become increasingly dependent on services provided by software systems. These services have become more and more complex, so that an isolated system has not been successful in meeting these services. Software-intensive systems have been increasingly required to interoperate among themselves, communicating, exchanging, and using information exchanged[1], so becoming more ubiquitous, larger, and complex, with considerable dissemination in various sectors and application domains (NAKAGAWA *et al.*, 2013). All types of equipment are receiving components shipped with software as part of its structure (DELICATO *et al.*, 2013). In a near future, many of the things on Earth will be identifiable, addressable, and accessed through the networks. Buildings, traffic, clothes, medical equipment, houses, alarm systems, and power distribution systems, just to mention a few, will have software embedded in their intrinsic structure, potentially connected and interoperating among them and resulting in larger complex systems.

In this scenario, a distinct class of systems known as Systems-of-Systems (SoS)[2] has arisen. An SoS results from operationally and managerially independent software-intensive systems (called constituents) working together to fulfill complex missions (MAIER, 1998). SoS are often present in critical domains, such as smart traffic control, crisis response management, and national defense. Examples of SoS are the military defense systems (SMITH; HARIKUMAR; RUTH, 2011; LAAR; TRETMANS, 2013), the global satellite observation system (S.FRITZ *et al.*, 2014), and the sensor-based flood monitoring system (HORITA *et al.*, 2015). In parallel to the creation of the term smart cities in 1997 due to the Kyoto Protocol and in parallel to the more intense use of that term in recent years (DAMERI; ROSENTHAL-SABROUX, 2014), the

[1] <http://www.himss.org/library/interoperability-standards/what-is-interoperability>

[2] For sake of simplicity, herein the acronym SoS will be used interchangeably to express both singular and plural.

research in SoS has intensified, in particular, in the sense of providing innovative techniques, processes, methods, and solutions in the context of a new research area: Software Engineering for Systems-of-Systems (SESoS). In fact, several other SoS have been proposed for the domains of smart grids, smart buildings, and smart houses (BRACCO *et al.*, 2016; DELSING *et al.*, 2014; PÉREZ *et al.*, 2013). SoS have been also recognized as a research challenge for the coming years (Graciano Neto; OQUENDO; NAKAGAWA, 2016). Thus, new technologies and approaches need to be designed to make it possible to build these new class of systems.

## 1.2 Motivation

SoS is a very challenging class of software-intensive systems due to its dynamism and heterogeneity. SoS constituents sometimes refer to as COTS (Commercial-Off-The-Shelf) components, engineered in all sorts of heterogeneities comprising distinct operational systems, communication mechanisms, programming languages, and middleware technologies (GOKHALE *et al.*, 2008). Efforts have been done to establish strategies, techniques, and methods to deal with the classic concerns of software engineering regarding software-intensive SoS (CALINESCU; KWIATKOWSKA, 2010; France; Rumpe, 2007).

In fact, the additional complexity of SoS arises new problems, namely:

- Integrating constituents, allowing them to interoperate, requires working on COTS with which they are usually engineered. Indeed, SoS requires configuring middleware support to enable constituents communication, abstracting inherent heterogeneity regarding external data representation, operating systems, programming languages, network patterns, and communication mechanisms (BALASUBRAMANIAN *et al.*, 2009; BAY, 2002; FARCAS *et al.*, 2010; GOKHALE *et al.*, 2008; KAZMAN *et al.*, 2013);

- Managing constituents systems requires controlling their presence in the SoS, taking into account their intrinsic dynamics and volatility in a running SoS (BATISTA, 2013; BRYANS *et al.*, 2013);

- Configuring and deploying SoS usually relies on manually creating and handling large text files. This is necessary to ensure a suitable and correct configuration and deployment. However, this is a laborious and error-prone task, since those files have huge dimensions and high complexity (Barbi *et al.*, 2012);

- Ensuring that the software generated for SoS faithfully corresponds to models used to specify them is currently a manual process. There is a lack of specialized tools to address SoS complexities, and a low consensus due to the diversity of models and languages for SoS modeling (FARCAS *et al.*, 2010); and

- Constituents must not only perform their functions o accomplish a mission of the whole SoS, but also perform its own functionalities independently (MAIER, 1998). This requires that code of independent work and mission need to co-exist in the same software entity, claiming for an adequate modularized architecture (MAIER; RECHTIN, 2010).

In this way, simulation techniques have been used successfully to support the visualization of dynamic systems behaviors, including in Software Engineering (FRANÇA; TRAVASSOS, 2015). Simulations support the visualization of dynamic architecture with the possibility of changes taking place in the simulation elements (in this case, the SoS constituents) at runtime, in addition to predicting and anticipating emerging behaviors and unforeseen abnormalities. Discrete Event System Specification (DEVS) (ZEIGLER *et al.*, 2012) is an example of a simulation formalism that supports the representation of the functioning of a SoS. In several areas, such as Cyber-Physical Systems, Internet of Things, and Embedded Systems, simulation is being used in conjunction with optimization and machine learning techniques, seeking to improve quality attributes (TAMPOURATZIS *et al.*, 2020; FINN; NUZZO; SANGIOVANNI-VINCENTELLI, 2015).

In this sense, optimization and machine learning techniques have been highlighted as candidates to overcome some of those barriers inherent to the SoS life cycle (CHANG *et al.*, 2015; Tsang, 1989a; Badr *et al.*, 2020), providing a mechanism to automatically enhance some pre-defined quality attributes regarding simulation results, being able to access dynamic characteristics of systems such as SoS.

As a summary, the main research problem addressed in this Master's project is that, although SoS are directly linked to critical systems, their architecture is not developed in a way that assure quality attributes such as performance relevant for SoS, in part, due to the lack of methodologies and tools that support this task.

## 1.3 Objectives

This Master's project aims at contributing to SESoS by proposing a framework to enhance quality attributes of SoS software architectures. To achieve this objective, this Master's dissertation proposes a simulation-based framework to the enhancement of quality attributes of SoS system architectures. This framework uses machine learning and optimization techniques and algorithms to dynamically perform enhancement in the architecture and overcomes the aforementioned problems. The scope of this research is to work on Directed SoS: "Directed" is one specific type of SoS with a central authority that controls constituents, publishing their services and interfaces specification.

We expect to contribute to the state of the art of SoS. For this, algorithms and procedures were defined to properly handle SoS dynamic characteristics. It was also established techniques

to couple them into a simulator. It is worth highlighting that we were especially interested in SoS like smart cities because there are previous works (MANZANO; NETO; NAKAGAWA, 2019) that offer a basis for the conduction of evaluations of the proposed framework.

In order to achieve the main objective, the following specific objectives were derived:

**First:** Identification of the knowledge on optimization and machine learning for system architectures as a result of our systematic mapping study (SMS);
**Second:** Investigation of optimization and machine learning approaches, techniques, and algorithms that could be used to enhance quality attributes of SoS architectures as a result of our SMS;
**Third:** Investigation of how the state of the art on simulation needs to be advanced to handle SoS characteristics and optimization needs;
**Forth:** Definition of a framework to SoS software architecture, including tools, algorithms, and techniques;
**Fifth:** Development of tools and algorithms to SoS software architecture optimization.

After accomplishing these objectives, we evaluate the framework, applying it using the software architecture of a smart city SoS, observing its viability and suitability.

## 1.4 Methodology

In order to guide the development of our research, we have established a methodology based on the design science research methodology (DSRM) (PEFFERS *et al.*, 2007), which incorporates principles, practices, and procedures required to carry out the research and to meet three main objectives: (i) it is consistent with prior literature; (ii) it provides a nominal process model for doing design science research; and (iii) it provides a mental model for presenting and evaluating design science. Figure 1 presents the methodology that guided the execution of this project. The methodology defined has five activities, (i) identify problem and motivation, which defines the specific research problem and justify the value of a solution; (ii) define objectives, which infer the objectives of a solution from the problem identified and knowledge of what is possible and feasible; (iii) design and development, which designs and creates the artifacts to accomplish the objectives; (iv) evaluation, which observes and measures how well the artifacts supports the solution proposed to the problem; and (v) communication, which reports the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to community.

**Identify problem and motivation.** Trustworthiness and safety have been shown to be very important elements for the design of systems-of-system software architecture, such systems should present optimal (or near-optimal) architectural configurations to assure that they achieve qualities required in their domains because malfunction or failures can lead to financial losses and

even serious impact on human lives. Thus, the main problem addressed by our work is to ensure the quality of Software-Intensive System Architecture. The literature shows that optimization and machine learning have been successfully used to improve the quality of software architectures. To better understand the problem we are addressing, we carry out a SMS to find the gaps that need to be filled and the techniques that can be used to address this problem.
**Define objectives.** The main objective of our research is to develop a framework that makes it possible to automatically generate a near-optimal software system-intensive architecture. To achieve this objective it is necessary: (i) to create/adapt an algorithm to generate the architecture; (ii) develop an infrastructure that supports the execution of this algorithm; and (iii) create a process to systematize the activities necessary to properly utilize this infrastructure and algorithm.

**Design and development.** To carry out the development of the necessary artifacts for our solution, it is first necessary to choose which approach to use, which techniques to use, which algorithms to use, and which tools to use. With the mapping results, we established an simulation-based optimization framework that uses a modified genetic algorithm to, together with a simulator, generate near-optimal architectures. Thus, it was necessary to develop a means to couple the developed algorithm to the chosen simulator.

**Evaluation.** The evaluation of the framework was divided into two stages, convergence analysis and benchmarking. During the convergence analysis step, the evaluation of the results generated by the algorithm was carried out in order to evaluate if the algorithm can reach local minima in a finite time, evaluate if the algorithm can diversify in order to find other better local minima and, by end, evaluate if the architectural configurations generated by the algorithm are near-optimal. Finally, during the second stage, a benchmarking experiment was carried out in order to analyze the amount of computational resources (CPU, memory and time) necessary for the execution of the developed infrastructure, as well as to analyze the algorithmic complexity to design how the computational resources increase as the size of the problem increases.

**Communication.** The communication of the framework will be carried out through the writing of scientific articles in order to carry out their publication in academic journals in the area.

## 1.5 Organization

This chapter presented an overview of the context and motivation for developing this Master's project, including its main objectives. The remaining chapters of this monograph are organized as follows. Chapter 2 provide the necessary background, introducing the main concepts about SoS, optimization, and machine learning. Chapter 3 presents the and overview about the

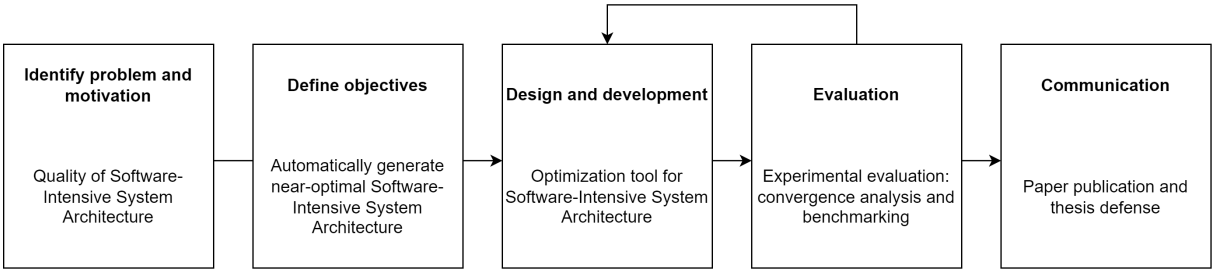


Figure 1 – DSR methodology process model. (Adapted from (PEFFERS *et al.*, 2007))

state-of-art in regards the use of optimization and machine learning in the context of software architectures. Chapter 4 presents the framework developed by this Master's project. Chapter 5 presents the evaluation performed to addresses the feasibility of the framework. Chapter 6 summarizes the main contribution of this Master's project, presents the main limitations, and future works.

CHAPTER

# 2

# BACKGROUND

The complexity of contemporary software-intensive systems has increased considerably. At the same time, these systems have spread in diverse critical application domains, e.g., Industry 4.0 (MORGAN *et al.*, 2021), smart buildings (GASSARA *et al.*, 2017), smart grids (MORTAJI *et al.*, 2017), and smart cities (MOHAMMADI *et al.*, 2018). These systems must ensure correct and non-stop operation and encompass diverse heterogeneous subsystems or components, such as software, equipment, and devices. Software-intensive systems often present large and complex architectures that should present optimal (or near-optimal) configurations to ensure that they achieve qualities required in their domains (HELLESTRAND, 2013) because malfunction or failures can lead to financial losses and even serious impact on human lives. Hence, high qualities are essential requirements of these systems but assuring such qualities has not been a trivial task. Simulation models can predict, still at design-time, properties of different architectural configurations that a system can assume at runtime (FRANÇA; TRAVASSOS, 2016; FRANçA; TRAVASSOS, 2015; Horeis *et al.*, 2020). Simulation can be combined with other techniques (such as machine learning and/or optimization) to achieve more precise and meaningful results (NGUYEN; REITER; RIGO, 2014; RUSSELL; NORVIG, 2009; SUTTON; BARTO, 1998). In particular, machine learning uses statistics in mathematical models can use example data or simulation results to extract knowledge(ABAEV *et al.*, 2013). Hence, from simulation results on a set of architectural configurations, it is possible to predict the behavior of a new configuration without indeed simulating it (EICHLER; KÖLMEL; SAUER, 1997). Simulation results can also be used as input to clustering algorithms to create profiles for a set of architectural configurations (UEDA *et al.*, 2005). They can also be used as input to classification algorithms to generate models that can classify new architectural configurations (Kawakami; Kahazu, 1996).

This chapter covers the concepts that were important during the execution of this project. Section 2.1 presents the main definitions related to SoS, compiles a set of the essential characteristics of SoS, discusses a taxonomy of SoS, presents some concepts of Systems Engineering processes to construct SoS, and brings some examples of current existing SoS. Section 2.2

presents the definition and the main concepts in regards of optimization, and presents some examples of optimization algorithms. Section 2.3 presents the definition and the main concepts of machine learning, as well as three approaches in machine learning, techniques, and examples, And, finally, Section 2.4 brings final considerations.

## 2.1 System-of-Systems

Systems are increasingly interoperating and collaborating with other systems, even for legacy systems they are being required to interoperate with other systems to satisfy society's needs. Increasing the size and complexity of software-intensive systems so that traditional software engineering methodologies and techniques are ineffective to deal with the demands of these systems. This phenomenon gave rise to a new class of large-scale software-intensive systems called Systems-of-Systems (SoS). SoS are composed by heterogeneous pre-existing software systems called constituents interconnected to cooperate among themselves, delivering behaviors as a composition of individual functionalities they offer (ACHESON; DAGLI; KILICAY-ERGIN, 2013; ANDREWS *et al.*, 2013). In military domain it is common to find a set of equipment and systems working in cooperation to accomplish goals, using satellites, GPS, missiles, and other embedded systems interacting to reach such a target during a simulation. The civilian software domain is starting to leave the traditional paradigms of monolithic systems engineering due to the need of an effective integration of socio-technical systems, specially those related to financial systems, Business-to-Business (B2B), and recommendation systems, to compose larger, more complex, and innovative functionalities, adding more domain-specific requirements to be accomplished. Engineering these SoS has been considered a challenging endeavor since there is a need to effectively accomplish a variety of requirements from a diversity of stakeholders (FITZGERALD; BRYANS; PAYNE, 2012)). This requires the development of increasingly complex and large SoS with society placing more and more reliance on them (BRYANS *et al.*, 2013).

### 2.1.1 Definitions and Characteristics

Software-intensive SoS combines heterogeneous software-intensive systems and are resulted from the interoperability of several managerially and operationally independent software systems, delivering capabilities and services which cannot be achieved by the constituent systems alone. Examples include enterprise information systems, integrated manufacturing systems, and emergency response collaborations.

Several definitions of SoS are presented in the literature, some are affected by particularities of the application area, demonstrating a low consensus on what is in fact considered an SoS. Some remarkable definitions and respective authors/sources are compiled below:

**Maier (1998).** An assemblage of constituents which individually may be regarded as systems,

and which possesses two main properties: Operational Independence of the constituents and Managerial Independence of the constituents (MAIER, 1998);

**SEI and DoD (2008).** A set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities (DEFENSE), 2008);

**INCOSE (2012).** A system-of-interest whose system elements are themselves systems, typically entailing large-scale inter-disciplinary problems with multiple, heterogeneous, and distributed systems (INCOSE, 2016);

**ISO/IEC/IEEE 21839:2019.** Set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own (21839:2019(E), 2019).

SoS often exhibit a well-defined set of characteristics postulated by Maier (MAIER, 1998) which includes:

**Operational independence.** Constituents may operate outside the SoS context, so its operation is not exclusively dedicated to the SoS;

**Managerial independence.** Constituents may act under the authority of different institutions so that each constituent has its own management strategy and each stakeholder independently deliveries and manages different constituents that compose the SoS;

**Evolutionary development.** The SoS evolves with new functions and purposes as the constituents and the environment evolves, culminating in an essential evolutionary development;

**Emergent behavior.** As constituents interoperate, the SoS can present some capabilities composed by individual smaller functionalities performed by the constituents, achieving emergent behaviors, and the main purposes of the SoS (Graciano Neto; OQUENDO; NAKAGAWA, 2017; OQUENDO, 2017). SoS behaviors can be classified as: *Simple*, in which emergent behaviors are emergent properties readily predicted by simplified models of the SoS; *Predicted*, in which emergent behaviors are those readily and consistently reproducible in simulations of the system, but not in static models; *Strong*, emergent behaviors are consistent with SoS known properties, but are not reproducible in any model of the system and simulations may reproduce the behavior, but inconsistently; and *spooky*, in which emergent behaviors are inconsistent with known properties of the SoS, not reproducible and not subject to simulation;

**Distribution.** SoS constituents are also an standalone system, so each constituent has a individual state, and the constituents usually are geographically distributed, so networks techniques are needed to the communication between constituents.

Besides the five traditional characteristics, Boardman and Sauser brought three other important characteristics: interoperability, heterogeneity, and hierarchy (BOARDMAN; SAUSER, 2006). SoS can also be a constituent of another SoS, such as control systems, smart buildings, and flood monitoring systems, which are themselves SoS and all can be constituents of a large-scale

SoS such as a smart city. Hence, it is imperative to investigate how to design such systems, predicting their behavior and structure still at design-time.

As the consequence of the operational independence of constituents, the SoS software architecture exhibit dynamic architecture since a constituent may not be exclusively dedicated to the SoS. Dynamic architectures exhibit a property called dynamic reconfiguration, i.e., their structure can change at runtime, inserting or removing parts, complying with the requirement of generating a new architectural arrangement that is still functional. A constituent can be part ofan SoS, contributing to the SoS accomplish its missions, but later leaving it, demanding for a reorganization of the structure of the SoS to maintain its operation in a valid operational state. In this case, a valid operational state is such that the constituents are properly connected to allow information exchange to perform the emergent behavior. An architectural change to the SoS dynamic architecture is considered successful when, besides transforming an SoS from an architectural configuration (also known as coalition) towards a new one, it also moves the architecture of an SoS from one valid operational state to another one.

### 2.1.2 Taxonomy

SoS can be classified by the level of managerial control (DAHMANN; JR.; LANE, 2008), and by their central authority, and collaboration level (BRYANS *et al.*, 2013; PÉREZ *et al.*, 2013). Maier established a taxonomy with four types of SoS (MAIER, 1998): directed, collaborative, acknowledge, and virtual. According to Maier's taxonomy, these are the essential types of SoS and its respective characteristics (DAHMANN; BALDWIN; JR., 2009; LANE, 2013; MAIER, 1998):

**Directed.** Directed SoS are built and managed to fulfill specific purposes and it is centrally managed to fulfill those purposes as well as any new ones the system managers might wish to address. Constituent systems long-term operation belongs to the SoS purposes, but they also can operate independently.

**Acknowledged.** Acknowledged SoS have recognized purposes, a designated manager, and resources for the SoS. However, constituent systems retain their independent ownership, objectives, funding, and development and sustainment approach. Changes in the systems are based on collaboration between the SoS and the system;

**Collaborative.** Collaborative SoS emerges from voluntary interaction of the constituent systems to fulfill agreed-upon central purposes. The central players decide collectivelly how to provide or deny service, thereby providing some means of enforcing and maintaining standards;

**Virtual.** Virtual SoS lack a central management authority and a centrally agreed-upon purpose for the system-of-systems. Large-scale behavior emerges—and may be desirable—but this type of SoS must rely upon relatively invisible mechanisms to maintain it.

Figure 2 presents a visualization of how constituent systems interact with each other

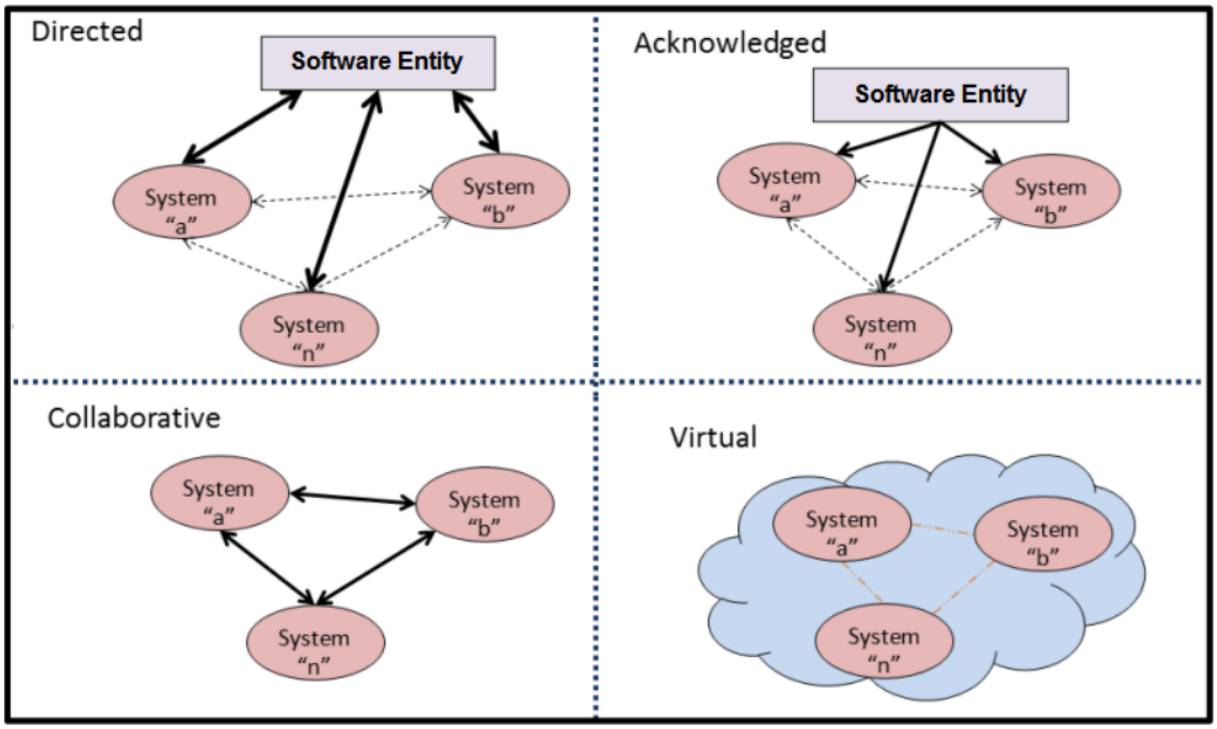


Figure 2 – SoS types and arrangements (adapted from (LANE, 2013))

regarding to types of SoS discussed (LANE, 2013). In the Directed type, the central authority has bidirectional communication with constituents, where those systems can query this entity to discover other constituents available and services provided by them, and the central authority delivers a response, enabling constituents to communicate among themselves. In the Acknowledged type, the software entity establishes unilateral communication, organizing constituents to collaborate and accomplish the SoS mission. Collaborative type lacks a central entity, and the cooperation among constituents arises spontaneously, but with a predefined and embedded purpose. Finally, the Virtual type, where constituents can interoperate supported by some mechanism, and this interaction is ad-hoc, without a predefined interaction, becoming spontaneous.

