**Architectural decision-making on interoperability in
software-intensive systems**

Pedro Henrique Dias Valle

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

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Pedro Henrique Dias Valle

Architectural decision-making on interoperability in software-intensive systems

Thesis submitted to the Institute of Mathematics and
Computer Science – ICMC-USP, in accordance with
the requirements of the Computer and Mathematical
Sciences Graduate Program, for the degree of
Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Profa. Dra. Elisa Yumi Nakagawa

USP – São Carlos
June 2021

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

D181 Dias Valle, Pedro Henrique
a Architectural decision-making on
interoperability in software-intensive systems /
Pedro Henrique Dias Valle; orientadora Elisa Yumi
Nakagawa. -- São Carlos, 2021.
132 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2021.

1. Software Architecture. 2. Interoperability.
3. Architectural Patterns. 4. Software-Intensive
Systems. 5. Decision-Making. I. Nakagawa, Elisa
Yumi, orient. II. Título.

Pedro Henrique Dias Valle

**Tomada de decisões arquiteturais sobre interoperabilidade
em sistemas intensivos em software**

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

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Elisa Yumi Nakagawa

**USP – São Carlos
Junho de 2021**

I dedicate this work to my parents and my grandmother!

Acknowledgements

This work is the result of a long journey that started many years ago. Getting here was only possible due to the support that I have received from incredible people, who have always been by my side. I want to thank each of these people at this moment.

Firstly, I thank God for all the graces granted and for always guiding, illuminating, and blessing my history. Thank you very much for this opportunity and for putting incredible people in my way along this trajectory. I also to thank to Most Holy Virgin Mary and to my guardian angel for all protection and intercession to God.

To my mother, Luzmaia, for all her love, sacrifices, confidence, and dedication to me. Thank you so much for always understanding my absence in these long years and also to believe in my dreams and life projects. I acknowledged that you have given up many things so that I could become who I am today. You are my great example and inspiration in life.

To my father, Randolfo, for all his love, confidence, advice over these years that helped me achieve my goals. Thank you very much for being by my side.

To my grandmother, Aurora, for all her love, care, affection, and example of faith. Thank you very much for your intercession to God, offering prayers, and praying Marian rosaries during each presentation, test, and proofs that were done along years so that everything works out. Her advice and words of wisdom were essential to conduct this doctoral thesis.

To Heronides (in memoriam) who generously fulfilled the role of grandfather in my life and for a distance of three months would have been able to see this beautiful moment happen. Thank you very much for encouraging me to believe in my dreams, and also for the many daily advice and concerns. I am sure you are very happy at this moment.

To my siblings and my niece for understanding my absence during these years. I also thank other members of my family who have always supported me during this journey.

To Prof. Elisa, my advisor, for receiving me so well in her research group, I felt very well received and welcomed. Thank you so much for your supervision, dedication, patience, and for your valuable advice. You were essential in my formation.

To Lina for the partnership in recent years, and for all her advice, incentives, and discussions that have generated great results. You were very important throughout this

process.

To Vitor for helping me to understand and apply the artificial intelligence techniques used in this work. In particular, I thank you for your friendship and for several discussions that made me reflect and find solutions to my problems.

To my friends Danillo, father Everton, Francielle, Iohan, João Paulo, Luana, Pedro Alexandre, Raphael, Renan, Renato, Ricardo and Rodrigo who always listened to me and help me in the most difficult moments. Thank you for smoothing this trajectory.

To my colleagues at LabES and START for their contributions during the conduct of this doctoral and for each moment of fun. In particular, I would like to thank Ana Allian, Brauner, Bruno Sena, Diógenes, Eliane, Jacson, Jorge, João Paulo, Katia, Leo, Lucas Lagôa, Maria Lydia, and Stevão.

To all teachers and professors from childhood education until to graduate who contributed to my formation.

To the Institute of Mathematics and Computer Science and the University of São Paulo (USP).

To CNPq (Grant: 141602/2017-1) and CAPES for financial support.

*“Your word is a lamp to my feet and a light to my path”
(Psalm 119:105)*

Resumo

VALLE, P.H.D. **Tomada de decisões arquiteturais sobre interoperabilidade em sistemas intensivos em software**. 2021. 132 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Sistemas intensivos em software estão cada vez mais presentes em diversos domínios, como Indústria 4.0, militar, cidades inteligentes, agricultura, entre outros. Devido ao aumento da complexidade e do tamanho desses sistemas, o projeto arquitetural tem-se tornado uma atividade consideravelmente complexa, enquanto a interoperabilidade tem desempenhado um importante papel no sucesso desses sistemas, uma vez que eles são, muitas vezes, compostos por sistemas individuais que precisam se comunicar. Apesar disso, os arquitetos de software têm enfrentado dificuldades para analisar, compreender, e guiar como a interoperabilidade pode ser abordada em arquiteturas de software de sistemas intensivos em software. Motivado por esse cenário, o objetivo principal desta tese de doutorado é fornecer aos arquitetos meios para a tomada de decisões sobre como abordar os diferentes níveis de interoperabilidade em sistemas intensivos em software. Mais especificamente, foram estabelecidos: (i) uma tipologia de estratégias arquiteturais organizadas por tipo de estratégia e pelos quatro diferentes níveis de interoperabilidade; e (ii) um catálogo composto dos principais problemas de interoperabilidade e um conjunto de padrões arquiteturais para mitigar esses problemas. Esse catálogo é considerado validado porque ele foi definido de acordo com as informações obtidas sistematicamente na literatura científica e nos resultados do *survey* que foi conduzido com 33 profissionais. Como resultado, observou-se que este catálogo pode apoiar os arquitetos a decidir quais soluções arquiteturais poderiam ser utilizadas para alcançar interoperabilidade em seus projetos de integração.

Palavras-chave: Arquitetura de Software, Interoperabilidade, Padrão Arquitetural.

Abstract

VALLE, P.H.D. **Architectural decision-making on interoperability in software-intensive systems**. 2021. 132 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Software-intensive systems are increasingly present in several domains, such as Industry 4.0, military, smart cities, and smart agriculture. Due to the increasing complexity and size of these systems, their architectural design has become a considerably complicated task. In contrast, interoperability has played an essential role in the success of software-intensive systems since this class of systems is sometimes composed of individual systems that need to communicate. However, software architects have faced difficulties in comprehending, analyze, and select suitable interoperability architectural solutions for these systems. Motivated by this scenario, the main objective of this doctoral thesis is to provide software architects means to support the decision-making on how to address the different levels of interoperability in software-intensive systems. More specifically, we established: (i) a typology of architectural strategies organized by strategy type and four different interoperability levels; and (ii) a catalog composed of main interoperability issues and the set of architectural patterns to mitigate these issues. This catalog is considered validated because it was defined according to information obtained systematically in the scientific literature and confirmed in the survey results that we performed with 33 practitioners. As a result, we observed that this catalog could support software architects to decide better which architectural solutions could solve each interoperability issue in their integration projects.

Keywords: Software Architecture, Interoperability, Architectural Pattern.

List of Figures

Figure 1 – Architectural design activities	12
Figure 2 – Three-part schema of a pattern description	18
Figure 3 – Process for the selection of relevant primary studies	26
Figure 4 – Process to design and conduct a survey	30
Figure 5 – Continents where practitioners work	34
Figure 6 – Experience of practitioners in years.	35
Figure 7 – Experience of practitioners on software integration projects.	35
Figure 8 – Overview of the process for association rules mining	48
Figure 9 – The TASIS consolidated and updated according to information from the industry	68
Figure 10 – Process to establish the catalog of interoperability architectural solutions	74
Figure 11 – Catalog of architectural patterns to solve interoperability issues	87

List of Tables

Table 1 – Elements that could be included in pattern formats	21
Table 2 – Search strings adapted for each publication database	25
Table 3 – Complete list of identified architectural strategies from the literature review	28
Table 4 – Frequency of answers on the type of interoperability at using each strategy from the practitioner’s perspective. Responses that obtained at least 50% of the practitioners’ opinions for each strategy are highlighted.	36
Table 5 – Frequency of difficulty levels at using each strategy from the practitioner’s perspective.	38
Table 6 – Practitioners impact perception of architectural strategies over quality attributes. Answers (P=Positive Impact, N=Negative Impact, IK= I Don’t Know, NI = None Impact). Responses that obtained at least 50% of the practitioners’ opinions for each strategy are highlighted	42
Table 7 – List complete of association rules generated for supporting the selection of architectural strategies	50
Table 8 – Architectural strategies suggested for each interoperability level	53
Table 9 – Typology of architectural strategies for interoperability	60
Table 10 – Architectural patterns to solve interoperability issues at the technical level	77
Table 11 – Architectural patterns to solve interoperability issues at the syntactic level	80
Table 12 – Architectural patterns to solve interoperability issues at the semantic level	82
Table 13 – Architectural patterns to solve interoperability issues at the organization level	85

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Problem Statement	4
1.3	Research Question and Objectives	4
1.4	Research Method	5
1.5	Summary of Contributions	6
1.6	Thesis Outline	7
2	Background	9
2.1	Software Architecture	9
2.1.1	Terminology	10
2.1.2	Architectural Process	12
2.1.3	Quality Attributes	14
2.2	Interoperability	15
2.2.1	Types of Interoperability	15
2.2.2	Levels of Interoperability	17
2.3	Patterns	17
2.3.1	Terminology and Pattern Elements	18
2.3.2	Description of Patterns	20
2.4	Final Considerations	22
3	Identification of State of the Art and Practice on Interoperability Strategies	23
3.1	State of the Art	23
3.1.1	Planning	23
3.1.2	Conduction	25
3.1.3	Results	26
3.1.3.1	Approaches for Interoperability	27
3.1.3.2	Architectural Strategies for Interoperability	28
3.2	State of the Practice	29
3.2.1	Research Method	29
3.2.1.1	Research Goal and Questions	29
3.2.1.2	Survey Design and Execution	30
3.2.1.3	Threats To Validity	32
3.2.2	Characterization of Target Audience	33
3.2.3	RQ ₁ : Strategies used by practitioners in integration projects	35

3.2.4	RQ ₂ : Difficulty at Using the Architectural Strategies	37
3.2.5	RQ ₃ : Impact Over the Quality Attributes	41
3.2.6	RQ ₄ : Challenges for Interoperability	43
3.2.7	RQ ₅ : Improvement of Integration Process	44
3.2.8	RQ ₆ : Technologies for Integration Projects	45
3.3	Mining Association Rules	46
3.3.1	Overview of the Process for Generating Association Rules	48
3.3.2	Association Rules	49
3.4	Final Considerations	54
4	TASIS: A Typology of Architectural Strategies for Interoperability of Software-Intensive Systems	57
4.1	Initial Version of TASIS	57
4.1.1	Methods	57
4.1.2	Threats to Validity	59
4.2	Architectural Strategies for Interoperability	61
4.2.1	Technical Interoperability	61
4.2.2	Syntactic Interoperability	62
4.2.3	Semantic Interoperability	64
4.2.4	Organizational Interoperability	65
4.3	Consolidation of TASIS	67
4.4	Discussion	69
4.5	Final Considerations	70
5	A Catalog of Architectural Patterns to Solve Interoperability Issues in Software-Intensive Systems	73
5.1	Research Method	74
5.2	Technical Interoperability	75
5.2.1	Interoperability Issues	75
5.2.2	Architectural Patterns	76
5.3	Syntactic Interoperability	78
5.3.1	Interoperability Issues	78
5.3.2	Architectural Patterns	79
5.4	Semantic Interoperability	81
5.4.1	Interoperability Issues	81
5.4.2	Architectural Patterns	82
5.5	Organizational Interoperability	83
5.5.1	Interoperability issues	83
5.5.2	Architectural Patterns	84
5.6	Discussion and Lessons Learned	86
5.7	Final Considerations	88

6 Conclusions 89

6.1 Thesis Contributions 90

6.2 Limitations 91

6.3 Future Work 91

References 93

APPENDIX A Survey Questionnaire 105

A.1 Initial Considerations 105

A.2 Consent for the Research Study 105

A.3 Participants 106

A.4 Architectural Style 108

A.5 Architectural Pattern 111

A.6 Difficulty Level 117

A.7 Background 131

A.8 Challenges 131

A.9 Contact 131

A.10 Final Considerations 132

Introduction

1.1 Context and Motivation

Due to the complexity of current software-intensive systems, an increasing interest in researching the development and evolution of these systems have been observed in the last years ([BALDWIN *et al.*, 2017](#); [BOUZIAT *et al.*, 2018](#); [VALLE *et al.*, 2021b](#)). This class of systems is required in several domains, such as connected health, Industry 4.0, military, smart cities, smart grids, and smart agriculture. Examples of software-intensive systems are those that need to exchange data to achieve their goal. These systems are sometimes based on microservices, IoT (Internet of Things), big data, Systems-of-Systems (SoS), among others ([VALLE *et al.*, 2021a](#)).

Software-intensive systems are often independently developed, operated, managed, and evolved. To achieve their goal, these independent systems can exchange information among them, and through that interaction, they may form a new kind of complex system, in other words, a system that is itself composed of systems ([GARCÉS *et al.*, 2019](#); [VALLE *et al.*, 2020](#)). In short, software-intensive systems are composed of individual systems that need to exchange information among them to fulfill a specific purpose. Their operations depend on interactions among other systems that are heterogeneous, distributed, and sometimes independent, large-scale, and software-intensive ([BOUZIAT *et al.*, 2018](#)).

Individual systems (also named constituent systems) participating in larger systems are developed in a distributed way and, consequently, have a different life cycle, operational environment, programming languages, platforms, goals, and business interests. It happens mainly because these systems are created and managed by different organizations ([MADNI; SIEVERS, 2014](#)). It is worth highlighting that the building of those systems includes a diversity of stakeholders and multidisciplinary practitioners, which are involved during their whole life cycle, e.g., requirements specification, architectural design,

development, testing, deployment, maintaining, and evolution (VALLE *et al.*, 2019).

Software engineers have faced difficulties in the development of software-intensive systems due to collaborations among their individual systems (ZOPPI *et al.*, 2017). In this sense, architects and developers of software-intensive systems are encouraged to adopt solutions to integrate heterogeneously, distributed, and independent software systems (ABUKWAIK; ROMBACH, 2017; KUBICEK *et al.*, 2011). Therefore, interoperability has an essential role in the composition of complex software-intensive systems. In the software systems context, interoperability is defined as the ability of different systems to share and manage semantically compatible information, enabling users to perform desired tasks (MADNI; SIEVERS, 2014). IEEE (2000) also defined interoperability as “*the ability of two or more systems or elements to exchange information and to use the information that has been exchanged*”.

In the scientific literature, interoperability has been widely classified into different levels (BENSON; GRIEVE, 2016; STADEN; MBALE, 1998; REZAEI *et al.*, 2014). In this thesis, we explore the four interoperability levels (KRUCHTEN, 2004; VEER; WILES, 2008; IBRAHIM; HASSAN, 2010): (i) *technical interoperability*: it is related to transmit data between components/systems. It is associated with the hardware and software components, networks, and equipment that enable machine-to-machine communication; (ii) *syntactic interoperability*: it focuses on rules about the type of data that is exchanged, specifically, on the way to put together the data and in which order (i.e., message content structure, size of headers, size of the message body, fields contained in a message); (iii) *semantic interoperability*: it refers to the exact meaning of the exchanged data. It concerns with the guarantee that the precise meaning of exchanged information is understandable by any other system; and (iv) *organizational interoperability*: it refers to the coordination of distributed workflows and activities that are well understood by systems, organizations, or people interacting in business processes.

Interoperability is a complex task since it is far more than getting to communicate with each other. It requires the organizations involved to employ some degree of compatible semantics and common interpretations of the information exchanged (ABUKWAIK; ROMBACH, 2017). To enable the successful integration of systems and their meaningful data exchange and interactions, an analysis of technical, conceptual, and organizational constraints is required (NIELSEN *et al.*, 2015). In short, when a system needs to exchange information, it is necessary to ensure that it understands the data, processing, and policies of other systems with which it interoperates and that they, in turn, understand their data, processing, and policies (MADNI; SIEVERS, 2014). Therefore, interoperability has been claimed as an essential requirement for the successful establishment of software-intensive systems.

Practitioners have faced several challenges of interoperability in their integration

projects (VALLE *et al.*, 2021a). Examples of these challenges are (i) assurance that other quality attributes are not affected by the scenarios required for the data/systems integration; (ii) lack of documentation, proven solutions, and guidelines to support the implementation of interoperability in the integration projects context; and (iii) a better understanding of the meaning of data and linkages of data via metadata. Therefore, solutions to mitigate such challenges should be proposed, as well as means to consider interoperability in software-intensive systems at the architectural level.

In this perspective, software architectures have been played a fundamental role in determining the quality of software systems (BASS, 2013), promoting several characteristics of quality, e.g., interoperability, performance, portability, adaptability, and maintainability. Therefore, software architectures are also essential to the success of complex software-intensive systems (NAKAGAWA *et al.*, 2015; ABTOY *et al.*, 2018; DELECOLLE *et al.*, 2020). The definition of software architecture has evolved over the years. Most of these definitions can be synthesized by the definition of Bass (2013): “*architecture of a software program or computing system is the structure or structures of the system, which is composed of software components, the externally visible properties of those components, and the relationships between them*”.

Software architectures are essential because decisions made at the architectural level directly enable, facilitate, or interfere in achieving business goals and in functional and quality requirements (BASS, 2013). To support the design of these architectures, Hofmeister *et al.* (2007) proposed a process composed of three main activities: (i) *architectural analysis*: this activity articulates architecturally significant requirements (ASRs) accordingly to architectural concerns and the context; (ii) *architectural synthesis*: architectural decisions are proposed to address the ASRs; and (iii) *architectural evaluation*: this activity ensures that architectural decisions are the right ones to enable the requirements to be satisfied. Due to the complexity of the architectural design, these activities are not executed sequentially; instead, they are conducted repeatedly at many levels of granularity until the architecture is complete and evaluated (HOFMEISTER *et al.*, 2007).

In particular, architectural synthesis is challenging, mainly due to the necessity of defining a *good* architectural design that addresses essential quality requirements in a system (BASS, 2013). Therefore, different architectural strategies have been proposed during the years to promote the reuse of good architectural designs for different quality requirements (MESBAH; DEURSEN, 2007; MICHENI *et al.*, 2014). For interoperability requirements, architectural style, patterns, and tactics have been proposed to address this quality attribute at the architectural level (REZAEI *et al.*, 2014).

1.2 Problem Statement

Despite the critical role of interoperability in the composition and execution of complex functionalities in software-intensive systems, software architects still face challenges that hamper the building and operation of those systems effectively and efficiently. In this perspective, we conducted a survey, and we identified several challenges faced by practitioners in their integration projects (VALLE *et al.*, 2021a): (i) the absence of documentation, architectural solutions used in practice, and directions to tackle interoperability issues in projects that need to exchange data, and (ii) the lack of understanding the vocabulary of the data exchanged among systems.

Abukwaik and Rombach (2017) also carried out an online survey with experienced software engineers and identified the main concerns that must be overcome in integration projects: (i) misunderstanding of data formats, data semantic, procedures, contracts, standards, quality, and interfaces provided by individual systems; and (ii) absence of high-level architectures to analyze, comprehend, and guide the way that interoperability can be considered. We intend to support software architects in decision-making to mitigate interoperability issues in software-intensive systems from these scenarios. Therefore, the main problem tackled in this thesis is:

The difficulty faced by software architects to consider interoperability in the software architectures of software-intensive systems that need to exchange useful data among them, but often present different data formats and interfaces.

In this perspective, the next section describes the research questions and objectives of this thesis that could contribute to mitigating the aforementioned problem.

1.3 Research Question and Objectives

Motivated by the challenges mentioned above and the problem, we intend to answer the following research question.

How could software architects use existing architectural solutions to mitigate the problems in each interoperability level in software-intensive systems?

Based on this question, the main objective of this thesis is **to provide architects means to decision-making on how to mitigate interoperability issues in software-intensive systems considering architectural solutions available in the scientific literature and industry**. We defined the following specific objectives to achieve the main objective and also answer the research question:

- *Identification of interoperability challenges:* The main challenges faced by software architects in the industry to consider interoperability requirements in their integra-

tion projects were investigated. Existing architectural solutions that practitioners have used to mitigate the main interoperability challenge were also investigated.

- *Establishment of a typology of architectural strategies for interoperability:* The current strategies for interoperability of software-intensive systems were investigated to comprehend how to address interoperability requirements. These strategies were organized in a typology, named TASIS, in which the interoperability levels were categorized into four classes, namely: technical, syntactic, semantic, and organizational interoperability, and, for each interoperability level, three types of solutions were associated (architectural patterns, styles, and tactics); and.
- *Establishment of a catalog:* A catalog composed of the main issues of each interoperability level and their respective solutions was defined. For this, we carried out a literature review and a survey with practitioners with expertise in software architecture, interoperability, and software-intensive systems in the industry. This catalog could help architects in decision-making on which architectural pattern can be considered to solve the identified issues in their integration projects.

1.4 Research Method

As aforementioned, the main objective of this thesis is to provide means to help software architects in decision-making on how to address interoperability in software-intensive systems. Hence, this thesis was conducted as a scientific research project that followed a methodology with well-defined steps, as presented:

1. **Research question elaboration:** The first step was the definition of the research question that motivates the conduction of this PhD research: *How could software architects use existing architectural solutions to mitigate the problems in each interoperability level in software-intensive systems?*. This research question aims to understand how architects can use the existing solutions to mitigate interoperability issues. Therefore, we needed to conduct researches to understand what solutions already existed and what research gaps would be addressed, as performed in step 2 discussed below.
2. **Literature review and gaps identification:** We conducted a literature review to identify gaps in the interoperability of software-intensive systems at the architecture level. As a result, we have identified many architectural solutions that can be considered in integration projects to promote interoperability. We classified such solutions into architectural tactics, architectural patterns, architectural styles, and techniques. The literature review was conducted from January 2019 to April 2019

using the following databases: Scopus, IEEE Xplore, Web of Science, ACM Digital Library, and Scopus.

3. **Solution proposal:** Using the information obtained in the literature review, we defined a TESIS, named typology. It organizes the identified architectural solutions according to the type of solution (i.e., architectural styles, patterns, and tactics) and the four interoperability levels (i.e., technical, syntactic, semantic, and organizational). The first version of TESIS was published in a conference paper (VALLE *et al.*, 2019). To validate the TESIS, we conducted a survey with several practitioners with expertise in software architecture and interoperability to understand whether the classification of the architectural solutions proposed in TESIS was correct. The result of this survey was also published in a conference paper (VALLE *et al.*, 2021a). We also have proposed a catalog to help software architects solve the main issues at each interoperability level. The catalog is composed of the main interoperability issues in each of its levels showing architectural patterns that can be used to mitigate such issues.
4. **Results communication:** Once the TESIS and the catalog were proposed, we concluded that they support software architects in the decision-making on interoperability of software-intensive systems. The first results of this thesis were communicated through conference publications. We also intend to submit the other results of this thesis to journal publications.

1.5 Summary of Contributions

This thesis contributes to the areas of interoperability, software architecture, and software-intensive systems. Specifically, the typology of architectural strategies (i.e., a catalog of architectural solutions) proposed in the context of this thesis may be used by practitioners to tackle interoperability at the architectural level in their integration projects. Hence, practitioners may choose the solutions according to their needs and interoperability requirements.

We make available an overview of the way that architectural strategies have been used in practice. This evidence is crucial because it shows the most strategies solve issues at all interoperability levels, contradicting the tendency of academics. We outline the difficulty of applying these strategies and how each may impact the quality attributes. We also provide a catalog with the main issues faced by architects and their respective solutions to support decision-making on interoperability. This catalog contributes directly to the problem addressed in this thesis, which is the difficulty of software architects to tackle interoperability in their systems. Therefore, the practitioners may have the means to make decisions on interoperability in their integration projects.

Finally, this thesis can contribute to the four interoperability levels, avoiding misunderstanding data formats, data semantic, procedures, contracts, standards, and interface structures provided by individual systems.

1.6 Thesis Outline

This chapter presented an overview of the context and problems that motivated the conduction of this thesis, followed by the research question and objectives. The remaining chapters of this thesis are organized as follows.

Chapter 2 provides an overview of the concepts directly associated with topics investigated in this thesis. Firstly, we introduce concepts related to software architectures, specifically the terminology, the architectural process, and the quality attributes. We also present concepts related to interoperability, the interoperability levels, main challenges for achieving this quality attribute, and suggestions for improving the integration process. Finally, we introduce the terminology and pattern elements.

Chapter 3 shows the survey that we carried out to validate the information obtained from the scientific literature about the main architectural strategies for interoperability. As a result, 33 practitioners of several countries participated in our survey and contributed to validated the previous information.

Chapter 4 presents the initial version of **TASIS**, which is the typology of architectural strategies for interoperability. The results are reported in (VALLE *et al.*, 2019). Then, we present the survey results carried out with practitioners with experience in integration projects about how they have used the strategies architectural in practice. The results are reported in (VALLE *et al.*, 2021a). We also used such information to consolidate TASIS, updating and validating it.

Chapter 5 describes the proposed catalog with the main interoperability issues and the architectural patterns to mitigate them in integration projects. This catalog was divided according to interoperability levels and was defined according to evidence provided by scientific literature and industry. Hence, this catalog can be used as a means for supporting the decision-making on interoperability.

Finally, Chapter 6 outlines the main contributions, the limitations of this thesis, and the future work.

Appendix A presents the survey questionnaire applied with practitioners from the industry. This questionnaire includes 14 to 104 questions. The number of questions depends on the number of architectural strategies selected by participants. We used filters to show only relevant questions for each participant.

Background

This chapter introduces the main concepts for the understanding and conduction of this thesis. Section 2.1 presents the main topics related to software architectures, showing the terminology related to this topic, as well as the concept of the architectural process and quality attributes that should be considered in the design of software architectures. Section 2.2 presents the definition of interoperability considered in this work, the levels of interoperability, challenges faced by practitioners in the industry, the improvements of the integration process, and technologies for integration projects used by practitioners in the industry. Finally, Section 2.3 describes the main concepts related to architectural pattern and associated terminology and elements since we investigated it as a solution for the the problem addressed in this thesis.

2.1 Software Architecture

Software architectures are considered essential to support the development of high-quality software systems (BASS, 2013). Such architectures provide good practices of development, involving models, reference architectures, design patterns, architectural patterns, and architectural styles to support the development of software systems (LAND, 2002; HASSELBRING, 2018). Therefore, software architecture plays an essential role in determining the system quality, as it is considered the backbone of any successful software-intensive system (CLEMENTS *et al.*, 2011). Decisions made at the architectural level may directly impact the achievement of business goals, and functional and quality requirements (HOFMEISTER *et al.*, 2007).

According to Bass (2013), “software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”. A software architecture focuses on the representation of element interfaces and does not show the internal details on the way that

such elements are implemented (CLEMENTS *et al.*, 2011). There are other definitions for software architecture, as that proposed by Jansen and Bosch (2005), who define software architecture as a collection of explicit architectural design decisions made over time. Despite this, we considered the definition proposed by Bass (2013) for the conduction of this work due to its good acceptance in the scientific community of software architecture.

2.1.1 Terminology

The concept of software architecture as it is understood nowadays arose in the early 1990s. In the software architecture area, many concepts and terminologies are used to establish a common glossary for practitioners and researchers of software architecture. To support this, we present some close concepts that are related to software architecture. Among them, we highlight:

- **Stakeholder:** refers to any individual, team, or organization that is interested in software systems and plays an important role during the development process of software-intensive systems (ISO, 2011b). Examples of stakeholders are clients, users, software architects, software engineers, developers, testers, product owners, network analysts, system analysts, and others.
- **Concern:** is an interest or need of one or more stakeholders on the software systems under development (ISO, 2011b). Examples of concerns are functionalities, qualities, the purpose of use, costs, and others.
- **Concrete Software Architecture:** it is the software architecture of a software system. The fundamental organization of a system considering its components, their relationships to each other and to the environment, and the principles guiding its design and evolution (ISO, 2011b).
- **Architectural Instance:** it is the concrete architecture of a software system, generated as a product of a reference architecture (NAKAGAWA *et al.*, 2007).
- **Architectural Patterns** are means to capture well-proven design structures, facilitating their reuse (BUSCHMANN *et al.*, 1996). It expresses a fundamental structural organizational schema for software systems. These patterns can be considered as a package of design decisions that are found repeatedly in practice (MICHENI *et al.*, 2014). Examples of architectural patterns that can address software systems interoperability are pipes and filters, scatter/gather, centralized architecture, mediator, contract monitor, bus, adapter, wrapper, facade, and broker (VALLE *et al.*, 2019; VALLE *et al.*, 2021a).

- **Architectural Tactics:** are design decisions that influence the obtaining of a quality attribute response. Tactics directly affect the system's response to some stimulus (BASS, 2013) related to a specific quality attribute requirement. Examples of tactics for the quality attribute of interoperability are discover service, tailor interface, orchestration, and choreography (VALLE *et al.*, 2019; VALLE *et al.*, 2021a).
- **Reference Model:** it refers to a standard decomposition of a known problem into parts that cooperatively solve the problem (BASS, 2013). It can be seen as a division of functionality with data flow between the pieces, defining the standard characteristics for an architectural domain (BASS, 2013).
- **Reference Architecture:** refers to an architecture that encompasses the knowledge about how to design concrete architectures of systems of a given application domain, promoting reuse of design expertise; therefore, it must address the business rules, architectural styles (sometimes also defined as architectural patterns that can also address quality attributes in the reference architecture), best practices of software development (for instance, architectural decisions, domain constraints, legislation, and standards), and the software elements that support the development of systems for that domain. All of this must be supported by a unified, unambiguous, and widely understood domain terminology (NAKAGAWA *et al.*, 2014).
- **Domain architecture:** domain-specific software architectures (DSSA) concept has been used as a synonym of reference architectures (TRACZ, 1995). DSSA and reference architecture definitions can be related in the sense that, a reference architecture for a specific domain is an input for developing DSSAs in such domain. In this context, it is possible to infer that a DSSA is more specific than a reference architecture since DSSA includes a selection of operating system, middleware, and data persistency services (BOSCH; RAN, 2000), that are not considered in reference architectures.
- **Product Line Architecture (PLA):** it is a type of software architecture used to build a product line, describing, explicitly, commonality and variability. In addition, a PLA is a basis for the architecture of all product line members (NAKAGAWA *et al.*, 2011).

The definition of such terms is important to avoid misconceptions since many of them are usually applied as synonyms. For instance, reference architectures are often used to refer to software architectures and reference, despite although the reference model is more general than other terms (BASS, 2013; ANGELOV *et al.*, 2009; NAKAGAWA *et al.*, 2014).

2.1.2 Architectural Process

Over the last years, several approaches, processes, techniques, best practices ([HOFMEISTER *et al.*, 2007](#); [KRUCHTEN, 2004](#); [BOSCH](#); [RAN, 2000](#)), an international standard ([ISO, 2011b](#)) have been proposed to support the development of software architectures. In this sense, [Hofmeister *et al.* \(2007\)](#) proposed a model to design software architecture in which activities of architectural analysis, synthesis, and evaluation have an important role in the successful deployment of the software architectures and, thereby, to the remaining activities of the software development. It is worth highlighting that these activities are not executed sequentially. They are performed repeatedly in many levels of granularity until the architecture is complete and evaluated ([HOFMEISTER *et al.*, 2007](#)). The overview of this model is shown in Figure 1, which is presented the three main activities and their input/output artifacts.

Figure 1 – Architectural design activities

Source: Adapted from [Hofmeister *et al.* \(2007\)](#).

As illustrated in Figure 1, the model to the design of software architectures consists of three main activities: architectural analysis, architectural synthesis, and architectural evaluation. These activities are described below:

- **Architectural Analysis:** its main objective the definition of problems that the software architecture must solve. The architectural concerns and the context of a system that should be developed are used as input to this activity. As a result of this activity, a set of ASRs are defined ([HOFMEISTER *et al.*, 2007](#)).
- **Architectural Synthesis:** defines a set of candidate architectural solutions that addresses the ASRs, impacting design decisions about the structure of software. This activity also includes information about the design rationale (for instance, commentary on why decisions were made, what decisions were considered and rejected, and traceability of decisions to requirements) ([HOFMEISTER *et al.*, 2007](#)).
- **Architectural Evaluation:** Its objective is to ensure that architectural design decisions made on architectural synthesis are the right ones to enable the achievement

of both functional and non-functional requirements. Therefore, candidate architectural solutions are verified upon ASRs built from the architectural analysis. As a result of multiple iterations, which consist of acceptance or disapproval of candidate solutions, validated architecture is expected as a result (HOFMEISTER *et al.*, 2007).

This thesis is focused on the architectural synthesis activity. This activity involves the architecture description that is considered the main artifact in determining software architectures since it executes a fundamental role in communication, quality assessment, evaluation, and evolution (CLEMENTS *et al.*, 2011; ISO, 2011b; KRUCHTEN *et al.*, 2006). Building a good architecture description is essential for the effective use of software architectures during the lifecycle of software-intensive systems (OLIVEIRA, 2015). In this context, The ISO/IEC/IEEE 42010 standard (ISO, 2011b) specifies how the architecture descriptions of software systems should be and expressed. This standard shows the elements of the architecture description, considering the creation, analysis, and sustenance of architectures of software-intensive systems. The main elements proposed by ISO/IEC/IEEE 42010 standard for building architecture descriptions are:

- **Architecture Viewpoint:** establishes the conventions for the construction, interpretation, and use of architecture views to frame specific systems concerns. In other words, it formalizes the idea that there is a different way of looking at the same systems;
- **Model Kind:** defines conventions for a given type of modeling by describing rules, meta-models, templates, and operations to represent the relationships and information in an architecture description. Examples of models kinds are data flow diagrams, class diagrams, statecharts, and state transitions, models;
- **Architecture View:** addresses one or more concerns of stakeholders of the system target following an architecture viewpoint. It is composed of one or more architecture models;
- **Architecture Model:** uses modeling conventions appropriate for the concerns to be addressed by one or more architecture views. These conventions are described by the model kind that governs the architecture description. An architecture model can be a part of more than one architecture view in the architecture description;
- **Architecture Rationale:** outlines the explanation, justification, or reasoning about architecture decisions that have been made. The rationale for a decision can include the basis for this decision, alternatives and trade-offs considered, as well a potential consequences and citations to source of additional information; and

- **Correspondence:** presents the relation between architecture elements by showing the architecture relations of interest within an architecture description (or between architecture descriptions). Architecture correspondences can be governed by correspondence rules used to enforce relations within an architecture description.

Another element important is the architectural description language (ADL) which is any form of expression for use in architecture descriptions that provides one or more model kinds as a means to frame concerns for the audience of stakeholders. In other words, ADL is a formal language it is used to represent the architecture of software-intensive systems. An ADL has different levels of formality (for instance: formal, semi-formal, or informal). This element supporting the creation, refinement, and validation ([MEDVIDOVIC; TAYLOR, 1997](#); [CLEMENTS *et al.*, 2011](#)).

2.1.3 Quality Attributes

Quality attributes have been aroused the interest of the software community at least since the 1970s. There are several taxonomies and definitions about quality attributes, and many of them have their research and practitioner communities ([BASS, 2013](#)). Quality attributes are some of the most important requirements of a system. Quality attributes are considered characteristics of the system, as opposed to functionality that describes what the system should be done. Examples of quality attributes are usability, maintainability, performance, and reliability.

[Sommerville \(2019\)](#) discusses the difference between functional and non-functional requirements. Specifically, functional requirements are services that should be provided by the target system, while the non-functional requirements (understood as requirements for quality attributes) are constraints on the functions offered by the system. The quality attribute requirements specify the responses of the system and realize the goals of the business. To achieve the required quality attributes, architectural tactics can be considered in the design of software architectures. A tactic is a design decision that influences the achievement of a quality attribute response ([BASS, 2013](#)).

The ISO/IEC/IEEE 25010 standard ([ISO, 2011a](#)) defines the product quality model that comprises the eight quality characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. To address quality attributes in a more concrete way, ([BASS, 2013](#)) proposed the use of quality attributes scenarios to describe quality attributes requirements. This approach is similar to use cases employed for functional requirements specifications and can be applied to support the understanding of quality goals from the perspective of different stakeholders types.

Interoperability is considered one of the most relevant quality attributes for software-intensive systems. This quality attribute can be handled as a cross-cutting concern that has a unique and crucial coordination role relative to the other quality concerns (ROTHENBERG, 2008; BIANCHI *et al.*, 2015; VALLE *et al.*, 2019). Interoperability is the quality attributes investigated in this thesis. Therefore, in the next section, we present the concepts related to this quality attribute and some challenges faced by practitioners in the industry, we also described some suggestions for improving the integration process provided by practitioners.

2.2 Interoperability

Interoperability is a topic very important both for academia and industry. It is considered a great challenge for developing and evolving complex systems. However, there are several definitions of interoperability, since this term can have many interpretations in different contexts. Following, we present some definitions for interoperability considered in the context of this work.

- *“The ability of two or more systems or elements to exchange information and to use the information that has been exchanged”* (IEEE, 2000).
- *“The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together”* (DOD, 2001).
- *“Interoperability refers to a property of diverse systems and organizations which enables them to work together”* (GOTTSCALK, 2009).
- *“The ability of organizations to interact towards mutually beneficial goals, involving the sharing of information and knowledge between these organizations, through the business processes they support, through the exchange of data between their ICT systems”* (EIRA, 2018).

From the definitions presented above, it is observed the interoperability is not related to one single system, but instead, it is related to the interaction between two or more systems. It is worth highlighting that interoperability is not related only to the exchange of information, but also to the use of the object of interoperation.

2.2.1 Types of Interoperability

When an organization is willing to share data with different architectures, vendors, or solutions, elements of associative autonomy have to be there to control autonomy at

different levels of data sharing across the organization for inter/intra communication and exchange of information (KUBICEK *et al.*, 2011). The interoperability in software systems perspective could be categorized into different types (i.e., technical, syntactic, semantic, and organizational) (MICHENI *et al.*, 2014). Following, we present a short description of each type of interoperability.

Technical Interoperability: it is related to the interoperability of infrastructure and software. The interoperability of infrastructure is the ability of hardware acquired by different organizations to work in a connected way. This type of interoperability is associated with the hardware and software components, networks, and equipment that enable machine-to-machine communication to take place, including aspects as open interfaces, connectivity, data integration, middleware, data presentation, data exchange, accessibility, and security issues (VEER; WILES, 2008; IBRAHIM; HASSAN, 2010).

Syntactic Interoperability: it is related to the data representation in machine-readable form and, in general, associated with data formats (VEER; WILES, 2008). This type of interoperability aim at identifying elements and rules for structuring the elements, mapping, bridging, and crosswalks among equivalent elements (VELTMAN, 2001; MICHENI *et al.*, 2014). In short, syntactic interoperability assists two or more systems in communicating and exchanging data (MICHENI *et al.*, 2014).

Semantic Interoperability: it is related to the guarantee that the precise meaning of exchanged information is understandable by any other application that was not initially developed for this purpose (MICHENI *et al.*, 2014). In this type, only relevant information can be exchanged or shared. It will support a high level, context-sensitive information request over heterogeneous information resources, hiding system, syntax, and structural heterogeneity. To achieve semantic interoperability, both sides must refer to a common information exchange reference model (KUBICEK *et al.*, 2011).

Organizational Interoperability: it is related to the ability of two or more units to provide services to and accept services from other units and to use the services to support them to operate effectively together (LEGNER; LEBRETON, 2007). Each organization has its own unique culture, capabilities, and operating procedures. Organizational Interoperability depends on the successful implementation of technical, syntactical, and semantic interoperability (MICHENI *et al.*, 2014).

It worth highlighting that does not exist a clear dependency among these types of interoperability. For some domains, organizational interoperability could be established before technical, syntactic, and semantic interoperability, or vice-versa (BENSON; GRIEVE, 2016). In this sense, a software system should be considered full interoperable when it requires and attends all types of interoperability.

2.2.2 Levels of Interoperability

The Levels of Information System Interoperability (LISI) is a model for SoS interoperability widely recognized in the scientific literature (MORRIS *et al.*, 2004). This model is composed of five levels of sophistication of SoS interoperability as discussed following.

Level 0: Isolated: this interoperability occurs in a manual environment between stand-alone systems. In other words, interoperability at this level comprises the manual extraction and integration of data from multiple systems.

Level 1 – Connected: this interoperability occurs in a peer-to-peer environment. This level is related to the electronic links with some form of simple electronic exchange of data. Some examples of simple, homogeneous data types, are: voice, text email, and graphics are shared. It is important to remember that at this level of interoperability, there is little capacity to fuse information.

Level 2 – Functional: this interoperability occurs in a distributed environment. Systems run on local area networks that allow data to be exchanged from system to system. Data is usually heterogeneous-containing information from many simple formats fused (e.g., images with annotations).

Level 3 – Domain: this interoperability occurs in an integrated environment. Usually, systems are connected through wide area networks, in which the information is exchanged between independent applications using shared domain-based data models. In this level of interoperability, it is enabled common business rules and processes, and direct database-to-database interactions. Besides, it supports group collaboration on fused information.

Level 4 – Enterprise: this interoperability occurs in a universal environment. Systems use a global information space across multiple domains, in which multiple users can access complex data simultaneously. At this level, advanced forms of collaboration are possible, in which data and applications are fully shared and distributed. In this case, the data has a common interpretation regardless of format.

It is important to highlight that the LISI focuses on technical interoperability and the complexity of interoperations among systems. This model does not address the environmental and organizational issues that contribute to the building and maintenance of interoperability systems (MORRIS *et al.*, 2004).

2.3 Patterns

Although there are different means to capture and encapsulate knowledge related to the description of reference architectures, still there is a lack of standardization and means to guide the description of interoperability in such architectures. In this perspective, patterns

could be considered as a method to describe tacit knowledge to allow it to be reapplied when a new problem is found. In other words, a pattern describes a solution to a problem in a recurring manner (PRESSMAN; MAXIM, 2014). Following, we present the concepts related to patterns language that should be known to understand the solution proposed by this thesis. In particular, we discuss the terminology and pattern elements, as well as, the concepts related to patterns language.

2.3.1 Terminology and Pattern Elements

The concept of patterns was defined by a building architect, named Christopher Alexander, who recognized that a recurring set of problems is found whenever the building is designed (ALEXANDER, 1977; PRESSMAN; MAXIM, 2014). These recurrent problems and their solutions are characterized as patterns. Therefore, each pattern describes a problem that occurs repeatedly in the environment. It also describes the core of the solution to that problem in such a way that anyone can use it many times without ever doing it the same way twice.

As mentioned aforementioned, Alexander (1977) proposed the first concepts about patterns in the building context. He affirms that a pattern can be described as “*a three-part rule which expresses a relation between a certain context, a problem, and a solution*”. Therefore, according to the definition mentioned before, we need to adopt a three-part schema that underlies every pattern (BUSCHMANN *et al.*, 1996): i) *context*: a situation giving rise to a problem; ii) *problem*: the recurring problem arising in that context; and iii) *solution*: a proven resolution of the problem. Figure 2 summarizes the whole schema.

Figure 2 – Three-part schema of a pattern description

Source: Adapted from Buschmann *et al.* (1996).

The schema presented aforementioned, expresses a type of rules that establishes a relationship between a given context, a problem that arises in this context, and an appropriate solution for the problem considered. It is worth highlighting that all three parts of this schema are closely coupled (BUSCHMANN *et al.*, 1996). In the software context, Beck and Cunningham (1987) used Alexander's concepts for a pattern language to design windows in Smalltalk. Then, other initiatives that considered patterns in software context have emerged (COAD, 1992; COPLIEN; SCHMIDT, 1995; GAMMA, 1995; BRAGA, 2002; FALBO *et al.*, 2016; FIORAVANTI, 2017; FASSBINDER, 2018; ARCAINI *et al.*, 2020).

The existing patterns showed that they cover several ranges of scale and abstraction. Some patterns could help in structuring a software system into subsystems, others could support the refinement of subsystems and components, or of the relationships between them, while others could help in implementing particular design aspects in a specific programming language (BUSCHMANN *et al.*, 1996; PRESSMAN; MAXIM, 2014). In this sense, Buschmann *et al.* (1996) classified the software patterns into three categories:

- **Architectural Patterns:** express a fundamental structural organizational schema for software systems. They provide a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships among them.
- **Design Patterns:** provide a scheme for refining the subsystems/components of software systems, or the relationships among them. They describe a commonly-recurring structure of communicating components that solves a general design problem within a particular context.
- **Idioms:** is a low-level pattern specific to a programming language. They describe how to implement particular aspects of components or the relationships among them using the features of the given language.

Besides the categories presented aforementioned, Pressman and Maxim (2014) identified more three types of software patterns, as shown following:

- **Data Patterns:** describe recurring data-oriented problems and the data modeling solution that can be used to solve them.
- **Component Patterns:** consider problems related to the development of subsystems/components, how they communicate with one other, and their placement within a large architecture.

- **Interface Design Patterns:** describe common user interface problems and their solution with a system that includes the specific characteristics of end-users at a lower level of abstraction.

Such types of patterns could provide many benefits for software development, as well as, the design of other types of products in different domains (GAMMA, 1995; BUSCHMANN *et al.*, 1996; PRESSMAN; MAXIM, 2014). Such benefits include:

1. a common vocabulary for communication among stakeholders.
2. a codified means for documenting problems and their solution in a way that allows the community to capture design knowledge for reuse in other projects.
3. acting as building blocks from which more complex projects can be built.
4. solutions that are successfully applied.

Different pattern formats have been used to describe the existing patterns. The formats proposed by Portland, Coplien, GoF, and Appleton are the best known (FOWLER, 2006). All formats are composed of several elements. However, there are mandatory elements that must be included in the description of patterns, such as name, problem, context, forces, and solutions (APPLETON, 1997; PRESSMAN; MAXIM, 2014). Table 1 presents an example of elements that could be included in the template for the description of patterns.

The description of a pattern can be dependent on the rules used to organize them into groups. A pattern could be grouped according to their abstraction level, allowing them to gather into collections, catalogs, systems, and language. In the next section, we present a brief description of how to describe the architectural patterns.

2.3.2 Description of Patterns

A pattern solves a problem of a particular context, but its applications can be results in other problems that could be solved also with the use of other patterns. In this sense, there is no pattern as an isolated entity since each pattern is supported by other patterns (ALEXANDER, 1977). In short, a pattern describes solutions to very similar problems, in which differs some of the influences involved. Therefore, establishing ways to group existing patterns according to criteria could facilitate their recovery and reuse.

From the software perspective, a pattern language is a structured collection of patterns that rely on each other to transform requirements and constraints into architecture (COPLIEN, 1998). A pattern language divides a general problem e its solutions into several related problems and their respective solutions. The problems are solved by

Table 1 – Elements that could be included in pattern formats

Element	Description
Name	It must have a meaningful name that represents a unique identifier to distinguish the pattern and express its important aspects. In order to define a name must choose one that a single word or short phrase to refer to the pattern, and the knowledge and structure it describes.
As known as	When other names are known for the pattern, they should be provided in their description.
Problem	A brief description of the problem the pattern addresses, showing the goals and objectives it wants to reach, including a discussion of its associated context and forces.
Context	A brief description of the problem context aiming to make it more comprehensible, in other words, the situation in which the pattern could be applied, besides preconditions under which the problem and its solutions seem to recur, and for which the solution is desirable.
Solution	A brief description of the solution suggested by the pattern that addresses the problem. Such a description may include figures, diagrams, and texts which identifies the pattern's structure, its participants, and their collaborations, to show the way that the problem is solved.
Variants	When there is variants or specialization of a pattern, they should be provided in their description.
Examples or Known Uses	Description of examples about when using the pattern addressed. For this, figures, hyperlinks, texts, and other sources may be used to represent the example of the solution. Besides known occurrences of the pattern and its application within existing systems. This helps validate the pattern, analyzing whether the pattern is indeed a proven solution to a recurring problem.
Related Patterns	The relationships among the pattern and others within the same context, that is, within the same systems or patterns language
Forces	A brief description of the benefits, as well as negative consequences that result from the pattern application. Therefore, information on the way that pattern interacts/conflict with one another and with goals that we wish to achieve should be shown in its description.

Source: Adapted from [Meszaros and Doble \(1996\)](#), [Buschmann et al. \(1996\)](#), [Fassbinder \(2018\)](#).

patterns that doing part of the same language. It is worth highlighting that each pattern can be applied separately or with several patterns of the languages.

Some researchers have proposed along with of years guidelines and techniques to support the process of writing and validation patterns and pattern languages. Among them, [Meszaros and Doble \(1996\)](#) proposed a pattern language that describes and demonstrates a collection of patterns writing practices. [Wellhausen and Fiesser \(2011\)](#) proposed a guide for first-time pattern authors. [Iba *et al.* \(2011\)](#) described a procedure for making a pattern language. [Braga *et al.* \(2007\)](#) also proposed a process to build pattern languages for a specific domain. In addition, [Guerra and Fernandes \(2010\)](#) proposed an evaluation technique for pattern languages.

2.4 Final Considerations

This chapter presented the main concepts related to this thesis. First, we introduced the concepts of software architecture, the main terminologies related the software architecture, the architectural process, and quality attributes. Then, we discussed the importance of interoperability to software-intensive systems, its four different types, as well as the main challenges for interoperability, and suggestions for improving the integration process. Finally, we presented an overview of architectural pattern, showing the main elements that should be considered in their description.

These contents are needed to understanding the contribution of this thesis since it proposes a means to address interoperability in the software architecture of software-intensive systems. The next chapter presents the identification of state of the art and practice on interoperability strategies.

Identification of State of the Art and Practice on Interoperability Strategies

Software-intensive systems are composed of Individual systems participating in larger systems that have different life cycles, the operational environment, programming languages, platforms, goals, and business interests (MADNI; SIEVERS, 2014). Therefore, developers of this class of systems are encouraged to adopt solutions for integrating such heterogeneous, distributed, and independent software systems (ABUKWAIK; ROMBACH, 2017; KUBICEK *et al.*, 2011). Therefore, interoperability has been considered an essential requirement for the successful establishment of software-intensive systems.