### 2.1.3 System-of-Systems Simulation

SoS software architectures exhibit dynamic architectures as a consequence of their fundamental characteristics, in particular regarding the *operational independence of constituents*, which states that a constituent is not exclusively dedicated to an SoS (MAIER, 1998). Dynamic architectures change its structure at run time performing dynamic reconfigurations, inserting or removing constituents and couplings, generating a new architectural configuration that is still functional.

Simulation techniques have been successfully used, especially in software engineering (FRANÇA; TRAVASSOS, 2016), to support the visualization of the system's dynamic behaviors. Simulations can be used to anticipate behaviors that can potentially occur at runtime, being able to failures and malfunctions. In particular, the simulation of software architectures can

be achieved by using simulation models with dynamic reconfiguration to predict architectural configuration changes at runtime (BOGADO; GONNET; LEONE, 2017).

Several studies have demonstrated the importance of using simulation in the SoS context (FRANÇA; TRAVASSOS, 2016; ZEIGLER *et al.*, 2012; MITTAL; RAINEY, 2015; NIELSEN *et al.*, 2015; MICHAEL; RIEHLE; SHING, 2009; SAUSER; BOARDMAN; VERMA, 2010; WACHHOLDER; STARY, 2015). These approaches (FRANÇA; TRAVASSOS, 2016; XIA *et al.*, 2013; MICHAEL *et al.*, 2011; WACHHOLDER; STARY, 2015): (i) support validation of expected emergent behaviors; (ii) empower the observation of unexpected emergent behaviors; (iii) enable the prediction of errors, diagnosing them and permitting corrections; and (iv) provide a visual and dynamic viewpoint, reproducing stimuli that the system can receive from an operational environment.

The languages currently adopted to describe software architectures, such as Unified Modeling Language (UML[1]), Systems Modeling Language (SysML[2]), and Compass Modeling Language (CML[3]), lack sufficient accuracy to effectively describe SoS architectures, specially in regards to (GUESSI *et al.*, 2015; ZEIGLER *et al.*, 2012; KLEIN; VLIET, 2013; CAVALCANTE *et al.*, 2016; NETO *et al.*, 2014; XIA *et al.*, 2013; CAVALCANTE; OQUENDO; BATISTA, 2014): (i) dynamic architecture; (ii) the description of constituents unknown at design time; (iii) environment modeling; (iv) the combination of formal foundations and high-level abstraction notations; and (v) the concept of SoS software architecture with adequate detail to guarantee precision in representation. UML, for instance, does not support a suitable specification of SoS software architectures, since a notation for specifying dynamic architectures is lacking (GUESSI *et al.*, 2015). UML diagrams must be adapted to tackle representation of multiple constituents through profiles, which do not represent SoS dynamism, including changes at runtime. In turn, SysML lacks specific structures to model various SoS architectural aspects such as dynamic architecture (GUESSI *et al.*, 2015). Moreover, simulating SoS using SysML often requires distributed simulation and simulators, requiring a heavy infrastructure and implementation to support the interoperability between simulations of single systems. CML is a formal language specially conceived for formal specification of Comprehensive Modeling for Advanced Systems of Systems (COMPASS) alliance (WOODCOCK *et al.*, 2012). However, it does not provide constructs for environment modeling, software architecture, or dynamic architecture (FITZGERALD *et al.*, 2013).

DEVS is a well-known simulation language (ZEIGLER *et al.*, 2012; NETO *et al.*, 2017). It consists of a modeling formalism based on the idea of atomic and coupled models. Atomic models represent individual entities in the SoS (for instance, systems), while coupled models represent a combination of atomic models. Such models exhibit the following elements (ZEIGLER *et al.*, 2012): (i) a labeled state diagram that performs transitions due to input or output

[1] UML, http://www.uml.org/
[2] SysML, http://sysml.org/
[3] CML, http://www.compass-research.eu/approach.html

events; (ii) variable initialization; and (iii) definition of abstract data types, global variables, ports, and events. In turn, coupled models are composed of atomic models, i.e., coupled models represent the entire SoS. DEVS Natural Language (DEVSNL) (ZEIGLER *et al.*, 2012) is a DEVS dialect which enables programming atomic and coupled models intuitive format using tools such as MS4ME[4]. Despite its advantages, DEVS and other simulation approaches do not properly support the software architecture description of SoS (GUESSI *et al.*, 2015). DEVS deals with the system architecture, i.e., a simulation model in DEVS considers the software and hardware aspects of all the constituents that compose an SoS, and for the SoS itself. DEVS deals with several important characteristics of software architectures, such as data types, constituent systems (represented as atomic models), constituent behaviors (expressed as labeled state machines), SoS structure (comprising the overall organization of such constituents) and how constituent exchange data (coupled models), and events. However, there are some important specific details of the software architectures of SoS that are not tackled by DEVS. DEVS only deals with the notion of ports. There is no notion of connections and gates separately, which are a paradigm of software architectures (components, connections, values, and constraints define a software architecture (BASS; CLEMENTS; KAZMAN, 2012; ISO/IEC/IEEE 42010, 2011)) used by SoS modeling (CAVALCANTE; BATISTA; OQUENDO, 2015). Equivalently, there is no differentiation among the concepts of mediators, nor constituents, nor stimuli generators. In DEVS, every major entity of an architecture is an atomic model. Thus, the SoS software architecture cannot be entirely characterized. The surrounding environment can be represented, but this is not natively supported, which is an important aspect of any software architecture, including SoS software architectures (ISO/IEC/IEEE 42010, 2011).

### 2.1.4 Examples of SoS

SoS appear in several application domains (NIELSEN *et al.*, 2013). Hellestrand shows a real example of SoS based on communication between cars for traffic management, bringing figures even towards smart cities (HELLESTRAND, 2013). Pérez (PÉREZ *et al.*, 2013) characterize Smart Grids as a kind of SoS. Some remarkable applications of SoS are the following:

**Earth Observation.** Earth observation is the gathering of information about planet Earth's physical, chemical, and biological systems via remote sensing technologies, usually involving satellites collecting images or via sensing station collecting measures from Earth to monitor and assess the status of, and changes in, the environment. An example of using SoS to earth observation is the Global Earth Observation System of Systems (GEOSS). which is a set of coordinated, independent Earth observation, information, and processing systems that interact to provide information to public and private sectors. GEOSS interoperate these systems to monitoring the state of the Earth and sharing environmental data collected from a large set of observing systems contributed by several countries and organizations;

[4] MS4ME, http://www.ms4systems.com/pages/ms4me.php

**Transportation.** Large-scale transport networks are made up of independent elements and distributed geographically and can evolve continuously to meet their demand. This problem is initially derived from the aerospace domain, emphasizing the role of networks in denying connectivity between the constituents and the heterogeneity of these systems;

**Emergency Management and Response.** Emergency response provides a particularly demanding application area, monitoring this area and notifying the authorities of the risk for the population. These systems require continuous monitoring and agility in the response to the emergence. An example of an emergency response SoS is the emergency management SoS in China, a hierarchically structured SoS with local, regional, and national SoS provide services at the level of jurisdiction required;

**Communication.** Since humanity is increasingly dispersed on the globe and the need for global communication, satellite communication systems have been developed, mainly in the military domain. An example of an SoS for communication is the Wideband Global SATCOM which was developed in partnership with the United States Department of Defense (DoD), Canadian Department of National Defence (DND), and the Australian Department of Defence, composed of a Space Segment satellites, Terminal Segment users, and a Control Segment operators.

## 2.2 Optimization

Optimization process consists on finding the best set of variables of a given model. Mathematical modeling of a problem consists on the indication of the objective, variables, and constraints of the problem (LUENBERGER; YE, 2008). In this way, the optimization will be as accurate as the model so that if the model is too simplistic, it will not give useful insights into the problem, but if it is too complex, it may become too difficult to solve. Once the model has been formulated, an optimization algorithm can be used to find its solution, usually with the support of a computational infrastructure. However, there is no universal optimization algorithm so that there are several algorithms, each specializing in a particular class of problem. In this way, choosing an algorithm is an important task that may determine whether the problem will be solved and, if it is, how fast the solution will be found (LUENBERGER; YE, 2008).

Another important task in the optimization process is to be able to recognize whether it has succeeded in its task of finding a solution to the problem or evaluate if the current solution can be improved. Finally, the model may be improved by applying techniques such as sensitivity analysis, which reveals the sensitivity of the solution to changes in the model and data.

An alternative to seek to reduce the computational resources required to perform the optimization is to seek only a local solution, which usually is not the global solution. As much as global solutions are ideal for solving the problem, they are usually quite difficult to identify and even more difficult to locate. An important special case is convex programming, in which all local solutions are also a global solution, such as linear programming problems. However,

general nonlinear problems, both constrained and unconstrained, may possess local solutions that are not global solutions.

Optimization algorithms are iterative algorithms that are initialized with a initial guess of the optimal solution, then perform a sequence of improved estimates until they reach an optimal solution. Each algorithm has its own strategy to iterate, some strategies uses derivatives of the model functions to iterate to the next solution and some algorithms accumulate local information from the current point to generate the next solution. Regardless of these specifics, algorithms seeks to posses the following properties:

**Robustness.** They should perform well on a wide variety of problems in their class, for all reasonable choices of the initial variables;

**Efficiency.** They should not require too much computer time or storage;

**Accuracy.** They should be able to identify a solution with precision, without being overly sensitive to errors in the data or to the arithmetic rounding errors that occur when the algorithm is implemented on a computer.

These goals may conflict. For example, a rapidly convergent method for nonlinear programming may require too much computer storage on large problems. On the other hand, a robust method may also be the slowest. Trade-offs between convergence rate and storage requirements, and between robustness and speed, and so on, are central issues in numerical optimization. The mathematical theory of optimization is used both to characterize optimal points and to provide the basis for most algorithms. It is not possible to have a good understanding of numerical optimization without a firm grasp of the supporting theory. Notable examples of optimization algorithms are evolutionary algorithm, genetic algorithm, and bee algorithm.

Evolutionary algorithms (EA) are rapidly developing associative analysis, in which a collection of techniques and systems are learned for managing a large complicated problem (MITCHELL, 1998). Many techniques are available under the class of evolutionary algorithms that differ in the representation of solution, implementation details, and how the particular problem is applied. The evolutionary process consists on the initial operand selection followed by fitness evaluation and population reproduction. The iteration continues until termination. Optionally, EA can perform adaptation of algorithm or local search. By choosing various options, new evolutionary methods can be derived.

Genetic algorithms are probabilistic search procedures designed to work on large spaces involving states that can be represented by strings. These methods are inherently parallel, using a distributed set of samples from the space (a population of strings) to generate a new set of samples (MITCHELL, 1998). They also exhibit a more subtle implicit parallelism. Roughly, in processing a population of m strings, a genetic algorithm implicitly evaluates substantially substrings. It then automatically biases future populations to exploit the above-average components as building blocks from which to construct structures that will exploit regularities in the environment

(problem space). Five phases are considered in a genetic algorithm (JONG, 2006): (i) initial population; (ii) fitness function; (iii) selection. (iv) crossover; and (v) mutation. The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve. An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution). In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score. Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

Bee algorithms are inspired by the foraging behaviour of bees. A few variants exist in the literature, including honeybee algorithm, artificial bee colony, bee algorithm, virtual bee algorithm, and honeybee mating algorithms (HADDAD; AFSHAR; MARIÑO, 2006). Honey bees live in a colony and they forage and store honey in their constructed colony. Honey bees can communicate by pheromone and 'waggle dance'. For example, an alarming bee may release a chemical message (pheromone) to stimulate attack response in other bees. Furthermore, when bees find a good food source and bring some nectar back to the hive, they will communicate the location of the food source by performing the so-called waggle dance as a signaling system. Such signaling dances vary from species to species, however, they are aimed at recruiting more bees by using directional dancing with varying strength so as to communicate the direction and distance of the food source. For multiple food sources such as flower patches, studies show that a bee colony seems to be able to allocate forager bees among different flower patches so as to maximize their total nectar intake (MORITZ; SOUTHWICK, 1992; NAKRANI; TOVEY, 2004; HADDAD; AFSHAR; MARIÑO, 2006).

## 2.3 Machine Learning

To solve a problem on a computer, we need an algorithm. An algorithm is a sequence of instructions that should be carried out to transform the input to output. However, in some tasks, we do not have an algorithm, we know what the output should be, but we do not know how to transform the input to the output. For example, classify whether or not an email is spam, we know the input (the body massage and meta-data), we know the output (yes/no indicating whether the message is spam or not), but we don't know how properly transform the input in the output (ALPAYDIN, 2014).

What we lack in knowledge, we make up for in data. We can easily compile thousands of

example messages, some of which we know to be spam and some of which are not, and what we want is to "learn" what constitutes spam from them. In other words, we would like the computer (machine) to extract automatically the algorithm for this task. We may not be able to fully identify the whole process, be e can construct a good and useful approximation. Fully identifying the complete process may not be possible, however, we can still detect certain patterns or regularities. This is the niche of machine learning. Such patterns may help to understand the process, or those patterns can be used to make predictions.

The application of machine learning methods to large databases is called data mining. Its application areas are abundant (ALPAYDIN, 2014): In addition to retail, in finance banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market. In manufacturing, learning models are used for optimization, control, and troubleshooting. In medicine, learning programs are used for medical diagnosis. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing, and searching for relevant information cannot be done manually. However, machine learning is not just a database problem, it is also a part of artificial intelligence. To be intelligent, a system that is in a changing environment should have the ability to learn. If the system can learn and adapt to such changes, the system designer need not foresee and provide solutions for all possible situations.

Machine learning also helps to find solutions to many problems in vision, speech recognition, and robotics (RUSSELL; NORVIG, 2009). For example, machine learning can be used to recognizing faces: This is a task that people do effortlessly to every day recognize family members and friends every by looking at their faces or from their photographs, despite differences in the pose, lighting, hairstyle, and so forth. However, people do it unconsciously and are unable to explain how they do it. Because we are not able to explain our expertise, we cannot write the computer program. At the same time, we know that a face image is not just a random collection of pixels; a face has structure. It is symmetric. There are the eyes, the nose, the mouth, located in certain places on the face. Each person's face is a pattern composed of a particular combination of these. By analyzing sample face images of a person, a learning program captures the pattern specific to that person and then recognizes it by checking for this pattern in a given image.

Machine learning is programming computers to optimize a performance criterion using example data or past experience (ALPAYDIN, 2014). We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data or both.

Machine learning uses the theory of statistics in building mathematical models because the core task is making inferences from a sample (ALPAYDIN, 2014). The role of computer science is twofold: First, in training efficient algorithms to solve the optimization problem, as

well as to store and process the massive amount of data. Second, once a model is learned, its representation and algorithmic solution for inference need to be efficient as well. In certain applications, the efficiency of the learning or inference algorithm, namely, its space and time complexity, may be as important as its predictive accuracy.

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning *signal* or *feedback* available to a learning system. These are Supervised learning, unsupervised learning, and reinforcement learning (ALPAYDIN, 2014; MOHRI; ROSTAMIZADEH; TALWALKAR, 2012).

In supervised learning, the goal is to learn the mapping (the rules) between a set of inputs and outputs. For example, the inputs could be the weather forecast, and the outputs would be the visitors to the beach. The goal of supervised learning would be to learn the mapping that describes the relationship between temperature and the number of beach visitors (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). Example labeled data is provided of past input and output pairs during the learning process to training the model how it should behave, hence, supervised learning. For the beach example, new inputs can then be fed in of forecast temperature and the Machine learning algorithm will then output a future prediction for the number of visitors.

Being able to adapt to new inputs and make predictions is the crucial generalization part of machine learning. In training, we want to maximize generalization, so the supervised model defines the real 'general' underlying relationship (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012; ALPAYDIN, 2014). If the model is over-trained, we cause over-fitting to the examples used and the model would be unable to adapt to new, previously unseen inputs. A side effect to be aware of in supervised learning that the supervision we provide introduces bias to the learning (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). The model can only be imitating exactly what it was shown, so it is very important to show it reliable, unbiased examples. Also, supervised learning usually requires a lot of data before it learns. Obtaining enough reliably labeled data is often the hardest and most expensive part of using supervised learning. Supervised learning techniques includes:

**Classification.** Which is technique to group similar data points into different sections to classify them. Machine Learning is used to find the rules that explain how to separate the different data points, using data and answers to discover rules that linearly separate data points (ALPAYDIN, 2014). Linear separability is a key concept in machine learning., classification approaches try to find the best way to separate data points with a line. The lines drawn between classes are known as the decision boundaries. The entire area that is chosen to define a class is known as the decision surface. The decision surface defines that if a data point falls within its boundaries, it will be assigned a certain class;

**Regression.** Which is another form of supervised learning. The difference between classification and regression is that regression outputs a number rather than a class (RUSSELL; NORVIG,

2009). Therefore, regression is useful when predicting number-based problems like stock market prices, the temperature for a given day, or the probability of an event. Regression uses statistical methods to create an estimation of a dependent variable to an independent variable;

**Structured Prediction.** Which involves predicting structured objects, rather than scalar discrete or real values (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). Similar to commonly used supervised learning techniques, structured prediction models are typically trained by means of observed data in which the true prediction value is used to adjust model parameters. Due to the complexity of the model and the interrelations of predicted variables the process of prediction using a trained model and of training itself is often computationally infeasible and approximate inference and learning methods are used.

In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor. In unsupervised learning, there is no such supervisor and we only have input data. The aim is to find the regularities in the input. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not. In statistics, this is called density estimation (ALPAYDIN, 2014).

Unsupervised learning can be harder than supervised learning, as the removal of supervision means the problem has become less defined. The algorithm has a less focused idea of what patterns to look for. By being unsupervised in a laissez-faire teaching style, algorithms start from a clean slate with less bias and may even find a new, better way to solve a problem (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). Therefore, this is why unsupervised learning is also known as knowledge discovery, being very useful when conducting exploratory data analysis. Unsupervised learning techniques includes:

**Clustering.** Which aims to find clusters or groupings of input. In the case of a company with data of past customers, the customer data contains the demographic information as well as the past transactions with the company, and the company may want to see the distribution of the profile of its customers, to see what type of customers frequently occur. In such a case, a clustering model allocates customers similar in their attributes to the same group, providing the company with natural groupings of its customers; this is called customer segmentation. Once such groups are found, the company may decide strategies, for example, services and products, specific to different groups; this is known as customer relationship management. Such a grouping also allows identifying those who are outliers, namely, those who are different from other customers, which may imply a niche in the market that can be further exploited by the company;

**Association.** Which aims to uncover the rules that describe your data, r discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness;

**Anomaly Detection.** Anomaly detection consists of the identification of rare or unusual items,

events, or observations that raise suspicions by differing significantly from the majority of a dataset. For example, the bank uses their client's data to detect fraudulent activity on their card. The normal spending habits will fall within a normal range of behaviors and values. But when someone tries to steal from you using the card the behavior will be different from your normal pattern. Anomaly detection uses unsupervised learning to separate and detect these strange occurrences.

In some applications, the output of the system is a sequence of actions. In such a case, a single action is not important, what is important is the policy that is the sequence of corrective actions to reach the goal. There is no such thing as the best action in any intermediate state, an action is good if it is part of a good policy. In such a case, the machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. Such learning methods are called reinforcement learning algorithms. To reinforce learning, software agents ought to take action in an environment so as to maximize some notion of cumulative reward.

Reinforcement learning differs from supervised learning in not needing labeled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Due to its generality, the field is studied in many other disciplines, such as control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms.

Once a system is mathematically modeled, computer-based simulations provide information about its behavior. Parametric simulation methods can be used to improve the performance of a system. In this method, the input of each variable is varied with other parameters remaining constant, and the effect on the design objective is observed. This is a time-consuming method and improves the performance partially. To achieve an optimal solution to a problem (or a solution near the optimum) with less time and labor, the computer building model is usually solved by iterative methods, which construct infinite sequences, of progressively better approximations to a solution, i.e. a point in the search-space that satisfies an optimality condition (WETTER, 2011). Due to the iterative nature of the procedures, these methods are usually automated by computer programming. Such methods are often known as numerical optimization or simulation-based optimization.

In a simulation experiment, the goal is to evaluate the effect of different values of input variables on a system (NGUYEN; REITER; RIGO, 2014). However, the interest is sometimes in finding the optimal value for input variables in terms of the system outcomes. One way could be running simulation experiments for all possible input variables. However, this approach is not always practical due to several possible situations and it just makes it intractable to run experiments for each scenario. For example, there might be too many possible values for input variables, or the simulation model might be too complicated and expensive to run for sub-optimal

input variable values. In these cases, the goal is to find optimal values for the input variables rather than trying all possible values. This process is called simulation optimization

## 2.4 Final Consideration

This chapter presented in the Section 2.1 all the concepts related to software-intensive SoS, covering their main characteristics, taxonomy, processes, and some examples, discussing the importance of Systems Engineering for SoS conception. Section 2.2 presented concepts related to optimization, presenting its definition, characteristics, and algorithms. Despite some limitations, optimization shows to be considered a valuable practice even for high-scale dynamic systems such as SoS. Section 2.3 concepts related to machine learning, covering their main characteristics, approaches, techniques, and some examples, discussing the importance of machine learning, advantages and disadvantages of machine learning approaches, limitations of the current state of practice, and gaps where machine learning approaches can be used.

Considering the complexity of architecting an SoS, it is imperative to develop approaches and tools to decrease efforts to build a quality SoS architecture. Simulation-based approaches show to be a great alternative to handle with SoS proprieties. Simulation enables the dynamic visualization of SoS behavior and the prediction of run-time failures, making it possible to extract metrics in the architecture and change the architecture configuration during the simulation (MANZANO; NETO; NAKAGAWA, 2019). In this way, simulation can provide useful data for optimization and machine learning algorithms, to enable the algorithms to apply operations in the architecture in order to improve a certain quality attribute. Next chapter presents the state of the art of how machine learning and optimization have been applied together with the simulation of software-intensive system architecture.

CHAPTER

3

# STATE-OF-THE-ART OF SIMULATION-BASED OPTIMIZATION

This chapter presents the state-of-the-art of the application of machine learning and optimization jointly with the simulation of software-intensive system architectures. The literature was examined and from 1,031 studies identified, 63 addressed specifically simulation, machine learning, and architectural optimization, demonstrating the combination of machine learning and optimization with simulation benefits the quality of software architectures with reduced human efforts, based on theoretical foundations that can assure near-optimal architectural configurations. Several issues for making *machine learning & optimization-based simulation* suitable for large and complex software-intensive systems still remain open.

Section 3.1 describes the research method adopted for SMS and Section 3.2 provides the results. The following section reports the main findings, research perspectives, and threats to validity. The last section is devoted to final considerations.

## 3.1 Research Method

The research method adopted followed a three-phase process for systematic mapping studies (SMS) (KITCHENHAM; BUDGEN; BRERETON, 2015), which includes planning, conduction, and results synthesis so that the main goal of the SMS, i.e., presentation of the state-of-the-art of simulation of software-intensive system architectures by machine learning and optimization techniques could be achieved.

### *3.1.1 Planning*

The SMS protocol that guided the research was defined in the planning phase and comprised research questions, search strategy, selection criteria, data extraction, and synthesis

methods.

According to the main goal of the SMS, the following five research questions (RQs) were defined:

- **RQ1:** What optimization techniques have been used in the simulation of system architectures?
- **RQ2:** What machine learning techniques have been used in the simulation of system architectures?
- **RQ3:** How machine learning and optimization have been combined to simulating system architectures?
- **RQ4:** What supporting tools are used when simulation, optimization, and machine learning are combined?
- **RQ5:** What are the maturity levels of the existing studies?

The search string defined comprised the main keywords contained in the RQs, namely*architecture*, *simulation*, *optimization*, and *machine learning* and, after calibration, resulted in a combination of keywords and their synonymous (also in the plural form): *(“software" OR "system" OR “systems") AND (“optimization" OR “meta-heuristic" OR “meta-heuristics" OR “metaheuristic" OR “metaheuristics” OR “genetic algorithm” OR “machine learning”) AND (“simulation" OR “simulations" OR “simulation-based") AND (“architecture" OR “architectures").*

The publication databases most commonly used for SMS in the software engineering area (DYBA; KITCHENHAM; JORGENSEN, 2005; KITCHENHAM; BUDGEN; BRERETON, 2015), namely Scopus[1], IEEExplore[2], and ACM Digital Library[3] were selected. The search string was adapted to coping with the different syntax rules required by each database’s search engine, and the searches were based on title, keywords, and abstract.

The two following inclusion criteria (IC) to be applied in studies returned from databases were defined for the selection of primary studies for the SMS:

- **IC1:** Study addresses optimization in the context of system architectures simulation.
- **IC2:** Study addresses machine learning in the context of system architectures simulation.

Five exclusion criteria (EC) were also defined: :

- **EC1:** Study does not address optimization in the context of system architectures simulation.
- **EC2:** Study does not address machine learning in the context of system architectures simulation.

[1] https://www.scopus.com
[2] https://ieeexplore.ieee.org
[3] https://dl.acm.org

- **EC3:** Study is a shorter version of another study.
- **EC4:** Study does not include an abstract.
- **EC5:** The full text of the study is unavailable.
- **EC6:** Study is written in a language other than English.

A data extraction form was defined for the obtaining of detailed information from studies and adequate answers to our RQs, collecting titles, authors, publication year of the studies, vehicles (event or journal where they were published), data on involvement with academia or industry, machine learning or/and optimization, tools used, quality attributes addressed, and maturity level.

### 3.1.2 Conduction

Following the SMS protocol, studies were searched in the databases according to the search string and 1,031 were returned.[4] As shown in Figure 3, after the removal of the duplicated ones, 948 remained for the next step. Each study was then examined regarding title, abstract, and, when necessary, keywords and introduction and conclusion sections. The application of IC and EC resulted in 96 studies. In this step, only studies that evidently were not in the scope of the SMS were excluded. After the full reading of all 96 studies and application of IC and EC again, 31 were excluded and 63 (listed in Table 1) were considered relevant for answering the RQs. The full text of each study was then read and data were extracted with the use of the data extraction form previously validated by an expert towards ensuring no field was missing. Moreover, an expert analyzed and verified the correctness of the extracted data.

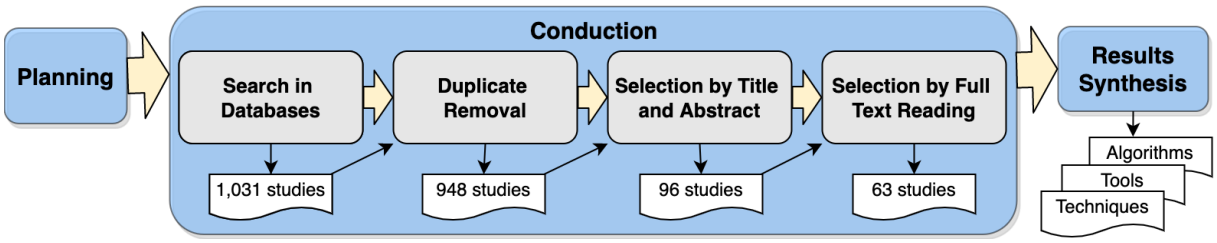


Figure 3 – SMS conduction process

Table 1 – Primary studies selected in this SMS

| ID | Title | Ref. |
|---|---|---|
| S1 | A blackboard software architecture for integrated intelligent control systems | (M.F. *et al.*, 2000) |
| S2 | A Framework for the Simulation and Validation of Distributed Control Architectures for Technical Systems of Systems | (NAZARI *et al.*, 2017) |

[4] Studies were collected from databases on June 21, 2020.

| | | |
|---|---|---|
| S3 | A mixed discrete-continuous optimization scheme for Cyber-Physical System architecture exploration | (FINN; NUZZO; SANGIOVANNI-VINCENTELLI, 2015) |
| S4 | A Mixed Signal Architecture for Convolutional Neural Networks | (LOU *et al.*, 2019) |
| S5 | A novel application architecture design for smart grid dispatching and control systems | (CHANG *et al.*, 2015) |
| S6 | A Novel, Highly Integrated Simulator for Parallel and Distributed Systems | (TAMPOURATZIS *et al.*, 2020) |
| S7 | A Reliability Engineering Based Approach to Model Complex and Dynamic Autonomous Systems | (Horeis *et al.*, 2020) |
| S8 | A simulation approach to structure-based software reliability analysis | (Gokhale; Michael Rung-Tsong Lyu, 2005) |
| S9 | A SVM Optimization Tool and FPGA System Architecture Applied to NMPC | (SANTOS *et al.*, 2017) |
| S10 | A workflow for the design of optimized system architectures using model-driven optimization | (WICHMANN *et al.*, 2018) |
| S11 | Adaptive wavefront correction with self-organized control system architecture | (VORONTSOV, 1998) |
| S12 | An architecture of a multi-model system for planning and scheduling | (ARTIBA; AGHEZZAF, 1997) |
| S13 | An intelligent driving simulation platform: architecture, implementation and application | (SUN *et al.*, 2020) |
| S14 | Application of genetic algorithms to Volterra filter structure determination for adaptive systems | (Mandyam, 1996) |
| S15 | Architecture and Performance of the Base Station Prototype for MN Systems | (Choi *et al.*, 2020) |
| S16 | Architecture design for communications simulation expert system using top-down information driven technique | (Tsang, 1989b) |
| S17 | Architecture-level performance estimation method based on system-level profiling | (UEDA *et al.*, 2005) |
| S18 | Automated generation of system-level AHDL architectures using a genetic algorithm | (White; Hallam; Binns, 1997) |

| | | |
|---|---|---|
| S19 | CA-Ex: A tuning-incremental methodology for communication architectures in embedded systems | (WANG *et al.*, 2005) |
| S20 | CACF: A Novel Circuit Architecture Co-Optimization Framework for Improving Performance, Reliability and Energy of ReRAM-Based Main Memory System | (ZHANG *et al.*, 2018) |
| S21 | Combining ab initio techniques with analytical potential functions for structure predictions of large systems: Method and application to crystalline silica polymorphs | (EICHLER; KÖLMEL; SAUER, 1997) |
| S22 | Compiler Optimization of Embedded Applications for an Adaptive SoC Architecture | (HARDNETT; PALEM; CHOBE, 2006) |
| S23 | DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference | (Gupta *et al.*, 2020) |
| S24 | Design and software architecture of SIP server for overload control simulation | (ABAEV *et al.*, 2013) |
| S25 | Design of communications simulation expert system architecture with common data structure | (Tsang, 1989a) |
| S26 | Device-architecture co-optimization of STT-RAM based memory for low power embedded systems | (XU *et al.*, 2011) |
| S27 | Distributed Optimization on Super Computers: Case Study on Software Architecture Optimization Framework | (ETEMAADI; CHAUDRON, 2014) |
| S28 | Effect of modular system structures on component reliability estimation | (ABOUL-SEOUD; USHER, 1996) |
| S29 | Efficient algorithm in model identification of multivariable and non-linear systems - GMDH plus optimization of model structure algorithm | (CHEN; YU; SHAO, 1988) |
| S30 | Efficient Heterogeneous Architecture Floorplan Optimization Using Analytical Methods | (KAHOUL *et al.*, 2010) |
| S31 | Enhancing microservices architectures using data-driven service discovery and QoS guarantees | (Houmani *et al.*, 2020) |
| S32 | ETH: An Architecture for Exploring the Design Space of In-situ Scientific Visualization | (Abram *et al.*, 2020) |
| S33 | General architecture for simulating complex systems able of auto-organization | (FIALHO; SANTOS, 1994) |
| S34 | Hidden Genes Genetic Algorithms for Systems Architecture Optimization | (ABDELKHALIK; DARANI, 2016) |

| | | |
|---|---|---|
| S35 | High level architecture for an advanced distributed complex systems engineering environment | (HOWELL; KARANGELEN, 1997) |
| S36 | Introducing a cloud based architecture for the distributed analysis of Real-Time Ethernet traffic | (Turcato *et al.*, 2020) |
| S37 | Know-how Protection and Software Architectures in Industry 4.0 | (FRICKE; SCHöNEBERGER, 2017) |
| S38 | Memory Access Optimization in Compilation for Coarse-Grained Reconfigurable Architectures | (KIM *et al.*, 2011) |
| S39 | Methods for Power Optimization in Distributed Embedded Systems with Real-Time Requirements | (RACU *et al.*, 2006) |
| S40 | Mocktails: Capturing the Memory Behaviour of Proprietary Mobile Architectures | (Badr *et al.*, 2020) |
| S41 | Model-based architecture for evolutionary intelligent systems | (CHI; LEE, 2001) |
| S42 | Obstacle-Avoiding Algorithm in X-Architecture Based on Discrete Particle Swarm Optimization for VLSI Design | (HUANG *et al.*, 2015) |
| S43 | On line multilevel controller structure for a class of dynamical systems | (El-Far; Nassef; Montasser, 1996) |
| S44 | Performance Optimization on Big.LITTLE Architectures: A Memory-Latency Aware Approach | (WOLFF; PORTER, 2020) |
| S45 | PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization | (KOZIOLEK; KOZIOLEK; REUSSNER, 2011) |
| S46 | Research of simulation software architecture optimization used with components based on reach-ability matrix | (NAN; LIAO; JIANG, 2012) |
| S47 | Robust ArcheOpterix: Architecture Optimization of Embedded Systems under Uncertainty | (MEEDENIYA *et al.*, 2012) |
| S48 | Scalable and Secure Architecture for Distributed IoT Systems | (Dhieb *et al.*, 2020) |
| S49 | Scheduling flexible manufacturing systems containing assembly operations based on Petri net structures and dynamics | (JENG; LIN, 1997) |
| S50 | Self-adaptive neuro-fuzzy systems: Structure and learning | (Lee; WANG, 2000) |
| S51 | Simulation and optimization software for axisymmetrical radiating structures | (BÉHÉ *et al.*, 1994) |

| | | |
|---|---|---|
| S52 | Simulation of a local computer network architecture applying a unified modeling system | (DIDIC; WOLFINGER, 1982) |
| S53 | Simultaneous optimum design of structure and control systems by sensitivity analysis | (KAJIWARA; NAGAMATSU, 1993) |
| S54 | SOC Test-Architecture Optimization for the Testing of Embedded Cores and Signal-Integrity Faults on Core-External Interconnects | (XU; ZHANG; CHAKRABARTY, 2009) |
| S55 | Software-Defined architecture for QoS-Aware IoT deployments in 5G systems | (TELLO-OQUENDO *et al.*, 2019) |
| S56 | Starchart: Hardware and Software Optimization Using Recursive Partitioning Regression Trees | (JIA; SHAW; MARTONOSI, 2013) |
| S57 | Statistical Timing and Power Optimization of Architecture and Device for FPGAs | (CHENG *et al.*, 2012) |
| S58 | Study on evolutionary synthesis of classifier system architectures | (Kawakami; Kahazu, 1996) |
| S59 | Supporting environmental and energy decisions through an open software structure | (GUARISO; BARACANI, 2006) |
| S60 | System Architecture for On-Line Optimization of Automated Trading Strategies | (FREITAS; FREITAS; SOUZA, 2013) |
| S61 | Tabu search versus evolutionary search for software structure optimization | (BALICKI; KITOWSKI, 2000) |
| S62 | The architecture of system for analysis and optimization of parallel programs | (OTWAGIN; SADYKHOV; BOGOMAZOV, 2004) |
| S63 | Using Quality of Service Bounds for Effective Multi-Objective Software Architecture Optimization | (NOORSHAMS; MARTENS; REUSSNER, 2010) |

### 3.1.3 Overview of Studies

According to Figure 4, the first study was published in 1982 and studies are somehow concentrated on the recent years. All 63 studies address the simulation of software architectures, of which 13 focus on machine learning, 40 are devoted to optimization, and 10 analyze both machine learning and optimization, showing similarly to the combination of simulation and machine learning, that of simulation, machine learning, and optimization has not been widely explored. 35 studies were published in events proceedings and 29 were published in journals, highlighting almost half of them seem to be more consolidated, since they were published in journals.

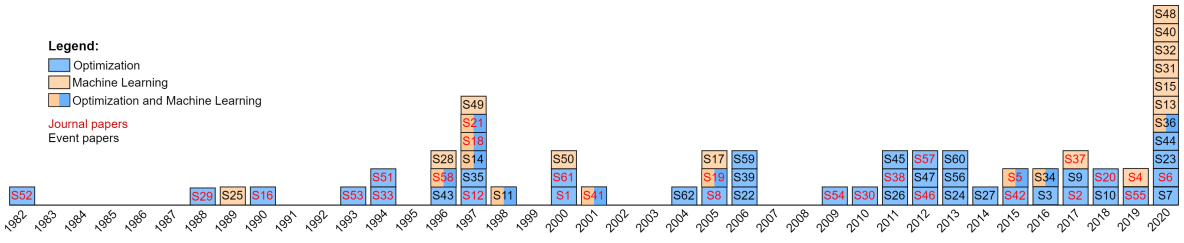


Figure 4 – Overview of primary studies distributed along the years

The studies were also classified into academia and industry, as shown in Figure 5, in function of their addressing application (e.g., case studies, experiments, or projects) in those contexts. 44 studies focused on academia and 19 addressed industry, i.e., according to those gathered in the SMS, most have been experimented in academia. Figure 5 also shows no trend regarding preference of industry or academia on the adoption of machine learning or optimization techniques for simulation has been detected.

The domain of the selected studies was also identified. Embedded systems were the most common, with 12 studies, followed by information systems (6 studies) and control systems (5). Others, such as expert systems, Internet of Things (IoT), and self-adaptive systems presented two studies each, while newer domains, such as smart cities, Industry 4.0, and microservices, presented just one study.

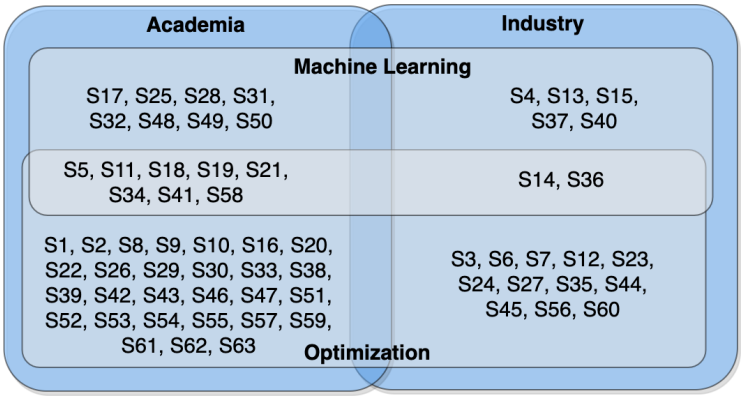


Figure 5 – Distribution of studies from academia and/or industry

The following subsections answer the RQs.

### 3.1.4 Optimization Techniques for Simulation of System Architectures

All 50 studies that addressed optimization were analyzed so that RQ1, *"What optimization techniques have been used in the simulation of system architectures?"*, could be answered. 17 different optimization techniques used in architecture simulation were returned, as shown in Figure 6. The most common one is *genetic algorithm*, with nine studies, which has been considered reliable and effective for architecture optimization (NGUYEN; REITER; RIGO, 2014) with positive results in experiments and to several domains and optimization problems. It has stood out due to its versatility for dealing with different quality attributes, such as adaptability (S14), flexibility (S18 and S58), performance (S18 and S47), reliability (S47), and safety (S47). On the other hand, the building of architecture representation for simulations is not trivial and, in some cases, its right representation and tuning are the most labor-intensive tasks in the optimization process.

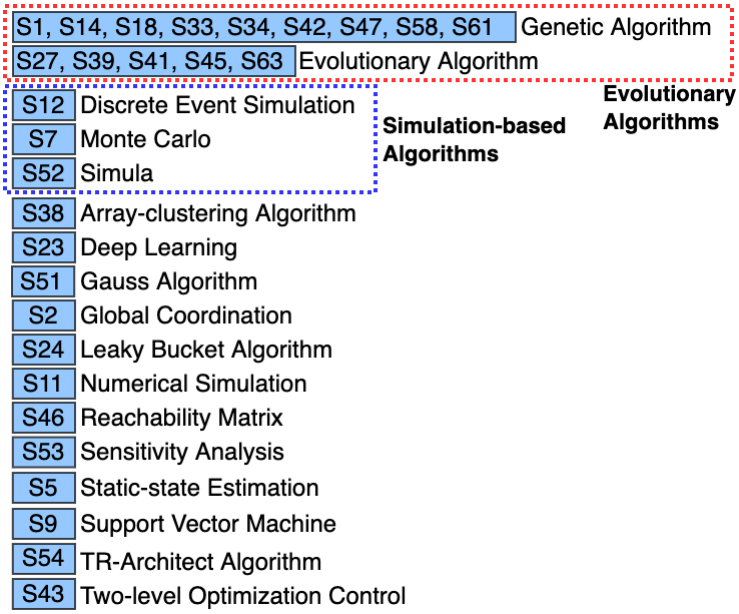


Figure 6 – Optimization techniques used in software architecture simulation

*Evolutionary algorithms* have also drawn attention from the community - five studies addressed them - mainly due to their ability to generate meta-heuristics that can better explore a search space through diversification and with no exhaustive searches. The algorithms can find near-optimal solutions with satisfactory performance. The studies reported different strategies to evolve the population of solutions and focused on the optimization of different quality attributes, such as performance (S27), energy (S39), and cost (S63), i.e., the algorithms show versatility regarding quality attributes to be optimized.

Other specific algorithms such as *reachability matrix*, *Monte Carlo algorithm*, and *numerical simulation* were used due to them being specialized to a certain problem, thus generating more concrete results and/or more efficiently. Reachability matrix was used only for assessing reachability. Although it requires a specific architecture representation, it performs better than generic algorithms according to S46. In S7, Monte Carlo addressed the probability aspects of autonomous systems with a simpler representation of the architecture, thus being able to evaluate the architecture in relation to reliability in a probabilistic environment. Concerning numerical simulation, S11 employed it to evaluate adaptability regarding the wavefront correction in self-organized control system architectures.

Among the 50 studies that addressed optimization in architecture simulation, S3, S6, S8, S10, S16, S19, S20, S21, S22, S26, S29, S30, S35, S36, S44, S57, S59, S60, and S62 did not make explicit the algorithm adopted or the use of proprietary algorithms. The creation of a new algorithm or meta-heuristics is not trivial, since it requires efforts into the mathematical specification and its formal verification. However, the efforts are worth when the algorithm is part of a tool or necessary due to legislation in very specific application domains.

The quality attributes measured by the studies were also a concern. 14 different and diverse quality attributes mainly on performance, reliability, flexibility, and efficiency were found in 35 studies (see Figure 7). Performance showed a relevant one, especially in relation to the response time of an architecture and the speed at which data become available for a given element of the architecture. Such aspects are significant in real-time systems, critical embedded systems, and control systems. Since architecture optimization is closely associated with critical systems, reliability and safety were also present among the studies. Autonomous, self-organizing, and self-adaptive systems reported in the studies (S11, S14, and S50) demonstrate the importance of quality attributes such as flexibility, availability, and adaptability. Although optimization in architecture simulation showed successful for several quality attributes, each study dealt with a restricted set of such attributes (one to three attributes) and developed its approach to addressing that set.

### 3.1.5 Machine Learning Techniques for the Simulation of System Architectures

The 23 primary studies on machine learning in architecture simulation were analyzed so that RQ2, *"What machine learning techniques have been used in the simulation of system architectures?"* could be answered. Figures8 and 9 show, respectively, the information extracted regarding the machine learning approaches adopted and the quality attributes addressed.

The most common approach was *supervised learning*, with 11 studies, of which five focused on structured prediction and six addressed classification. The analysis of the studies that used *structured prediction* and the way the architecture adapts at runtime (S17) provided a

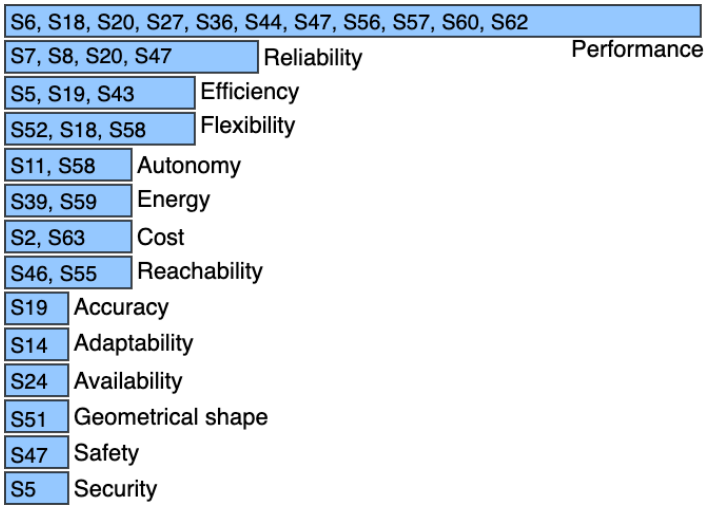


Figure 7 – Quality attributes addressed by optimization techniques in software architecture simulation

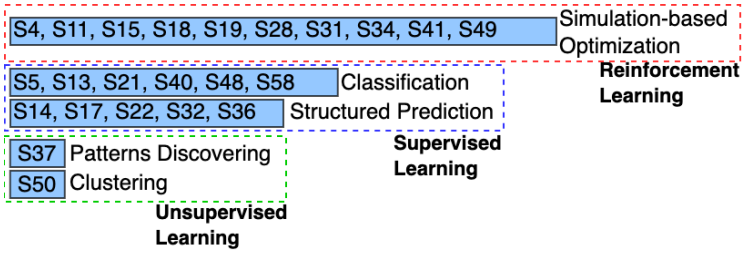


Figure 8 – Machine learning approaches used in software architecture simulation.