Several strategies have been proposed to address interoperability in software-intensive systems to exchange helpful information in this context. To know these strategies, we reviewed the literature to identify strategies used during the construction of software-intensive systems and surveyed practitioners to understand how they have considered these strategies in their integration projects. Therefore, in this chapter, we present the findings found about the architectural strategies.

3.1 State of the Art

To identify the architectural strategies used to addressed interoperability in software-intensive systems, we conducted a literature review and examined the main strategies reported in the literature. The planning and conduction of the research method are presented in the following.

3.1.1 Planning

We adopted the GQM (Goal Question Metric) (BASILI *et al.*, 1994) approach to support the conduction of this literature review. Hence, the goal of this literature review is:

Analyze existing architectural strategies
for the purpose of identifying them
with respect to adherence to interoperability levels
from the viewpoint of the software engineering research
in the context of software-intensive systems

To achieve this goal, we defined two research questions (RQs):

- **RQ₁** Which are the approaches applied to address interoperability in software-intensive systems? (e.g., patterns, technologies, techniques, frameworks).
- **RQ₂** Which are architectural strategies have been used to address interoperability in software-intensive systems?

RQ₁ aims to identify the types of strategies for interoperability that have been reported in the scientific literature. This information is essential because it could be used to understand which group of solutions have received more attention from the scientific community. RQ₂ intends to identify which are strategies have been considered to achieve interoperability in systems that need to communicate. This information is essential because it could be used to create a catalog of solutions reported in the scientific literature.

To define the search string of the literature review, we selected two keywords: *interoperability* and *software-intensive systems*. As interoperability is a well-known, disseminated term, we considered more one synonym for this word that is *integration*. We also considered other terms a synonym for *software-intensive systems*, that are: *systems of systems*, *big data*, *internet of things*, and *smart cities*. Hence, the final search string was: *TITLE ((“integration” OR “interoperability”) AND (“software-intensive systems” OR “systems of systems” OR “systems-of-systems” OR “system-of-systems”) OR “system of systems” OR “big data” OR “internet of things” OR “smart cities”))*. Table 2 shows the adapted string to the specific syntax of each publication database to perform the searches.

Concerning the publication databases, we selected those recommended in Kitchenham *et al.* (2009): Scopus¹, Web of Science², IEEE Xplore³, and ACM Digital Library⁴.

Scopus and Web of Science are general indexing systems and allow us to cover a broader scope for our search. IEEE Xplore and ACM Digital Library publish works of the

¹ <<http://scopus.com>>

² <<http://isiknowledge.com>>

³ <<https://ieeexplore.ieee.org>>

⁴ <<http://dl.acm.org>>

Table 2 – Search strings adapted for each publication database

Source	Search string
Scopus	TITLE-ABS-KEY(("integration" OR "interoperability") AND ("software-intensive systems" OR "systems of systems" OR "systems-of-systems" OR "system-of-systems" OR "system of systems" OR "big data" OR "internet of things" OR "smart cities"))
IEEE Xplore	("Abstract": integration') AND ("Abstract": integration') AND ("Abstract": software-intensive systems OR "Abstract": systems of systems OR "Abstract": systems-of-systems OR "Abstract": system-of-systems OR "Abstract": system of systems OR "Abstract": big data OR "Abstract": internet of things OR "Abstract":smart cities)
ACM Library	[[[Abstract: "integration"] AND [[Abstract: "interoperability"] OR [Abstract: "software-intensive systems"] OR [Abstract: "systems of systems"] OR [Abstract: "systems-of-systems"] OR [Abstract: "system-of-systems"] OR [Abstract: "system of systems"]OR [Abstract: "big data"]][Abstract: "internet of things"] OR [Abstract: "smart cities"]]]]
Web of Science	AB=((("integration") AND ("interoperability" OR "software-intensive systems" OR "systems of systems" OR "systems-of-systems" OR "system-of-systems" OR "system of systems" OR "big data" OR "internet of things" OR "smart cities"))

most important venues (conferences and journals) related to architectural strategies. We also defined one inclusion criterion (IC) and three exclusion criteria (EC):

1. IC1: The study proposes a strategy to address interoperability.
2. EC1: The study does not consider a strategy to address interoperability.
3. EC2: The study does not permit identifying information about the architectural strategies because it is a table of contents, short course description, invited talk of events, a summary of events, among others, or written in other languages than English.
4. EC3: The study was not peer-reviewed.

3.1.2 Conduction

This literature review was conducted from January 2019 to April 2019 by researchers with expertise in software architecture, interoperability, and software-intensive systems. Figure 3 shows the steps of the selection process of primary studies.

We adapted the search string for each database, performed the searches, and obtained 13429 studies. The number of studies returned may be significant because we kept

Figure 3 – Process for the selection of relevant primary studies

Source: Elaborated by the author.

the string wide to find several architectural strategies that could be used by systems that need to communicate. After removing the duplicated studies, we kept 11258 studies remained. After the first selection, we applied the selection criteria on title, abstract, and keywords; 254 studies were selected. After reading the full text of such studies and applying the selection criteria again, 65 studies were selected.

For the data extraction and analysis of each study, we used an online form that supported collecting the information to answer the RQs. Data from each study were extracted by two researchers involved in this literature review, and when there were doubts, discussions with other researchers were performed to solve this. Hence, the dataset generated from this form supported us in synthesizing results, answering the RQs, and further drawing conclusions on this topic.

3.1.3 Results

The complete list of studies selected to answer our RQs comprises 65 studies, and they can be consulted in⁵. Our main objective in conducting this literature review was to identify architectural strategies that could be applied in integration projects to promote interoperability in software-intensive systems. In this perspective, we establish two RQs. The first RQ is related to the types of strategies that could be considered in integration projects, and the second RQ is related to architectural strategies used to promote interoperability. In this sense, the following sections present the answers for these RQs.

⁵ <<https://github.com/pedrohduvalle/Literature-Review-on-Stratregies>>

3.1.3.1 Approaches for Interoperability

After reading the 65 studies, different approaches were identified, among them, technologies, techniques, frameworks, platforms, middleware, cloud infrastructures, architectural patterns, architectural styles, and architectural tactics. In this thesis, techniques, and architectural styles, patterns, and tactics were the architectural strategies explored. This decision was made based on the fact that technologies, frameworks, platforms, middleware, and cloud infrastructures are broader solutions that adopt the architectural strategies investigated in this thesis. These approaches are described below:

Architectural styles are packages of design decisions that explain a generic design approach for a software system (CLEMENTS *et al.*, 2011). Another definition used by Bass (2013) is a specialization of element and relation types, together with a set of constraints on how they can be used. The architectural styles contribute to a good design of architectures, in which they are defined as a coordinated set of architectural constraints that restricts the roles of architectural elements and the allowed relationships among those elements within any architecture (MESBAH; DEURSEN, 2007).

Architectural patterns are ways to capture well-proven design structures, facilitating their reuse (BUSCHMANN *et al.*, 1996; MARK, 2015). Architectural patterns provide a mechanism to capture domain experience and knowledge to allow it to be reapplied when a new architectural problem is found (PRESSMAN; MAXIM, 2014). In short, an architectural pattern can be considered as a package of design decisions that are found repeatedly in practice (HARRISON; AVGERIOU, 2007). Such patterns have known properties that permit reuse, besides that they describe a class of architectures.

Architectural tactics are design decisions that influence the achievement of a quality attribute response tactics directly affect the system's response to some stimulus (BASS, 2013). Architectural tactics are simple ideas opposed to patterns since they are described in just a few sentences, which they can be refined into a hierarchical structure of tactics. Most patterns are composed or created from several different tactics, and although these tactics might all serve a common purpose, they are often chosen to promote different quality attributes (FERNANDEZ *et al.*, 2015).

Techniques: are the use of a mechanism to implement different techniques to achieve technical interoperability requirements (REZAEI *et al.*, 2014). In this approach, the architects are concerned with specific technology patterns, practices, and tools applicable across many services (BITTNER *et al.*, 2005). In general, techniques define a set the technology standards. Examples of techniques are the communication protocols, services, standards, anthologies, among others.

In each approaches to consider interoperability in software-intensive systems, we identified a set of architectural strategies e listed each of them in the next section.

3.1.3.2 Architectural Strategies for Interoperability

This section shows the identified architectural strategies to promote interoperability among different systems that need to communicate, responding to RQ₂. Table 3 summarizes the answers to this RQ.

Table 3 – Complete list of identified architectural strategies from the literature review

Approaches	Name of Strategy
Architectural Styles	Microservices
	Service-Oriented Architecture (SOA)
	Layer
	Representational State Transfer (REST)
	File Transfer
	Shared Database
	Remote Procedure Invocation (RPI)
	Messaging
Architectural Patterns	Pipes and Filters & Pipeline
	Scatter/Gather
	Centralized Architecture
	Mediator
	Construct Monitor
	Bus
	Adapter
	Wrapper
	Broker
	Facade
Architectural Tactics	Orchestrate
	Coreography
	Discover Service
	Tailor Interface
Techniques	Business Process Execution Language (BPEL)
	Web Service Level Agreement (WSLA)
	OGSA (Migration)
	Business Process Modeling Notation (BPMN)
	XML Process Definition Language (XPDL)
	APIs
	Taxonomies
	Ontologies
	Thesaurus
	Web Services Languages
	JSON, YAML, XML
	Web Services Security (WS-Security)
	OGSA-DATA Access
	Integration (OGSA-DAI)
	Simple Object Access Protocol (SOAP)

We organized the architectural solutions according to the corresponding type of architectural strategy. This classification facilitates to have an overview of existing solu-

tions to promote interoperability in software-intensive systems. In this sense, in the next section, we show how each architectural strategy has been applied in integration projects by practitioners considering each interoperability level (i.e., technical, syntactic, semantic, and organizational).

3.2 State of the Practice

We conducted a survey with several practitioners of different countries to understand how the identified architectural strategies in the literature review have been addressed in integration projects in the industry. Surveys are used extensively by software and systems engineering organizations to provide insight into complex issues, assist with problem-solving, and support effective decision-making (KASUNIC, 2005).

A survey will allow us to generalize about the beliefs and opinions of software engineering practitioners and researchers on the use of architectural strategies for promoting interoperability in industry integration projects. Following, we present the research method used to carry out our survey and the results obtained.

3.2.1 Research Method

This section presents the research method used for the conduction of this survey that is composed of research goals and questions, survey design and execution, and data analysis. As mentioned before, software engineers still face interoperability challenges that hamper the building and operation of complex systems effectively and efficiently. In this sense, Valle *et al.* (2019) identified several strategies from the literature review that have been used to achieve interoperability in complex systems, and addressed in the software architectures of such systems.

3.2.1.1 Research Goal and Questions

The goal of this survey was to have an overall panorama about architectural strategies that have been used in industry to support four levels of interoperability, namely, technical, syntactic, semantic, and organizational. The following research questions (RQ) were to achieve this goal:

- RQ₁: Which are the architectural strategies (patterns, styles, and tactics) that have been used by practitioners to resolve interoperability issues of software-intensive systems?
- RQ₂: Which is the difficulty level to apply such strategies in integration projects?

- RQ₃: Which is the level of impact over the quality attributes at using such strategies in integration projects?
- RQ₄: How the process of integrating software systems could be improved? (e.g., in terms of, agility, simplicity, and documentation)
- RQ₅: Which are the technologies used in integration projects?
- RQ₆: Which are the main challenges perceived by practitioners during the software integration?

3.2.1.2 Survey Design and Execution

We followed the guidelines offered by (KASUNIC, 2005) to properly conduct our survey. Such guidelines define a seven-stage process, as shown in Figure 4.

Figure 4 – Process to design and conduct a survey

Source: Adapted from Kasunic (2005).

Target Audience: the target audience of our survey consisted of software industry practitioners (i.e., system analysts, architects, testers, developers, requirements engineers, or project managers) that have experience in software integration/interoperability projects. We sent a massive invitation to possible participants (volunteers) using international and national email lists (e.g., AISWORLD, CEES, SBC) and professional and social networks, such as ResearchGate, Twitter, Facebook, WhatsApp, and LinkedIn. We also sent personal invitations to industry researchers that were identified in the program committees of conferences on software architecture and interoperability (e.g., European Conference on Software Architecture, IEEE International Conference on Software Architecture, Brazilian Symposium on Software Components, Architectures, and Reuse, and International Conference on Interoperability for Enterprise Systems and Applications).

Questionnaire Structure: based on our goal and research questions, we prepared an on-line questionnaire considering the following aspects:

- *Demographic information:* we defined seven questions about participant's profiles, asking them about their current role (i.e., researcher, architect, manager project, developer, product owner, tester, network analyst, system analyst), geographical localization, application domains of their projects, size of their organization with respect of numbers of employees, organizations' type, and years of experience working in integration projects.
- *Interoperability Strategies:* we defined two questions to identify which the architectural strategies practitioners have used in practice. For this, we provided a list of 22 strategies obtained from the literature (VALLE *et al.*, 2019) that practitioners could select. Following, practitioners were asked to inform for which interoperability types each selected strategy was used. Moreover, practitioners informed the reasons for not using the selected strategies.
- *Application of interoperability strategies:* we defined two closed questions about the application of selected strategies. In the first question, practitioners should inform the difficulty level (i.e., very high, high, medium, low, very low) to apply the selected strategies. The second question was related to the perceived impact level (i.e., highly negative, lowly negative, none impact, lowly positive, highly positive) that each selected strategy could have over quality attributes of security, performance, functionality, reliability, usability, portability, and maintainability.
- *Qualitative questions:* we defined two questions about the agreement level of the participants regarding the architectural strategies. These questions were related to technologies and reference architectures they have used in integration projects.
- *Open questions:* we defined three open questions to obtain as more information as possible on the challenges that practitioners have perceived in integration projects, and the way that, from their perspective, integration processes of systems could be improved.

Consent for the Research Study: we provided a consent form to practitioners to manifest their interest in participating voluntarily in the survey. In this form, we informed that all responses would be treated as confidential. No personal information was gathered and practitioners could withdraw from the study at any time.

Questionnaire Length: the questionnaire included from 14 to 104 questions, depending on the number of architectural strategies applied by participants. The questionnaire was written in English. Due to this large number of questions and to avoid overwhelming to practitioners with irrelevant questions to their experience, we used filters to show only relevant questions for each participant. Questions regarding the difficulty level to apply strategies, the impact over quality attributes, and negative examples of their use

were asked to practitioners only for the selected architectural strategies. The final version of the questionnaire is available in [Appendix A](#)

Pilot Studies: Two pilot studies were performed to evaluate the quality of our survey. In the first study, three experts in interoperability and software architecture reviewed the questionnaire. They analyzed the relevance and clarity of the questionnaire and suggested some improvements. Some questions were re-categorized and shortened. We carried out a second pilot study with an additional three researchers with expertise in software architecture, interoperability, and software-intensive systems to analyze the understanding of the questions and estimate the time required to answer the questionnaire. For this, the experts answered the online questionnaire with questions about whether: (i) instructions are clear; (ii) questions are understandable; (iii) the order of the questions seems logical; (iv) answer choices (for closed-ended questions) are appropriate and no answer choices are overlooked; and (v) there is any difficulty navigating through the questionnaire. As a result, the feedback contributed to improving our questionnaire, in which two questions were reorganized to simplify the questionnaire. The estimated time required to answer the questionnaire was between 25 and 30 minutes.

Questionnaire Distribution Strategy: we implemented the on-line questionnaire using LimeSurvey, and it was available for three months. In this period, we sent three times the invitation to the target audience of our survey using the selected e-mail lists and professional networks, as planned.

Data Analysis: Results collected on *LimeSurvey* were automatically recorded in spreadsheets. We conducted qualitative data analysis based on bar charts, pie charts, and other types of graphics that allowed us to analyze results.

To ensure that the practitioners know the terms used in the survey, we used a resource called tooltips that supported participants to access the definition of concepts related to interoperability levels along with the entire questionnaire. We also asked participants to select only those strategies that they have used in their integration projects, i.e., that they had the knowledge and previous experience. Hence, the answers collected in our survey regarding these strategies refer to those that the practitioners know the terms.

3.2.1.3 Threats To Validity

To minimize biases of our survey, we present below the potential threats to validity and some actions performed to mitigate them.

Inconsistencies on survey protocol: two researchers validated the survey protocol aims to eliminate inconsistencies or differences among the survey's objectives, audience, questionnaire, and strategies. Also, we conduct two survey pilot with software engineering researchers with expertise in software architecture, interoperability, and complex systems

to analyze the understanding of survey questions, besides following guidelines offered by Kasunic (KASUNIC, 2005).

Selection of architectural strategies: some architectural strategies for interoperability may not have been addressed in this survey. To mitigate this, we select the identified strategies in (VALLE *et al.*, 2019), and we create an open question on the questionnaire to identify other types of architectural strategies that have been used by practitioners in their integration projects.

Amount of participants: the number of participants in our survey could be considered small, once 33 practitioners answered our survey. However, the participants of our survey have a large experience in practice and have been participated in many integration projects. Also, the participants are of different continents and they have different roles in enterprises/companies (as, architect, research, project manager, develop, product owners, among others). Thereby, even with a small number of participants, we obtained good evidence, which we report in this work.

Understanding of interoperability types: practitioners may not have understood or been familiar with the classification of types of interoperability, and consequently, they may have classified the strategies into different interoperability types. For minimizing this bias, we put a tooltip on each interoperability type for that practitioners could consult when in doubt about the definition of interoperability types.

3.2.2 Characterization of Target Audience

We received 33 responses from practitioners of different countries. We present next to a brief overview of participants' demographic characterization and we answer the RQs.

Localization: to get an overview of the characterization of the target audience, we asked practitioners the continents where they work. The result of this question is shown in Figure 5.

South America is the localization where more respondents work (45%), followed by Europe (33%), North America (12%), and Asia (9%). Although we sent the survey to practitioners through email lists and social networks around the world, we did not obtain participation from other continents (i.e., Africa, Oceania, and Antarctica). This may have occurred because we did not find people of these continents in the program committees of conferences and email lists of these continents to invite them to participate in our study.

Size and Kind of Organizations: The 45.5% (15/33) of the respondents work in extra-large organizations (with more than 1000 employees), making it possible they have experience in several integration projects. 30% (10/33) of practitioners work in organizations with small size (less than 50 employees), 15% (5/33) work in organizations with large size (between 251 and 999 employees), and 9% (3/33) work in organizations medium

Figure 5 – Continents where practitioners work

Source: Elaborated by the author.

size (between 51 and 250 employees). Respondents have worked in more than one kind of organization, such as universities (48.5%), industry research centers (18.8%), IT companies (33.3%), entertainment enterprises (3%), public agencies (3%), and government administration (3%).

Current Role of Practitioners: The practitioners that participated in our study have performed in more than one role in their organization. 61% (20/33) of practitioners identified themselves also as researchers in their companies, 27% (9/33) as architects, 15% (5/33) as project managers, 6% (2/33) as developer, and 3% (1/33) as product owner, testers, network analysts, or system analysts.

Experience in Integration Projects: To ensure the respondents were the right person to answer our survey, we defined two questions about the experience of the respondents. The first question asked them about the years of experience in integration projects. The second question asked them about the number of integration projects they had already participated in.

As shown in Figure 6, 85% (28/33) had more than two years of experience, from which 55% (18/33) accumulated more than eight years participating in projects for software integration and leading with interoperability issues.

The Figure 7 illustrates that 78% (26/33) had participated in at least two integration projects, from which 42% (14/33) had worked in at least five integration projects. We consider the years of experience and number of integration projects in which the respondents took part were enough to ensure that the respondents had accumulated important

Figure 6 – Experience of practitioners in years.

Source: Elaborated by the author.

experience with system integration.

Figure 7 – Experience of practitioners on software integration projects.

Source: Elaborated by the author.

Application Domains: Surveyed practitioners had participated in projects for integrating software systems in 28 different application domains. The most representative domains were healthcare and smart cities, both with 15% (5/33) of practitioners working on them. Other representative domains were manufacture, governmental, IoT, automation technology, and education.

3.2.3 RQ₁: Strategies used by practitioners in integration projects

Practitioners were asked to indicate whether, in their integration projects, they had (or not) already used the 22 architectural strategies presented in the first column of Table 4.

Such strategies were identified after analyzing related works (VALLE *et al.*, 2019; FERNANDES *et al.*, 2018; VARGAS *et al.*, 2016). For each architectural strategy, practitioners chose the type of interoperability (i.e., technical, syntactic, semantic, and organizational) in which the strategy was used.

Table 4 lists the surveyed architectural strategies, amount of practitioners that selected each strategy, the frequency that practitioners had utilized a strategy for achieving each interoperability type.

Table 4 – Frequency of answers on the type of interoperability at using each strategy from the practitioner’s perspective. Responses that obtained at least 50% of the practitioners’ opinions for each strategy are highlighted.

Strategies	Amount of Practitioners that used the strategy	Types of Interoperability			
		Technical	Syntactic	Semantic	Organizational
REST	16	16	6	5	3
Microservices	16	11	6	3	8
SOA	15	15	8	5	4
Messaging	14	14	9	8	8
Adapter	14	14	10	11	2
Shared DataBase	13	13	5	4	3
Layer	13	13	6	2	3
Pipes and Filters	13	13	4	1	3
Wrapper	12	12	6	9	5
RPI ⁶	12	12	5	4	3
Mediator	12	12	5	8	3
File Transfer	11	11	4	5	7
Centralized Architecture	10	10	4	4	5
Bus	10	10	5	6	5
Orchestration	9	9	5	4	7
Broker	9	9	8	9	6
Facade	9	9	7	5	2
Discover Service	8	8	5	3	1
Contract Monitor	8	7	3	2	1
Tailor Interface	6	5	3	3	0
Choreography	5	4	5	1	4
Scatter/Gather	3	3	2	1	1

Source: Elaborated by the author.

Based on information from Table 4, all 22 strategies for interoperability, reported in the literature, have been used at least by one practitioner. Moreover, differently from what literature reports, e.g., in (VALLE *et al.*, 2019), the results of this survey evidence the difficulty to categorize a set of strategies for one specific interoperability level. For instance, it seems that any architectural strategy could consider interoperability requirements at any level, excluding the *tailor interface* strategy that was not used for organizational interoperability by any practitioner.

From the 22 architectural strategies for interoperability, only the 36.4% (8/22) of them had been used by at least 40% (13/33) of practitioners that answered our survey. The most used strategies are *REST*, *Microservices*, *SOA*, *Messaging*, *Adapter*, *Shared*

Database, *Layer*, and *Pipes & Filters*, which were separated in Table 4. The majority of this set of strategies are architectural styles. Only two architectural patterns, i.e., *Adapter* and *Pipes & Filters*, were among the most used strategies in practice. Architectural tactics are the strategy less used among practitioners since less than 27.3% (9/33) of them have adopted tactics in integration projects.

The eight most used strategies have been applied for achieving interoperability requirements in the four levels, namely, technical, syntactic, semantic, and organizational. The popularity of *REST*, *Microservices*, and *SOA* among practitioners can be explained by the existence of several mature technologies for creating interoperable service-oriented systems, such as those tools and technologies offered by cloud providers (e.g., Amazon, Google Cloud, IBM).

We highlighted the frequency of answers that obtained at least 50% of positive responses from practitioners that used a determined strategy. For instance, from the 16 practitioners that applied microservices in their projects, the 68.7% (11/16) intended to tackle technical interoperability requirements, and 50% (8/16) of them were oriented to achieve organizational interoperability requirements.

As displayed in Table 4, it seems that all strategies could be used for overcoming technical interoperability issues, since practically 100% of practitioners that used each strategy agreed it allowed technical interoperability. This can justify that practitioners can be more familiar with the concept of technical interoperability than the other three levels. Perhaps, a clear differentiation between different interoperability levels is lacking among practitioners.

From Table 4 is possible to observe that, from the set of the most used strategies (8/22), the major proportion of practitioners related that *REST*, *Shared Database*, *Layers*, and *Pipes & Filters* were only oriented to support technical interoperability, whilst the remaining strategies support additional purposes. For syntactic interoperability, using *SOA*, *Messaging*, and *Adapter* could be a good option. Semantic interoperability can be realized through messaging and adapter. At the organizational level, *Microservices* and *Messaging* could be a good strategy. We also observed that *Messaging*, *Bus*, and *Broker* were the most frequent strategies used to mitigate issues in all interoperability levels, as reported by at least 50% of practitioners that applied them in integration projects.

3.2.4 RQ₂: Difficulty at Using the Architectural Strategies

For each architectural strategy used by practitioners, they were asked to estimate the difficulty level for its implementation, independently from any interoperability type. This information could help the architects to select strategies based on effort estimation. For

each strategy, the practitioner selected one of the following options of difficulty levels: very low, low, medium, high, and very high. The results of this are presented in Table 5. Numbers indicate the number of practitioners who notified the difficulty level for applying each strategy. Highlighted numbers denote that at least 50% of practitioners who used each strategy selected the specific difficulty level.

Table 5 – Frequency of difficulty levels at using each strategy from the practitioner’s perspective.