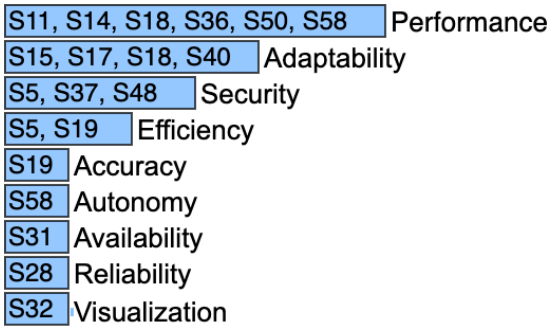


Figure 9 – Quality attributes addressed by machine learning techniques in architecture simulation.

dynamic visualization of the architecture (S32). Those on *classification* focused on the use of simulation results and, in some cases, historical data for undertaking tasks such as classification of the security level presented by the architecture (S5 and S48), classification of the architecture's behavior in relation to its adaptability (S40), and classification of the performance level of the architecture (S58). Therefore, classification was used for categorizing architectures according to different aspects.

Ten selected studies addressed *reinforcement learning* and adopted meta-heuristics to improve some performance metrics such as transfer rate and response time (S11 and S18), the efficiency of the architecture in terms of cost and energy expenditure (S19), and its reliability (S28). Reinforcement learning was also used for recommending changes to the architecture

towards keeping it available (S31) and generating policies to maintain a self-organizing system running (S31). Its adoption is related to nature of the domain of systems architectures, where, due to the complexity and nondeterminism of the environment, it is unfeasible to find an optimal solution. The use of techniques to improve architectures regarding quality attributes and generate a "good enough" solution is valuable for complex domains.

Only two studies addressed *unsupervised learning* for clustering architectural configurations, generating profiles regarding their flexibility (S50), or discovering patterns to extract knowledge regarding security and associate architectural decisions with security properties (S37).

12 different machine algorithms were used by the selected studies (see Figure 10), of which four adopted *genetic algorithm*, showing versatility in relation to the approaches, and yielding good results in terms of learning performance and accuracy of learned models. Genetic algorithm was used for both reinforcement learning (S18 and S34) and supervised learning (S14 and S58) and others were adopted for reinforcement learning (e.g., Convolutional Algorithm (S4), Numerical Simulation (S11), and ScaleDown Algorithm (S31)), supervised learning (e.g., Static-state Estimation (S5), Support Vector Machine (S48), and Graph Analysis (S17)), and unsupervised learning (e.g., Neuro-fuzzy networks (S50)).

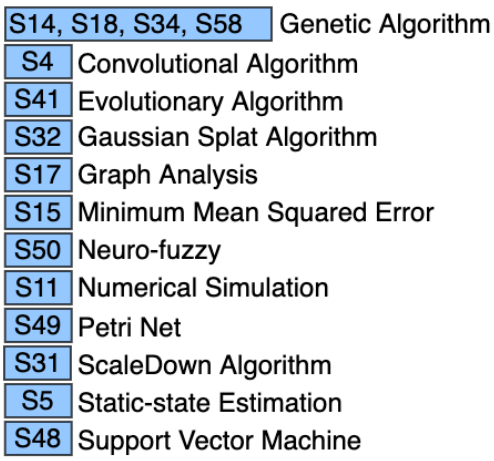


Figure 10 – Machine learning algorithms used in architecture simulation

Parts of the studies proposed the *prediction of different configurations of system architectures* - specifically in systems found in dynamic environments, where a same configuration cannot be kept at all times at runtime (S3, S6, S7, S8, S12, S13, and S31). Therefore, such studies dealt with dynamic architectures, i.e., their components were added, removed, and replaced, or evolved at runtime. Connections among them are also dynamic, i.e., the systems assume different architectural configurations at runtime (CAVALCANTE *et al.*, 2016; MANZANO; NETO; NAKAGAWA, 2019). As a main result of the RQ, machine learning techniques have drawn attention for supporting simulation of system architectures. Although few studies described the experience with supervised learning and genetic algorithms, approaches or techniques that would work better than others cannot be cited. No study has performed a deeper comparison among

them, but experimented different ways.

### 3.1.6 Combination of Machine Learning and Optimization for the Simulation of System Architectures

This section answers RQ3, "How machine learning and optimization have been combined to simulating system architectures?", since, after answering RQ1 and RQ2, our interest lay in understanding a better way to combine machine learning and optimization during architecture simulation. Therefore, the 10 studies listed in Figure 5 were analyzed. Five (S11, S18, S19, S34, and S41) addressed reinforcement learning using simulation-based optimization, including genetic algorithm (S18 and S34), evolutionary algorithm (S41), and numerical simulation (S11), and five others (S5, S14, S21, S36, and S58) focused on supervised learning. In particular, three addressed classification together with static-state estimation (S5) and genetic algorithm (S58), and two were devoted to structured prediction – S14 adopted genetic algorithm and S36 did not specify the algorithm used. Certain preference for genetic algorithms can be observed, since all studies that adopted genetic algorithms and addressed machine learning also addressed optimization, due to the innate characteristics of genetic algorithms.

The studies showed optimization and machine learning can be combined in three main ways, i.e., (i) using machine learning techniques to diversify optimization meta-heuristics (S18, S34, and S41), (ii) using machine learning techniques to parameterize optimization algorithms (S11 and S19), and (iii) using optimization to generate optimal or near-optimal machine learning models (S5, S14, S21, S36, and S58). Machine learning and optimization can be combined, for example, towards the use of machine learning techniques for diversifying the population of a genetic algorithm, parameterizing iterations of a numerical simulation, and learning a classification model.

### 3.1.7 Supporting Tools

This section answers RQ4, *"What supporting tools are used when simulation, optimization, and machine learning are combined?"* All 63 selected studies were examined towards the identification of the supporting tools used in architecture simulation. The simulators, simulation toolkits, or simulation API extracted are summarized in Figure 11. 14 studies used commercial (proprietary) tools, including GEM5[5], Matlab[6], and Modelica[7] for different tasks, such as modeling and simulation of intelligent control systems (S1), analysis of real-time network traffic for cloud architectures (S36), optimization and simulation of system architectures (S10), simulation of energy consumption in embedded systems (S26), and validation and simulation of distributed control systems (S2).

[5] <https://www.gem5.org/>
[6] <https://www.mathworks.com/products/matlab.html>
[7] <https://modelica.org/>

Nine studies used tools developed by their authors and addressed optimization (no author developed tools to address machine learning). The authors integrated their approach to some technique or framework supported by the tool – as an example, S33 incorporated an architectural pattern defined into a tool. Other authors developed new features not natively supported by tools (e.g., statistical timing and power optimization into the simulator (S57)).

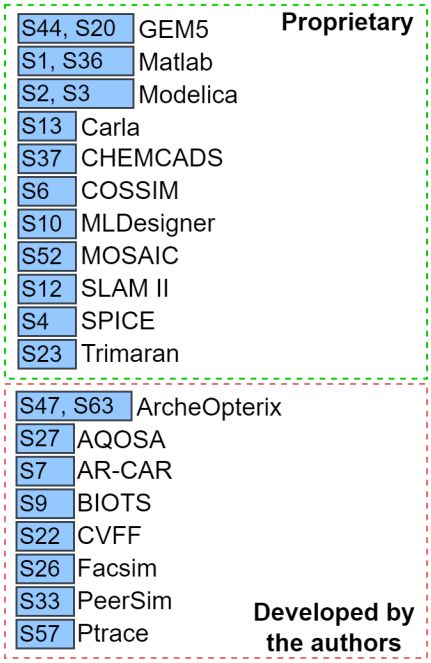


Figure 11 – Simulators used in the studies

The tools displayed two important abilities, namely flexibility and interoperability. The former enables tools to incorporate new algorithms or techniques – as an example, Matlab/Simulink, presented in S1 and S36, enabled users to design their algorithms and access vast literature and implementation libraries towards facilitating development and incorporation of elements built in other languages. Interoperability is the ability to communicate with other tools, either enabling developers to implement artifacts based on a framework supplied by the tool, or providing a co-simulation protocol for communication with other simulators and tools. For instance, COSSIM[8], in S6, enabled integration with several other simulators and tools, i.e., it can natively perform co-simulation with simulators such as GEM5, OMNeT++[9], and McPAT[10].

The studies showed the selection of a simulator is not an ordinary task, since it requires analyses of different alternative tools regarding several aspects, including performance and robustness. Commercial tools seem a good alternative due to their robustness and support provided. Moreover, the capability of commercial tools for communicating with external tools and integrating user-made algorithms has shown relevant for the selection of a commercial tool. The tools enabled authors to implement artifacts based on a framework they provided, or to use a

[8] <https://omnetpp.org/download-items/COSSIM.html>
[9] <https://omnetpp.org/>
[10] <https://www.hpl.hp.com/research/mcpat>

co-simulation protocol to communicate with other simulators and tools. Since some tools either are not flexible for integration with external algorithms, or may not be effective for some classes of systems, some studies proposed their own simulators, although their development is a highly complex task.

### 3.1.8 Maturity Level of Studies

This section addresses RQ5, *"What are the maturity levels of the existing studies?"*. Since any new solution should have a certain level of maturity for effectively contributing to a given area, this RQ elucidates the maturity of the studies. Technology Readiness Level (TRL) inspired its definition, since it enables the estimation of the maturity level of not only technologies and/or systems on a 1 (lowest level) to 9 (highest one) scale (MANKINS, 2009), but also other artifacts with proper adaptations (JR; ALBERT; GARCIA-MILLER, 2010). TRL was adjusted to our study into: Level 1 (Study reports basic principles with no examples), Level 2 (Study reports basic principles with running examples), Level 3 (Study presents a proof-of-concept), Level 4 (Study vas validated in a laboratory environment), Level 5 (Study was validated in a relevant environment), Level 6 (Study involves demonstration in a relevant environment), Level 7 (Study involves demonstration in a real-world environment), Level 8 (Study involves demonstration and testing in a real-world environment), and Level 9 (Study proved successful in a real-world environment).

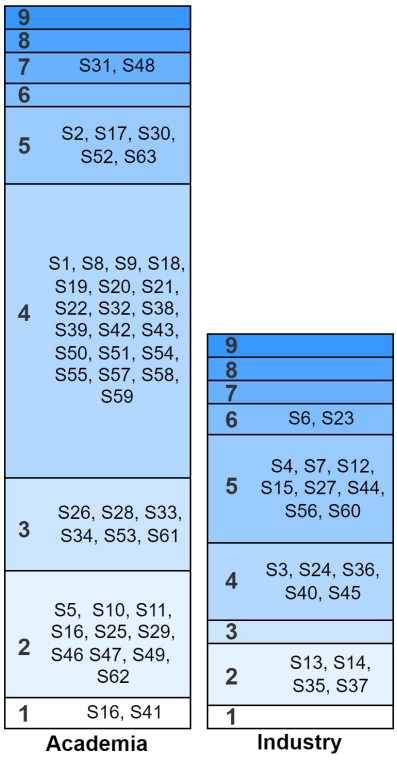


Figure 12 – Distribution of the primary studies according to maturity levels (Level 1 to 9)

Each study was deeply analyzed and categorized into Levels 1 to 9, as shown in Figure

12. Approximately 1/2 of studies from Level 5 (10 out of 19) are related to industry, whereas around 1/6 (7 out of 44) come from academia. Regarding Level 3 and below, approximately 1/5 of studies (4 out of 19) are related to industry, and around 2/5 (18 out of 44) come from academia. As expected, such numbers show industry studies aim at higher levels of maturity, validating studies outside the laboratory environment, whereas academic ones show a lower maturity level and are validated in a laboratory environment, or even not validated. More than half of the studies (24, of which 19 come from academia and 5 are industry ones) showed a medium-high maturity level, i.e., Level 4.

Lower-level studies can be considered valuable, since they may offer innovative ideas and solutions. For instance, S16, in Level 1, presented a design for the communication of expert systems architecture; the authors proposed an architecture with some component and data structures and the use of simulation to optimize the overall communication in. However, no examples or evaluation of the proposal were provided.

On the other hand, two studies in Level 7 (S31 and S48) reached the highest level. In particular, S31 presented a standalone data-driven service discovery framework deployed in a real-world environment that enables client programs to discover available functionalities and microservices. S48 introduced an architecture - available in a real-world environment - for IoT based on blockchain and artificial intelligence technologies for decentralizing authorities and improving interoperability and security.

From a broader view, the studies returned in the SMS revealed important initiatives for improving the quality of system architectures and/or facilitating the architect's work in seeking for those close to the ideal ones. However, several of them still lack extensive validation

## 3.2 Discussions

This section describes the main findings, research perspectives, and threats to validity related to this research.

### *3.2.1 Main Findings*

In what follows are the main findings from the insights gathered from the SMS:

- **Various initiatives combine optimization and/or machine learning with simulation:**: Important initiatives, including several with industry involvement, combining optimization and/or machine learning with simulation have been proposed and published over the past years, increasingly drawing the attention of academic and industrial communities, which experimented with different combinations through testing a number of optimization and machine learning techniques. Some preference for the adoption of genetic algorithms with simulation has been observed. Concerning the combination of three topics, namely

simulation, optimization, and machine learning, studies have used machine learning to diversify optimization meta-heuristics or parameterize optimization algorithms and optimization to generate machine learning models. Therefore, such a union seems to be interesting for further research.

- **Diverse quality attributes have been addressed**: The application of simulation supported by optimization and machine learning techniques has proven successful in measuring and predicting different quality attributes. The adoption of numerical or meta-heuristics techniques can improve quality attributes - in particular, performance has been highlighted, followed by reliability, adaptability, and efficiency. Such attributes were stressed due to the considerable costs and efforts spent on the evaluation of architectures of those critical and large-scale software-intensive systems, where the attributes may need to be assured.

- **Software-intensive systems in critical domains have been the focus of selected studies**: The selected studies addressed heterogeneous application domains and are, to some extent, related to complex and critical application domains, such as cyber-physical systems, smart cities, control systems, and Industry 4.0. Therefore, there are concerns over the development of those critical systems and new solutions have been tested. Machine learning and optimization were adopted in the present study for simulating the architecture of those systems. S2 displayed a structured framework for the simulation-based validation of architectures of Systems-of-Systems (SoS), i.e., large-scale, distributed, and coordinated systems. The framework focused on reductions in the significant engineering efforts for validating the systems by increasing the reusability of simulation models and supporting the plug-and-play of different algorithms.

- **Simulators are essential for studies involving simulation, optimization, and machine learning**: A successful architectural simulation depends directly on good supporting tools such as simulators, which are also essential when the simulation is combined with optimization and machine learning. A diversity of proprietary simulators, including the well-known Matlab and Modelica were employed in the studies returned by the SMS. Studies have also introduced new tools developed by their authors possibly due to a lack of tools that support their approaches. Moreover, the ability to interoperate with external tools and/or the ability to extend and aggregate new implementation (for instance, new machine learning algorithms) is a requirement that would make the tools more widely adopted.

### 3.2.2 Threats to Validity

The threats to validity identified in the SMS refer to search and selection of studies, extraction of data from studies, and synthesis of results. The countermeasures for mitigating them are discussed below.

- **Search for studies:** The search string might have imposed threats during the search for studies. Towards mitigating them, the process described in KITCHENHAM; BUDGEN; BRERETON was adopted for the definition of the string. Studies were searched in three publication databases, namely IEEE Xplore, ACM DL, and Scopus, which are the most relevant sources in software engineering (DYBA; KITCHENHAM; JORGENSEN, 2005; KITCHENHAM; BUDGEN; BRERETON, 2015). No limits to the publication date were imposed so that as many relevant primary studies as possible could be gathered. Studies suggested by an expert in the field were also considered and the returning of the most cited ones in the area was checked. The search included conference papers, journal articles, technical reports, and book chapters; despite the efforts for the inclusion of all relevant studies, some may have been missed.

- **Selection of studies:** Towards an unbiased selection of studies, RQs and selection criteria were defined in advance. The SMS protocol was reviewed by four people with extensive experience in conducting secondary studies and both questions and criteria were detailed so as to provide an assessment of how the final set of studies was obtained. At least two researchers read each study for increasing the reliability of the results and a third participant reviewed the studies for making a final decision in case of conflicts in the application of inclusion and exclusion criteria. Some studies might have been excluded in the first step, since title, abstract, keywords, introduction, and conclusions sections lacked important information. The SMS protocol was systematically followed for avoiding bias and assuring the inclusion of all relevant evidence; therefore, the final 63 studies are expected to reflect the state-of-the-art of simulation of system architectures by optimization and machine learning.

- **Data extraction validity:** Some data from primary studies were interpreted, since, apparently, they were not easily extracted. As a countermeasure for such a threat, apart from the data extraction form used, a specialist doubled checked all data extracted and interpretations required. The specialist identified some inconsistencies and incompatibilities, which were then discussed until a consensus had been reached. The specialist and the authors of the study also used the extracted data for triggering discussion sessions.

- **Result synthesis validity:** It refers to the possibility of our opinion or knowledge having influenced the synthesis of the results. As a countermeasure to assure their validity, all the SMS authors have researched into software architecture, simulation, and SoS area for years, so that our previous knowledge has probably supported such a synthesis better. Moreover, a detailed SMS protocol (as addressed in Section 3) based on well-established guidelines for SMS was previously defined and systematically followed (KITCHENHAM; BUDGEN; BRERETON, 2015).

## 3.3 Final Considerations

This chapter has provided a panorama of how optimization and machine learning have supported architectural simulation. Despite several important initiatives, many issues remain open and should be further investigated towards being consolidated and making *machine learning & optimization-based simulation* suitable and widely adopted for critical, complex, and large software-intensive systems.

The evaluation of architectures of a complex software-intensive system is an arduous, time-consuming, and costly task. Simulation has offered important benefits to the system architecture evaluation in a scenario where such architectures have increasingly assumed diverse architectural configurations at runtime and the impact of architectural changes is complex to be assessed and predicted still at design time. Whereas architectures have become progressively large and complex, simulation must advance through the incorporation of means that leverage its results. The combination of machine learning and optimization with simulation can offer several benefits and has particularly shown effective in (i) improving several quality attributes in software architectures, (ii) finding near-optimal architectural solution reorganizing architectures in case of changes and maintaining near-optimal configurations, and (iv) creating prediction models that assess the impact of events on the architectures. The main benefit is the quality of software architectures, since all theoretical foundations involved can assure near-optimal architectural configurations with fewer human efforts in comparison with simulation with no machine learning and optimization. The next chapter presents the framework developed for the simulation-based optimization of SoS software architectures.

CHAPTER

# 4

# FRAMEWORK FOR A SIMULATION-BASED OPTIMIZATION OF SOS ARCHITECTURES

Systems-of-systems (SoS) will likely form the next generation of software-intensive systems (JAMSHIDI, 2009; BOEHM, 2006), often supporting missions in critical domains. Although high qualities are essential requirements of those systems, their assurance has not been a trivial task. Simulation can be combined with other techniques, such as machine learning and/or optimization, towards assisting architects in evaluating an architectural configuration and generating a better-quality one. This chapter presents a framework that uses simulation-based optimization to generate a near-optimal configuration of an SoS software architecture and MS4Me simulator as a base; it adopts DEVS as the language to describe the architecture models for simulation.

The framework consists of three main parts, namely (i) an algorithm, described in Section 4.1, which performs the optimization process; (ii) a process that defines a set of phases required for the simulation-based optimization in an SoS software architecture (described in Section 4.2); and (iii) an infrastructure for the optimization process (described in Section 4.3).

## 4.1 Optimization Algorithm

Genetic algorithm was chosen for the optimization after analyses of the results of the mapping conducted, which showed it has been massively adopted in the context of software and systems architecture due to its versatility and power to search for near-optimal solutions. It also works well when an objective function cannot be mathematically established, since it enables the use of simulations to evaluate the individuals of a population. However, since genetic algorithms cannot work with DEVS models, such models must be treated for the generation of those the algorithms can work with. Figure 13 shows the encoding process that transforms the model described with the use of DEVS in a gene defined by a binary string. For such a purpose, DEVS

is described as a labeled directed multigraph, seen as a set of tuples (origin, destination, label) where origin is the atomic model sending, destination is the atomic model receiving, and label is the port where communication occurs. The tuples enable the encoding of the model names and are joined to forming a chromosome.

The established optimization process consists of the four classic spaces of a genetic algorithm, namely (i) fitness evaluation, (ii) selection, (iii) crossover, and (iv) mutation and is conducted at each iteration of its execution, as shown in Figure 13.

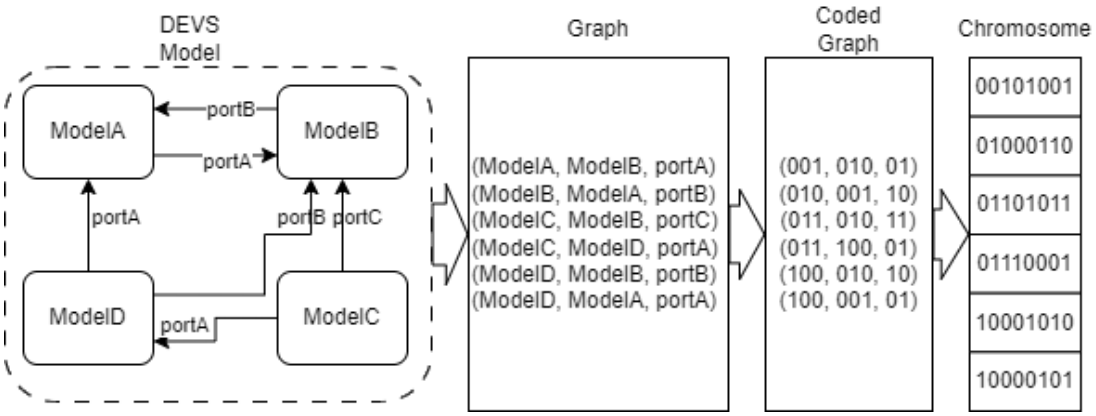


Figure 13 – Encoding process to represent DEVS as a chromosome.

**Fitness Evaluation.** The evaluation process consists in assigning a value to the quality of each individual in the population. In our case, results of the simulation of the architecture referring to each individual of the population were used.

**Selection.** The selection process aims to find individuals with highest quality so that they remain in the population and generate new ones through crossover. Elitist selection was the process adopted; it selects individuals with the best results in the fitness evaluation from the population.

**Crossover.** The crossover process takes place from selected individuals - the chromosome of an individual is divided into n parts that are exchanged with parts of the chromosome of another individual for generating a new individual containing parts of the solution of each of the two individuals. Therefore, the individuals generated by crossover will contain part of the graph of one of the parents and part of the other. The point that separates the chromosomes can be either fixed, or random; however, the chromosome size of the generated individuals and the order of the genes must be maintained.

**Mutation.** The mutation process aims to increase the genetic variability in a population. Individuals are randomly selected and one of their genes are randomly altered, thus enabling the generation of solutions with characteristics different from those of the individuals selected by crossover.

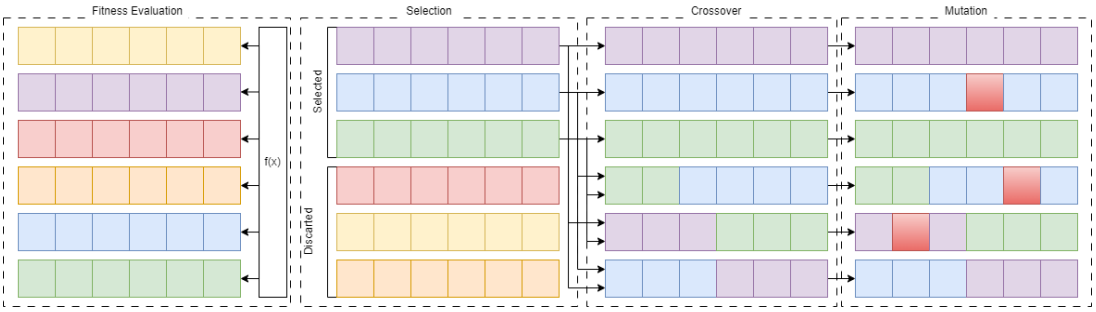


Figure 14 – Genetic Algorithm developed.

## 4.2 Process

A process was established for the use and execution of the developed infrastructure and divided into two main stages, namely (i) preparation, in which the architect develops the elements to be used by the infrastructure, and (ii) execution, in which the infrastructure uses the elements developed in the first phase and conducts an internal optimization process (see Figure 15).

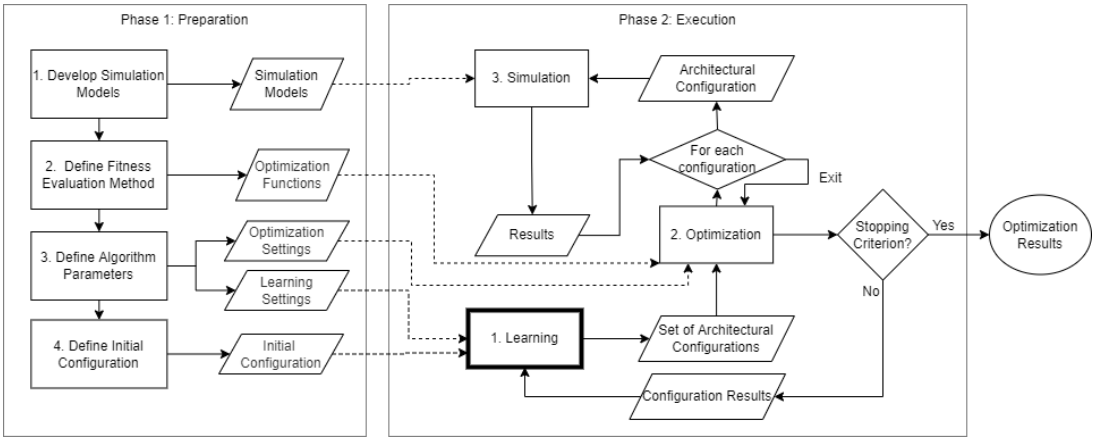


Figure 15 – Process for the simulation-based optimization of SoS architectures

### 4.2.1 Phase 1: Preparation

The preparation phase involves the specification of the artifacts necessary for the optimization and the definition of the parameters of the genetic algorithm and contains the following steps:

**Development of Simulation Models:** the architect must develop all simulation models to be used during the optimization, i.e., the atomic models of all constituents that can be part of the SoS. Therefore, all state machines and internal logic of the constituents must be developed and all types of data to be used and their generation or collection must be defined.

**Definition of Fitness Evaluation Method:** both quality attributes to be extracted from the simulation and the method the simulation will adopt to generate them must be defined.

**Definition of Algorithm Parameters:** the parameters of both learning and optimization algorithms are defined. Regarding the learning algorithm, population size, mutation rate, and crossover strategy must be determined, whereas convergence criterion and optimization restrictions must be defined for the optimization algorithm.

**Definition of Initial Configuration:** the initial architectural configuration must be defined, i.e., a coupled model that characterizes the initial configuration, or defines an initial set of constituents and the rules for coupling between the constituents.

### 4.2.2 Phase 2: Execution

The simulation-based optimization process is conducted in this phase towards the generation of a near-optimal architectural configuration. Phase 2 contains the following steps
**Learning:** a genetic algorithm is executed for generating a population with a set of architectural configurations. The algorithm performs an elitist selection of the previous population, i.e., it chooses individuals with the best fitness function evaluation and, from the set, performs the crossover step, i.e., the chromosomes of the individuals are divided and merged with other individuals to generating a new individual. The mutation step is then performed, and genes whose values have not been changed are randomly selected according to a mutation rate.

**Optimization:** the settings generated by the learning algorithm are filtered, i.e., individuals that satisfy the constraints of the problem are selected. Each architectural configuration is then sent to the simulator for its evaluation. With the results, the optimizer analyzes whether any configuration that satisfied the convergence criterion has been found. If so, the process is terminated and the configuration is returned as a semi-optimal solution to the problem; otherwise, the optimizer sends the evaluations performed by the simulator for the learning stage for the generation of a new set of architectural configurations, thus repeating the entire execution process.

**Simulation:** the architectural configuration for which the simulation models are executed is evaluated, generating the evaluation of the fitness function of the architectural configuration based on the developed evaluation method.

## 4.3 Infrastructure

A framework that receives the artifacts created in the preparation phase and performs the necessary processing towards generating a close to optimal architecture was designed so that the execution phase of the process established in the previous section could be conducted, as show in Figure 16. The infrastructure of the framework contains:

**Machine learning Algorithm:** a Genetic algorithm that adopts probabilistic search proce-

dures to work on architectural configuration represented by strings was developed; it uses the five standard phases of a genetic algorithm, namely (i) initial population, (ii) fitness function, (iii) selection, (iv) crossover, and (v) mutation. The process begins with an initial population of which each individual is an architectural configuration. An individual is characterized by a set of parameters - in our case, vertices and edges (genes). Genes are joined into a string to forming a graph (chromosome). In our genetic algorithm, the set of genes of an individual is represented by a string. The fitness function determines the fitness of an individual, i.e., the ability of an individual to compete with other individuals, providing a fitness score to each individual. The probability of an individual being selected for reproduction by our algorithm is based on its fitness score. In the crossover of the algorithm, for each pair of parents to be mated, a crossover point is randomly chosen within the genes. In certain new offspring formed, some of their genes can be subjected to a mutation at a low random probability, implying some of the bits in the bit string can be flipped. The algorithm generates a set of architectural configurations at the end of each iteration.
**Configuration:** Size of population, mutation probability, and crossover strategy.
**Input:** Fitness evaluation of the current population.
**Output:** New population of architectural configurations.

**Optimization Algorithm:** a population-based optimization algorithm that receives a population and a set of constraints to be checked in each individual of the population was developed. For each selected individual, the algorithm evaluates its fitness function and then analyzes if the population has fitted the convergence criteria. In a positive case, the algorithm outputs the best individual; otherwise, it proceeds to the next iteration.
**Configuration:** convergence criteria and constraints.
**Input:** Architectural configurations.
**Output:** Selected architectural configurations.

**Simulation:** The simulation evaluates each architectural configuration by executing simulation models that characterize them and evaluate their execution through a previously defined method. Each constituent of the SoS must be defined as an atomic model, i.e., by a timed state machine, and define the state machine interface.
**Configuration:** quality attribute evaluation method.
**Input:** architectural configuration models.
**Output:** quality attribute evaluation.

**Interface:** The interface has two main functionalities, namely (i) connection of simulation and optimization and (ii) translation of the architectural configuration models. It creates a TCP connection towards message exchange between the optimization and the simulation so that the optimization sets a new simulation of an architectural configuration and the simulation sends the

simulation results. The interface also translates the architectural configuration described mathematically as a graph in the optimization and generates an equivalent coupled model defining the architectural configuration to be used by the simulation.
**Configuration:** TPC settings.
**Input:** Graph describing the architectural configuration.
**Output:** Coupled Model describing the architectural configuration.

**Encoder:** responsible for encoding the architectural configuration described as a graph by the optimization algorithm in a genetic string.
**Configuration:** Encoding method.
**Input:** Graph describing the architectural configuration.
**Output:** String describing the architectural configuration.

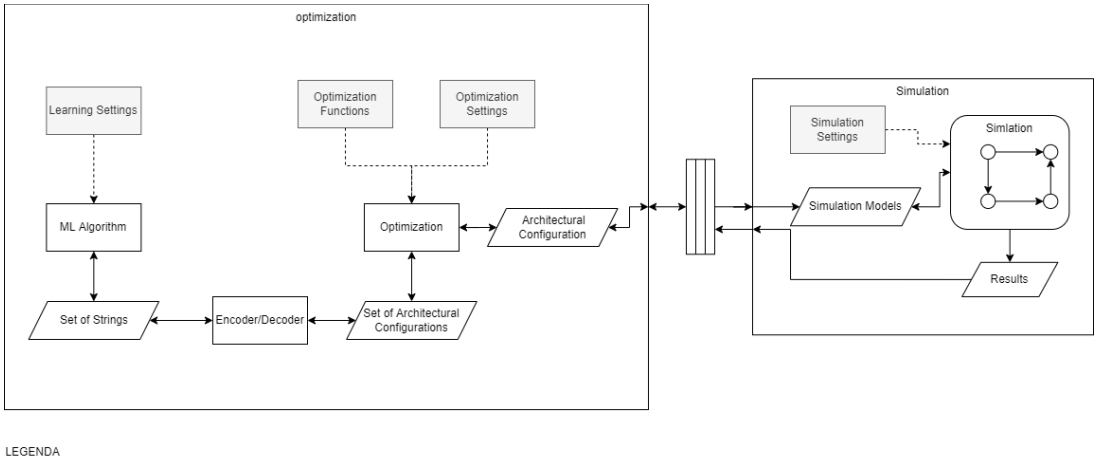


Figure 16 – Infrastructure for the simulation-based optimization of SoS architectures

## 4.4 Final Considerations

This chapter has introduced a framework that performs simulation-based optimization in SoS software architectures. The two main elements of the framework were defined and detailed, establishing a process towards the simulation-based optimization. A process with steps to be performed by the architect to prepare the artifacts required by the framework and the steps for the simulation-based optimization on SoS software architectures were described. An infrastructure also established defined all the components necessary for the process.

The next chapter addresses a case study conducted for the evaluation of the framework.

CHAPTER

# 5

# EVALUATION

A scenario that supported a case study was defined for the evaluation of the framework. It consisted of an intelligent building comprised of sensors, actuators, and control units hierarchically deployed throughout the building. The evaluation was performed in two stages, involving a convergence analysis that assessed the framework's capacity to generate near-optimal SoS software architectures and a benchmarking that checked the computational cost of the framework execution.

Section 5.1 presents the scenarios established; Section 5.2 reports the convergence analysis on the capability of the framework for providing meaningful results; Section 5.3 describes the benchmarking that exercised the scalability of the framework; Section 5.4 focuses on threats to validity; finally, Section 5.5 is devoted to the final considerations.

## 5.1 Scenario

In the context of smart cities, smart buildings provide important services to their residents and visitors, such as energy savings by light control by sensors and fire monitoring, as specified by Gassara et al. (GASSARA *et al.*, 2017). Such an SoS is an important instance of problems addressed, in particular because such SoS is composed of constituents that are, themselves, other entire SoS. Smart Building is an important instance of problems addressed, in particular because it is composed of constituents that are another entire SoS. Smart Building is composed of three other SoS, namely Fire System, Lighting System, and Room and SBS has a Smart Building Control Unity (SBCU) that manages the constituent systems of the building. Figure 17 exemplifies its architecture.

Fire System is an SoS that controls fire alarm in smart building public areas and consists of: :

- **Fire System Control Unities (FSCU)**: constituent systems that manage the Fire System

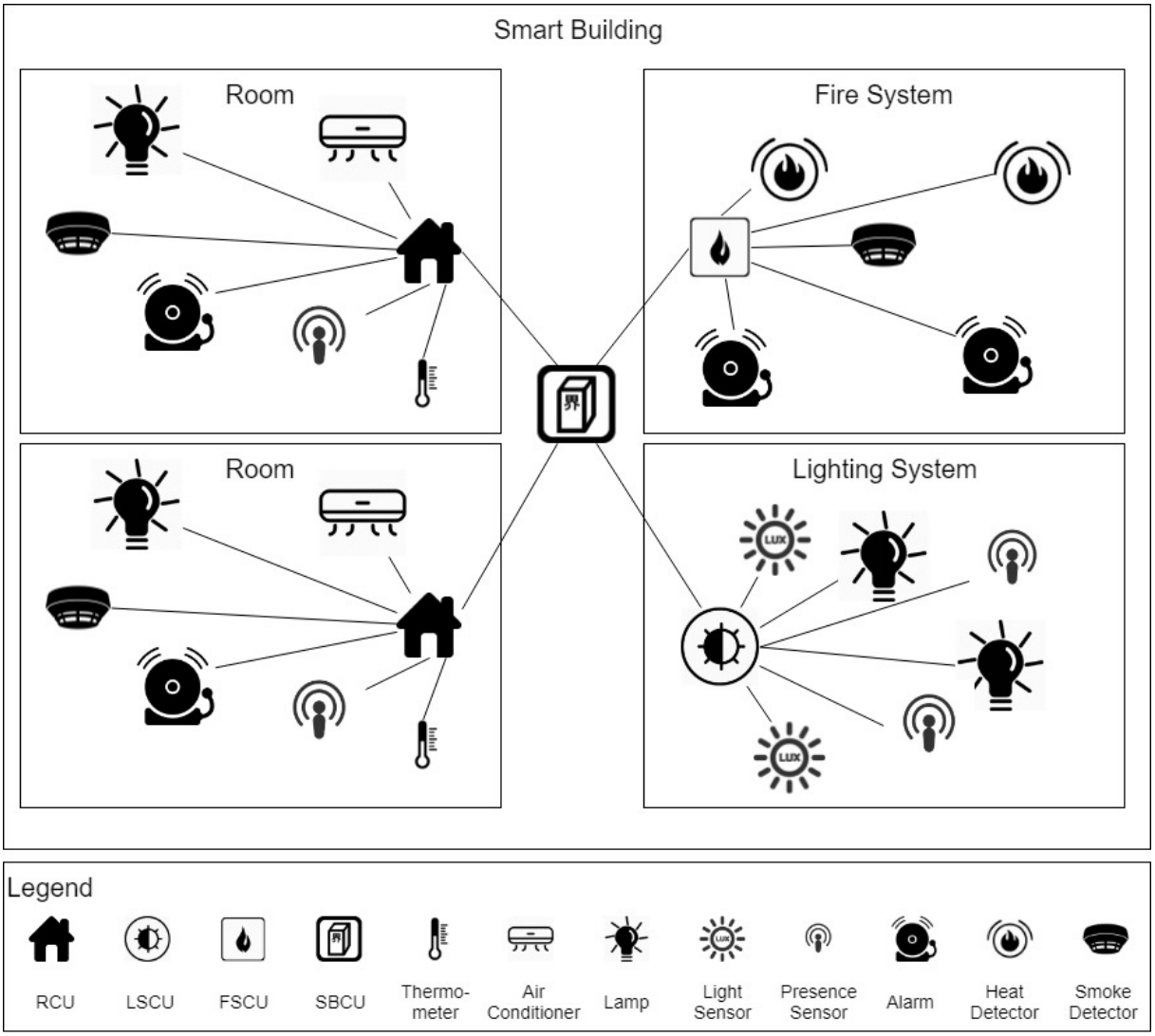


Figure 17 – A representation of a Smart Building SoS.

receiving sensor data and, if necessary, trigger alarms and sprinklers in the area where fire was detected. Once fire has been identified, FSCU informs SBCU about it;

- **Smoke Detectors**: detect the presence of smoke and send a signal to FSCU. Smoke Detector (an Ionization Smoke Detector was used) opens to receive air and assess the presence of smoke. Once smoke has been detected, a signal is sent to the fire alarm;

- **Heat Detectors**: temperature sensors that check the ambient temperature and emit a signal to the fire sprinkler when the temperature is higher than 58°C (136.4°F);

- **Fire Sprinklers**: once triggered, such systems fire water for the elimination of fire outbreaks; and

- **Fire Alarms**: once triggered, such systems trigger audible alerts signaling the occurrence of fire in the Smart Building.

Lighting System is an SoS that lights the public areas of the Smart Building and consists of:

- **Lighting System Control Unities (LSCU)**: units that manage the Lighting System, receiving sensor data and, when necessary, turning lamps on;
- **Light Sensors**: convert the energy of visible or infrared light (photons) into an electrical signal (electrons). They capture environment light measures and send the values to sensors that cover the area to receive light;
- **Presence Sensors**: devices that detect presence in an environment; and
- **Lamps**: smart lamps that can be digitally controlled and turned on / off and whose intensity is remotely controlled.

Room consists of private environments (e.g., apartments, offices, and stores) and can contain elements such as Light Sensors, Presence Sensor, Lamp, Smoke Detector, and Fire Alarm. They also encompass

- **Room Control Unities (RCU)**: a system that manages the room in which it is inserted and communicates with SBCU. It also controls temperature, room lamps, and the internal fire system and receives/sends fire alerts to SBCU;
- **Thermometers**: temperature sensors that collect the ambient temperature and send it to RCU; and
- **Air Conditioners**: devices that cool environment; they are triggered when the temperature is higher than a stipulated value and the presence sensor is triggered

Smart Building must accomplish three main missions, namely *fire control*, *light management*, and *air conditioning*. Towards undertaking the former, it has a Fire System, which uses smoke sensors and heat sensors to detect a fire and notify FSCU. The Fire System uses FSCU to trigger an alarm to inform people about fire occurrence and an eminent need to evacuate the building. Fire Sprinklers aim at extinguishing the fire and notify SBCU about its occurrence. Rooms also may have sensors, alarms, and sprinklers to assist Smart Building in the accomplishment of the fire control mission. The light management mission is undertaken through the interoperation of light sensors, presence sensors, smart lamps, and LSCU. The latter receives data from sensors and, if necessary, properly activates smart lamps at a suitable intensity. Rooms also have presence sensors and lamps for internal lighting. The air conditioning mission is accomplished in a room through thermometers, presence sensors, and air conditioners towards keeping the air conditioner on only when necessary.

Smart Building also exhibits the following characteristics of an SoS (MAIER, 1998; OQUENDO, 2016):

- **Operational independence of the constituents**: Internal constituents of a Room are not exclusively dedicated to the Intelligent Building, since they may accomplish internal missions and become unavailable to the Smart Building. Therefore, the operational independence of constituents is preserved;

- **Managerial independence of constituents**: A diversity of stakeholders and companies can have Rooms in the Smart Building; each Room has its internal policy for internal composition and data availability for the Smart Building;

- **Distribution**: Smart Building requires a network to establish communication with its constituent systems and accomplish its missions due to the interoperability constituent;

- **Evolutionary Development**: A Room may change its internal composition and make available new functionalities for the Smart Building towards contributing to its accomplishing new missions or improving the existing mission, hence, evolving; and

- **Emergent behavior**: For fire control, Smart Building requires Rooms and Fire Systems notice SBCU about the occurrence of a fire in the building so that SBCU sends an alarm to all Rooms, areas, and firefighters. Smart Building also requires Light System and Rooms interoperate towards managing the lighting and energy consumption of the building. Therefore, the entire Smart Building operation for accomplishing pre-established missions is possible due to behaviors that emerge from the interoperability of constituent systems.

A problem related to such architectures is loss of data, which occurs mainly when the number of constituents is increased, thus increasing competition for resources (MANZANO; NETO; NAKAGAWA, 2019). The framework can help architectures minimize such a loss. The optimization problem to be treated was modeled towards containing the following elements:

- **Objective Function:** finds an architectural configuration that minimizes data loss from the constituent systems by the Control Unities.

- **Fitness function:** data loss is calculated by a simulation using a pre-defined dataset.

- **Variables:** couplings between the constituents and control unities.

- **Constraints.**

  - Each constituent must be connected to an adequate control unity (e.g. light Sensors must be connected to an LSCU).
  - A constituent can connect with control unity only in the range of communication.
  - A path must exist between each constituent and the SBCU.

Table 2 – Architectural configurations adopted in the studies

| # | T | AC | RCU | LS | PS | L | LSCU | SD | HD | FS | FA | FSCU | BCU | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1 | 20 | 20 | 10 | 20 | 20 | 20 | 2 | 20 | 5 | 20 | 20 | 2 | 2 | 161 |
| A2 | 20 | 20 | 10 | 25 | 25 | 25 | 2 | 25 | 5 | 25 | 25 | 2 | 2 | 191 |
| A3 | 20 | 20 | 10 | 30 | 30 | 30 | 2 | 30 | 10 | 30 | 30 | 2 | 2 | 226 |
| A4 | 20 | 20 | 10 | 30 | 30 | 30 | 5 | 30 | 10 | 30 | 30 | 5 | 5 | 235 |
| A5 | 30 | 30 | 10 | 30 | 30 | 30 | 5 | 30 | 10 | 30 | 30 | 5 | 5 | 245 |
| A6 | 30 | 30 | 15 | 30 | 30 | 30 | 5 | 30 | 10 | 30 | 30 | 5 | 5 | 250 |
| A7 | 40 | 40 | 15 | 40 | 40 | 40 | 5 | 40 | 15 | 40 | 40 | 5 | 5 | 325 |
| A8 | 40 | 40 | 20 | 40 | 40 | 40 | 7 | 40 | 15 | 40 | 40 | 7 | 7 | 336 |
| A9 | 50 | 50 | 20 | 60 | 60 | 60 | 7 | 40 | 20 | 50 | 50 | 10 | 10 | 437 |
| A10 | 50 | 50 | 25 | 60 | 60 | 60 | 10 | 40 | 20 | 50 | 50 | 10 | 10 | 445 |

An SoS architecture was specified for Smart Building (A1) and involved a Fire System, a Lighting System, 10 Rooms, and two Building Control Unities. Altogether, the initial configuration contained 20 Thermometers (T), 20 Air Conditioners (AC), 10 Room Control Unities (RCU), 20 Light Sensors (LS), 20 Presence Sensors (PS), 20 Lamps (L), two (2) Lighting System Control Unities (LSCU), 20 Smoke Detectors (SD), five (5) Heat Detectors (HD), 20 Fire Sprinklers (FS), 20 Fire Alarms (FA), two (2) Fire System Control Unities (FSCU), and two (2) Building Control Unities (BCU).