Strategies	Amount of Practitioners that used the strategy	Difficulty Level				
		Very High	High	Medium	Low	Very Low
REST	16	0	0	9	4	3
Microservices	16	0	3	10	1	2
SOA	15	0	0	12	3	0
Messaging	14	0	5	3	5	1
Adapter	14	1	1	8	2	2
Shared DataBase Layer	13	2	0	7	1	3
Pipes and Filters	13	0	2	6	5	0
	13	0	0	5	4	4
Wrapper	12	0	2	8	0	2
RPI	12	0	2	7	0	3
Mediator	12	0	0	6	3	3
File Transfer	11	1	1	6	2	1
Centralized Architecture	10	1	2	4	2	1
Bus	10	0	3	3	2	2
Orchestration	9	0	1	4	0	4
Broker	9	0	2	7	0	0
Facade	9	0	0	2	6	1
Discover Service	8	0	1	3	2	2
Contract Monitor	8	1	1	4	0	2
Tailor Interface	6	0	0	3	2	1
Choreography	5	0	1	4	0	0
Scatter/Gather	3	0	0	1	1	1

Source: Elaborated by the author.

From Table 5, practitioners estimated that 63.6% (14/22) of strategies is required medium or low effort to realize them in integration projects. For the remaining 36.4% (8/22) of strategies, it was not a consensus between practitioners about the difficulty level of using these solutions.

Moreover, when practitioners answered a difficulty level as high or very high, they were asked to provide the reason for this estimation. Following, we summarize the main challenges that practitioners have faced when used architectural strategies for interoperability. The quoted text presents the exact practitioner’s comments about such challenges.

Difficulties at using Microservices: The main challenge of using this style is dealing with the granularity level of microservices. The greater a microservice is, the deeper the level of details is required. Another problem is related to the testability of microservices, since their implementation is complex, which creates very difficult testing challenges. Finally, some participants reported a possible reason that makes it tricky to use

this strategy in integration projects is the lack of practitioners trained to use microservices.

Difficulties at using Contract Monitor: According to practitioners, one possible challenge of this architectural pattern is related to the complexity of defining service's behavior and synchronization with other services, especially when working at organizational interoperability level.

Difficulties at using Bus: Although this pattern is a good idea for achieving interoperability at all levels, many practitioners said that there is a great challenge for this, due to the need to understand the Bus' API and the need to manage the Bus' infrastructure. Therefore, there is a tendency to prefer point-to-point connections between applications, instead of going through the Bus.

Difficulties at using Orchestration: The main challenge before applying this tactic is the need to have a good definition of the business processes to correctly execute orchestration.

Difficulties at using Choreography: In general, this tactic must be predetermined before its execution. This requires systems or services to be integrated to change their rules for executing a correct choreography.

Difficulties at using Layers: Normally, it is complex to have a good definition of layers, which requires a good specification of each layer scope. Besides this, multiple layer issues are difficult to handle at the same time.

Difficulties at using Messaging: The application of this style depends on network availability and requires previous knowledge about using and configuring MOM (Message Oriented Middleware) systems. Additionally, several developers must work with functions that are executed synchronously, at the same time that, distributed coordination with message-based systems are executed asynchronously.

Difficulties at using Adapter: Some challenges when using adapters are related to their: (i) reusability, since sometimes, it necessary to create the own adapters to integrate systems; (ii) configuration, because sometimes, adapters are tricky to parameterize; (iii) hard customization, since sometimes, the lack of functionality that the system would require; (iv) cost, since in general, they are very expensive; and (v) there is often a high manual effort for developing an adapter.

Difficulties at using Wrapper: The principal challenge of this pattern is the high manual effort required for its development and implementation.

Difficulties at using Broker: Applications that interact with each other via the Broker could take an additional development effort (when compared to point-to-point integration), and requires specific knowledge/training required for its development. Besides, the learning curve stiff for the development of the Broker.

Difficulties at using Centralized Architecture: The application of this architectural pattern requires political buy-in from the stakeholders of data owners who want proprietary solutions to their data needs, besides that time to market of the application may be an issue. In general, application with centralized architecture has a monolithic structure.

Difficulties at using Remote Procedure Invocation (RPI): Based on the practitioner's opinion, RPI is a style that is not commonly used in modern integration projects, since middleware tool vendors may use these capabilities within their platform, but the level of abstraction is now much higher than RPI. This is happening because RPI requires that interfaces are defined at a very low level of granularity. To use RPI at that level is intolerable for performance and reliability reasons. Besides, the need to change level and provide synchronization with larger chunks over RPI is too burdensome, and interoperable adoption could fail.

Difficulties at using File Transfer: The correct use of this style depends on the data structure and the possibility of alteration. This strategy also depends on scheduled processes, and often the files are not available (because it still being generated) when the scheduled task is executed. Hence, it could generate low availability levels of the integrated system.

Difficulties at using Discover Service: The main challenge of using this tactic is the lack of advanced tools for service discovering. For instance, the use of UDDI (Universal Discovery Description and Integration) or ebXML Registry / Repository never had good results to me, so the tooling is weak.

Difficulties at using Share Database: This style is useful when very low-level integration is required. In general, software systems are built by different companies and those who integrate them must know the data structure. Besides, any change in integrated software has an impact on the integration process. Therefore, independence is fundamental.

We did not obtain evidence from practitioners on the difficulties to apply the strategies SOA, REST, Mediator, Facade, Pipes and Filters, Tailor Interface, and Scatter/Gather in integration projects. This may have happened because we asked the reasons for the difficulties level only for the strategies that were classified as difficulty level very high and high. Therefore, we consider that difficulties of applying such strategies need to be better investigated in future work.

3.2.5 RQ₃: Impact Over the Quality Attributes

Practitioners were also questioned the way that strategies used to tackle interoperability levels in their integration projects have impacted the quality attributes. For this analysis, we considered seven quality attributes proposed by the product quality model defined in ISO/IEC 25010 (ISO, 2011a), namely, security, performance, functionality, reliability, usability, portability, and maintainability. Not all practitioners agreed to participate in this question. The second column of Table 6 presents the number of practitioners that responded to this survey's query.

Each practitioner indicated the way that strategies impact quality attributes at selecting one of the following options: positive impact (P), negative impact (N), I do not know (IK), or none impact (NI). Table 6 shows the number of practitioners that selected an impact level option for the architectural strategies used in their projects. Highlighted numbers refer to the impact tendency of each strategy over a specific quality attribute. Being more specific, for each quality attribute, we highlighted the impact chosen by at least 50% of practitioners that used a specific strategy. For instance, from the practitioner's perspective, in addition to interoperability, *REST* also positively contributes to quality attributes as performance, functionality, reliability, portability, and maintainability. Since few practitioners reported negative impacts of *REST* over attributes, it is not possible to conclude that this style could negatively affect, e.g., security.

We observed that practitioners coincided with possible impacts for the 95.5% (21/22) of strategies, i.e., strategies for which an impact tendency was found and highlighted in Table 6. A tendency for the *Scatter/Gatter* strategy was not possible to observe.

Table 6 – Practitioners impact perception of architectural strategies over quality attributes. Answers (P=Positive Impact, N=Negative Impact, IK=I Don't Know, NI = None Impact). Responses that obtained at least 50% of the practitioners' opinions for each strategy are highlighted

		Quality Attributes																											
		Security				Performance				Functionality				Reliability				Usability				Portability				Maintainability			
Strategies	Amount of Practitioners	P	N	IK	NI	P	N	IK	NI	P	N	IK	NI	P	N	IK	NI	P	N	IK	NI	P	N	IK	NI	P	N	IK	NI
REST	16	4	5	4	3	9	5	1	1	11	1	1	3	10	1	3	2	9	2	3	2	11	0	3	2	11	1	3	1
Microservices	15	6	6	1	2	6	5	2	2	7	2	1	5	9	5	0	1	8	2	2	3	9	3	2	1	12	3	0	0
SOA	15	7	4	1	3	4	10	0	1	11	1	0	3	7	4	1	3	11	1	1	2	13	2	0	0	11	3	0	1
Messaging	9	2	3	2	2	4	3	1	1	3	2	1	4	2	2	2	3	3	2	3	1	3	2	2	2	5	4	0	0
Adapter	9	1	2	3	3	1	2	3	3	1	3	2	3	1	1	3	4	0	2	2	5	2	2	2	3	2	2	2	3
Shared DataBase Layer	10	5	2	1	2	7	2	0	1	9	0	0	1	5	2	0	3	6	3	1	0	4	2	2	2	6	2	1	1
Pipes and Filters	13	9	0	2	2	6	3	2	2	8	0	2	3	7	2	2	4	7	2	4	0	9	2	2	0	8	4	0	1
Wrapper	8	0	1	3	4	2	1	1	4	2	0	2	4	4	0	1	3	2	1	2	3	4	0	1	3	4	0	2	2
RPI	8	1	1	3	3	0	4	2	2	1	2	2	3	1	2	2	3	2	0	3	3	2	2	3	1	2	1	2	3
Mediator	11	4	5	0	2	7	3	0	1	6	0	0	5	4	6	0	1	1	5	0	5	3	6	0	2	3	8	0	0
File Transfer	8	1	2	2	3	2	1	3	2	4	1	2	1	4	0	3	1	4	0	3	1	3	1	3	1	3	1	2	2
Centralized Architecture	11	5	4	0	2	8	2	1	0	5	4	2	0	6	3	2	0	8	2	1	0	10	1	0	0	8	2	0	1
Bus	9	4	1	2	2	4	3	1	1	4	0	1	4	3	1	2	3	5	0	1	3	3	3	1	2	5	0	2	2
Orchestration	8	3	1	1	3	1	5	1	1	2	1	1	4	6	1	1	0	3	0	3	2	4	2	1	1	4	2	1	1
Broker	6	0	1	3	2	1	2	2	1	1	1	2	2	1	0	2	3	1	0	3	2	0	2	3	1	1	1	3	1
Facade	6	2	1	1	2	0	5	1	0	2	1	1	2	2	2	1	1	0	0	2	4	2	1	2	1	3	2	1	0
Discover Service	3	1	0	1	1	0	0	1	2	2	0	1	0	1	1	1	0	1	1	1	0	0	0	1	2	1	0	1	1
Tailor Interface	5	1	0	1	3	0	1	1	3	0	2	1	2	1	0	1	3	1	0	1	3	1	0	1	3	1	0	1	3
Choreography	3	0	1	1	1	0	1	1	1	0	0	1	2	0	1	1	1	0	0	1	2	0	0	1	2	0	0	1	2
Scatter/Gather	5	0	0	3	2	0	2	3	0	1	0	3	1	0	1	4	0	1	0	4	0	0	2	3	0	0	1	4	0
	2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1

Source: Elaborated by the author.

Mostly, practitioners tended to report more positive than negative impacts. For instance, practitioners reported a negative impact over quality attributes of only 22.7% (5/22) of strategies, namely, *SOA*, *Wrapper*, *RPI*, *Bus*, and *Broker*. From them, practitioners considered *RPI* as a strategy that implies more trade-offs over quality attributes comparing with other solutions. Despite *RPI* seems to support interoperability, performance, and functionality, it could have negative effects on reliability, portability, and maintainability.

REST, *Layers*, *Shared Database*, and *File Transfer* were the strategies that, besides interoperability, seem to have positive impacts on at least the other five quality attributes. Moreover, practitioners did not report significant negative impacts of these three strategies over any of the listed quality attributes.

Regarding tactics for interoperability (i.e., *Tailor Interface*, *Discovery Service*, *Orchestration*, and *Choreography*) we observed that practitioners did not offer enough information to establish a trend on their impact (positive or negative) over any quality attributes. Perhaps, tactics trade-offs are not so well-known by practitioners as trade-offs of architectural styles and patterns, as supported by recent research works (MOGHADDAM *et al.*, 2018).

Performance was the quality attribute that tended to be more negatively impacted by the use of architectural strategies, namely *SOA*, *Wrapper*, *Bus*, and *Broker*. Since these strategies are used in service-oriented approaches, the trade-offs perception among these strategies and performance were aligned with academic literature (JOSUTTIS, 2007).

3.2.6 RQ₄: Challenges for Interoperability

Interoperability is considered a wide challenge for the development and evolution of complex systems, as SoS, IoT, systems based on microservices, smart cities, among others (MOTTA, 2016). Aiming to identify the main challenges for interoperability of software architecture of software-intensive systems, we carried out a survey with 33 practitioners from the whole world who had experience in project integration and software architecture (VALLE *et al.*, 2021a). From the practitioners' experience, we wanted to know some challenges they have faced in their projects and that could be considered as gaps for the state-of-the-practice/art.

1. To support technology evolution in a short time-frame. Currently, new technologies are adopted in a short time frame, and consequently, their suppliers also changed. This scenario makes it difficult the communicate among systems that adopt such technologies.
2. To perform a preliminary analysis of data that needs to be exchanged among the

organization's systems, before acquiring new technologies or discontinuing the technologies already deployed in the organization.

3. To ensure that other quality attributes (e.g., performance, scalability, minute ability, reliability) are not affected by the scenarios required data/systems integration.
4. To ensure the exchange of data among systems that wish to interoperate when such systems have monolithic architectures.
5. To govern different integration types aiming always the integrity preservation of systems' architecture. It requires also some alignment with the development process being used by the company.
6. To not extrapolate costs of allocation of human and financial resources in integration projects. Sometimes, the allocation of human resources can involve conflicts of interest since one party of stakeholders can need to adapt to a technology that is imposed in the project context.
7. To overcome the lack of documentation, proven solutions, and guidelines to support the implementation of interoperability in the integration projects context.
8. To avoid unwanted dependencies in integration projects. For instance, dependencies with legacy systems or with obsolete technology.
9. To improve the understanding of the meaning of data and linkages of data via metadata. Misunderstanding of these data leads to integration problems.

3.2.7 RQ₅: Improvement of Integration Process

Still, regarding the survey that we conducted to obtain an overview of architectural strategies for interoperability (VALLE *et al.*, 2021a), we asked practitioners the way that the process of integrating software systems could be improved. We obtained 14 answers to this question. This section summarizes several improvements suggested by practitioners to support the integration process. We created eight categories of recommendations based on practitioners' responses, which are presented as follows. Quoted text exactly presents respondents' suggestions.

Agile methodologies: The 35.7% (5/14) of practitioners consider that integration processes must be followed by agile approaches. The highly changing conditions of business must be supported by organizational interoperability strategies, making possible the development of software-intensive systems incrementally and iteratively. Moreover, business dynamism also requires to adapt technical systems information sharing and collaboration. Additionally, to consider interoperability strategies that allow continuous changes in the system, practitioners also consider that the integration process must be

done following agile methodologies. The system integration must be done incrementally and iteratively, and it is necessary to take into account evaluations (at the end of each iteration) with the participation of various types of stakeholders.

Standardized APIs were suggested by 21.4% (3/14) of practitioners. They consider the use of APIs very important to support interoperability since they make it possible that applications communicate with each other without users' knowledge or intervention. However, from their perspective, the state of practice lacks well-designed APIs exposed using REST or vendor-neutral protocols, standards-based APIs, or infrastructures that support loose coupling and clean simple interfaces.

Abstract architectural solutions, were proposed by 14.3% (2/14) of practitioners. They highlighted the importance to create *light* reference architectures or sets of integration patterns that allow practitioners to understand the way that each software systems exchange data among them. These solutions could simplify the integration process due to knowledge about the standards that must be considered for such systems to interoperate among them. However, considering that organizations are looking at time-to-market all the time, maybe some (light) automation is required.

Enough documentation requirement, was pointed out by 14.3% (2/14) of practitioners. They believe that, in some projects, having good documentation of the required integration is an enabler to understand which systems (or software elements), data, processes, strategies must be considered for the software integration.

The remaining practitioners proposed the use of **ontologies** and business processes to understand the meaning of exchanged data among software systems, **simulation tools** to evaluate integrated systems and find out possible problems/errors as early as possible, and **artificial intelligence** to help understand the semantic of exchanged data among systems and the coordination of distributed workflows in business processes.

3.2.8 RQ₆: Technologies for Integration Projects

To identify means used in practice to promote interoperability, in our survey, we asked practitioners about the technologies they have used in their integration projects. To answer this question, we obtained 20 answers and classified such answers into five categories of technologies, namely, middleware, share database, bus, programming languages, and machine learning, and are described as follows (VALLE *et al.*, 2021a).

The 30% (6/20) of practitioners affirmed have used **middleware**, i.e., a type of software that offers runtime services for communications, integration application execution, monitoring, and operations. Some technologies listed by them are Microsoft environments, WebLogic server, Webservices (i.e., a solution used in systems integration and communication among different applications), Message-oriented middleware (MOM), and

remote procedure calls (e.g., custom SSH system-to-system tunnels).

Shared database was listed by the 20% (4/20) of practitioners, who used database technologies as Oracle or MySQL. Moreover, some practitioners have to make use of file-sharing protocols (e.g., sftp, scp). They commented that many times, the information that should be exchanged is stored their data in a single shared database.

Some **bus** technologies were mentioned by 20% (4/20) of practitioners, who have utilized the IBM Service Bus or Apache MQ (Message Queuing). Usually, a bus provides an abstraction of layers in the implementation of a messaging system, which allows integration of the architecture to exploit the value of messages without writing code.

Programming languages as C, C++, C#, Java, and Python, and the XML Schema⁷ were suggested by the 25% (5/20) of practitioners.

Finally, one practitioner used **machine learning** techniques to support their integration projects. He/She proposed that the use of reinforcement learning is a technique that enables an agent to learn in an interactive environment by trial and error using feedback from its actions and experiences.

3.3 Mining Association Rules

The information provided by practitioners in the survey was used to give insights on how these architectural strategies could be combined to consider interoperability in integration projects. For this, association rules that are usually applied to discover elements that occur in common within a specific dataset were used. This technique is commonly applied in data mining and machine learning. Our dataset is composed of the practitioner's responses about the use of architectural strategies in practice. It is important to highlight that these association rules are suggestions and are not considered valid for all possible situations.

The association rule mining emerged in the context of supermarket applications, in which rules are generated as "*a customer who buys products $a_1, a_2, \dots, a_n$ will also buy the product a_m with a probability of $c\%$* " (AGRAWAL *et al.*, 1993). However, association rules are not restricted only in the context of supermarkets, since they have been applied in different domains to solve a wide range of problems (Nuwangi *et al.*, 2010; Choi; Kim, 2014; JADHAV; DESHPANDE, 2016; Wu *et al.*, 2016).

The goal of association rules is identifying relationships among the items of a specific dataset, with the purpose of discovering implicit knowledge in the dataset, which normally would not be discovered by manual means of exploration (AGGARWAL, 2015). In general, the number of association rules generated can exceed the capacity for hu-

⁷ A language based on an XML (Extensible Markup Language) format for defining validation rules ("schemas") in documents in XML format

man understanding, making the process of association rules mining very complex. It is necessary to use means to interpret them, in a more intuitive way (PANDEY *et al.*, 2009).

In general, association rules are widely used in the scientific literature to discover correlations among items on a given dataset. The application of association pattern mining is defined in unordered set-wise data. It is assumed that the dataset T contains a set of n transactions, denoted by $T_1 \dots T_n$. Each transaction T_i is drawn on the universe of items U and can also be represented as a multidimensional record of dimensionality, $d = |U|$, containing only binary attributes (AGGARWAL, 2015). An itemset is a set of items. A k -itemset is an itemset that contains exactly k items. In other words, a k -itemset is a set of items of cardinality k .

The association rules are defined on the format $LHS \Rightarrow RHS$, where LHS stands for the left-hand side and RHS for the right-hand side, both of them contain a set of items such as $LHS \cap RHS = \emptyset$ (CALCADA, 2019). An association rule is only generated whether it has the minimum values of confidence and support that has been pre-established (AGGARWAL, 2015; PADUA, 2019). The *support* shows the percentage of transactions that contain the items that are being analyzed, as presented in Equation 3.1, in which $T(LHS)$ is the number of the transaction containing T and $T(\text{dataset})$ is the total number of transactions. The *confidence* shows the number of transactions that RHS occurs given that LHS has occurred. Such a metric can be calculated using the Equation 3.2. If the rule has a confidence value greater than the minimum confidence that was previously defined, then this rule is presented to the user; otherwise, it is removed.

$$Support(LHS) = \frac{T(LHS)}{T(\text{dataset})} \quad (3.1)$$

$$Confidence(LHS \Rightarrow RHS) = \frac{Support(LHS \cup RHS)}{Support(LHS)} \quad (3.2)$$

Apriori is the best-known algorithm to obtain the association rules. This algorithm finds all sets of frequent items, namely frequent itemsets (L_k). *Apriori* uses a depth-first search and generates a set of candidate items (patterns) of k elements and eliminates patterns that are not frequent. The dataset is completely tracked and the sets of frequent items are obtained from the sets of candidate items (AGGARWAL, 2015). The algorithm uses two functions, one to generate the candidate items and eliminate those that are not frequent, and the other to extract the association rules (CALCADA, 2019).

To generate the association rules, the *Apriori* algorithm was used and it was performed three steps to obtain the association rules for our context as presented following:

3.3.1 Overview of the Process for Generating Association Rules

The association rules mining process is divided into three phases, namely: preprocessing, extraction of association rules, and post-processing. Figure 8 shows an overview of the the association rules mining process in the context of this thesis.

Figure 8 – Overview of the process for association rules mining

Source: Elaborated by the author.

Following, we present the way that each phase of the association rules mining was tackled in the context of this thesis.

Phase 1 - Preprocessing: the first phase consists of preparing the dataset for the extraction of the association rules. In this phase, we need to make our dataset suitable for the association rules algorithm. In our case, we considered the *Apriori* algorithm, which expects our data to be in the form of transactions.

Our original dataset consisted of the architectural strategies related to the interoperability level that the practitioners have used in practice in their integration projects. To transform this dataset into transactions, we concatenated the interoperability level with the architectural strategies mentioned by practitioners. For example, if a participant indicated the Adapter architectural strategy for Semantic interoperability, we considered the term *Semantic_Adapter*. This was done for each architectural strategy considered by the practitioner.

As a result of this phase, for each participant, we have a list of such terms that were used as input to the *Apriori* algorithm.

Phase 2 - Extraction of Association Rules: the second phase consists of the extraction of association rules, by analyzing the input domain and by making associations among items. We must define the minimum *support* and minimum *confidence*. In our context, we generated the association rules relating to the different strategies and interoperability level with minimum *support* of 15% and minimum *confidence* 80%.

We chose these values because we observed that they generated a good number of rules to analyze, in other words, the association rules with less than 15% support and association rules with less than 80% confidence would not occur as often, which would not be interesting for our analysis. For example, the rule *Semantic_Broker, Syntactic_Adapter* $\Rightarrow$ *Technical_Messaging* indicated that when we have Broker on the Semantic level and Adapter on the Syntactic level there is a high probability of having Messaging on the Technical level.

Phase 3 - Post-Processing: the third phase consists of selecting the most relevant association rules since the *Apriori* algorithm generates a high number of rules, most of them being irrelevant to our domain.

A list of all the strategies considered in the association rules was created, and as a result, eight architectural strategies were selected: Adapter, Broker, Messaging, Pipes & Filters, Wrapper, Share Database, REST, and Microservices. In the next section, we present the results of this analysis.

3.3.2 Association Rules

The rules were obtained after of carried out the process of mining association rules for our context. The generated association rules can be used to select a minimum set of architectural strategies.

For this, it were defined that the association rules should be generated with minimum *support* of 12% and minimum *confidence* of 80%. The *support* is related to the probability of association rules being present in the dataset, and the *confidence* is related to the probability of *LHS* $\Rightarrow$ *RHS* existing in the dataset. Table 7 shows the the list complete of association rules generated from the dataset using the *Apriori* algorithm.