**Data Preparation:** a realistic dataset composed of data representing 10 days that fed the simulation was built for simulating a Smart Building. A type of datum known as Lux (lx), which is the total luminous flux incident on a surface per unit area (illuminance), stimulated Light Sensors and data were generated in a 0.1 lx (at night) to 10,000 lx (in broad daylight) range. Data received from Light Sensors by the BCU are used to turn external lamps on when illumination is lower than 100. Data were randomly generated between 10 and 60 presences per sensor per day for feeding the Presence Sensors and used by BCU to switch lamps on in the Presence Sensor area. Data used for stimulating Smoke Sensors consisted of binary values (one, for smoke detected, and zero, for non-detected smoke) and 10 fires were chosen in a random area. The data were sent to an FSCU or RCU. If the value was 1, the alarm was triggered and fire sprinklers were activated in the detected smoke area. Finally, data were generated between 10°C (30°F) and 30°C (86°F) for thermometers and used by RCU to turn the air conditioners on in case of a presence detection in the area and when the temperature was higher than a given value.

## 5.2 Convergence Analysis

The following protocol was established for the analysis of whether the developed framework successfully performed simulation- based optimization on SoS software architectures:
**Method.** Perform the optimization on all the architecture configurations and monitoring the metrics.

**Research Question.** Can the infrastructure proposed successfully produce relevant results?

**Rationale.** This research question assesses whether the infrastructure can generate good results.

**M1 metric.** Percentage of data lost.

### 5.2.1 Conduction

The case study followed the phases and steps specified by the framework. Table 2 shows the architectures configurations established by DEVS, the constraints specified into DEVS models, and the population size of the genetic algorithms defined as 15. The execution was conducted for each architectural configuration, monitoring the evolution of M1 during the development of the case study.

### 5.2.2 Results

The simulation required approximately 35 hours to optimize all ten architectural configurations and was conducted on an Intel Core i5-10210U CPU 3.90 GHz x64, 8GB RAM, 256GB SSD, running Windows 11 64bits. Figure 18 displays the results and the data loss rate of the architecture at each iteration of the genetic algorithm and Table 3 shows excerpts of data collected during the simulation.

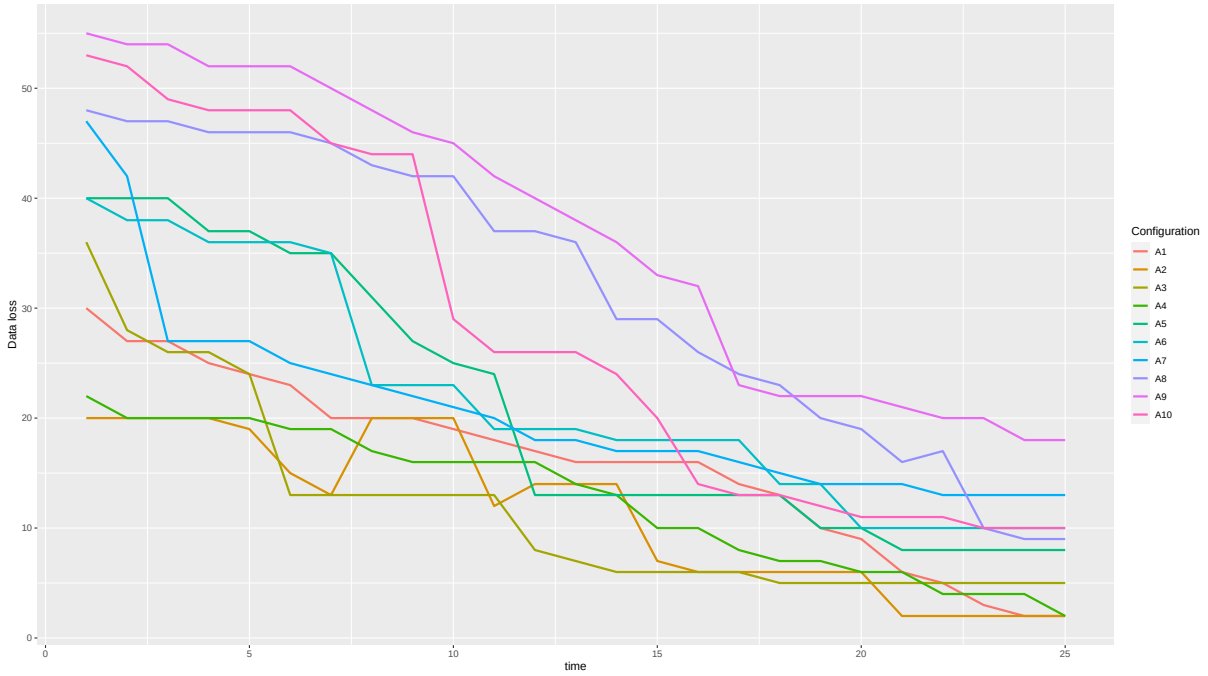


Figure 18 – Data lost by the architectures through the iterations.

According to the results, the framework generated architectural configurations with a low level of data loss, even in architectures with many constituents such as A10, which contains

Table 3 – Sample of data loss in smarting building.

| | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1 | 30 | 27 | 24 | 20 | 20 | 18 | 16 | 16 | 14 | 10 | 6 | 3 | 2 |
| A2 | 20 | 20 | 19 | 13 | 20 | 12 | 14 | 7 | 6 | 6 | 2 | 2 | 2 |
| A3 | 36 | 26 | 24 | 13 | 13 | 13 | 7 | 6 | 6 | 5 | 5 | 5 | 5 |
| A4 | 22 | 20 | 20 | 19 | 16 | 16 | 14 | 10 | 8 | 7 | 6 | 4 | 2 |
| A5 | 40 | 40 | 37 | 35 | 27 | 24 | 13 | 13 | 13 | 10 | 8 | 8 | 8 |
| A6 | 40 | 38 | 36 | 35 | 23 | 19 | 19 | 18 | 18 | 14 | 10 | 10 | 10 |
| A7 | 47 | 27 | 27 | 24 | 22 | 20 | 18 | 17 | 16 | 14 | 14 | 13 | 13 |
| A8 | 48 | 47 | 46 | 45 | 42 | 37 | 36 | 29 | 24 | 20 | 16 | 10 | 9 |
| A9 | 55 | 54 | 52 | 50 | 46 | 42 | 38 | 33 | 23 | 22 | 21 | 20 | 18 |
| A10 | 53 | 49 | 48 | 45 | 44 | 26 | 26 | 20 | 13 | 12 | 11 | 10 | 10 |

445. The best result was achieved for A1 and A2, reaching 2% of data loss, and the worst one was reported for A9, which generated an architectural configuration with 18% data loss. Therefore, the answer to the research question "*Can the infrastructure proposed successfully produce relevant results?*" is “Yes”, since the framework generated architectures with less than or equal to 10% of data loss for eight out of ten architectures.

## 5.3 Benchmarking

A second experiment with the smart building aimed at the evaluation of the computational performance of the framework. An evaluation protocol was established with respect to the following elements for the planning of the case study:
**Method.** Performs optimization with architecture configuration A1, changes population size, fixes it, and changes architecture configuration.

**Research Question.** Can the infrastructure successfully optimize architectures with good performance?

**Rationale.** This research question assesses whether the infrastructure can perform the optimization with good performance, showing it is functional and provides the desired result without high computational costs.

**M1 metric.** Amount of memory used for running simulation and optimization.

**M2 metric.** Amount of CPU used for running simulation and optimization.

**M3 metric.** Time used for performing the optimization.

### 5.3.1 Conduction

The evaluation of the framework performance was divided into two stages. The first referred to the optimization process with the use of the initial architectural configuration previously defined, followed by a change in the population size of the genetic algorithm for the verification of its impact on performance. In the second stage, the population size was maintained at 10 individuals and the number of constituents was increased, as shown in Table2. During the entire optimization process, the consumption of computational resources of both simulator and optimizer was monitored and the data were stored in a data file.

### 5.3.2 Results

The simulation required approximately 17 hours in the first phase of the benchmarking and around 9 hours and 40 minutes in the second phase and was performed on an Intel Core i5-10210U CPU 3.90 GHz x64, 8GB RAM, 256GB SSS, running Windows 11 64bits. During the evaluation, information on the computational cost of optimization was collected - Figure 19 shows the memory usage information (Metric 1), according to which up to 1.5 GB of memory were required for the optimization.

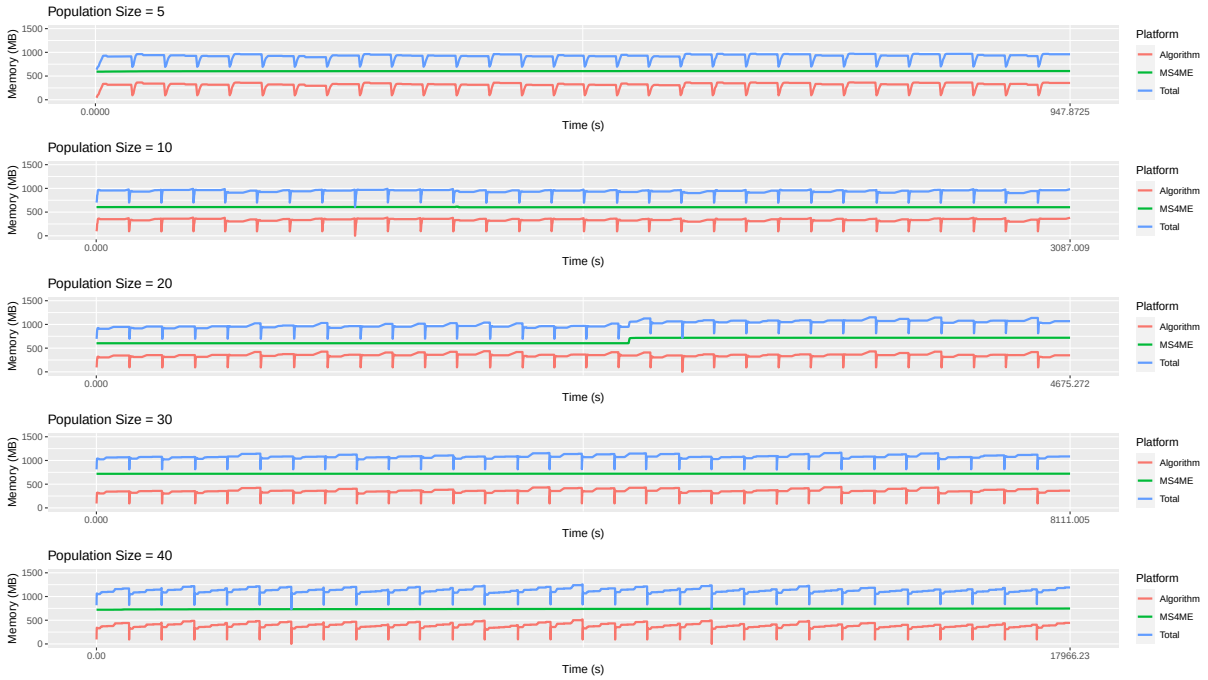


Figure 19 – Collection of memory usage data.

Data on CPU usage (Metric 2) were also collected (Figure 20). The infrastructure consumed up to 75% of the CPU of the machine that ran the simulation; however, on average, the utilization was 28% of the CPU.

Data regarding the time required for the optimization based on each absolute value (Metric 3) were collected (see Figure 21, which also shows an exponential growth of time as the

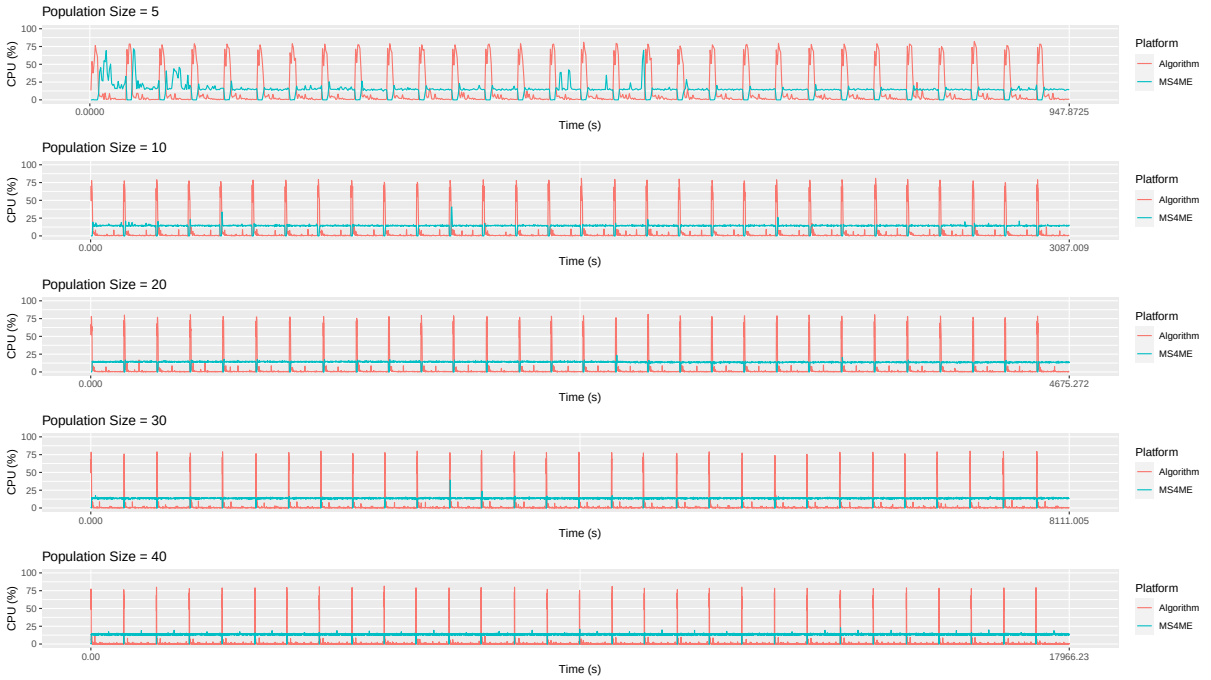


Figure 20 – Collection of CPU usage data.

absolute error value decreases with the simulation requiring 4 hours to perform the optimization with a population size of 40).

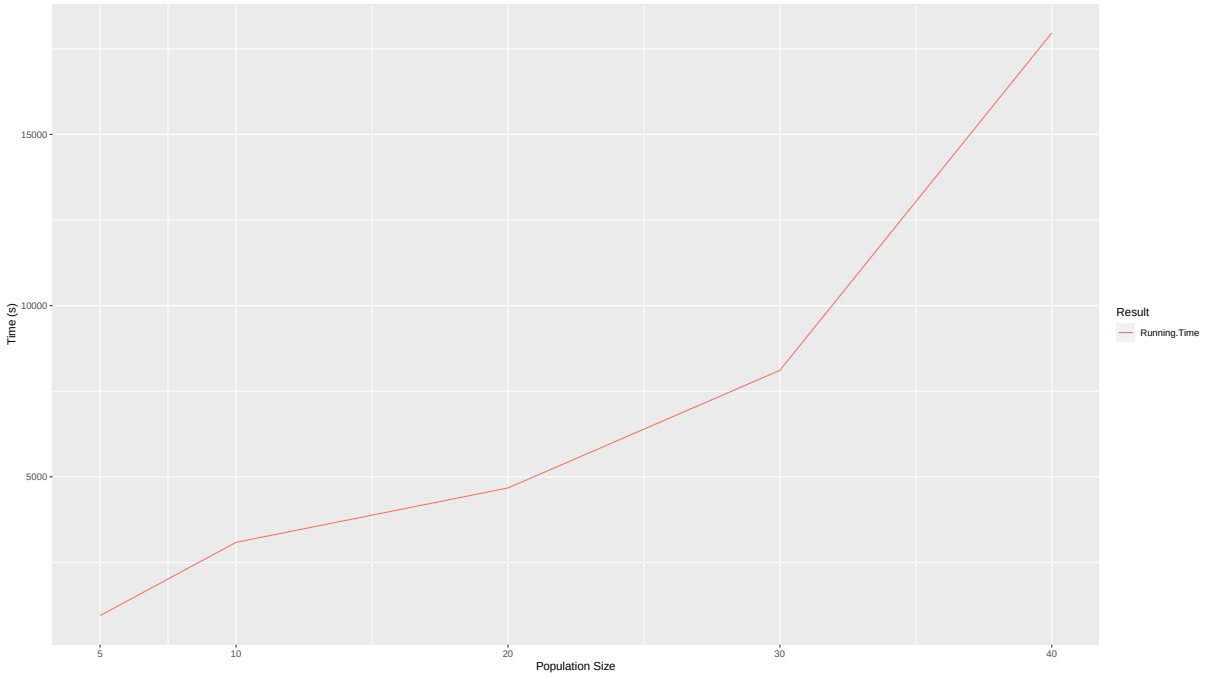


Figure 21 – Time required for the optimization.

Metrics 1 and 3 regarding the study exercising the architectural configuration scale and population size fixed at 10 (see Figure 22) were collected. The initial architecture configuration consumed 1.1GB of memory with 161 elements and the last one consumed 3.1GB of memory with 445 elements. Figure 23 shows the time taken (in seconds) for perform the optimization.

The initial optimization took 3123 seconds (52 minutes) and the larger one took 10943 seconds (180 minutes) with 445 elements.

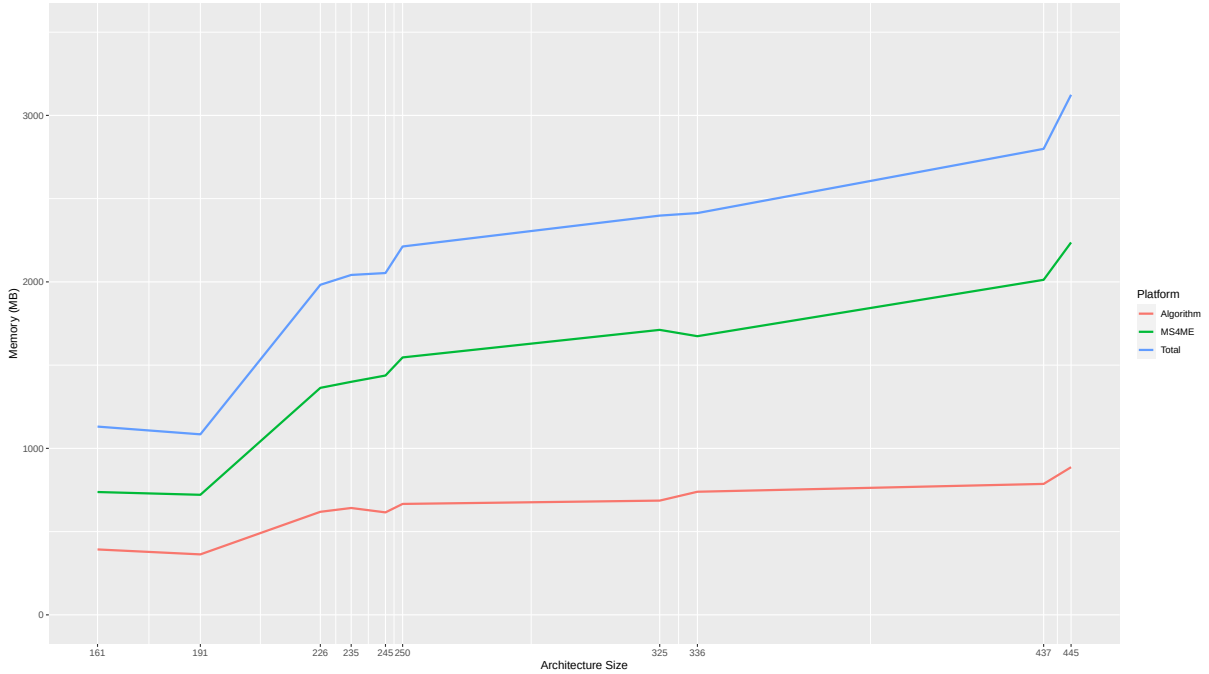


Figure 22 – Memory required for the optimization increasing the size of the architectural configuration.

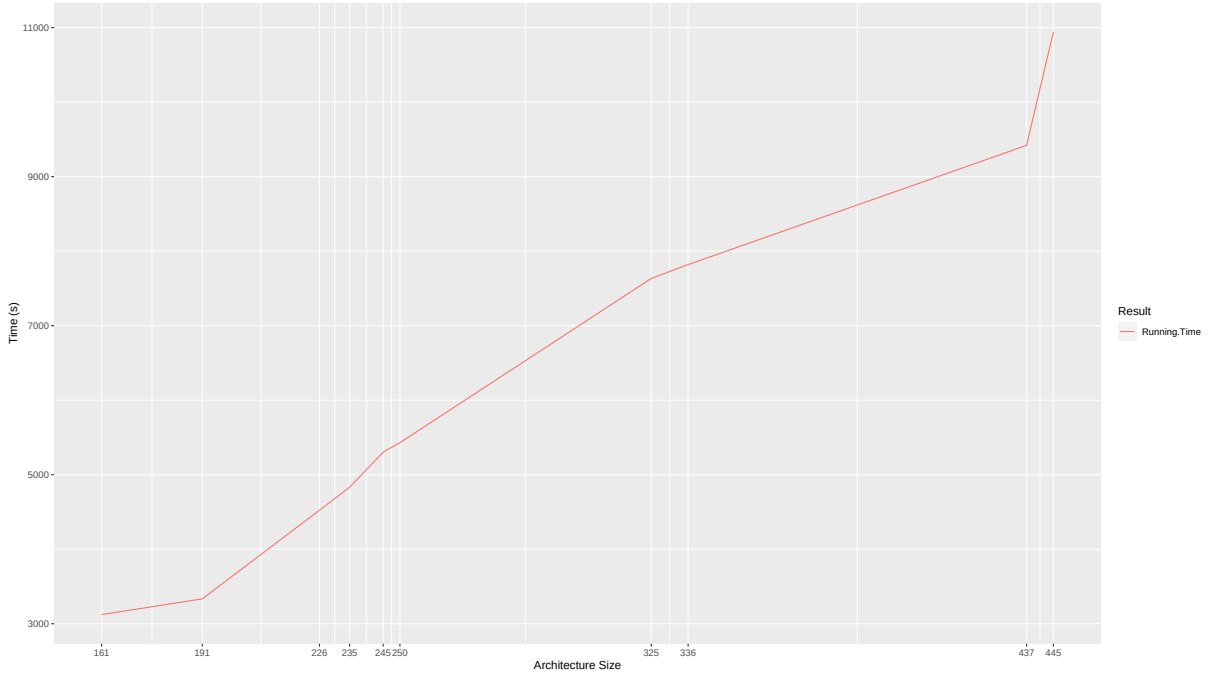


Figure 23 – Time taken to perform the optimization increasing the size of the architectural configuration.

According to the metrics collected, the answer to the research question "*Can the infrastructure successfully optimize architectures with good performance?*", is “Yes”, since it optimized them at a relatively low computational cost.

## 5.4 Threats to Validity

Scale, correctness of transformation rules, selection of dataset, and selection of changes set were some of the threats to validity identified in the study. Scale is addressed through a continuous increase in the number of constituents during the study until a couple hundred of systems had been simulated. As a side effect, it hampers data visualization and processing, since the visualization of a large number of constituents is difficult in a simulation and a more powerful processor is required for the simulation of a larger SoS. However, such problems were reduced by saving results in external files to be further analyzed.

The selection of data may display bias. For the simulation of Smart Building, our dataset was artificially constructed, which might have caused it. Towards minimizing that possible effect, several events, such as presence and fire, was randomly created, thus increasing the diversity of scenarios represented and reducing bias through the introduction of casuality in the data produced and used to stimulate constituents.