Table 7 – List complete of association rules generated for supporting the selection of architectural strategies

ID	Association Rules	Confidence	Support
1	Semantic_Adapter, Technical_Adapter $\Rightarrow$ Syntactic_Adapter	1.000	0.152
2	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_Messaging	1.000	0.152
3	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_Pipes and Filters	1.000	0.152
4	Semantic_Adapter, Syntactic_Adapter $\Rightarrow$ Technical_Adapter	0.833	0.152
5	Semantic_Broker, Technical_Messaging $\Rightarrow$ Syntactic_Adapter	0.833	0.152
6	Syntactic_Messaging, Technical_Share Database $\Rightarrow$ Semantic_Broker	0.833	0.152
7	Semantic_Broker, Syntactic_Messaging $\Rightarrow$ Technical_Share Database	0.833	0.152
8	Semantic_Wrapper, Syntactic_Adapter $\Rightarrow$ Technical_Messaging	0.833	0.152
9	Semantic_Wrapper, Syntactic_Adapter $\Rightarrow$ Technical_REST	0.833	0.152
10	Organizational_Broker, Semantic_Messaging $\Rightarrow$ Technical_Pipes and Filters	1.000	0.121
11	Organizational_Broker, Semantic_Wrapper $\Rightarrow$ Technical_Pipes and Filters	1.000	0.121
12	Syntactic_Bus, Technical_SOA $\Rightarrow$ Organizational_Microservices	1.000	0.121
13	Organizational_Microservices, Technical_SOA $\Rightarrow$ Syntactic_Bus	1.000	0.121
14	Organizational_Microservices, Syntactic_Bus $\Rightarrow$ Technical_SOA	1.000	0.121
15	Semantic_Adapter, Technical_Wrapper $\Rightarrow$ Syntactic_Adapter	1.000	0.121
16	Syntactic_REST, Technical_Share Database $\Rightarrow$ Semantic_Adapter	1.000	0.121
17	Semantic_Broker, Technical_Bus $\Rightarrow$ Syntactic_Adapter	1.000	0.121
18	Semantic_Broker, Technical_REST $\Rightarrow$ Syntactic_Adapter	1.000	0.121
19	Semantic_Broker, Syntactic_Broker $\Rightarrow$ Technical_File Transfer	1.000	0.121
20	Semantic_Broker, Syntactic_Broker $\Rightarrow$ Technical_Messaging	1.000	0.121

21	Syntactic_Share Database, Technical_Share Database $\Rightarrow$ Semantic_Broker	1.000	0.121
22	Semantic_Broker, Syntactic_Share Database $\Rightarrow$ Technical_Share Database	1.000	0.121
23	Semantic_Wrapper, Syntactic_Broker $\Rightarrow$ Technical_Messaging	1.000	0.121
24	Syntactic_Broker, Technical_REST $\Rightarrow$ Semantic_Wrapper	1.000	0.121
25	Semantic_Wrapper, Syntactic_Broker $\Rightarrow$ Technical_REST	1.000	0.121
26	Semantic_Wrapper, Syntactic_Broker $\Rightarrow$ Technical_SOA	1.000	0.121
27	Syntactic_REST, Technical_Layer $\Rightarrow$ Semantic_Wrapper	1.000	0.121
28	Syntactic_REST, Technical_REST $\Rightarrow$ Semantic_Wrapper	1.000	0.121
29	Semantic_Messaging, Technical_Pipes and Filters $\Rightarrow$ Organizational_Broker	0.800	0.121
30	Organizational_Broker, Technical_Pipes and Filters $\Rightarrow$ Semantic_Messaging	0.800	0.121
31	Organizational_Broker, Technical_Pipes and Filters $\Rightarrow$ Semantic_Wrapper	0.800	0.121
32	Syntactic_Bus, Organizational_Microservices $\Rightarrow$ Technical_SOA	0.800	0.121
33	Semantic_Adapter, Technical_REST $\Rightarrow$ Syntactic_Adapter	0.800	0.121
34	Semantic_Adapter, Technical_Remote Procedure Invocation $\Rightarrow$ Syntactic_Adapter	0.800	0.121
35	Semantic_Adapter, Syntactic_Broker $\Rightarrow$ Technical_Messaging	0.800	0.121
36	Semantic_Adapter, Syntactic_Broker $\Rightarrow$ Technical_SOA	0.800	0.121
37	Syntactic_REST, Technical_Messaging $\Rightarrow$ Semantic_Adapter	0.800	0.121
38	Syntactic_REST, Technical_SOA $\Rightarrow$ Semantic_Adapter	0.800	0.121
39	Semantic_Adapter, Technical_Share Database $\Rightarrow$ Syntactic_REST	0.800	0.121
40	Semantic_Broker, Technical_Adapter $\Rightarrow$ Syntactic_Adapter	0.800	0.121
41	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_Adapter	0.800	0.121
42	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_Bus	0.800	0.121

43	Syntactic_Adapter, Technical_File Transfer $\Rightarrow$ Semantic_Broker	0.800	0.121
44	Semantic_Broker, Technical_File Transfer $\Rightarrow$ Syntactic_Adapter	0.800	0.121
45	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_File Transfer	0.800	0.121
46	Semantic_Broker, Technical_Layer $\Rightarrow$ Syntactic_Adapter	0.800	0.121
47	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_Layer	0.800	0.121
48	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_REST	0.800	0.121
49	Semantic_Broker, Technical_SOA $\Rightarrow$ Syntactic_Adapter	0.800	0.121
50	Semantic_Broker, Syntactic_Adapter $\Rightarrow$ Technical_SOA	0.800	0.121
51	Semantic_Broker, Technical_File Transfer $\Rightarrow$ Syntactic_Broker	0.800	0.121
52	Syntactic_Messaging, Technical_Pipes and Filters $\Rightarrow$ Semantic_Broker	0.800	0.121
53	Semantic_Messaging, Syntactic_Messaging $\Rightarrow$ Technical_Share Database	0.800	0.121
54	Semantic_Wrapper, Technical_Adapter $\Rightarrow$ Syntactic_Adapter	0.800	0.121
55	Syntactic_Messaging, Technical_REST $\Rightarrow$ Semantic_Wrapper	0.800	0.121
56	Semantic_Wrapper, Syntactic_REST $\Rightarrow$ Technical_Layer	0.800	0.121
57	Syntactic_REST, Technical_Messaging $\Rightarrow$ Semantic_Wrapper	0.800	0.121
58	Semantic_Wrapper, Syntactic_REST $\Rightarrow$ Technical_Messaging	0.800	0.121
59	Semantic_Wrapper, Syntactic_REST $\Rightarrow$ Technical_REST	0.800	0.121
60	Syntactic_REST, Technical_SOA $\Rightarrow$ Semantic_Wrapper	0.800	0.121
61	Semantic_Wrapper, Syntactic_REST $\Rightarrow$ Technical_SOA	0.800	0.121
62	Syntactic_SOA, Technical_Remote Procedure Invocation $\Rightarrow$ Semantic_Wrapper	0.800	0.121
63	Semantic_Wrapper, Syntactic_SOA $\Rightarrow$ Technical_Remote Procedure Invocation	0.800	0.121
64	Semantic_Wrapper, Syntactic_SOA $\Rightarrow$ Technical_SOA	0.800	0.121

To select the strategies, we selected all strategies with minimum *support* of 15% and minimum *confidence* of 80% that were considered by the association rules, analyzing for which interoperability requirements they could be used. As a result, we selected eight strategies as suggestions for architects to consider in their integration projects. Such strategies are Adapter, Broker, Messaging, Pipes & Filters, Share Database, REST, Wrapper, and Microservices. The original set of strategies is composed of 22 different strategies. Therefore, after applying the association rules mining in our database, we reduced our set to approximately 63%.

The complete list of selected strategies is presented in Table 8, as well as which interoperability levels can be tackled by such strategies.

Table 8 – Architectural strategies suggested for each interoperability level

Patterns	Technical	Syntactic	Semantic	Organizational
Adapter	✓	✓	✓	
Broker			✓	✓
Messaging	✓	✓		
Pipes and Filters	✓			
Share Database	✓			
Wrapper			✓	
REST	✓			
Microservices				✓

Source: Elaborated by the author.

Based on the practitioners' answers, it is worth highlighting that there are strategies that are suggested to be applied to only one interoperability level (e.g., Microservices for organizational interoperability according to our association rules). Despite this, other architectural patterns can be applied to achieve more than one interoperability level (e.g., Adapter that could be used for three interoperability levels, namely: technical, syntactic, and semantic).

The application of association rules mining provided an overview of the main strategies used in practice and provided suggestions for their applications according to the data obtained from the survey carried out with practitioners. To conduct manually this analysis could hamper the process of discovering strategies on how practitioners have been applied these strategies together in integration projects. Therefore, association rules provided an insight into which are the main architectural strategies that have been used and the way that they relate to each other. It is worth noticing that although we have obtained these results systematically, we also consulted experts to validate the extracted evidence.

3.4 Final Considerations

Regarding the results obtained in the survey, most strategies were selected by practitioners to solve issues at all interoperability levels, contradicting the tendency of academics, which have proposed different solutions for each type of interoperability. This may have occurred because (possibly) there is a lack of common understanding of practitioners about the different concepts surrounding interoperability. Consequently, practitioners may be selected as the strategy for different interoperability levels.

Another interesting finding is regarding the perceived difficulties levels for using the strategies. For practitioners, most strategies have low levels of difficulty (e.g., medium, low, and very low). This may have occurred because, in general, the great number of practitioners who answered our survey have more than six years of experience and have already worked in more than five integration projects. We also noted that those strategies that have technologies and infrastructures as service (i.e., SOA, REST, microservices) facilitated their use in integration projects, especially in domains as IoT, smart cities, and big data. Such strategies are the most used by practitioners.

We found some confirmation of the impacts that architectural strategies have over quality attributes. Literature reports that Layers positively impact security, maintainability, reliability, and portability (HARRISON; AVGERIOU, 2007). Among practitioners that already used Layers in their projects, 69.2% (9/13) of them confirmed that this strategy improves security, while 61.5% (8/13) said this for maintainability, 69.2% (9/13) for portability (9/13), and 53.8% reported that Layers positive impact for reliability.

A disagreement was found regarding usability, since 53.8% (7/13) of practitioners said that Layers improve this attribute, however, the literature (HARRISON; AVGERIOU, 2007) shown that this strategy is neutral for usability. In the same perspective, we found some disagreements between the information of literature and practitioners' opinions on how *Pipes and Filters* impact the quality attributes. The literature reports that security and reliability are negatively impacted (HARRISON; AVGERIOU, 2007).

Practitioners reported that this strategy has none impact on security (4/8) and positively impacts reliability (4/8). The results of this study confirmed that portability and maintainability are positively impacted by Pipes and Filters. Regarding the *Broker*, practitioners confirm that this strategy contributes to maintainability (3/6) as reported in the literature. On the other hand, the literature reports that the Broker improves usability, but practitioners said that this strategy has not impact usability (4/6).

The literature also reports that some listed strategies are specific for one interoperability level (VALLE *et al.*, 2019; VARGAS *et al.*, 2016; FERNANDES *et al.*, 2018). However, the evidence obtained in this survey reports that practitioners have used one same strategy for different interoperability levels, in contrast to what is shown in the

literature. This happened regardless of the strategy type considered (i.e., architectural pattern, style, and tactic) by practitioners. For instance, the literature reports that REST has been used to support technical interoperability, but practitioners related that they have used REST to achieve the four interoperability types. This is similar to SOA, Messaging, and Microservices where the literature reports that such strategies are used to promote technical interoperability, but in practice, we observed that these strategies have applied to achieve the four interoperability types.

The architectural tactics reported in the literature for organizational interoperability (i.e., orchestration and choreography) have also been applied to lower levels of interoperability, as technical and syntactic. This scenario can be explained because theoretical studies may not consider new technologies that implement these strategies and facilitate their application by practitioners in current integration projects. Therefore, it is important to update the existing contributions according to the practitioners' perspective regarding the difficulty in applying the strategies and trade-offs over quality attributes.

We also identify important issues perceived by practitioners that must be overcome in next years, as well as suggestions to tackle difficulties of integration projects. For instance, the lack of a good documentation and guides to allow the understanding on how a system must be integrated could be supported by the creation of light reference architectures (including well-defined patterns and standardized APIs) that could be easily adopted in organizations' agile environments.

Therefore, survey results could be used by practitioners to choose candidate architectural strategies based on the desired interoperability level. They can analyze the difficulty of applying each strategy and how each of these strategies could positively or negatively impact the quality attributes considered important for their architectures. Thus, they can select or reject an architectural strategy for their integration project.

TASIS: A Typology of Architectural Strategies for Interoperability of Software-Intensive Systems

This chapter introduces a set of architectures strategies for supporting achieving interoperability in integration projects. The strategies were organized in an typology, named **TASIS** (Typology of Architectural Strategies for Interoperability of Software-Intensive Systems), which is categorized into four classes and four types of strategies (i.e., techniques, patterns, styles, and architectural tactics). The evidence obtained in the survey reported in Chapter 3 was used to consolidate TASIS from the industry perspective. This information was obtained from 33 practitioners that had experience in software architecture and interoperability.

4.1 Initial Version of TASIS

A typology is considered a conceptual classification of a problem, and must represent relationships among different concepts types found in the problem domain. A typology is restricted to a system of conceptually derived groupings (BAILEY, 1994; DOTY; GLICK, 1994). In this thesis, requirements are categorized into four classes, namely: technical, syntactic, semantic, and organizational interoperability. For each requirement class, four types of strategies were associated, i.e., technologies, architectural patterns, styles, and tactics. Following, we present the methods for building the typology and its description.

4.1.1 Methods

To create TASIS, the guidelines proposed by Bailey (1994) were followed.

Step 1: The first step to create the typology was the selection of *types of interoperability*. To this, a search for interoperability classifications was made using scientific data libraries, such as Scopus and Google Scholar. These search engines were selected since they index most of the scientific research made in the computer science area. The string used for this purpose was *TITLE ((“integration” OR “interoperability”) AND (“classification” OR “taxonomy” OR “typology” OR “characterization”))*. As a result, 11 studies were selected because of they present some type of classification of interoperability types, namely DoD (1998), Morris *et al.* (2004), Chen *et al.* (2008), Kubicek *et al.* (2011), Benson and Grieve (2016), Li. *et al.* (2013), Neiva *et al.* (2016), Wegner (1996), Vargas *et al.* (2016), Blair *et al.* (2011).

Based on these studies, the following classes of interoperability were identified: programmatic, constructive, operational, conceptual, technological, organizational, semi-otic, syntactic, semantic, pragmatic, technology, data, human, institutional, technical, data, infrastructure, application, procedures, service, process, and business. After a careful analysis, they were grouped in more representative classes, since some of them are synonyms. Therefore, the classification proposed by Kubicek *et al.* (2011) was selected to group them. This classification defines four types of interoperability, namely technical, syntactic, semantic, and organizational.

Step 2: To identify approaches that tackle each type of interoperability, a substruction strategy was followed, as suggested by Bailey (1994). Such strategy consists of finding most *dimensions* for a single *type of concepts* in the typology. In this work, *dimensions* are referred to as strategies for accomplishing interoperability, and *types of concepts* are related to the four interoperability types defined in Step 1. For this, a new search on scientific databases (i.e., ACM Digital Library, IEEE Xplore, Scopus, and Web of Science) was made to investigate approaches applied for interoperability in contemporary software projects, e.g., Software-Intensive Systems, SoS, IoT, big data. The search string used for this search was:

TITLE ((“integration” OR “interoperability”) AND (“software-intensive systems” OR “systems of systems” OR “systems-of-systems” OR “system-of-systems” OR “system of systems” OR “big data” OR “internet of things” OR “smart cities”))

As a result, 13429 studies were initially obtained. After removing duplicated studies, a set of 11258 was under analysis. From these, only 65 studies reported at least one strategy for interoperability. All studies can be consulted in the GitHub repository¹. After reading the 65 studies, different approaches were identified, among them, technologies, techniques, frameworks, platforms, middleware, cloud infrastructures, architectural patterns, architectural styles, and architectural tactics. In this work, techniques, architectural styles, patterns, and tactics were the architectural strategies explored. This decision was

¹ <<https://github.com/pedrohduvalle/Literature-Review-on-Strategies>>

made based on the fact that technologies, frameworks, platforms, middleware, and cloud infrastructures are broader solutions that adopt the architectural strategies investigated in this work.

Step 3: The relation among interoperability types and strategies was made. This mapping was based on evidence reported by the 65 studies about which kind of interoperability requirement a specific strategy was applied. This classification was carried out by two researchers, which discussed and agreed on strategies categorization for each type of interoperability requirement. As a result, the typology with four types of concepts and four dimensions was constructed, which is explained in Section 4.2 and depicted in Table 9.

4.1.2 Threats to Validity

In order to minimize biases of this study, the potential threats to validity and some actions carried out to mitigate those threats are discussed below:

- **Missing important strategies:** Aiming to include all architectural strategies for interoperability, a systematic search in scientific data libraries and patterns repositories was carried out. Besides, backward and forward snowballing was conducted using the references and the citations of the initial list of papers returned by the systematic search. Despite this, it is possible we did not identify some architectural solutions for interoperability, henceforth they might have been omitted in the typology proposed in this paper.
- **Appropriateness of strategies:** Architectural strategies presented in the typology were selected based on their applicability to resolve interoperability requirements. Selection was made supported by evidence presented in the analyzed studies and through additional consults in pattern repositories. Therefore, the typology only presents relevant strategies to address interoperability requirements at technical, semantic, syntactic, and organizational levels.
- **Incorrect understanding of strategies:** To ensure a correct interpretation of architectural strategies, several examples of their use were consulted, analyzed by two researchers, showing how each solution can be applied at each level of interoperability.
- **Scientific databases:** The architectural strategies were identified using scientific databases, which could limit the amount of studies to be analyzed, and hence, the number of identified strategies. To mitigate this threat, pattern repositories were used to found all possible solutions that can be used to approach interoperability in

Table 9 – Typology of architectural strategies for interoperability

Interoperability Level	Categories	Strategies
Organizational	Architectural Styles	Microservices Service-Oriented Architecture (SOA) Layer
	Architectural Patterns	Pipes and Filters & Pipeline Scatter/Gather Centralized Architecture Mediator Contract Monitor Bus
	Architectural Tactics	Orchestrate Choreography
	Techniques	Business Process Execution Language (BPEL) Web Service Level Agreement (WSLA) Business Process Modeling Notation (BPMN) XML Process Definition Language (XDL) APIs
Semantic	Architectural Styles	Layer Service-Oriented Architecture (SOA) Microservices
	Architectural Patterns	Adapter Wrapper Bus
	Architectural Tactics	Discover Service Tailor Interface
	Techniques	Taxonomies Ontologies Thesaurus
Syntactic	Architectural Styles	Microservices Service-Oriented Architecture (SOA) Layer
	Architectural Patterns	Pipes and Filters Wrapper Facade Adapter
	Architectural Styles	Discover Service Tailor Interface
	Techniques	Business Process Execution Language (BPEL) Web Service Languages JSON, YAML, XML
Technical	Architectural Styles	Representational State Transfer (REST) File Transfer Shared Database Remote Procedure Invocation Messaging
	Techniques	Web Services Security (WS-Security) OGSA-Data Access Integration (OGSA-DAI) Simple Object Access Protocol (SOAP)

Source: Adapted from [Valle et al. \(2019\)](#).

software architectures. Therefore, these repositories were used as a complementary mechanisms to increase the searching scope of our solution space.

4.2 Architectural Strategies for Interoperability

Table 9 presents TESIS. At each interoperability level (i.e., technical, syntactic, semantic, and organizational), strategies were divided by the category that their address, i.e., technique, or architectural styles, patterns, and tactics.

In the remainder of this section, each architectural strategy in each category of interoperability is presented. Specifically, for each strategy, it is offered its general description, as well as an example that shows how the strategy contributes to achieving the interoperability level for which it was classified.

4.2.1 Technical Interoperability

Technical interoperability is related to transmitting data between components or individual systems independently of their distance. This type of interoperability is domain-independent and the meaning of data exchanged is not known by systems exchanging the data (BENSON; GRIEVE, 2016). Therefore, technical interoperability only guarantees the correct transmission of bits, but the meaning of these bits is unknown (KUBICEK *et al.*, 2011). Thus, technical interoperability is focused on communication protocols and the infrastructure needed for those protocols to operate (VEER; WILES, 2008).

Technical interoperability is the first interoperability requirement that must be tackled in an integration project (REZAEI *et al.*, 2014; KUBICEK *et al.*, 2011). For supporting technical interoperability requirements, four different architectural styles were included in TESIS, namely *File Transfer*, *Shared Database*, *Remote Procedure Invocation*, and *Messaging* (REPOSITORIO, 2021). To exemplify the use of those architectural styles, consider multiple applications, built independently, with different languages and platforms, and it is required those applications to work together exchanging diverse information. A possible solution for this requirement is the use of the *File Transfer* architectural style, in which each application produces files containing the information required by the other application. An implementation of this style is FTP (File Transfer Protocol) used by operational systems, like Linux.

Another solution is the *Messaging* architectural style, which can be used to transfer packets of data frequently, immediately, reliably, and asynchronously, using customizable formats (e.g., as those specified in the TCP - Transmission Control Protocol), enabling communication among the different systems. The *Shared Database* is another style for information exchange, and it suggests that applications are integrated to store their data in a common shared database or repository. This strategy is one of the most common in

big data systems, where information produced by different entities are homogenized and stored in a central repository, and after consulted for visualization or decision-making purposed. For the same requirement, *Remote Procedure Invocation* architectural style suggests that each application is developed as a large-scale object or component with encapsulated data. For implementing this strategy, an interface should be provided to allow other applications to interact with the running application (REPOSITORIO, 2021). Moreover, *REST* (*REpresentational State Transfer*) is an architectural style widely used in Web-based systems, such as web services and microservices. The correct implementation of the *REST* style is denominated RESTful.

Some of these styles have been implemented in different techniques to achieve technical interoperability requirements. Among them, *OGSA-Data Access* and *Integration* (*OGSA-DAI*) and were identified which orient the way that data can be stored, accessed, and transferred (REZAEI *et al.*, 2014). Besides, *SOAP* (*Simple Object Access Protocol*) is a communication protocol mainly used in service-oriented systems, and allows information exchanging at applying RPC.

4.2.2 Syntactic Interoperability

Syntactic interoperability is related to providing data formats, well-defined syntax, and encoding (such as message content structure, size of headers, size of the message body, fields contained into a message) (KUBICEK *et al.*, 2011; GRACE *et al.*, 2012). As shown in Table 9, solutions of all types (e.g., techniques, architectural tactics, styles, and patterns) were identified.

Regarding architectural styles, *SOA* (*Service-Oriented Architecture*), *Microservices*, and *Layers* have been used to achieve syntactic interoperability requirements (KUBICEK *et al.*, 2011; NEWMAN, 2015; REPOSITORIO, 2021). *SOA* has been widely used to facilitate the publication of capabilities offered by distributed systems in form of services with well-defined interfaces. Interesting examples about using *SOA* for interoperability are provided in (GARLAN *et al.*, 2010). *Microservices* has risen as a sub-style of *SOA* (ZIMMERMANN, 2017) for overcoming challenges related to interoperability among services. Such challenges were mainly associated with interface specifications and continuous integration. Examples of using microservices for interoperability can be consulted in (NEWMAN, 2015). *Layers* are one of the most used architectural styles for organizing software systems and are used to separate the type of information embedded in message packages. An application of layers for semantic interoperability is the TCP/IP packet format.

Five different architectural patterns to achieve requirements of syntactic interoperability were identified. The *Broker* architectural pattern has been used for the coordination of communication among components (REPOSITORIO, 2021). For example, servers pub-

lish their capabilities (services and characteristics) to a *Broker*, clients request a service from the broker, and then they redirect the clients to a suitable service from their registry (KESHAV; GAMBLE, 1998). In a complementary way, *Pipes & Filters* architectural patterns could be used (KESHAV; GAMBLE, 1998). An example of using this pattern can be a sequence of message queues that can be used to provide the infrastructure required to implement a pipeline. An initial message queue receives unprocessed messages. A component implemented as a filter task listens for a message on this queue, performs its work, and then posts the transformed message to the next queue in the sequence (MICROSOFT, 2021).

The *Wrapper* architectural pattern can also support syntactic interoperability (KESHAV; GAMBLE, 1998). For example, to achieve integration and interoperation, a *Wrapper* encodes information in a specific message format to be understood by other participants (GARCÉS, 2018). Wrappers can be used to aggregate new data types in a message, for facilitating the processing of such a message by peers. Another architectural pattern included in TESIS is *Facade*. Such pattern hides the complexity of a system and provides a unified interface for a system with multiples components. The unified interface defines an only entry point for clients that would use all functionalities of the system (KESHAV; GAMBLE, 1998). The *Adapter* architectural pattern also can be used to assist the syntactic interoperability among several systems (KESHAV; GAMBLE, 1998). Adapters can be used for transforming data formats, vocabularies, and protocols used by different systems in a common standard. In web systems, adapters are a fundamental strategy to transform messages formats, e.g., from XML to JSON message specification.

Two architectural tactics that can contribute to achieving this interoperability requirement were selected in TESIS. The tactic of *Discover Service* could be used when different systems that interoperate must be verified and integrated at runtime. For this, this tactic locates a service provider for a required capability, through searching in a known services directory. In this sense, services can be deployed as sets of capabilities that are accessible through some kind of interface (BASS, 2013). In this same perspective, the *Tailor Interface* architectural tactic could be used for syntactic interoperability. This tactic adds or removes capabilities to an interface. An example of its application is to hide functionalities from untrusted users or to add new permissions to clients over a service (BASS, 2013).

Different techniques were also selected for supporting syntactic interoperability. The *Business Process Execution Language (BPEL)* could be used for enabling easy and flexible composition of services. Moreover, web service languages for the implementation of services (REZAEI *et al.*, 2014), namely: *JSON*, *YAML*, *XML*, are also techniques that guide the way that a message can be formatted to standardize its interpretation by external systems or components.

4.2.3 Semantic Interoperability

Semantic interoperability is related to data that are conceived as information to be shared, processed, and without ambiguity by systems. In other words, this interoperability level is related to the meaning of the exchanged data as clear definitions of terminology or vocabulary (DABC, 2004). It is important to highlight that this interoperability level is specific to the domain and context and requires the use of unambiguous information and identifiers (BENSON; GRIEVE, 2016).

As depicted in Table 9, the architectural styles *SOA*, *Layer*, and *Microservices* also can be used to support semantic interoperability requirements, as reported in (REPOSIT6RIO..., 2019; KUBICEK *et al.*, 2011; NEWMAN, 2015; REPOSITORIO, 2021). The use of these styles has allowed the dissemination of the semantic web 3.0, where services can share well-structured and understood information to homogenize, link, and generate new knowledge, for instance in the case of answer-question services systems (Jang *et al.*, 2007).