The set of reconfiguration operators and order in which changes occur were the last threat identified. Different changes could be performed, substituting or eliminating constituents, or even reorganizing the SoS configuration. However, the purpose of this study was to check whether our approach could successfully (i) support automatic generation of functional dynamic simulation models for SoS software architectures, (ii) reproduce effects of the reconfigurations in the SoS architecture, and (iii) support analysis of the effect of different architectural configurations on the generation of new coalitions in valid operational states. Since the approach succeeded in all those endeavors, that threat exerted no influence on the results.

## 5.5 Final Considerations

This chapter has addressed an evaluation conducted for exercising the developed framework regarding its results and the expenditure of computational resources. According to the results, the framework proved able to generate satisfactory solutions to the problem even in cases of architectural configurations containing a high number of elements and economical in relation to computational costs.

CHAPTER

# 6

# CONCLUSIONS

During this Master's project, a framework was developed to supporting the optimization of SoS software architectures. Machine learning and optimization approaches, techniques, and algorithms that might enhance the quality attributes of SoS architectures were investigated regarding their possible handling of SoS characteristics and/or the way they could be adapted to SoS system architectures. The results enabled the establishment of a framework that defines tools, algorithms, and techniques that handle the simulation-based optimization of SoS software architectures and evaluation of such elements concerning their ability to provide good solutions and deal with the SoS characteristics. An infrastructure was designed towards supporting the framework process and developing software elements for connecting the simulator with an optimization algorithm.

## 6.1 Contributions

The main contribution of this project is the design of a framework that supports the optimization of SoS software architectures and automatically generates near-optimal architectural configurations, improving the quality of SoS architectures and reducing both workload and time required by the architect for their creation.

The following contributions have also been achieved:

- **Design of a genetic algorithm that optimizes SoS software architectures.** A method that converts architectural configurations in DEVS into the genetic paradigm and another that initializes and evolves populations of architectural configurations have been established. The algorithm designed finds local minima and contains a method for the diversification of population to spread out in the search space, thus providing near-optimal solutions.
- **Establishment of a process that performs a simulation-based optimization of SoS software architectures.** The process encompasses activities and steps – also defined -

for the architect to generate the necessary artifacts for optimization. The elements and their interactions are established and the way the simulation results are combined with the optimization algorithm is reported. The process uses the artifacts defined by the software architect together with the optimization algorithm for producing close-to-optimal architectural configurations.

- **Design and development of an infrastructure for the simulation-based optimization.** The infrastructure developed enables the automatic execution of activities established in the process. The communication and translation between the paradigms used by the genetic algorithm and the simulator are also defined. As a result, the infrastructure can automatically generate near-optimal configurations.

## 6.2 Limitations

The following limitations related to the framework have been found:

- **MS4Me does not support parallelism.** The framework depends on the simulator used, therefore, scalability must be considered, mainly because MS4ME does not enable the parallel execution of each constituent. Large-scale SoS such as smart cities with thousands or even millions of constituents can be part of those SoS through different technologies, heterogeneous means of communication, different platforms, and different levels of complexity ranging from simple objects with sensors (e.g., those of IoT systems and cyber-physical systems (CPS)) to complex information systems or other entire SoS. Therefore, scale for SoS is still an open problem specially for the simulation domain.

- **DEVS is not an architectural description language.** DEVS is a simulation language of systems whose structure supports their accurate description with a high level of detail and low level of abstraction. It is not considered an architectural description language, since it does not have the desired abstractions for an architecture. However, Graciano Neto et al.(Graciano Neto *et al.*, 2018) developed SosADL, an approach that generates simulation models in DEVS from a model in an architectural description language built to support the SoS architecture activity for overcoming the limitation.

## 6.3 Future Work

The following research opportunities have been identified for future work:

- **Simulation should evolve to handling dynamic aspects of system architectures.** The evaluation of architectures of complex and heterogeneous software-intensive systems is a challenging task due to the diverse architectural configurations such systems can assume at

runtime and the complex assessments of the impact of architectural changes at design time. Therefore, research on simulation of system architectures must evolve to better addressing all dynamics aspects of those architectures, including their components, connectors, and the environment. In other words, simulation should focus more on dynamic architectures, as in S7, in which it uses a dynamic operation model to specify the behavior of the architecture in case of software and hardware failure. More efforts should be devoted to the exploration of the contributions of machine learning and optimization to simulation.

- **Scalability is still a challenge for architectural simulation.** Complex and critical application domains, such as smart cities, Industry 4.0, and transportation, are sometimes composed of smartphones, autonomous vehicles, smart buildings/houses, and many other IoT-based elements with different granularity, potentially reaching millions of constituents. The simulation of the overall architecture of systems that support such domains is a complex task, especially regarding scalability. When simulation is combined with machine learning and optimization, scalability is even more challenging; therefore, the scaling of solutions for dealing with simulation of large and complex architectures is still an open problem. Strategies such as parallel processing and distributed simulations combined with huge computational processing power should be investigated for the handling of such huge dimensions of real-world forthcoming complex systems.

- **Tools are still limited to support simulation based on optimization and machine learning.** The supply of relevant results in a timely manner is challenging for machine learning and optimization techniques, especially for population-based algorithms that require the evaluation of several individuals separately. Simulation tools must provide not only performance and parallelism for the simulation, but also means for evaluations of different individuals of a population and enable algorithms to explore a larger search space and achieve better results. The choice of a simulation tool depends on its performing simulations in parallel. In some cases, researchers use specific domain tools that do not have that feature and search for tools that do not have a significant overhead of resources allocated for the simulator to enable the execution of multiple instances. However, computational resources are wasted for each instance of the simulator and require inter-process communication among those instances, thus adding more processing overhead and computational resources. Research must still focus on the design of simulation tools that enable multiple simulations and reduce the amount of resources used.

# REFERENCES

21839:2019(E), I. Iso/iec/ieee international standard – systems and software engineering – system of systems (sos) considerations in life cycle stages of a system. *ISO/IEC/IEEE 21839:2019(E)*, p. 1–40, 2019. Citation on page 9.

ABAEV, P.; GAIDAMAKA, Y.; SAMOUYLOV, K.; SHORGIN, S. Design and software architecture of sip server for overload control simulation. In: **27th Conference on Modelling and Simulation**. [S.l.: s.n.], 2013. p. 580–586. ISBN 9780956494467. Citations on pages 7 and 27.

ABDELKHALIK, O.; DARANI, S. Hidden genes genetic algorithms for systems architecture optimization. In: **Genetic and Evolutionary Computation Conference (GECCO)**. [S.l.: s.n.], 2016. p. 629–636. ISBN 9781450342063. Citation on page 27.

ABOUL-SEOUD, M. H.; USHER, J. S. Effect of modular system structures on component reliability estimation. In: **5th Industrial Engineering Research Conference**. [S.l.: s.n.], 1996. p. 464–467. Citation on page 27.

Abram, G.; Adhinarayanan, V.; Feng, W.; Rogers, D.; Ahrens, J. Eth: An architecture for exploring the design space of in-situ scientific visualization. In: **2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.: s.n.], 2020. p. 515–526. ISSN 1530-2075. Citation on page 27.

ACHESON, P.; DAGLI, C.; KILICAY-ERGIN, N. Model based systems engineering for system of systems using agent-based modeling. **Procedia Computer Science**, v. 16, p. 11–19, 2013. ISSN 1877-0509. 2013 Conference on Systems Engineering Research. Available: <https://www.sciencedirect.com/science/article/pii/S1877050913000033>. Citation on page 8.

ALPAYDIN, E. **Introduction to Machine Learning, third edition**. [S.l.]: MIT Press, 2014. (Adaptive Computation and Machine Learning series). ISBN 9780262325752. Citations on pages 16, 17, 18, and 19.

ANDREWS, Z.; PAYNE, R. J.; ROMANOVSKY, A.; DIDIER, A.; MOTA, A. Model-based development of fault tolerant systems of systems. **2013 IEEE International Systems Conference (SysCon)**, p. 356–363, 2013. Citation on page 8.

ARTIBA, A.; AGHEZZAF, E. H. An architecture of a multi-model system for planning and scheduling. **International Journal of Computer Integrated Manufacturing**, Taylor Francis, v. 10, n. 5, p. 380–393, 1997. Citation on page 26.

Badr, M.; Delconte, C.; Edo, I.; Jagtap, R.; Andreozzi, M.; Jerger, N. E. Mocktails: Capturing the memory behaviour of proprietary mobile architectures. In: **47th Annual International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2020. p. 460–472. Citations on pages 3 and 28.

BALASUBRAMANIAN, K.; SCHMIDT, D. C.; MOLNÁR, Z.; LÉDECZI, Á. System integration using model-driven engineering. In: ____. Hershey, PA, USA: IGI Global, 2009. (Designing Software-Intensive Systems: Methods and Principles), p. 474–504. Citation on page 2.

BALICKI, J.; KITOWSKI, Z. Tabu search versus evolutionary search for software structure optimisation. **WIT Transactions on The Built Environment**, WIT Press, v. 46, 2000. Citation on page 29.

Barbi, E.; Cantone, G.; Falessi, D.; Morciano, F.; Rizzuto, M.; Sabbatino, V.; Scarrone, S. A model-driven approach for configuring and deploying systems of systems. In: **2012 7th International Conference on System of Systems Engineering (SoSE)**. [S.l.: s.n.], 2012. p. 214–218. Citation on page 2.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2012. ISBN 0-201-19930-0. Citation on page 13.

BATISTA, T. Challenges for SoS Architecture Description. In: **1st SESoS**. New York, NY, USA: ACM, 2013. (SESoS '13), p. 35–37. ISBN 978-1-4503-2048-1. Citation on page 2.

BAY, J. S. Recent advances in the design of distributed embedded systems. In: SURESH, R.; ROPER, W. E.; ROPER, W. E. (Ed.). **Battlespace Digitization and Network-Centric Warfare II**. [S.l.]: SPIE, 2002. v. 4741, p. 36 – 45. Citation on page 2.

BÉHÉ, R.; BILLS, R.; BRACHAT, P.; DEDEBAN, C.; VILLES, J.-L. D. Simulation and optimization software for axisymmetrical radiating structures. **Annales Des Télécommunications**, v. 49, n. 9, p. 575–588, Sep 1994. ISSN 1958-9395. Citation on page 28.

BOARDMAN, J.; SAUSER, B. System of systems - the meaning of of. In: **SOSE**. Los Angeles, California, USA: [s.n.], 2006. p. 6 pp.–. Citation on page 9.

BOEHM, B. A view of 20th and 21st century software engineering. In: **28th International Conference on Software Engineering**. New York, NY, USA: ACM, 2006. (ICSE '06), p. 12–29. ISBN 1-59593-375-1. Citation on page 43.

BOGADO, V.; GONNET, S.; LEONE, H. DEVS-based methodological framework for multi-quality attribute evaluation using software architectures. In: **2017 XLIII Latin American Computer Conference (CLEI)**. Cordoba, Argentina: IEEE, Piscataway, NJ, USA, 2017. p. 1–10. Citation on page 12.

BRACCO, S.; DELFINO, F.; PAMPARARO, F.; ROBBA, M.; ROSSI, M. A pilot facility for analysis and simulation of smart microgrids feeding smart buildings. **Renewable and Sustainable Energy Reviews**, v. 58, p. 1247–1255, 2016. Citation on page 2.

BRYANS, J.; PAYNE, R.; HOLT, J.; PERRY, S. Semi-formal and formal interface specification for system of systems architecture. In: **2013 IEEE International Systems Conference (SysCon)**. Orlando, FL, USA: [s.n.], 2013. (SysCon 2013), p. 612–619. Citations on pages 2, 8, and 10.

CALINESCU, R.; KWIATKOWSKA, M. Software engineering techniques for the development of systems of systems. In: CHOPPY, C.; SOKOLSKY, O. (Ed.). **Foundations of Computer Software. Future Trends and Techniques for Development: 15th Monterey Workshop 2008, Budapest, Hungary, September 24-26, 2008, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 59–82. Citation on page 2.

CAVALCANTE, E.; BATISTA, T. V.; OQUENDO, F. Supporting dynamic software architectures: From architectural description to implementation. In: **IEEE/IFIP Conference on Software Architecture (WICSA) 2015**. Montreal, Canada: IEEE, 2015. p. 31–40. Available: <http://dx.doi.org/10.1109/WICSA.2015.21>. Citation on page 13.

CAVALCANTE, E.; OQUENDO, F.; BATISTA, T. V. Architecture-based code generation: from $\pi$-ADL architecture descriptions to implementations in the Go language. In: **8th European Conference on Software Architecture, ECSA 2014**. Vienna, Austria: Springer, 2014. p. 130–145. Citation on page 12.

CAVALCANTE, E.; QUILBEUF, J.; TRAONOUEZ, L.; OQUENDO, F.; BATISTA, T.; LEGAY, A. Statistical model checking of dynamic software architectures. In: **10th European Conference on Software Architecture (ECSA)**. [S.l.: s.n.], 2016. p. 185–200. Citations on pages 12 and 34.

CHANG, N.; ZHANG, Z.; LU, Q.; GUO, J.; YAO, J.; NAN, G.; TAO, H.; LIU, J. A novel application architecture design for smart grid dispatching and control systems. **Dianli Xitong Zidonghua/Automation of Electric Power Systems**, v. 39, p. 53–59, 01 2015. Citations on pages 3 and 26.

CHEN, Y.; YU, J.; SHAO, H. Efficient algorithm in model identification of multivariable and non-linear systems - gmdh plus optimization of model structure algorithm. **Journal of East China Institute of Chemical Technology**, v. 14, 1988. Citation on page 27.

CHENG, L.; XU, W.; GONG, F.; LIN, Y.; WONG, H.-Y.; HE, L. Statistical timing and power optimization of architecture and device for fpgas. **ACM Trans. Reconfigurable Technol. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 5, n. 2, p. 1–19, Jun. 2012. ISSN 1936-7406. Citation on page 29.

CHI, S.; LEE, J. Model-based architecture for evolutionary intelligent systems. **Transactions of the Society for Computer Simulation**, v. 18, p. 2–8, 03 2001. Citation on page 28.

Choi, S.; Kim, J.; Kim, S.; Chung, H.; Kim, I. Architecture and performance of the base station prototype for mn systems. In: **2020 14th European Conference on Antennas and Propagation (EuCAP)**. [S.l.: s.n.], 2020. p. 1–5. Citation on page 26.

DAHMANN, J.; BALDWIN, K. J.; JR., G. R. **Systems of Systems and Net-Centric Enterprise Systems**. Loughborough University, 2009. Citation on page 10.

DAHMANN, J. S.; JR., G. R.; LANE, J. A. Systems engineering for capabilities. **CrossTalk Journal - The Journal of Defense Software Engineering**, v. 21, n. 11, p. 4–9, November 2008. Citation on page 10.

DAMERI, R. P.; ROSENTHAL-SABROUX, C. **Smart City: How to Create Public and Economic Value with High Technology in Urban Space**. [S.l.]: Springer, 2014. Citation on page 1.

DEFENSE), U. D. of. **System engineering guide for system-of-systems engineering**. [S.l.], 2008. Citation on page 9.

DELICATO, F. C.; PIRES, P. F.; BATISTA, T.; CAVALCANTE, E.; COSTA, B.; BARROS, T. Towards an iot ecosystem. In: **Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems**. New York, NY, USA: Association for Computing Machinery, 2013. (SESoS '13), p. 25–28. ISBN 9781450320481. Available: <https://doi.org/10.1145/2489850.2489855>. Citation on page 1.

DELSING, J.; ELIASSON, J.; GUSTAFSSON, J.; KYUSAKOV, R.; MCLOAD, A. K. S.; HARRISON, R.; COLOMBO, A.; MENDES, J. M. Industrial cloud-based cyber-physical systems: The imc-aesop approach. In: ____. Armando w. colombo. [S.l.]: Springer, 2014. chap.

Building System of Systems with SOA Technology : A Smart House Use Case, p. 219–230. Citation on page 2.

Dhieb, N.; Ghazzai, H.; Besbes, H.; Massoud, Y. Scalable and secure architecture for distributed iot systems. In: **IEEE Technology Engineering Management Conference (TEMSCON)**. [S.l.: s.n.], 2020. p. 1–6. Citation on page 28.

DIDIC, M.; WOLFINGER, B. Simulation of a local computer network architecture applying a unified modeling system. **Computer Networks (1976)**, v. 6, n. 2, p. 75 – 91, 1982. ISSN 0376-5075. Citation on page 29.

DYBA, T.; KITCHENHAM, B. A.; JORGENSEN, M. Evidence-based software engineering for practitioners. **IEEE Software**, IEEE Computer Society Press, Washington, DC, USA, v. 22, n. 1, p. 58–65, Jan. 2005. ISSN 0740-7459. Citations on pages 24 and 40.

EICHLER, U.; KÖLMEL, C.; SAUER, J. Combining ab initio techniques with analytical potential functions for structure predictions of large systems: Method and application to crystalline silica polymorphs. **Journal of Computational Chemistry**, v. 18, n. 4, p. 463–477, 1997. Citations on pages 7 and 27.

El-Far, G. Z.; Nassef, M. D.; Montasser, A. A. Online multilevel controller structure for a class of dynamical systems. In: **Thirteenth National Radio Science Conference. NRSC '96**. [S.l.: s.n.], 1996. p. 371–383. Citation on page 28.

ETEMAADI, R.; CHAUDRON, M. R. Distributed optimization on super computers: Case study on software architecture optimization framework. In: **Annual Conference on Genetic and Evolutionary Computation - Companion (GECCO Comp)**. [S.l.: s.n.], 2014. p. 1125–1132. ISBN 9781450328814. Citation on page 27.

FARCAS, C.; FARCAS, E.; KRUEGER, I. H.; MENARINI, M. Addressing the integration challenge for avionics and automotive systems—from components to rich services. **Proceedings of the IEEE**, v. 98, n. 4, p. 562–583, 2010. Citation on page 2.

FIALHO, F.; SANTOS, N. dos. General architecture for simulating complex systems able of auto-organization. v. 4, p. 57–62, 01 1994. Citation on page 27.

FINN, J.; NUZZO, P.; SANGIOVANNI-VINCENTELLI, A. A mixed discrete-continuous optimization scheme for cyber-physical system architecture exploration. In: **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2015. p. 216–223. ISBN 9781467383899. Citations on pages 3 and 26.

FITZGERALD, J.; BRYANS, J.; PAYNE, R. A formal model-based approach to engineering systems-of-systems. In: CAMARINHA-MATOS, L. M.; XU, L.; AFSARMANESH, H. (Ed.). **Collaborative Networks in the Internet of Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 53–62. ISBN 978-3-642-32775-9. Citation on page 8.

FITZGERALD, J.; FOSTER, S.; INGRAM, C.; LARSEN, P. G.; WOODCOCK, J. **Model-based Engineering for Systems of Systems: the COMPASS Manifesto**. [S.l.], 2013. Available: <http://www.compass-research.eu/Project/Publications/MBESoS.pdf>. Citation on page 12.

FRANÇA, B. B. N. de; TRAVASSOS, G. H. Simulation based studies in software engineering: A matter of validity. **CLEI Electron. J.**, v. 18, n. 1, 2015. Available: <http://www.clei.org/cleiej/paper.php?id=329>. Citation on page 3.

____. Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. **Empirical Software Engineering**, v. 21, n. 3, p. 1302–1345, 2016. ISSN 1573-7616. Citations on pages 7, 11, and 12.

France, R.; Rumpe, B. Model-driven development of complex software: A research roadmap. In: **Future of Software Engineering (FOSE '07)**. [S.l.: s.n.], 2007. p. 37–54. Citation on page 2.

FRANçA, B. B. N. de; TRAVASSOS, G. H. Simulation based studies in software engineering: A matter of validity. **CLEI Electronic Journal**, v. 18, n. 1, p. 1–18, April 2015. Citation on page 7.

FREITAS, F.; FREITAS, C. D.; SOUZA, A. System architecture for on-line optimization of automated trading strategies. In: **6th Workshop on High Performance Computational Finance (WHPCF)**. [S.l.: s.n.], 2013. p. 1–8. ISBN 9781450325073. Citation on page 29.

FRICKE, A.; SCHöNEBERGER, J. Know-how protection and software architectures in industry 4.0. **Computer Aided Chemical Engineering**, p. 2287–2292, 01 2017. Citation on page 28.

GASSARA, A.; RODRIGUEZ, I. B.; JMAIEL, M.; DRIRA, K. A Bigraphical Multi-scale Modeling Methodology for System of Systems. **Computers and Electrical Engineering**, Elsevier, v. 58, n. 1, p. 113–125, 2017. Citations on pages 7 and 49.

GOKHALE, A.; BALASUBRAMANIAN, K.; KRISHNA, A. S.; BALASUBRAMANIAN, J.; EDWARDS, G.; DENG, G.; TURKAY, E.; PARSONS, J.; SCHMIDT, D. C. Model driven middleware: A new paradigm for developing distributed real-time and embedded systems. **Science of Computer Programming**, v. 73, n. 1, p. 39–58, 2008. ISSN 0167-6423. Special Issue on Foundations and Applications of Model Driven Architecture (MDA). Available: <https://www.sciencedirect.com/science/article/pii/S0167642308000518>. Citation on page 2.

Gokhale, S. S.; Michael Rung-Tsong Lyu. A simulation approach to structure-based software reliability analysis. **IEEE Transactions on Software Engineering**, v. 31, n. 8, p. 643–656, 2005. Citation on page 26.

Graciano Neto, V. V.; GARCES, L.; GUESSI, M.; PAES, C. E. B.; MANZANO, W.; OQUENDO, F.; NAKAGAWA, E. ASAS: An Approach to Support Simulation of Smart Systems. In: **Hawaii International Conference on System Sciences**. Hawaii, USA: [s.n.], 2018. p. 1–10. Citation on page 62.

Graciano Neto, V. V.; OQUENDO, F.; NAKAGAWA, E. Y. Systems-of-systems: Challenges for information systems research in the next 10 years. In: **Big Research Challenges in Information Systems in Brazil (2016-2026) at Brazilian Symposium on Information Systems**. Florianópolis, Brazil: SBC, 2016. (GRANDSI-BR/SBSI), p. 1–3. Citation on page 2.

____. Smart systems-of-information systems: Foundations and an assessment model for research development. In: BOSCARIOLI, C.; ARAUJO, R.; MACIEL, R. S. P. (Ed.). **GranDSI-BR: Big Research Challenges in Information Systems in Brazil (2016-2026)**. Brazil: Brazilian Computer Society, 2017. p. 13–24. Citation on page 9.

GUARISO, G.; BARACANI, M. Supporting environmental and energy decisions through an open software structure. In: **3rd International Congress on Environmental Modelling and Software (iEMSs)**. [S.l.: s.n.], 2006. p. 1–6. Citation on page 29.

GUESSI, M.; Graciano Neto, V. V.; BIANCHI, T.; FELIZARDO, K. R.; OQUENDO, F.; NAKAGAWA, E. Y. A systematic literature review on the description of software architectures for systems of systems. In: **ACM Symposium on Applied Computing (SAC)**. Salamanca, Spain: ACM, 2015. p. 1433–1440. Citations on pages 12 and 13.