Four architectural patterns were selected for TASIS. The *Adapter* architectural pattern allows semantic interoperability between systems exchanging data (KESHAV; GAMBLE, 1998). This type of interoperability is done through the translation or mapping between different data formats. For example, it is possible to adapt *X* into *Y*, where the data type *X* is different from data type *Y*, using a transformation rule *Z* (GARCÉS *et al.*, 2018). The *Wrapper* architectural pattern also can be used for this purpose (KESHAV; GAMBLE, 1998), since it adds extra information to data exchanged. For instance, in hospital systems, to construct the whole clinical record of a patient, a wrapper can add information obtained from multiple devices sensing the patient's vital signs. The *Enterprise Service Bus (ESB)* is another architectural pattern that could be adopted for achieving semantic interoperability. An *ESB* establishes connectivity protocols, offers capabilities for data and protocol transformation, and message routing (ARSANJANI *et al.*, 2007).

Three techniques have been used to help the building of systems that require semantic interoperability, namely *Thesaurus*, *Taxonomies*, and *Ontologies*. An example of the thesaurus is the MetaNet that is a metadata term thesaurus that provides the additional semantic knowledge that is non-existent within declarative XML-encoded metadata descriptions (HUNTER, 2001). An example of *Taxonomy* the one established to support interoperability in the internet of things, which helps the understanding of the meaning of the data that are exchanged among the different systems (NOURA *et al.*, 2019). The *Ontologies* also can be used to help in information exchange among systems, since *Ontologies* support the specification of the semantics for each terminology used by different systems in an unambiguous fashion (BITTNER *et al.*, 2005).

4.2.4 Organizational Interoperability

Organizational interoperability is related to the coordination of distributed workflows and activities that are understood by systems, organizations, or people interacting in business processes. In this case, business processes can be seen as the interaction of multiple organizations to obtain their respective business goals (CHOPRA, 2008). Organizations that participate in a business process must commit to performing several activities, and the commitments are specified in contracts agreed with other organizations that are taking part in the business process (BENSON; GRIEVE, 2016).

SOA has been widely the preferred architectural style for organizational interoperability. *SOA* (and its sub-style, microservices) allows for the common description of inter-organizational processes when business process definition languages are standardized. Some example of languages is web services defined in *WSDL* (*Web Services Definition Language*) or *BPML* (*Business Process Modeling Language*) (KUBICEK *et al.*, 2011). Additional evidence was identified in the literature about the use of *Layer* and *Microservices* architectural styles to also assist in organizational interoperability (REPOSIT6RIO..., 2019; KUBICEK *et al.*, 2011; NEWMAN, 2015; REPOSITARIO, 2021).

Several architectural patterns also have been used to help in organizational interoperability (KESHAV; GAMBLE, 1998). In organizational interoperability context, an example of using the *Pipes % Filters* architectural pattern is when a system just requires to receive messages from its peers, but no request is sent to its systems, for instance in an IoT based system responsible for monitoring environmental conditions in a smart building. The *Mediator* architectural pattern could intercede interactions among distributed systems. For example, a *Mediator* has a gate used to make available specifications about its interface, as protocols and possible agreements. Middlewares, platforms, and even the cloud serve as an example of capabilities that mediators could implement.

The *Scatter/Gather* is a conversation pattern that helps the systems to interoperate among them. To exemplify this, consider an order processing, where each order item that is not currently in stock could be supplied by one of the multiple external suppliers. The suppliers may or may not have the respective item in stock themselves. For this, it should be requested quotes from all suppliers and decide which one provides the best term for the requested item. A problem is maintaining the overall message flow when a message needs to be sent to multiple recipients and each of which may send a reply. To mitigate this, the architectural pattern suggests that use a *Scatter/Gather* that broadcasts a message to multiple recipients and re-aggregates the responses back into a single message. This pattern routes a request message to several recipients. It then uses an aggregator to collect the responses and distill them into a single response message (ONLINE, 2021a).

The *Centralized Architecture* pattern provides a hub that acts as a central point of

control and is responsible for ensuring correct communication among systems (INGRAM *et al.*, 2015). An example of this architectural pattern is an anti-guerrilla operation SoS that has a central Theatre Command system, and different systems including UAV scouts; artillery; troops; and communication infrastructures. The central makes operational decisions based upon data sourced from other systems. Therefore, the goals of the SoS are achieved due to the commands of the central which accepts responsibility for delivering SoS functions (HALL-MAY; KELLY, 2006).

The *Contract Monitor* pattern should monitoring strategies which indicate precisely how and when a given contract may interact with a component/system (INGRAM *et al.*, 2015). An example of a *Contract Monitor* in the SoS context is a health monitor observing the interface of a specific system only, to detect contract violations of this system (HUANG *et al.*, 2014). The service *Bus* pattern can be used to achieve interoperability. For designing and implementing the interaction and communication between mutually interacting software applications in a *SOA* is used. An example is to use the *Bus* within an organization, or enterprise to manage and govern the use of services within that organization (JOSUTTIS, 2007).

Orchestration and *Choreography* are two architectural tactics that have been used as strategies to achieve organizational interoperability (BASS, 2013). *Orchestration* is a tactic, mainly used in *SOA* and *Microservices*, that uses a control means to coordinate and manages the invocation of particular services that could be ignorant of each other (BASS, 2013). An example of the use of this tactic is workflows engines. The *Mediator* pattern can be used for defining a simple orchestration. The *Orchestration* can be specified using the language BPEL (NEWMAN, 2015). In the same perspective, *Choreography* describes a collection of services that work together to achieve a common and unique goal described by the different web services in interactions with their uses (BENANY; BEQQALI, 2018). An example is to use the *Choreography* for proposing an architecture based on the business process transformation centralized technique using graphical representation *BPEL* (NEWMAN, 2015).

Several techniques have been found to support organizational interoperability. In this interoperability level, *Web Service Level Agreement (WSLA)* can be used to specify a commitment between a service provider and a client. The *OGSA (Open Grid Services Architecture)* can be employed for service level attainment and migration. The *Business Process Modeling Notation (BPMN)* can be applied for understandable and interpretable business process descriptions. The *XML Process Definition Language (XPDL)* can be applied for process modeling and workflow (MICHENI *et al.*, 2014). Moreover, in *SOA*, services can expose their capabilities through the use of APIs (Application Programming Interface). Therefore, *APIs* are a strategy for collaboration and communication of distributed services (ONLINE, 2021b).

4.3 Consolidation of TESIS

TESIS was consolidated using the information obtained in the survey reported in chapter 3. This information was provided by practitioners that indicated the architectural strategies that they have considered in their integration projects for each type of interoperability (technical, syntactic, semantic, and organizational). The information obtained from the industry was used to update and validate TESIS. Figure 9 illustrates the consolidated TESIS.

In the consolidated version of TESIS, we kept the four classes (that is, the four types of interoperability) and we removed technologies from the types of strategies considered. Only three types of strategies were considered, that are: architectural patterns, styles, and tactics. Technologies were removed due to TESIS considers only strategies proven in the literature and that can be reused in integration projects without relying on technical knowledge about technologies. In this sense, these architectural strategies were prioritized to obtain evidence from the industry perspective.

As illustrated in Figure 9, the architectural strategies were reorganized according to four types of interoperability based on the use of such strategies in integration projects as reported by practitioners. In the initial version of TESIS, the identification and categorization of strategies were based on scientific studies available in the database. In this perspective, we considered in the initial version of TESIS, strategies used or cited at least once in studies and/or repositories of architectural patterns and styles. Therefore, we had no evidence about the frequency with which each strategy has been reported in the scientific literature. To consolidate and update TESIS using information from the industry, we performed the four steps following:

- **Step 1:** strategies most used by practitioners for each interoperability requirements were ranked. Thus, it was possible to verify which are the most common strategies for achieving each type of interoperability in integration projects.
- **Step 2:** from the available set (22 architectural strategies), the main strategies used in practice for each type of interoperability were selected. As a criterion, only the strategies that were used by at least 10 % of practitioners were considered.
- **Step 3:** the information on a mind map showing which strategies were categorized into the three types of strategies for each type of interoperability was plotted. Besides, it was added among parentheses in front of each strategy the way that many practitioners indicated that they have used such strategies in their integration projects.

Figure 9 – The TASIS consolidated and updated according to information from the industry

Source: Elaborated by the author.

- **Step 4:** the letter “I” was added among parentheses in front of each strategy to show that such evidence is provided from the industry. Then, the initial version of TESIS was analyzed to check which were strategies considered for each type of interoperability and strategy. Next, this information was added in the new version together with the letter “L” among parentheses in front of strategies to show that such evidence is provided by the literature.

In this sense, TESIS was consolidated and validated with information from the scientific literature and also from the industry perspective. TESIS provides an overview of the possible solutions to achieve interoperability at the architectural level, mainly for the context of software-intensive systems. Therefore, software architects could also use TESIS to support address interoperability in their integration projects, once they could have a catalog with possible solutions that are proven in practice, and that provide information on the frequency of their use.

4.4 Discussion

The strategies presented in the typology in Figure 9 are proven solutions and have been used by software architects and engineers to support the achieving of interoperability requirements on different levels. TESIS provides an overview of existing strategies that can be used during the building of software-intensive systems to understand and guide how the interoperability could be addressed. Based on typologies theory (BAILEY, 1994), it is expected TESIS supports practitioners works at selecting the strategies for their integration projects.

In this sense, TESIS contributes to solving some of the interoperability challenges identified by Abukwaik and Rombach (2017), such as: the misunderstanding about data structures, procedures, and other interface settings of systems to be integrated, and the lack of a high-level architecture to guide how interoperability requirements could be achieved. In the first case, architects could use the strategies grouped in the level of syntactic interoperability for message formatting. Regarding the misunderstanding of data semantic, the practitioners could use the adapter pattern (grouped in the level of semantic interoperability) to translate the information contained in a message to be interpreted by another system. Regarding the interfaces structures provided by individual systems, practitioners can use the tailor interface tactic to add or remove capabilities to an interface, so it can modify published capabilities according to specific requirements.

For the second interoperability challenge, TESIS can help practitioners to select among architectural styles and patterns that have been used for different types of interoperability. Specifically, SOA, microservices, and layers can be considered high-level architectures guiding the structure of the system to be integrated, independently of the

desired interoperability level. Moreover, architectural patterns related in TASIS are proven solutions that can be used for specific types of interoperability requirements. In particular, architectural patterns are well-proven solutions for recurring issues in software architecture.

These solutions assist software architectures to understand the impact of their decisions on multiples quality attributes requirements due to they contain information about benefits, consequences, and context of their use. TASIS is composed of eleven architectural patterns that could contribute to achieving interoperability on different levels. Hence, software architects could use TASIS to select an architectural pattern according to the interoperability issue faced, selecting the architectural pattern that best solves their problem. Finally, architectural tactics can be used as a refinement to construct a well-structured solution by practitioners, since they can be as atomic solutions for very specific interoperability requirements.

The technical interoperability is related, mainly, to messaging exchanges among different systems, in other words, the correct transmission of bits. Few architectural strategies were identified to technical interoperability, due to this kind of requirements can be addressed using well-known techniques and styles highly promoted in webservices.

In this perspective, it was possible to show that TASIS can be used by software architects to identify and understand different solutions for achieving interoperability in different levels, furthermore TASIS contributes to overcome the “*absence of high-level architectures to analyze, comprehend, and guide how interoperability can be addressed in integration projects*” (ABUKWAIK; ROMBACH, 2017). Moreover, TASIS is not exclusive to software-intensive systems and it can be used for any architect to design any class of systems requiring to inter operate and exchange information between them.

4.5 Final Considerations

Architectural synthesis is the activity, within the software architecting process, in which the architect must focus on selecting the most feasible strategies to achieve quality attributes requirements, e.g., security, performance, reliability, safety, and interoperability. In order to contribute to the software-intensive systems architecting process, TASIS was proposed to organize the existing and proven architectural strategies for achieving interoperability requirements. It was composed of 22 different architectural strategies from evidence found in the scientific literature.

TASIS was divided into three groups (namely, architectural tactics, styles, and patterns) for each type of interoperability requirement. In this sense, the main contribution of TASIS is supporting architects to identify and understand solutions for achieving interoperability requirements during the composition of software-intensive systems.

To consolidate TESIS, we carried out a survey with 33 practitioners from several countries with expertise in interoperability and software architectures. The procedures to create the typology were carried out systematically to minimize the possible bias of this research. Besides, this research was conducted using information from the industry, gathering practical evidence and an overview of the way that interoperability has been addressed in integration projects.

A Catalog of Architectural Patterns to Solve Interoperability Issues in Software-Intensive Systems

This chapter provides a catalog of architectural patterns to address of interoperability issues that have been occurred in the context of software-intensive systems. Such issues are organized according to the interoperability level (technical, syntactic, semantic, or organizational) that they impact. For each identified issue, we present the architectural patterns can be considered to solve them. In this sense, we provided a mapping among identified interoperability issues and the architectural patterns to solve them reported in our survey results. Hence, the knowledge presented in this chapter can be used by architects to mitigate interoperability issues in software-intensive systems.

This chapter is structured as follows: Section 5.1 presents the research method performed to propose the catalog. Section 5.2 describes the interoperability issues for the technical level and a set of architectural patterns that can be used to solve these issues. Section 5.3 presents the interoperability issues for the syntactic level and some architectural patterns that can be applied to solve such issues. Section 5.4 outlines the interoperability issues for the semantic level and a set of architectural patterns that can be used to mitigate these issues. Section 5.5 describes the interoperability issues for the organizational level and a set of architectural patterns that can be used to solve these issues. Section 5.6 presents the discussion and the lessons learned about the conduction of the thesis and how to promote interoperability in software-intensive systems. Section 5.7 presents the final considerations of this chapter.

5.1 Research Method

We defined a research process to establish the catalog of interoperability issues and their possible architectural solutions for software-intensive systems. Figure 10 presents the overview of this process. The process was divided into five steps and can be applied interactively. The steps are briefly described as follows:

Figure 10 – Process to establish the catalog of interoperability architectural solutions

Source: Elaborated by the author.

- **Step 1 - Identification of architectural strategies:** usually, architectural strategies are reusable solutions to common issues in software architecture in a given context. These solutions are documented according to the experience and practice of practitioners or experts. Therefore, the starting point for identifying architectural strategies was to perform searches in publications available in the literature and conduct interviews with experts. In our context, we conducted a literature review search in several databases and pattern repositories. As a result, we identified 22 strategies for interoperability at the architectural level, as presented in Chapter 3.
- **Step 2 - Survey with practitioners:** to understand how the identified strategies in the literature review have been used in practice, we conducted a survey with practitioners from different countries to understand how these strategies have been applied in their integration projects. As a result, we received 33 responses from

participants that help us understand how these strategies could be applied at each interoperability level.

- **Step 3 - Prioritization of interoperability issues:** based on the evidence collected, we prioritized the main interoperability issues at each level reported both in the literature and in the industry. Therefore, in this doctoral thesis, we selected 11 different issues that the software architects have faced to exchange data correctly.
- **Step 4 - Writing of interoperability issues:** for each selected interoperability issue, we describe them showing the main challenges faced by architects and developers in the industry. This step also contributes to research on interoperability because it provides an overview at the architectural level on the main challenges for correct data exchange among heterogeneous systems.
- **Step 5 - Establishment of the catalog:** finally, we mapped among issues and solutions, showing which are the architectural patterns that can mitigate or even solve such interoperability issues. In this context, the practitioners can analyze which architectural pattern they have to consider in their projects.

5.2 Technical Interoperability

Technical interoperability is related to the ability of two or more applications to accept data from each other and perform a given task appropriately and satisfactorily without extra operator intervention (EHGI, 2017). In other words, technical interoperability is related to the hardware and software components, networks, and equipment that enable machine-to-machine communication, including aspects as open interfaces, connectivity, data integration, accessibility, and security issues (VEER; WILES, 2008; IBRAHIM; HASSAN, 2010).

5.2.1 Interoperability Issues

This section describes three main interoperability issues for the technical level that the architects have been faced in their integration projects.

1. **Peer-to-peer communication:** the heterogeneous systems that need to exchange information among them have been faced with the difficulty of performing peer-to-peer communication (CHEN, 2018). It has contributed to a low level of interoperability between the systems, mainly due to incompatibilities of metadata and interfaces of the operations used in the systems' communications (CHAINHO *et al.*, 2017). Therefore, mechanisms that provide ways for systems that do not have peer-to-peer

communication are required for heterogeneous systems to achieve interoperability at the technical level.

2. **Lack of infrastructure:** the absence of infrastructure can difficulty heterogeneous systems to achieve interoperability, and consequently, and make the exchange of information among systems unfeasible (EHGI, 2017). In short, the infrastructure is the ability of hardware acquired by different organizations to work in a connected way. It is associated with the hardware and software components, networks, and equipment that enable peer-to-peer communication, including open interfaces, connectivity, data integration, middleware, and data exchange (VEER; WILES, 2008; IBRAHIM; HASSAN, 2010).
3. **Different communication protocols:** in the software-intensive systems context, the individual systems can use different protocols to communicate with others software systems (NOURA *et al.*, 2019). Protocols are rules that govern the syntax of communication among different systems. In other words, they are a convention that controls and enables such communication (IBRAHIM; HASSAN, 2010; FAROOQ *et al.*, 2020). They can be implemented by hardware, software, or a combination of the two. Implementing a mechanism that standardizes the protocols of systems can allow, through a single interface, uniform access to a diversity of heterogeneous information sources synchronously and transparently for the end-user (PANG *et al.*, 2015).

5.2.2 Architectural Patterns

In this section, we suggest architectural patterns to solve or mitigate the identified interoperability issue at the technical level. The suggested architectural solutions are based on the information provided in the scientific literature and the survey we conducted (VALLE *et al.*, 2019; VALLE *et al.*, 2021a). Table 10 presents the mapping between issues and solutions.

The **peer-to-peer communication** is a recurring issue when technical interoperability is required for heterogeneous systems to interoperate. Software architects and practitioners have related by surveys and studies reported in the scientific literature that architectural solutions can be considered in their integration projects to solve/mitigate these issues.

Shared Database pattern can be used to solve peer-to-peer communication issues. This pattern makes it possible heterogeneous systems are integrated, storing their data in a common shared database/repository, to then they are consulted by other systems that need to interoperate (GUO *et al.*, 2011). This solution is one of the most common in big

Table 10 – Architectural patterns to solve interoperability issues at the technical level

Issues	Patterns	Description	References
Peer-to-Peer Communication	Share Database	It stores the data of the system in a common shared database/repository, to then they are consulted by other systems that need to interoperate.	(GUO <i>et al.</i> , 2011; REPOSITORIO, 2021)
	File Transfer	It produces files containing the information required by the other systems for each system.	(KUBICEK <i>et al.</i> , 2011; REPOSITORIO, 2021)
	RPI	It uses the principle of encapsulation to interoperate, providing an interface to allow other systems to interact with the running systems.	(VERNADAT, 2009; REPOSITORIO, 2021)
Lack of Infrastructure	REST	It widely used in Web-based systems, providing a set of restrictions for communication and data format.	(AL-ZOUBI; WAINER, 2010; REZAEI <i>et al.</i> , 2014)
	Messaging	It transfers packets of data, immediately, reliably, and asynchronously and it helps in the definition of infrastructure for systems interoperate.	(CHEN <i>et al.</i> , 2008; REPOSITORIO, 2021).
Different Communication Protocols	Adapter	It transforms protocols used by heterogeneous systems to a common standard.	(KESHAV; GAMBLE, 1998; HARRER <i>et al.</i> , 2008)

data systems because data produced by different entities are homogenized and stored in a central repository (BOUGUETTAYA; MILLINER, 1995; REPOSITORIO, 2021).

File Transfer pattern is another possible solution for this interoperability issue. For each system, it produces files containing the information required by the other systems (KUBICEK *et al.*, 2011). The FTP is an implementation of this pattern used by operational systems, as Linux. A benefit is that there is less risk of misplaced or misdirected files because the production can automatically route sensitive files as needed to and from the correct locations and follow proper workflows (REPOSITORIO, 2021).

RPI pattern uses the principle of encapsulation to interoperate. For this, it is required that systems be developed as a large-scale object or component with encapsulated data (MAYBEE *et al.*, 1996). Thus, it is made possible to provide an interface to allow other systems to interact with the running systems. When a system needs data that is owned by another system, it asks that system directly. If one system needs to modify the data of another, then it does so by making a call to the other system (VERNADAT, 2009; REPOSITORIO, 2021).

The **lack of infrastructure** is the second interoperability issue at the technical level addressed in this thesis. As aforementioned, this issue is related to the software

and hardware that provide communication among heterogeneous systems. *REST* pattern has been used to mitigate this issue. It is widely used in Web-based systems, such as web services and microservices, and their correct implementation is denominated RESTful (AL-ZOUBI; WAINER, 2010). In this sense, a set of restrictions for communication and data format should be defined so that there is communication among heterogeneous systems (REZAEI *et al.*, 2014; REPOSITORIO, 2021).

Messaging pattern can be used to transfer packets of data frequently, immediately, reliably, and asynchronously. Sending a message does not require both systems to be up and ready at the same time (BICER *et al.*, 2005). This pattern also helps in the definition of infrastructure for that heterogeneous systems can interoperate (CHEN *et al.*, 2008). The infrastructure can be established through a message bus that transfers data packets among systems in an asynchronous using customizable formats (REPOSITORIO, 2021).

Finally, the last interoperability issue addressed at the technical level is using **different communication protocols** by the heterogeneous systems. *Adapter* pattern could transform protocols used by heterogeneous systems to a common standard. Thus, these systems can transmit data among them independently of their distance (HARRER *et al.*, 2008). This pattern also can act as a messaging client to the messaging system and invokes systems functions via a system-supplied interface. So, any system can connect to the messaging system and be integrated with other systems (KESHAV; GAMBLE, 1998).

5.3 Syntactic Interoperability

Syntactic interoperability is based on data coding, using markup languages for the development of systems, document management models, electronic records, and formats of information presentation (VEER; WILES, 2008; MICHENI *et al.*, 2014). Issues at this level arise when the sender's encoding rules are incompatible with the receiver's decoding rules, which leads to (the construction of) mismatching message parse trees. Web Services standards address syntactic interoperability by providing XML-based standards (HOSSEINI; DIXON, 2016; MACIEL *et al.*, 2017).

5.3.1 Interoperability Issues

This section presents two issues of syntactic interoperability that have been occurred in integration projects according to evidence obtained in the scientific literature and reports provided by architects and developers.

- **Different data formats:** The existence of legacy systems, the various data formats in use, and the heterogeneity of solutions from different vendors (computer

networks, operating systems, application servers, database systems, among others) contribute to the complexity of achieving interoperability at the syntactic level (DI-VÁN; REYNOSO, 2018). These systems use different data formats to represent and store their information, difficult to achieve interoperability at the syntactic level (MICHENI *et al.*, 2014). Therefore, the different data formats of systems challenge building a structure and syntax that enables the information to be ascertainable by both systems that need to communicate (VELTMAN, 2001; MICHENI *et al.*, 2014).

- **Encoding rules incompatible:** software-intensive systems are often developed by different organizations using different technologies and programming languages (BALDWIN *et al.*, 2017; BOUZIAT *et al.*, 2018). In general, these systems are loosely coupled systems and exchange messages (in synchronous or asynchronous modes) using neutral formats (preferably XML or XML-based) and simple transfer protocols (e.g., HTTP/HTTPS, SMTP, MIME, JMS, or SOAP over TCP/IP) (VERNADAT, 2009). Despite that, many systems have problems with the encoding rules incompatible. This has hampered communication among heterogeneous systems, difficulting the achieving interoperability at the syntactic level. Therefore, identifying elements and rules for structuring the elements, mapping, and crosswalks among equivalent elements should be established (VELTMAN, 2001; MICHENI *et al.*, 2014).

5.3.2 Architectural Patterns

This section presents architectural solutions that can be used in integration projects to solve each identified issue, as we listed aforementioned. As a result, Table 11 shows an overview of architectural solutions proposed for each identified issue of interoperability syntactic.

The **different data formats** are one of the main challenges when software systems need to interoperate at the syntactic level. In this perspective, architectural solutions have been proposed to mitigate this issue. The *SOA* (Service-Oriented Architecture) pattern has been used to help the publication of capabilities using services with well-defined formats and interfaces (GARLAN *et al.*, 2010; ZIMMERMANN, 2017). The *SOA* pattern represents the strategy most adopted to support syntactic interoperability among multi-channel and composite real-time applications implemented within large-scale distributed environments (GAZZARATA *et al.*, 2017). This pattern is formed by a set of network-accessible that can encapsulate the functionality and information of software systems and provides them through well-defined interfaces, facilitating interoperability among different systems (KAWAMOTO; LOBACH, 2007; GAZZARATA *et al.*, 2014).