Gupta, U.; Hsia, S.; Saraph, V.; Wang, X.; Reagen, B.; Wei, G.; Lee, H. S.; Brooks, D.; Wu, C. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In: **47th Annual International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2020. p. 982–995. Citation on page 27.

HADDAD, O. B.; AFSHAR, A.; MARIÑO, M. A. Honey-bees mating optimization (hbmo) algorithm: A new heuristic approach for water resources optimization. **Water Resources Management**, v. 20, n. 5, p. 661–680, Oct 2006. ISSN 1573-1650. Available: <https://doi.org/10.1007/s11269-005-9001-3>. Citation on page 16.

HARDNETT, C. R.; PALEM, K. V.; CHOBE, Y. Compiler optimization of embedded applications for an adaptive soc architecture. In: **International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)**. [S.l.: s.n.], 2006. p. 312–322. ISBN 1595935436. Citation on page 27.

HELLESTRAND, G. Engineering safe autonomous mobile systems of systems using specification (model) based systems architecture & engineering. In: **IEEE International Systems Conference (SysCon)**. [S.l.: s.n.], 2013. p. 599–605. Citations on pages 7 and 13.

Horeis, T. F.; Kain, T.; Müller, J.; Plinke, F.; Heinrich, J.; Wesche, M.; Decke, H. A reliability engineering based approach to model complex and dynamic autonomous systems. In: **International Conference on Connected and Autonomous Driving (MetroCAD)**. [S.l.: s.n.], 2020. p. 76–84. Citations on pages 7 and 26.

HORITA, F. E.; ALBUQUERQUE, J. P. de; DEGROSSI, L. C.; MENDIONDO, E. M.; UEYAMA, J. Development of a spatial decision support system for flood risk management in Brazil that combines volunteered geographic information with wireless sensor networks. **Computers & Geosciences**, v. 80, n. 1, p. 84 – 94, 2015. Citation on page 1.

Houmani, Z.; Balouek-Thomert, D.; Caron, E.; Parashar, M. Enhancing microservices architectures using data-driven service discovery and qos guarantees. In: **2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)**. [S.l.: s.n.], 2020. p. 290–299. Citation on page 27.

HOWELL, S.; KARANGELEN, N. E. High level architecture for an advanced distributed complex systems engineering environment. In: **IEEE International Conference on the Engineering of Computer-Based Systems (ECBS)**. [S.l.: s.n.], 1997. p. 296. Citation on page 28.

HUANG, X.; LIU, G.; GUO, W.; NIU, Y.; CHEN, G. Obstacle-avoiding algorithm in x-architecture based on discrete particle swarm optimization for vlsi design. **ACM Trans. Des. Autom. Electron. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 20, n. 2, p. 1–28, Mar. 2015. ISSN 1084-4309. Citation on page 28.

INCOSE. **The Guide to the Systems Engineering Body of Knowledge (SEBoK)**. 2016. Citation on page 9.

ISO/IEC/IEEE 42010. ISO/IEC/IEEE Systems and software engineering – Architecture description. **ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)**, v. 1, p. 1–46, Dec 2011. Citation on page 13.

JAMSHIDI, M. **System of Systems Engineering - Innovations for 21st century**. 1st. ed. Hoboken, New Jersey, United States: Wiley, 2009. (Wiley Series in Systems Engineering and Management, v. 1). ISBN 0470195908,9780470195901. Citation on page 43.

JENG, M.; LIN, C. Scheduling flexible manufacturing systems containing assembly operations based on petri net structures and dynamics. In: **IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation**. [S.l.: s.n.], 1997. p. 4430–4435. Citation on page 28.

JIA, W.; SHAW, K.; MARTONOSI, M. Starchart: Hardware and software optimization using recursive partitioning regression trees. In: **22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)**. [S.l.: s.n.], 2013. p. 257–268. ISBN 9781479910212. Citation on page 29.

JONG, K. A. de. **Evolutionary computation**. Cambridge, MA: Bradford Books, 2006. (Evolutionary Computation). Citation on page 16.

JR, S. B.; ALBERT, C.; GARCIA-MILLER, S. **Beyond technology readiness levels for software: US Army workshop report**. [S.l.], 2010. Citation on page 37.

KAHOUL, A.; SMITH, A. M.; CONSTANTINIDES, G. A.; CHEUNG, P. Y. K. Efficient heterogeneous architecture floorplan optimization using analytical methods. **ACM Trans. Reconfigurable Technol. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 4, n. 1, p. 1–23, Dec. 2010. ISSN 1936-7406. Citation on page 27.

KAJIWARA, I.; NAGAMATSU, A. Simultaneous optimum design of structure and control systems by sensitivity analysis. **Finite Elements in Analysis and Design**, v. 14, n. 2, p. 187 – 195, 1993. ISSN 0168-874X. Citation on page 29.

Kawakami, T.; Kahazu, Y. A study on evolutionary synthesis of classifier system architectures. In: **IEEE International Conference on Evolutionary Computation**. [S.l.: s.n.], 1996. p. 155–160. Citations on pages 7 and 29.

KAZMAN, R.; SCHMID, K.; NIELSEN, C.; KLEIN, J. Understanding patterns for system of systems integration. In: . [S.l.: s.n.], 2013. p. 141–146. ISBN 978-1-4673-5596-4. Citation on page 2.

KIM, Y.; LEE, J.; SHRIVASTAVA, A.; PAEK, Y. Memory access optimization in compilation for coarse-grained reconfigurable architectures. **ACM Trans. Des. Autom. Electron. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 16, n. 4, p. 1–27, Oct. 2011. ISSN 1084-4309. Citation on page 28.

KITCHENHAM, B.; BUDGEN, D.; BRERETON, P. **Evidence-based software engineering and systematic reviews**. [S.l.]: CRC Press, 2015. Citations on pages 23, 24, and 40.

KLEIN, J.; VLIET, H. van. A systematic review of system-of-systems architecture research. In: **9th International ACM Sigsoft Conference on Quality of Software Architectures**. New York, NY, USA: ACM, 2013. (QoSA '13), p. 13–22. Citation on page 12.

KOZIOLEK, A.; KOZIOLEK, H.; REUSSNER, R. Peropteryx: automated application of tactics in multi-objective software architecture optimization. In: **7th International Conference on the Quality of Software Architectures (QoSA) and 2nd International Symposium on Architecting Critical Systems (ISARCS)**. [S.l.: s.n.], 2011. p. 33–42. Citation on page 28.

LAAR, P. van de; TRETMANS, J. **Introduction: Situation Awareness, Systems of Systems, and Maritime Safety and Security**. [S.l.]: Springer, 2013. 3-20 p. Citation on page 1.

LANE, J. A. **What is a System of Systems and Why Should I Care?** [S.l.], 2013. Citations on pages 10 and 11.

Lee, C.; WANG, J.-S. Self-adaptive neuro-fuzzy systems: structure and learning. In: **IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.: s.n.], 2000. p. 52–57. Citation on page 28.

LOU, Q.; PAN, C.; MCGUINNESS, J.; HORVATH, A.; NAEEMI, A.; NIEMIER, M.; HU, X. S. A mixed signal architecture for convolutional neural networks. **J. Emerg. Technol. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 15, n. 2, p. 1–26, Mar. 2019. ISSN 1550-4832. Citation on page 26.

LUENBERGER, D. G.; YE, Y. **Linear and nonlinear programming**. 3. ed. New York, NY: Springer, 2008. (International series in operations research & management science). Citation on page 14.

MAIER, M. W. Architecting principles for systems-of-systems. **Systems Engineering**, v. 1, n. 4, p. 267–284, 1998. Citations on pages 1, 3, 9, 10, 11, and 51.

MAIER, M. W.; RECHTIN, E. **The Art of Systems Architecting (2nd Ed.)**. USA: CRC Press, Inc., 2010. ISBN 0849304407. Citation on page 3.

Mandyam, G. Application of genetic algorithms to volterra filter structure determination for adaptive systems. In: **Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers**. [S.l.: s.n.], 1996. p. 1147–1151. Citation on page 26.

MANKINS, J. C. Technology Readiness Assessments: A Retrospective. **Acta Astronautica**, v. 65, p. 1216–1223, 2009. Citation on page 37.

MANZANO, W.; NETO, V. V. G.; NAKAGAWA, E. Y. Dynamic-sos: An approach for the simulation of systems-of-systems dynamic architectures. **The Computer Journal**, v. 63, n. 5, p. 709–731, 04 2019. ISSN 0010-4620. Citations on pages 4, 21, 34, and 52.

MEEDENIYA, I.; ALETI, A.; AVAZPOUR, I.; AMIN, A. Robust archeopterix: Architecture optimization of embedded systems under uncertainty. In: **2nd International Workshop on Software Engineering for Embedded Systems (SEES)**. [S.l.: s.n.], 2012. p. 23–29. ISBN 9781467318532. Citation on page 28.

M.F., A.; D.A., L.; A., B.; N., C. A blackboard software architecture for integrated intelligent control systems. **Kybernetes**, MCB UP Ltd, v. 29, n. 7/8, p. 999–1015, Jan 2000. ISSN 0368-492X. Citation on page 25.

MICHAEL, J. B.; DRUSINSKY, D.; OTANI, T. W.; SHING, M.-T. Verification and validation for trustworthy software systems. **IEEE Software**, IEEE Computer Society, Los Alamitos, CA, USA, v. 28, n. 6, p. 86–92, 2011. ISSN 0740-7459. Citation on page 12.

MICHAEL, J. B.; RIEHLE, R.; SHING, M. T. The verification and validation of software architecture for systems of systems. In: **International Conference on Systems-of-Systems Engineering (SoSE)**. Albuquerque, USA: IEEE, 2009. p. 1–6. Citation on page 12.

MITCHELL, M. **An introduction to genetic algorithms**. Cambridge, MA: Bradford Books, 1998. (Complex Adaptive Systems). Citation on page 15.

MITTAL, S.; RAINEY, L. Harnessing Emergence: The Control and Design of Emergent Behavior in System of Systems Engineering. In: **Conference on Summer Computer Simulation**. Chicago, Illinois, USA: Society for Computer Simulation International, 2015. p. 1–10. ISBN 978-1-5108-1059-4. Citation on page 12.

MOHAMMADI, M.; AL-FUQAHA, A.; GUIZANI, M.; OH, J.-S. Semisupervised deep reinforcement learning in support of iot and smart city services. **IEEE Internet of Things Journal**, v. 5, n. 2, p. 624–635, 2018. Citation on page 7.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of Machine Learning**. [S.l.]: The MIT Press, 2012. ISBN 026201825X. Citations on pages 18 and 19.

MORGAN, J.; HALTON, M.; QIAO, Y.; BRESLIN, J. G. Industry 4.0 smart reconfigurable manufacturing machines. **Journal of Manufacturing Systems**, v. 59, n. 1, p. 481–506, 2021. ISSN 0278-6125. Citation on page 7.

MORITZ, R. F. A.; SOUTHWICK, E. E. **Bees as superorganisms: An evolutionary reality**. Berlink, Germany: Springer-Verlag, 1992. Citation on page 16.

MORTAJI, H.; OW, S.; MOGHAVVEMI, M.; ALMURIB, H. Load shedding and smart-direct load control using internet of things in smart grid demand response management. **IEEE Transactions on Industry Applications**, v. 53, n. 6, p. 5155–5163, 2017. Citation on page 7.

NAKAGAWA, E. Y.; GONcALVES, M.; GUESSI, M.; OLIVEIRA, L. B. R.; OQUENDO, F. The state of the art and future perspectives in systems of systems software architectures. In: **International Workshop on Software Engineering for Systems of Systems**. Montpellier, France: ACM, 2013. (SESoS '13), p. 13–20. ISBN 978-1-4503-2048-1. Citation on page 1.

NAKRANI, S.; TOVEY, C. On honey bees and dynamic server allocation in internet hosting centers. **Adaptive Behavior - ADAPT BEHAV**, v. 12, p. 223–240, 12 2004. Citation on page 16.

NAN, S.; LIAO, X.; JIANG, N. Research of simulation software architecture optimization used with components based on reach-ability matrix. In: **Frontiers of Manufacturing and Design Science II**. [S.l.: s.n.], 2012. (Applied Mechanics and Materials), p. 2174–2178. Citation on page 28.

NAZARI, S.; WENZEL, S.; Samuel Maxeiner, L.; SONNTAG, C.; ENGELL, S. A framework for the simulation and validation of distributed control architectures for technical systems of systems. **IFAC-PapersOnLine**, v. 50, n. 1, p. 12458 – 12463, 2017. ISSN 2405-8963. Citation on page 25.

NETO, V. V. G.; GUESSI, M.; OLIVEIRA, L. B. R.; OQUENDO, F.; NAKAGAWA, E. Y. Investigating the model-driven development for systems-of-systems. In: **2014 European Conference on Software Architecture Workshops**. Vienna, Austria: ACM, 2014. (ECSAW '14), p. 22:1–22:8. ISBN 978-1-4503-2778-7. Citation on page 12.

NETO, V. V. G.; PAES, C. E. B.; GARCÉS, L.; GUESSI, M.; MANZANO, W.; OQUENDO, F.; NAKAGAWA, E. Y. Stimuli-SoS: a model-based approach to derive stimuli generators for simulations of systems-of-systems software architectures. **Journal of the Brazilian Computer Society**, v. 23, n. 1, p. 1–22, Oct 2017. ISSN 1678-4804. Citation on page 12.

NGUYEN, A.-T.; REITER, S.; RIGO, P. A review on simulation-based optimization methods applied to building performance analysis. **Applied Energy**, v. 113, n. 1, p. 1043–1058, 2014. ISSN 0306-2619. Citations on pages 7, 20, and 31.

NIELSEN, C. B.; LARSEN, P. G.; FITZGERALD, J.; WOODCOCK, J.; PELESKA, J. **Model-based engineering of systems of systems**. [S.l.], 2013. Citation on page 13.

____. Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 48, n. 2, p. 18:1–18:41, Sep. 2015. ISSN 0360-0300. Citation on page 12.

NOORSHAMS, Q.; MARTENS, A.; REUSSNER, R. Using quality of service bounds for effective multi-objective software architecture optimization. In: **2nd International Workshop on the Quality of Service-Oriented Software Systems (QUASOSS)**. [S.l.: s.n.], 2010. p. 1–6. ISBN 9781450302395. Citation on page 29.

OQUENDO, F. Formally Describing the Software Architecture of Systems-of-Systems with SosADL. In: **11th IEEE System of Systems Engineering Conference (SoSE)**. Kongsberg, Norway: IEEE, 2016. p. 1–6. Citation on page 51.

____. Software architecture of self-organizing systems-of-systems for the internet-of-things with SosADL. In: **12th International Conference on System of Systems Engineering (SoSE)**. Waikoloa, HI, USA: IEEE, 2017. p. 1–6. Citation on page 9.

OTWAGIN, A.; SADYKHOV, R.; BOGOMAZOV, A. The architecture of system for analysis and optimization of parallel programs. p. 204–207, 01 2004. Citation on page 29.

PEFFERS, K.; TUUNANEN, T.; ROTHENBERGER, M. A.; CHATTERJEE, S. A design science research methodology for information systems research. **Journal of Management Information Systems**, Routledge, v. 24, n. 3, p. 45–77, 2007. Citations on pages 15, 4, and 6.

PÉREZ, J.; DÍAZ, J.; GARBAJOSA, J.; YAGüE, A.; GONZALEZ, E.; LOPEZ-PEREA, M. Large-scale smart grids as system of systems. In: **First International Workshop on Software Engineering for Systems-of-Systems**. New York, NY, USA: ACM, 2013. p. 38–42. Citations on pages 2, 10, and 13.

RACU, R.; HAMANN, A.; ERNST, R.; MOCHOCKI, B.; HU, X. S. Methods for power optimization in distributed embedded systems with real-time requirements. In: **International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)**. [S.l.: s.n.], 2006. p. 379–388. ISBN 1595935436. Citation on page 28.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. USA: Prentice Hall Press, 2009. ISBN 0136042597. Citations on pages 7, 17, and 19.

SANTOS, C.; COELHO, L.; SAMPAIO, R.; JACOBI, R.; AYALA, H.; LLANOS, C. A svm optimization tool and fpga system architecture applied to nmpc. In: **30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands (SBCCI)**. [S.l.: s.n.], 2017. p. 96–102. ISBN 9781450351065. Citation on page 26.

SAUSER, B.; BOARDMAN, J.; VERMA, D. Systomics: Toward a Biology of System of Systems. **IEEE Trans. on Systems, Man, and Cybernetics**, v. 40, n. 4, p. 803–814, 2010. ISSN 1083-4427. Citation on page 12.

S.FRITZ; MCCALLUMX, S.; WILLIAMS, M.; KRAXNER, F.; OBERSTEINER, M. Encyclopedia of remote sensing. In: . New York: Springer-Verlag, 2014. chap. Global Earth Observation System of Systems (GEOSS). Citation on page 1.

SMITH, J. A.; HARIKUMAR, J.; RUTH, B. G. An Army-Centric System of Systems Analysis (SoSA) definition. 2011. Citation on page 1.

SUN, Y.; YANG, X.; XIAO, H.; FENG, H. An intelligent driving simulation platform: architecture, implementation and application. In: **2020 International Conference on Connected and Autonomous Driving (MetroCAD)**. [S.l.: s.n.], 2020. p. 71–75. Citation on page 26.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. [S.l.]: MIT Press, 1998. Citation on page 7.

TAMPOURATZIS, N.; PAPAEFSTATHIOU, I.; NIKITAKIS, A.; BROKALAKIS, A.; ANDRIANAKIS, S.; DOLLAS, A.; MARCON, M.; PLEBANI, E. A novel, highly integrated simulator for parallel and distributed systems. **ACM Transactions on Architecture and Code Optimization**, v. 17, n. 1, p. 1–28, Mar. 2020. ISSN 1544-3566. Citations on pages 3 and 26.

TELLO-OQUENDO, L.; LIN, S.-C.; AKYILDIZ, I. F.; PLA, V. Software-defined architecture for qos-aware iot deployments in 5g systems. **Ad Hoc Networks**, v. 93, p. 101911, 2019. ISSN 1570-8705. Citation on page 29.

Tsang, C. . Design of communications simulation expert system architecture with common data structure. In: **IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)**. [S.l.: s.n.], 1989. p. 249–252. Citations on pages 3 and 27.

Tsang, C. A design of communications simulation expert system architecture with common data structure. In: **IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)**. [S.l.: s.n.], 1989. p. 249–252. Citation on page 26.

Turcato, A.; Dias, A.; Sestito, G.; Flauzino, R.; Brandão, D.; Sisinni, E.; Ferrari, P. Introducing a cloud based architecture for the distributed analysis of real-time ethernet traffic. In: **2020 IEEE International Workshop on Metrology for Industry 4.0 IoT**. [S.l.: s.n.], 2020. p. 235–240. Citation on page 28.

UEDA, K.; SAKANUSHI, K.; TAKEUCHI, Y.; IMAI, M. Architecture-level performance estimation method based on system-level profiling. **Computers and Digital Techniques**, v. 152, n. 1, p. 12–19, 02 2005. Citations on pages 7 and 26.

VORONTSOV, M. A. Adaptive wavefront correction with self-organized control system architecture. In: GONGLEWSKI, J. D.; VORONTSOV, M. A. (Ed.). **Artificial Turbulence for Imaging and Wave Propagation**. [S.l.]: SPIE, 1998. v. 3432, p. 68 – 72. Citation on page 26.

WACHHOLDER, D.; STARY, C. Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In: **2011 6th International Conference on Systems of Systems Engineering (SoSE)**. [S.l.: s.n.], 2015. p. 334–339. Citation on page 12.

WANG, H.; BIAN, J.; NIU, Y.; TONG, K.; WANG, Y. Ca-ex: A tuning-incremental methodology for communication architectures in embedded systems. In: WU, Z.; CHEN, C.; GUO, M.; BU, J. (Ed.). **Embedded Software and Systems**. [S.l.]: Springer Berlin Heidelberg, 2005. p. 74–80. Citation on page 27.

WETTER, M. **GenOpt Generic Optimization Program, User Manual, Version 3.1.0**. Berkeley: Simulation Research Group, Lawrence Berkeley National Laboratory, 2011. Citation on page 20.

White, A. M.; Hallam, P.; Binns, R. J. Automated generation of system-level ahdl architectures using a genetic algorithm. In: **IEE Colloquium on Mixed-Signal AHDL/VHDL Modelling and Synthesis**. [S.l.: s.n.], 1997. p. 5/1–5/7. Citation on page 26.

WICHMANN, A.; MASCHOTTA, R.; BEDINI, F.; ZIMMERMANN, A. A workflow for the design of optimized system architectures using model-driven optimization. In: **Model-Driven Approaches for Simulation Engineering Symposium (Mod4Sim)**. [S.l.: s.n.], 2018. p. 1–12. ISBN 9781510860186. Citation on page 26.

WOLFF, W.; PORTER, B. Performance optimization on big.little architectures: A memory-latency aware approach. In: **21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)**. [S.l.: s.n.], 2020. p. 51–61. ISBN 9781450370943. Citation on page 28.

WOODCOCK, J.; CAVALCANTI, A.; FITZGERALD, J.; LARSEN, P.; MIYAZAWA, A.; PERRY, S. Features of cml: A formal modelling language for systems of systems. In: **7th International Conference on System of Systems Engineering (SoSE)**. Genova, Italy: IEEE, 2012. p. 1–6. Citation on page 12.

XIA, X.; WU, J.; LIU, C.; XU, L. A model-driven approach for evaluating system of systems. In: **International Conference on Engineering of Complex Computer Systems (ICECCS)**. [S.l.: s.n.], 2013. p. 56–64. Citation on page 12.

XU, C.; NIU, D.; ZHU, X.; KANG, S. H.; NOWAK, M.; XIE, Y. Device-architecture co-optimization of stt-ram based memory for low power embedded systems. In: **International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2011. p. 463–470. ISBN 9781457713989. Citation on page 27.

XU, Q.; ZHANG, Y.; CHAKRABARTY, K. Soc test-architecture optimization for the testing of embedded cores and signal-integrity faults on core-external interconnects. **ACM Trans. Des. Autom. Electron. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 14, n. 1, p. 1–27, Jan. 2009. ISSN 1084-4309. Citation on page 29.

ZEIGLER, B. P.; SARJOUGHIAN, H. S.; DUBOZ, R.; SOULI, J.-C. **Guide to Modeling and Simulation of Systems of Systems**. Berlim, Germany: Springer, 2012. ISBN 085729864X, 9780857298645. Citations on pages 3, 12, and 13.

ZHANG, Y.; FENG, D.; TONG, W.; HUA, Y.; LIU, J.; TAN, Z.; WANG, C.; WU, B.; LI, Z.; XU, G. Cacf: A novel circuit architecture co-optimization framework for improving performance, reliability and energy of reram-based main memory system. **ACM Trans. Archit. Code Optim.**, Association for Computing Machinery, New York, NY, USA, v. 15, n. 2, p. 1–26, May 2018. ISSN 1544-3566. Citation on page 27.


ICMC
USP
SÃO CARLOS