The *Microservice* pattern also has been used to support software-intensive systems to achieve interoperability when there are software systems with different data for-

Table 11 – Architectural patterns to solve interoperability issues at the syntactic level

Issues	Patterns	Description	References
Different data formats	SOA	It encapsulates the functionality and information of software systems and provides them through well-defined interfaces, facilitating interoperability among the different systems.	(GAZZARATA <i>et al.</i> , 2017; ZIMMERMANN, 2017)
	Microservice	It helps in the standardization of data format of systems that need to communicate using services through interface specifications and continuous integration.	(NEWMAN, 2015; ZIMMERMANN, 2017)
	Adapter	It encodes information in a specific message format to be understood by other systems.	(GARCÉS, 2018)
Encoding rules incompatible	Layers	It separates the type of information embedded in message packages helping to identify elements and rules for structuring the information that will be exchanged.	(DALIYA; RAMESH, 2019; AYDIN; AYDIN, 2020)
	Adapter	It supports transforming data formats and adapts the coding rules that are incompatible.	(KESHAV; GAMBLE, 1998)

mats (NEWMAN, 2015; ZIMMERMANN, 2017). This pattern can be used to help in the standardization of data format of software systems that need to communicate using services through interface specifications and continuous integration (NEWMAN, 2015; ZIMMERMANN, 2017). The *Adapter* pattern encodes information in a specific message format to be understood by other software systems (GARCÉS, 2018). This pattern is used to aggregate new data types in a message to facilitate understanding these messages by others software systems. The *Adapter* pattern is a fundamental strategy to transform message formats, for example, transform XML to JSON message specification (KESHAV; GAMBLE, 1998).

Regarding the issue of **encoding rules incompatible** among software systems, there also solutions that have been proposed. This challenge is related to elements and rules for structuring the communication. In this sense, the *Layers* pattern can be considered to mitigate this issue in integration projects (DALIYA; RAMESH, 2019). This pattern is used to separate the type of information embedded in message packages helping to identify elements and rules for structuring the information that will be exchanged among software-intensive systems (AYDIN; AYDIN, 2020). Another architectural solution for this interoperability issue is the use of the *Adapter* pattern. Besides supporting to transform data formats, this pattern also helps to adapt coding rules that are incompatible (KESHAV; GAMBLE, 1998).

Other techniques can be considered to support syntactic interoperability (REZAEI

et al., 2014). Are examples: (i) BPEL (Business Process Execution Language) to easy and flexible the composition of services; (ii) web service languages for the implementation of services and guide how a message can be formatted to standardize its interpretation, such as JSON, YAML, XML.

5.4 Semantic Interoperability

Semantic interoperability aims to provide systems with a way to interpret the meaning of data, information, or knowledge. To provide communication at this level is very hard, and unfortunately, there is no packaged or out-of-the-box solution (VERNADAT, 2009). In short, semantic interoperability is the ability of heterogeneous systems to communicate among them, exchanging data, and having this data interpreted by the receiving system with the same meaning as provided by the sending system (RAHMAN; HUSSAIN, 2020).

5.4.1 Interoperability Issues

This section discusses the three main interoperability issues for the semantic level of software-intensive systems. These issues were identified through studies reported in the scientific literature and the survey conducted with the support of practitioners composed of software architects and developers with experience in industrial projects (VALLE *et al.*, 2019; VALLE *et al.*, 2021a).

- **Inconsistent information:** understanding the information exchanged among heterogeneous systems is a significant challenge in semantic interoperability (VERNADAT, 2009). In particular, the information exchanged among these systems is often inconsistent or even conflicting (MACIEL *et al.*, 2017). In this context, often the message sender does not always know the subject domain of the message receiver. Consequently, depending on their knowledge, the message sender makes assumptions about the receiver's subject domain, leading to the exchange of inconsistent information.
- **Different interpretations of the same concepts:** at this interoperability level, the exchanged message should ensure the same meaning for both the message sender and receiver (MACIEL *et al.*, 2017). Data in both messages have meaning only when are interpreted regarding the respective domain. In this sense, software architects and developers have faced problems understanding the information exchanged among heterogeneous systems (VERNADAT, 2009). In particular, there is a divergence of similar concepts, and consequently, different definitions of messages are

exchanged. It is related to different interpretations of the same concept for a given application domain.

- **Naming problems:** heterogeneous systems can work with data that has different nomenclatures (MACIEL *et al.*, 2017). The data exchanged among the different systems often have synonyms (i.e., several terms with the same meaning) and antonyms (i.e., one term with several meanings), leading to an increase in the complexity in understanding the meaning of information exchanged (VERNADAT, 2009). Another issue of this interoperability level is related to the translations between several languages.

5.4.2 Architectural Patterns

This section discusses architectural solutions that can be applied to mitigate the identified issues or even solve them. These architectural patterns are well-established solutions in the scientific literature and industry to solve interoperability issues at the semantic level. Table 12 shows an overview of architectural solutions suggested for each identified issue at the semantic interoperability level.

Table 12 – Architectural patterns to solve interoperability issues at the semantic level

Issues	Patterns	Description	References
Inconsistent information	Bus	It establishes connectivity and transform and message routing among systems and enables runtime semantic mediation within traditional SOA infrastructure.	(ZHU, 2012; ARSANJANI <i>et al.</i> , 2007)
Different interpretation of the same concepts	Adapter	It adds extra information to information exchanged. The extra information added helps the systems to found the same interpretation for information exchanged.	(GARCÉS <i>et al.</i> , 2018)
Naming problems	Adapter	It standardizes the understanding of data exchanged by analyzing their synonyms and antonyms.	(KESHAV; GAMBLE, 1998; MOREIRA <i>et al.</i> , 2018)

The **inconsistent information** exchanged among heterogeneous systems is one of the significant challenges of semantic interoperability. The *Bus* pattern can establish connectivity and transform and message routing among systems to mitigate this problem. This pattern enables runtime semantic mediation within traditional SOA infrastructure, creating a semantic mediation bus (ZHU, 2012). The *Bus* pattern can also simplify message transformations based on technologies such as the Extensible Stylesheet Language Transformation (XSLT) (ARSANJANI *et al.*, 2007). The *Adapter* pattern can add extra information to information exchanged to solve the problem of **different interpretations of the same concept**. The extra information added helps the systems to found the same interpretation for information exchanged (GARCÉS *et al.*, 2018).

The *Adapter* pattern can still solve the **naming problems** since it does the translation of information exchanged among heterogeneous systems aiming to standardize the understanding of data exchanged by analyzing their synonyms and antonyms (KESHAV; GAMBLE, 1998; MOREIRA *et al.*, 2018). Besides these architectural patterns, other techniques can support heterogeneous systems to achieve semantic interoperability, as thesaurus, taxonomies, and ontologies. These techniques have been used in the health domain help to undertafing of the meaning of the data that are exchanged among the different systems (BITTNER *et al.*, 2005; NOURA *et al.*, 2019).

5.5 Organizational Interoperability

The organizational interoperability deal with defining business goals, aligning and coordinating business processes, and bringing collaboration capabilities to organizations that wish to exchange information and may have different internal structures and processes. Therefore, the aim of organizational interoperability is to address the requirements of the user community by making services available, easily identifiable, accessible, and usercentric (VERNADAT, 2009).

5.5.1 Interoperability issues

This section discusses the main issues that occur at the organizational level and make it difficult the exchange information among software-intensive systems; in other words, they are the main issues of interoperability organizational. These issues result from research about interoperability and responses provided by practitioners who have faced them in their integration projects (VALLE *et al.*, 2019; VALLE *et al.*, 2021a).

- **Different legal bases, legislation, cultures:** organizational interoperability is the ability of business organizations to provide services to each other, users or customers, or the wider public. In this context, software-intensive systems that are part of such organizations have different legal bases, legislation, culture (VERNADAT, 2009). This makes it challenging to coordinate the business processes of cooperating business entities. Therefore, architectural solutions should be made when the individual systems made part of this class of systems have different legal bases, legislation, culture, and need to communicate.
- **Different business process:** the individual systems that are part of the software-intensive systems often have organizational aspects that deal with defining business

processes (MICHENI *et al.*, 2014; ADAMO *et al.*, 2018). These systems need to have the ability of diverse business processes to work together even when they are developed by different organizations. The collaboration of different business processes can compromise interoperability at the business level (VERNADAT, 2009).

- **Different organizational structures:** an organizational structure describes what employees do, whom they report to, and how decisions are made across the business (MICHENI *et al.*, 2014; ZARKO *et al.*, 2017). In general, Organizational structures can use functions, markets, products, geographies, or processes for businesses of specific sizes and industries. However, coordinating the different organizational structures is a great challenge because this involves the coordination and communication among the different individual systems that are part of software-intensive systems (VERNADAT, 2009).

5.5.2 Architectural Patterns

This section presents the architectural solutions to solve/mitigate the issues of organizational interoperability in software-intensive systems. These solutions can be found in the scientific literature and were validated through surveys with practitioners that have faced such issues in integration projects. Table 13 shows the main identified issues for the business level and their respective possible solutions.

Organizational interoperability is related to the coordination of distributed workflows and activities understood by different systems, organizations, or people interacting in business processes (BENSON; GRIEVE, 2016). In this perspective, individual systems that need to interoperate have faced **different legal bases, legislation, and cultures**. To mitigate this issue, architects and developers have considered some architectural solutions. To mitigate this issue, architects and developers have considered some architectural solutions. The *Pipes & Filters* pattern can be used when individual systems have different legislation and cultures, and they need to obtain messages from other systems without sending requests for them (KESHAV; GAMBLE, 1998).

The *Contract Monitor* pattern also can be used to mitigate the problem of different legal bases, legislation, and cultures. For this, the pattern should monitoring strategies which indicate precisely how and when a given contract may interact with a system that has these different characteristics of the sender system (INGRAM *et al.*, 2015). Therefore, this pattern captures the implementation requirements and the behavioral guarantees that result when these requirements are satisfied (HALLSTROM *et al.*, 2006).

In the context of the software-intensive systems, the individual systems can have **different business processes**, making it difficult to achieve interoperability at the business level. The *SOA* and *Microservices* patterns have been used to mitigate issues when

Table 13 – Architectural patterns to solve interoperability issues at the organization level

Issues	Patterns	Description	References
Different legal bases, legislation, and cultures	Pipes & Filters	It helps to obtain messages from other systems without sending requests for them.	(KESHAV; GAMBLE, 1998)
	Contract Monitor	It monitors strategies that indicate precisely how and when a given contract may interact with a system.	(HALLSTROM <i>et al.</i> , 2006; INGRAM <i>et al.</i> , 2015)
Different business process	SOA	It supports the description of organizational processes.	(BASS, 2013)
	Microservice	It standardizes different business processes using languages for this purpose, such as WSDL and BPMN.	(BASS, 2013)
	Orchestration	It uses a control mechanism to coordinate and manages the invocation of particular services that could be ignorant of each other.	(NEWMAN, 2015)
	Choreography	It describes a collection of services that work together to achieve a common and unique goal described by the different web services in interactions with their uses.	(BENANY; BEQQALI, 2018)
Different organization structures	Centralized Architecture	It provides a central point of control facilitating proper communication among individual systems.	(HALL-MAY; KELLY, 2006; INGRAM <i>et al.</i> , 2015)

software systems that need to interoperate have different business processes (BASS, 2013). These patterns support to describe organizational processes and also to standardize different business processes using languages for this purpose, such as WSDL and BPMN (NEWMAN, 2015). The *Orchestration* and *choreography* patterns also contribute to mitigating issues related to different organizational processes of individual systems that need interoperate (NEWMAN, 2015; BENANY; BEQQALI, 2018).

Architectural solutions have been adopted in integration projects to mitigate the problem faced by architects and developers regarding the **different organizational structures** of individual systems that need interoperability at the business level. For this problem, the *Centralized Architecture* pattern can be considered to provide a central point of control facilitating proper communication among individual systems even when they have different organizational structures (HALL-MAY; KELLY, 2006; INGRAM *et al.*, 2015).

5.6 Discussion and Lessons Learned

Software architects and developers have faced several challenges to achieve interoperability in software-intensive systems. In general, this class of systems is formed by individual systems developed by different organizations and technologies. Due to these characteristics, different issues can arise during the design of such systems, especially when their goal is to promote interoperability among individual systems. The issues can be divided into four interoperability levels (i.e., technical, syntactic, semantic, and organizational).

In this perspective, a set of interoperability issues were identified in the scientific literature, and to validate them, we conducted a survey with 33 practitioners of several countries of the world. Hence, we selected 11 different interoperability issues to be addressed in this doctoral thesis. These problems represent the main challenges that software architects and developers can face in their projects when they need to design software-intensive systems.

In this sense, this doctoral thesis aims to provide software architects with a catalog of the main interoperability issues and possible architectural patterns that can be used in integration projects to mitigate such issues. The catalog is divided according to the four interoperability levels. The catalog was defined according to the evidence available in the scientific literature. We also conducted a survey with practitioners with expertise in interoperability, software architecture, and software-intensive systems to validate the suggested solutions in the catalog, as reported in Chapter 3. Figure 11 illustrates an overview of this catalog.

All information used to define the catalog (i.e., interoperability issues and architectural patterns) was extracted from studies that received peer review. To confirm this evidence, we collected information from practitioners that participate in our survey. Therefore, we do not carry out experimental studies to provide the effectiveness and efficiency of the proposed catalog since all the information has already been validated and made available in the scientific literature, besides confirmed in the survey we conducted.

The main scientific contribution of this catalog is the grouping of the available information in the scientific literature on the way to mitigate/solve the main issues of each interoperability level. Therefore, architects will be able to consult it and make the decision about which architectural patterns should be considered to mitigate/solve the existing issues in their integration projects.

We highlight the following lessons learned and feelings in conducting this thesis:

1. The same architectural pattern can solve more than one interoperability issue, and

Figure 11 – Catalog of architectural patterns to solve interoperability issues

Source: Elaborated by the author.

also it can be applied at different interoperability levels;

2. Few studies detail the existing issues at each interoperability level;
3. Few studies report information on the way software architects have addressed interoperability issues in the industry;
4. There is a lack of guidelines for supporting software architects to address interoperability in their integration projects; and
5. There is little participation of industry practitioners in research on interoperability at the architectural level.

5.7 Final Considerations

This chapter proposed a catalog composed of interoperability issues and architectural solutions to mitigate/solve such issues. The main contribution of this catalog was to provide software architects and developers an overview of existing architectural solutions to mitigate/solve the 11 main interoperability issues considered in this doctoral thesis, which were organized according to four interoperability levels. Therefore, using this catalog, architects can use the knowledge gathered, categorized, and prioritized to make decisions about how to address each interoperability level considered in their integration projects.

The next chapter presents the main contributions of this doctoral thesis, identified limitations, and future work.

Conclusions

Due to the complexity of software-intensive systems, an increase in the interest to the research, development, and evolution of these systems has been observed (BOUZIAT *et al.*, 2018). Software-intensive systems are composed of individual systems that need to interoperate to exchange useful information with others. Individual systems, oftentimes, are developed by different organizations using different platforms and programming languages (BALDWIN *et al.*, 2017). Therefore, interoperability has performed an essential role in the success of software-intensive systems (MICHENI *et al.*, 2014; ABUKWAIK; ROMBACH, 2017).

In the software-intensive systems context, interoperability can be defined as the ability of two or more systems or elements to exchange information and to use the information that has been exchanged (IEEE, 2000). Despite interoperability be recognized as an important factor for the quality of software systems, there a great difficulty to analyze, comprehend, and guide the way that interoperability can be addressed (ABUKWAIK; ROMBACH, 2017). In this sense, practitioners have faced several challenges of interoperability as a lack of documentation, proven solutions, and guidelines to support the implementation of interoperability in the integration projects (VALLE *et al.*, 2021a).

In the scientific literature, there are several solutions to support the interoperability of software-intensive systems that need to interoperate (VALLE *et al.*, 2019; ABUKWAIK; ROMBACH, 2017). However, architects have faced a lack of means to help them make decisions about which solutions should be considered to address each interoperability level required in the requirements of a software system (VALLE *et al.*, 2021a). Based on this scenario, it is possible to observe the need of creating guidelines that help to guide and understand the way that interoperability should be addressed considering the different interoperability levels: technical, syntactic, semantic, and organizational.

In this perspective, the main contribution of this thesis was to provide means to

support the architects to make decisions and to guide, analyze, and comprehend the way that interoperability can be addressed in software-intensive systems. The main contributions of this thesis are summarized in Section 6.1, the difficulties and limitations are discussed in Section 6.2 and, finally, Section 6.3 presents the future work.

6.1 Thesis Contributions

The main objective of this thesis was to provide means to help architects consider architectural solutions to mitigate/solve interoperability issues in software-intensive systems. Specifically, this thesis contributes to the software-intensive systems, interoperability, and software architecture topics. The main contributions are summarized as follows:

- **Architectural solutions used in practice on integration projects:** we investigated how existing architectural strategies have been used in practice to promote the interoperability of software-intensive systems from the industry perspective. For this, we carried out a survey with practitioners from several countries to identify the: (i) difficulty level to apply such strategies; (ii) possible strategies impact over other quality attributes than interoperability; (iii) solutions to improve the integration process of software systems; (iv) technologies used for the integration in software projects; and (v) main challenges for achieving such integration. This thesis can help researchers guide new research and support sharing valuable architectural knowledge for decision-making in new integration projects. The results of this contribution can be consulted in (VALLE *et al.*, 2021a).
- **Typology of architectural strategies for interoperability:** We also proposed a typology of architectural strategies, named TASIS, which proposed architectural solutions for four interoperability levels (technical, syntactic, semantic, organizational). The contribution is to help architects select strategies from TASIS to mitigate interoperability issues at different architectural levels. To evaluate TASIS, we carried out a survey with 33 practitioners from various countries with experience in integration projects can be found in (VALLE *et al.*, 2019).
- **Catalog of interoperability issues and their architectural solutions:** to support the software architects in the decision-making to mitigate/solve the main interoperability issues in software-intensive systems, we proposed a catalog composed of 11 interoperability issues and their respective architectural solutions. The issues were organized according to each interoperability level. This catalog was defined ac-

cording to scientific literature and industry evidence and can support the architects in decision-making the interoperability of this class of systems.

6.2 Limitations

This section describes the limitations identified during the conduction of this thesis that may impact the validity of the results of our work.

1. **Experimental studies:** even though the proposed catalog was defined using scientific literature and industry evidence, it would be essential to conduct experimental studies to validate its efficiency and effectiveness to support architects and developers in decision-making on interoperability in their integration projects.
2. **Amount of practitioners:** we invited practitioners from the software industry from different countries to participate in our survey about architectural strategies for interoperability. In this study, we received 33 responses from different parts of the world. Although the amount of practitioners who participated in our survey is small and, consequently, it is not easy to generalize the evidence obtained, we believe that this information is reliable due to the expertise of practitioners in integration projects. Besides, we performed some actions to minimize threats to the validity of our study, as shown in (VALLE *et al.*, 2021a). Despite this, we know that more participants could have provided a better overview of the architectural strategies.
3. **The set of architectural patterns:** we selected the architectural patterns from TESIS based on the information found in the scientific literature. In this sense, other architectural patterns focused on interoperability of software architecture may not have been identified, and, consequently, they may not be considered in this work. To minimize this fact, we validated the list of patterns with practitioners of the software industry, asking them whether they knew other architectural patterns. As a result, we did not identify new architectural patterns.
4. **Description of interoperability:** the main contributions of this thesis are focused on the decision-making that software architects make to consider interoperability in software architectures. To make our contribution more complete, models (such as views and viewpoints) to describe the software architectures focusing on interoperability could be proposed.

6.3 Future Work

During the conduction of this thesis, many research opportunities were identified. In short, they could contribute directly to the software-intensive systems, interoperability, and software architecture areas. In this sense, the possible future work is described as follow:

- **Evaluation of the effectiveness of catalog:** experimental studies could be conducted to evaluate the effectiveness of the catalog and compare it with other existing approaches that help the software architects in decision-making. Therefore, these new studies could help to evaluate the catalog.
- **Consolidation of method to support the use of the catalog:** guidelines to support the use of the catalog could be proposed. The guidelines could help the software architects make decisions on how to mitigate/solve the interoperability issues.
- **Views and viewpoint for interoperability:** architectural models could be proposed to represent the architectural solutions made using the catalog. For example, architectural views could be built to represent each interoperability level.
- **Method to decision-making on interoperability in reference architectures:** solutions to consider interoperability in reference architectures could be proposed. These solutions could support the design and evolution of reference architectures, as well as their architectural instances.

References

ABTOY, A.; TOUHAFI, A.; TAHIRI, A. *et al.* Ambient assisted living system's models and architectures: A survey of the state of the art. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 32, n. 1, p. 1–10, 2018. Citation on page 3.

ABUKWAIK, H.; ROMBACH, D. Software interoperability analysis in practice: A survey. In: **International Conference on Evaluation and Assessment in Software Engineering (EASE)**. Karlskrona, Sweden: ACM, 2017. p. 12–20. Citations on pages 2, 4, 23, 69, 70, and 89.

ADAMO, G.; BORGO, S.; FRANCESCOMARINO, C. D.; GHIDINI, C.; GUARINO, N. On the notion of goal in business process models. In: SPRINGER. **International Conference of the Italian Association for Artificial Intelligence**. [S.l.], 2018. p. 139–151. Citation on page 84.

AGGARWAL, C. C. **Data mining: the textbook**. Chicago, USA: Springer, 2015. Citations on pages 46 and 47.

AGRAWAL, R.; IMIELIUNDEFINEDSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. In: **ACM International Conference on Management of Data (SIGMOD)**. New York, USA: ACM, 1993. p. 207–216. Citation on page 46.

AL-ZOUBI, K.; WAINER, G. Rise: Rest-ing heterogeneous simulations interoperability. In: **Proceedings of the 2010 Winter Simulation Conference**. [S.l.: s.n.], 2010. p. 2968–2980. Citations on pages 77 and 78.

ALEXANDER, C. **A pattern language: towns, buildings, construction**. California, USA: Oxford university press, 1977. Citations on pages 18 and 20.

ANGELOV, S.; GREFFEN, P.; GREEFHORST, D. A classification of software reference architectures: Analyzing their success and effectiveness. In: **Joint Working IEEE/IFIP Conference on Software Architecture (ECSA) & European Conference on Software Architecture (WICSA)**. Cambridge, UK: IEEE, 2009. p. 141–150. Citation on page 11.

APPLETON, B. **Patterns and software: Essential concepts and terminology**. Accessed in 26/03/2021, 1997. Available: <http://www.enteract.com/~bradpp/docs/patterns-intro.html>. Citation on page 20.

ARCAINI, P.; MIRANDOLA, R.; RICCOBENE, E.; SCANDURRA, P. Msl: a pattern language for engineering self-adaptive systems. **Journal of Systems and Software**, Elsevier, v. 164, p. 1–18, 2020. Citation on page 19.

ARSANJANI, A.; ZHANG, L.-J.; ELLIS, M.; ALLAM, A.; CHANNABASAVIAH, K. S3: A service-oriented reference architecture. **IT professional**, IEEE, v. 9, p. 10–17, 2007. Citations on pages 64 and 82.

AYDIN, S.; AYDIN, M. N. Semantic and syntactic interoperability for agricultural open-data platforms in the context of iot using crop-specific trait ontologies. **Applied Sciences**, Multidisciplinary Digital Publishing Institute, v. 10, n. 13, p. 4460, 2020. Citation on page 80.

BAILEY, K. D. **Typologies and taxonomies: an introduction to classification techniques**. California, USA: Sage, 1994. Citations on pages 57, 58, and 69.

BALDWIN, W. C.; SAUSER, B. J.; BOARDMAN, J. Revisiting “the meaning of of” as a theory for collaborative system of systems. **IEEE Systems Journal**, v. 11, n. 4, p. 2215–2226, 2017. Citations on pages 1, 79, and 89.

BASIL, V.; CALDIERA, G.; ROMBACH, H. The goal question metric approach. **Encyclopedia of software engineering**, v. 1, p. 528–532, 1994. Citation on page 23.

BASS, L. **Software architecture in practice**. 3rd. ed. Massachusetts, USA: Addison-Wesley, 2013. Citations on pages 3, 9, 10, 11, 14, 27, 63, 66, and 85.

BECK, K.; CUNNINGHAM, W. Using pattern languages for object-oriented programs. **Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)**, 1987. Citation on page 19.

BENANY, E.; BEQQALI, E. Choreography for interoperability in the e-government applications. In: **International Conference on Intelligent Systems and Computer Vision (ISCV)**. Fez, Morocco: IEEE, 2018. p. 1–4. Citations on pages 66 and 85.

BENSON, T.; GRIEVE, G. **Principles of health interoperability: SNOMED CT, HL7 and FHIR**. London, UK: Springer, 2016. Citations on pages 2, 16, 58, 61, 64, 65, and 84.

BIANCHI, T.; SANTOS, D. S.; FELIZARDO, K. R. Quality attributes of systems-of-systems: A systematic literature review. In: **IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS)**. Florence, Italy: ACM, 2015. p. 23–30. Citation on page 15.

BICER, V.; LALECI, G. B.; DOGAC, A.; KABAK, Y. Artemis message exchange framework: Semantic interoperability of exchanged messages in the healthcare domain. **ACM**, New York, USA, v. 34, n. 3, 2005. Citation on page 78.

BITTNER, T.; DONNELLY, M.; WINTER, S. Ontology and semantic interoperability. In: **Large-Scale 3D Data Integration: Challenges and Opportunities**. New York, USA: Press, 2005. p. 139–160. Citations on pages 27, 64, and 83.

BLAIR, G. S.; PAOLUCCI, M.; GRACE, P.; GEORGANTAS, N. Interoperability in complex distributed systems. In: **International School on Formal Methods for the Design of Computer, Communication and Software Systems (USENIX)**. Bertinoro, Italy: Springer Berlin Heidelberg, 2011. p. 1–26. Citation on page 58.

- BOSCH, J.; RAN, A. Evolution of software product families. In: **International Workshop on Software Architectures for Product Families (IW-SAPF)**. Las Palmas de Gran Canaria, Spain: Springer, 2000. p. 168–183. Citations on pages [11](#) and [12](#).
- BOUGUETTAYA, A.; MILLINER, S. Co-database approach to database interoperability. **IEICE TRANSACTIONS on Information and Systems**, The Institute of Electronics, Information and Communication Engineers, v. 78, n. 11, p. 1388–1395, 1995. Citation on page [77](#).
- BOUZIAT, T.; CAMPS, V.; COMBETTES, S. A cooperative sos architecting approach based on adaptive multi-agent systems. In: **International Workshop on Software Engineering for Systems-of-Systems (SESoS)**. Gothenburg, Sweden: ACM, 2018. p. 8–16. Citations on pages [1](#), [79](#), and [89](#).
- BRAGA, R. T.; RÉ, R.; MASIERO, P. C. A process to create analysis pattern languages for specific domains. In: **Latin American Conference on Pattern Languages of Programs (SugarLoafPloP)**. Recife, Brazil: ACM, 2007. p. 251–265. Citation on page [22](#).
- BRAGA, R. T. V. **Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico**. Phd Thesis (PhD Thesis) — São Paulo, 2002. Citation on page [19](#).
- BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern-oriented software architecture: a system of patterns**. Pennsylvania, USA: John Wiley & Sons, 1996. Citations on pages [10](#), [18](#), [19](#), [20](#), [21](#), and [27](#).
- CALCADA, D. B. **Redes de regras de associação filtradas e multialvo**. Phd Thesis (PhD Thesis) — Universiade de São Paulo, 2019. Citation on page [47](#).
- CHAINHO, P.; DRÜSEDOW, S.; PEREIRA, R. L.; CHAVES, R.; SANTOS, N.; HAENSGE, K.; PORTABALES, A. R. Decentralized communications: Trustworthy interoperability in peer-to-peer networks. In: **2017 European Conference on Networks and Communications (EuCNC)**. [S.l.: s.n.], 2017. p. 1–5. Citation on page [75](#).
- CHEN, D.; DOUMEINGTS, G.; VERNADAT, F. Architectures for enterprise integration and interoperability: Past, present and future. **Computers in Industry**, v. 59, n. 7, p. 647–659, 2008. Citations on pages [58](#), [77](#), and [78](#).
- CHEN, J. Devify: Decentralized internet of things software framework for a peer-to-peer and interoperable iot device. ACM, New York, USA, v. 15, n. 2, 2018. Citation on page [75](#).
- Choi, Y. K.; Kim, S. K. An auxiliary recommendation system for repetitively purchasing items in e-commerce. In: **International Conference on Big Data and Smart Computing (BIGCOMP)**. Bangkok, Thailand: IEEE, 2014. p. 96–98. Citation on page [46](#).
- CHOPRA, A. Business process interoperability:extended abstract. In: **International Foundation for Autonomous Agents and Multiagent Systems(AAMAS)**. Estoril, Portugal: ACM, 2008. Citation on page [65](#).

CLEMENTS, P.; BACHMANN, F.; BASS, L.; GARLAN, D.; IVERS, J.; LITTLE, R.; MERSON, P.; NORD, R.; STAFFORD, J. **Documenting Software Architecture: Views and Beyond**. Pittsburgh, USA: Addison-Wesley, 2011. 537 p. Citations on pages 9, 10, 13, 14, and 27.

COAD, P. Object-oriented patterns. **Communications of the ACM**, ACM, v. 35, n. 9, p. 152–159, 1992. Citation on page 19.

COPLIEN, J. O. Software design patterns: common questions and answers. **The patterns handbook: Techniques, strategies, and applications**, Cambridge University Press New York, USA, v. 13, p. 311, 1998. Citation on page 20.

COPLIEN, J. O.; SCHMIDT, D. C. **Pattern languages of program design**. New York, USA: ACM Press/Addison-Wesley Publishing Co., 1995. 1–576 p. Citation on page 19.

DABC. **European interoperability framework for pan-european e-government services (eif 1.0)**. Accessed in 26/03/2021, 2004. On-line. Available: <<http://ec.europa.eu/idabc/servlets/Docd552.pdf?id=19529>>. Citation on page 64.

DALIYA, V.; RAMESH, T. Data interoperability enhancement of electronic health record data using a hybrid model. In: **International Conference on Smart Systems and Inventive Technology**. [S.l.: s.n.], 2019. p. 318–322. Citation on page 80.

DELECOLLE, A.; LIMA, R. S.; NETO, V. V. G.; BUISSON, J. Architectural strategy to enhance the availability quality attribute in system-of-systems architectures: a case study. In: **International Conference of System of Systems Engineering (SoSE)**. Budapest, Hungary: IEEE, 2020. p. 93–98. Citation on page 3.

DIVÁN, M.; REYNOSO, M. L. S. Fostering the interoperability of the measurement and evaluation project definitions in pabmm. In: **International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)**. [S.l.: s.n.], 2018. p. 231–238. Citation on page 79.

DOD. **Levels of Information Systems Interoperability (LISI)**. C4ISR. AWG - Architectures Working Group, 1998. Available: <<http://web.cse.msstate.edu/~hamilton/C4ISR/LISI.pdf>>. Citation on page 58.

DOD. **DoD Dictionary of Military and Associated Terms**. 2001. Accessed in 26/03/2021. Available: <http://www.dtic.mil/doctrine/jel/new_pubs/jp1_02.pdf>. Citation on page 15.

DOTY, D. H.; GLICK, W. H. Typologies as a unique form of theory building: Toward improved understanding and modeling. **Academy of management review**, Academy of Management, Arkansas, USA, p. 230–251, 1994. Citation on page 57.

EHGI, E. G. I. **Discussion paper on semantic and technical interoperability**. 2017. Citations on pages 75 and 76.

EIRA. **European Interoperability Reference Architecture**. 2018. Access in 03/08/2018. Available: <<https://joinup.ec.europa.eu/solution/eira/>>. Citation on page 15.

- FALBO, R. A.; QUIRINO, G. K.; NARDI, J. C.; BARCELLOS, M. P.; GUIZZARDI, G.; GUARINO, N.; LONGO, A.; LIVIERI, B. An ontology pattern language for service modeling. In: **Annual ACM Symposium on Applied Computing (SAC)**. Pisa, Italy: ACM, 2016. p. 321–326. Citation on page 19.
- FAROOQ, M. O.; WHEELLOCK, I.; PESCH, D. Iot-connect: An interoperability framework for smart home communication protocols. **IEEE Consumer Electronics Magazine**, v. 9, n. 1, p. 22–29, 2020. Citation on page 76.
- FASSBINDER, A. **A contribution to the process of designing for learning in Massive Open Online Courses (MOOCs)**. Phd Thesis (PhD Thesis) — São Paulo, 2018. Citations on pages 19 and 21.
- FERNANDES, J. C.; NETO, V. V. G.; SANTOS, R. P. d. Interoperability in systems-of-information systems: A systematic mapping study. In: **Brazilian Symposium on Software Quality (SBQS)**. Curitiba, Brazil: ACM, 2018. p. 131–140. Citations on pages 36 and 54.
- FERNANDEZ, E. B.; ASTUDILLO, H.; PEDRAZA-GARCÍA, G. Revisiting architectural tactics for security. In: SPRINGER. **European Conference on Software Architecture**. [S.l.], 2015. p. 55–69. Citation on page 27.
- FIORAVANTI, M. L. **MLearning-PL: a pedagogical pattern language for mobile learning applications**. Phd Thesis (PhD Thesis) — São Paulo, 2017. Citation on page 19.
- FOWLER, M. **Writing software patterns**. 2006. Accessed in 26/03/2021. Available: <<https://www.martinfowler.com/articles/writingPatterns.html>>. Citation on page 20.
- GAMMA, E. **Design patterns: elements of reusable object-oriented software**. Boston, USA: Pearson Education India, 1995. Citations on pages 19 and 20.
- GARCÉS, L. **A Reference Architecture for Healthcare Supportive Home (HSH) systems**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2018. Citations on pages 63 and 80.
- GARCÉS, L.; OQUENDO, F.; NAKAGAWA, E. Y. Towards a taxonomy of software mediators for systems-of-systems. In: **Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARs)**. São Carlos, Brazil: ACM, 2018. p. 53–62. Citations on pages 64 and 82.
- GARCÉS, L.; OQUENDO, F.; NAKAGAWA, E. Y. Software mediators as first-class entities of systems-of-systems software architectures. **Journal of the Brazilian Computer Society**, Springer, v. 25, n. 1, p. 8, 2019. Citation on page 1.
- GARLAN, D.; BACHMANN, F.; IVERS, J.; STAFFORD, J.; BASS, L.; CLEMENTS, P.; MERSON, P. **Documenting Software Architectures: Views and Beyond**. Montreal, Canada: Addison-Wesley Professional, 2010. 31–39 p. Citations on pages 62 and 79.
- GAZZARATA, R.; GIANNINI, B.; GIACOMINI, M. A soa-based platform to support clinical data sharing. **Journal of healthcare engineering**, Hindawi, v. 2017, 2017. Citations on pages 79 and 80.

GAZZARATA, R.; VERGARI, F.; CINOTTI, T. S.; GIACOMINI, M. A standardized soa for clinical data interchange in a cardiac telemonitoring environment. **IEEE journal of biomedical and health informatics**, IEEE, v. 18, n. 6, p. 1764–1774, 2014. Citation on page 79.

GOTTSCHALK, P. Maturity levels for interoperability in digital government. **Government Information Quarterly**, Elsevier, v. 26, n. 1, p. 75–81, 2009. Citation on page 15.

GRACE, P.; BROMBERG, Y.-D.; RÉVEILLÈRE, L.; BLAIR, G. Overstar: An open approach to end-to-end middleware services in systems of systems. In: **International Conference on Distributed Systems Platforms (IFIP) and Open Distributed Processing (USENIX)**. Montreal, Canada: Springer, 2012. p. 229–248. Citation on page 62.

GUERRA, E.; FERNANDES, C. An evaluation process for pattern languages. In: **Latin American Conference on Pattern Languages of Programs (SugarLoafPLOP)**. Salvador, Brazil: ACM, 2010. p. 1–11. Citation on page 22.

GUO, Y.; HU, Y.; AFZAL, J.; BAI, G. Using p2p technology to achieve ehealth interoperability. In: **International Conference on Service Systems and Service Management**. [S.l.: s.n.], 2011. p. 1–5. Citations on pages 76 and 77.

HALL-MAY, M.; KELLY, T. Using agent-based modelling approaches to support the development of safety policy for systems of systems. In: **International Conference on Computer Safety, Reliability, and Security (SafeComp)**. Heidelberg, Germany: Springer, 2006. p. 330–343. Citations on pages 66 and 85.

HALLSTROM, J. O.; DALTON, A. R.; SOUNDARAJAN, N. Parallel monitoring of design pattern contracts. In: CITESEER. **SEKE**. [S.l.], 2006. p. 236–241. Citations on pages 84 and 85.

HARRER, A.; PINKWART, N.; MCLAREN, B. M.; SCHEUER, O. The scalable adapter design pattern: Enabling interoperability between educational software tools. **IEEE Transactions on Learning Technologies**, v. 1, n. 2, p. 131–143, 2008. Citations on pages 77 and 78.

HARRISON, N. B.; AVGERIOU, P. Leveraging architecture patterns to satisfy quality attributes. In: **European conference on software architecture (ECSA)**. Heidelberg, Germany: Springer, 2007. p. 263–270. Citations on pages 27 and 54.

HASSELBRING, W. **Software architecture: Past, present, future**. Cham, Switzerland: Springer, 2018. 169–184 p. Citation on page 9.

HOFMEISTER, C.; KRUCHTEN, P.; NORD, R. L.; OBBINK, H.; RAN, A.; AMERICA, P. A general model of software architecture design derived from five industrial approaches. **Journal of Systems and Software**, v. 80, n. 1, p. 106 – 126, 2007. Citations on pages 3, 9, 12, and 13.

HOSSEINI, M.; DIXON, B. E. Syntactic interoperability and the role of standards. In: **Health Information Exchange**. [S.l.]: Elsevier, 2016. p. 123–136. Citation on page 78.

HUANG, W. L.; PELESKA, J.; SCHULZE, U. Contract support for evolving sos. **Public Document D**, v. 1, 2014. Citation on page 66.

HUNTER, J. Metanet: a metadata term thesaurus to enable semantic interoperability between metadata domains. **Journal of Digital information**, Texas A & M University Libraries, v. 1, 2001. Citation on page 64.

IBA, T.; SAKAMOTO, M.; MIYAKE, T. How to write tacit knowledge as a pattern language: Media design for spontaneous and collaborative communities. **Procedia Social and Behavioral Sciences**, Elsevier, v. 26, p. 46–54, 2011. Citation on page 22.

IBRAHIM, N. M.; HASSAN, M. F. bin. A survey on different interoperability frameworks of soa systems towards seamless interoperability. In: **International Symposium in Information Technology (ITSim)**. Kuala Lumpur, Malaysia: IEEE, 2010. v. 3, p. 1119–1123. Citations on pages 2, 16, 75, and 76.

IEEE. The authoritative dictionary of ieee standards terms. **IEEE Std 100**, v. 2000, p. 1–1362, 2000. Citations on pages 2, 15, and 89.

INGRAM, C.; PAYNE, R.; FITZGERALD, J. Architectural modelling patterns for systems of systems. In: **Annual International Council on Systems Engineering (INCOSE)**. Seattle, USA: Wiley Online Library, 2015. v. 25, n. 1, p. 1177–1192. Citations on pages 66, 84, and 85.

ISO, I. **ISO/IEC 25010 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models**. 2011. On-line. Available: <<https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en>>. Citations on pages 14 and 41.

ISO, I. **ISO/IEC/IEEE 42010: Systems and software engineering—architecture description**. 2011. Accessed in 26/03/2021. Available: <<http://www.iso-architecture.org/ieee-1471/ads/>>. Citations on pages 10, 12, and 13.

JADHAV, R. D.; DESHPANDE, A. An efficient and interactive approach for association rules generation by integrating ontology and filtering technique. In: **International Conference on ICT for Sustainable Development (ICT4SD)**. New York, USA: Springer, 2016. p. 233–241. Citation on page 46.

Jang, M.; Sohn, J.; Cho, H. K. Automated question answering using semantic web services. In: **IEEE Asia-Pacific Service Computing Conference (APSCC)**. Tsukuba, Japan: IEEE, 2007. p. 344–348. Citation on page 64.

JANSEN, A.; BOSCH, J. Software architecture as a set of architectural design decisions. In: **Working IEEE/IFIP Conference on Software Architecture (WICSA)**. Pittsburgh, USA: IEEE, 2005. p. 109–120. Citation on page 10.

JOSUTTIS, N. M. **SOA in practice: the art of distributed system design**. Montreal, Canada: “O’Reilly Media, Inc.”, 2007. Citations on pages 43 and 66.

KASUNIC, M. Designing an effective survey. **Software Engineering Institute | Carnegie Mellon University**, Pittsburgh, USA, 2005. Citations on pages 29, 30, and 33.

KAWAMOTO, K.; LOBACH, D. F. Proposal for fulfilling strategic objectives of the us roadmap for national action on decision support through a service-oriented architecture leveraging hl7 services. **Journal of the American Medical Informatics Association**, BMJ Group BMA House, Tavistock Square, London, WC1H 9JR, v. 14, n. 2, p. 146–155, 2007. Citation on page [79](#).

KESHAV, R.; GAMBLE, R. Towards a taxonomy of architecture integration strategies. In: **International Workshop on Software Architecture (ISAW)**. Orlando, USA: ACM, 1998. p. 89–92. Citations on pages [63](#), [64](#), [65](#), [77](#), [78](#), [80](#), [82](#), [83](#), [84](#), and [85](#).

KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering—a systematic literature review. **Information and software technology**, Elsevier, v. 51, n. 1, p. 7–15, 2009. Citation on page [24](#).

KRUCHTEN, P. **The rational unified process: an introduction**. Boston, USA: Addison-Wesley Professional, 2004. 1–21 p. Citations on pages [2](#) and [12](#).

KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The past, present, and future for software architecture. **IEEE Software**, v. 23, p. 22–30, 2006. Citation on page [13](#).

KUBICEK, H.; CIMANDER, R.; SCHOLL, H. J. Chapter 7 - layers of interoperability. In: **Organizational Interoperability in E-Government (ICSOC)**. Heidelberg, Germany: Springer, 2011. p. 85–96. Citations on pages [2](#), [16](#), [23](#), [58](#), [61](#), [62](#), [64](#), [65](#), and [77](#).

LAND, R. **A brief survey of software architecture**. 2002. Malardalen Real-Time Research Center (MRTC) Report. Citation on page [9](#).

LEGNER, C.; LEBRETON, B. Business interoperability research: present achievements and upcoming challenges. **Electronic Markets**, Taylor & Francis, v. 17, n. 3, p. 176–186, 2007. Citation on page [16](#).

LI, W.; LIU, K.; LIU, S. Semiotic interoperability - a critical step towards systems integration. In: **International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)**. Vilamoura, Portugal: SciTePress, 2013. p. 508–513. Citation on page [58](#).

MACIEL, R. S. P.; DAVID, J. M. N.; CLARO, D.; BRAGA, R. Full interoperability: Challenges and opportunities for future information systems. **Sociedade Brasileira de Computação**, 2017. Citations on pages [78](#), [81](#), and [82](#).

MADNI, A. M.; SIEVERS, M. System of systems integration: Key considerations and challenges. **Systems Engineering**, Wiley Online Library, v. 17, n. 3, p. 330–347, 2014. Citations on pages [1](#), [2](#), and [23](#).

MARK, R. **Software Architecture Patterns-Understanding Common Architecture Patterns and When to Use Them**. [S.l.]: O'Reilly, Febrero, 2015. Citation on page [27](#).

MAYBEE, M.; HEIMBIGNER, D.; OSTERWEIL, L. Multilanguage interoperability in distributed systems. experience report. In: **International Conference on Software Engineering**. [S.l.: s.n.], 1996. p. 451–463. Citation on page [77](#).

- MEDVIDOVIC, N.; TAYLOR, R. N. A framework for classifying and comparing architecture description languages. In: **International Symposium on Foundations of Software Engineering (SIGSOFT)**. Zurich, Switzerland: ACM, 1997. p. 60–76. Citation on page 14.
- MESBAH, A.; DEURSEN, A. V. An architectural style for ajax. In: **Working IEEE/IFIP Conference on Software Architecture (WICSA)**. Mumbai, India: IEEE, 2007. p. 1–9. Citations on pages 3 and 27.
- MESZAROS, G.; DOBLE, J. Metapatterns: A pattern language for pattern writing. In: **Pattern Languages of Programming conference (PLoP)**. Monticello, Illinois: Cite-seer, 1996. p. 1–36. Citations on pages 21 and 22.
- MICHENI, E.; MUKETHA, G. M.; WAMOCHO, L. A review of agent based interoperability frameworks and interoperability assessment models. **Scholars Journal of Engineering and Technology (SJET)**, v. 2, 2014. Citations on pages 3, 10, 16, 66, 78, 79, 84, and 89.
- MICROSOFT. **Cloud Design Patterns**. Accessed in 26/03/2021, 2021. Available: <<https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters>>. Citation on page 63.
- MOGHADDAM, F. A.; PROCACCIANTI, G.; LEWIS, G. A.; LAGO, P. Empirical validation of cyber-foraging architectural tactics for surrogate provisioning. **Journal of Systems and Software**, Elsevier, v. 138, p. 37–51, 2018. Citation on page 43.
- MOREIRA, M. W.; RODRIGUES, J. J.; SANGAIAH, A. K.; AL-MUHTADI, J.; KOROTAEV, V. Semantic interoperability and pattern classification for a service-oriented architecture in pregnancy care. **Future Generation Computer Systems**, Elsevier, v. 89, p. 137–147, 2018. Citations on pages 82 and 83.
- MORRIS, E.; LEVINE, L.; MEYERS, C.; PLACE, P.; PLAKOSH, D. **System of systems interoperability (SOSI)**. Pittsburgh, USA: Software Engineering Institute | Carnegie Mellon University, 2004. Available: <<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=7045>>. Citations on pages 17 and 58.
- MOTTA, R. C. **On Context-Aware Software Systems and Interoperability: A Discussion Grounded in Data**. Master's Thesis (Master's Thesis) — Rio de Janeiro, 2016. Citation on page 43.
- NAKAGAWA, E.; ANTONINO, P. O.; BECKER, M. Reference architecture and product line architecture: A subtle but critical difference. **European Conference on Software Architecture (ECSA)**, Springer, Essen, Germany,, p. 207–211, 2011. Citation on page 11.
- NAKAGAWA, E. Y.; GUESSI, M.; MALDONADO, J. C.; FEITOSA, D.; OQUENDO, F. Consolidating a process for the design, representation, and evaluation of reference architectures. In: **IEEE/IFIP Conference on Software Architecture (WICSA)**. Sydney, Australia: IEEE, 2014. p. 143–152. Citation on page 11.
- NAKAGAWA, E. Y.; OQUENDO, F.; AVGERIOU, P.; CUESTA, C. E.; DRIRA, K.; MALDONADO, J. C.; ZISMAN, A. Foreword: towards reference architectures for systems-of-systems. In: **International Workshop on Software Engineering for Systems-of-Systems (SESoS)**. Florence, Italy: IEEE, 2015. p. 1–4. Citation on page 3.

- NAKAGAWA, E. Y.; SIMAO, A. da S.; FERRARI, F. C.; MALDONADO, J. C.; CARLOS, S. Towards a reference architecture for software testing tools. In: **International Conference on Software Engineering and Knowledge Engineering (SEKE)**. Boston, USA: IEEE, 2007. p. 1–6. Citation on page 10.
- NEIVA, F. W.; DAVID, J. M. N.; BRAGA, R.; CAMPOS, F. Towards pragmatic interoperability to support collaboration: A systematic review and mapping of the literature. **Information and Software Technology**, v. 72, p. 137 – 150, 2016. Citation on page 58.
- NEWMAN, S. **Building microservices: designing fine-grained systems**. New York, USA: “O’Reilly Media, Inc.”, 2015. Citations on pages 62, 64, 65, 66, 80, and 85.
- NIELSEN, C. B.; LARSEN, P. G.; FITZGERALD, J.; WOODCOCK, J.; PELESKA, J. Systems of systems engineering: basic concepts, model-based techniques, and research directions. **ACM Computing Surveys**, ACM, v. 48, n. 2, p. 18, 2015. Citation on page 2.
- NOURA, M.; ATIQUZZAMAN, M.; GAEDKE, M. Interoperability in internet of things: Taxonomies and open challenges. **Mobile Networks and Applications**, Springer, v. 24, n. 3, p. 796–809, 2019. Citations on pages 64, 76, and 83.
- Nuwangi, S. M.; Oruthotaarachchi, C. R.; Tilakaratna, J. M. P. P.; Caldera, H. A. Usage of association rules and classification techniques in knowledge extraction of diabetes. In: **International Conference on Advanced Information Management and Service (IMS)**. Seoul, Korea (South): IEEE, 2010. p. 372–377. Citation on page 46.
- OLIVEIRA, L. B. R. de. **Architectural design of service-oriented robotic systems**. Phd Thesis (PhD Thesis) — Universiade de São Paulo, 2015. Citation on page 13.
- ONLINE, R. **Architectural Patterns (3)**. Accessed in 26/03/2021, 2021. Available: <<http://pautasso.info/\lectures/s14/sad/8-patterns/patterns3.pdf>>. Citation on page 65.
- ONLINE, R. **Pattern: Service Level Agreement**. Accessed in 26/03/2021, 2021. Available: <www.microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/\ServiceLevelAgreement>. Citation on page 66.
- PADUA, R. d. **Redes de regras de associação**. Phd Thesis (PhD Thesis) — Universiade de São Paulo, 2019. Citation on page 47.
- PANDEY, G.; CHAWLA, S.; POON, S.; ARUNASALAM, B.; DAVIS, J. G. Association rules network: Definition and applications. **Statistical Analysis and Data Mining: The ASA Data Science Journal**, Wiley Online Library, v. 1, n. 4, p. 260–279, 2009. Citation on page 47.
- PANG, L.; ZHONG, R. Y.; FANG, J.; HUANG, G. Q. Data-source interoperability service for heterogeneous information integration in ubiquitous enterprises. **Advanced Engineering Informatics**, Elsevier, v. 29, n. 3, p. 549–561, 2015. Citation on page 76.
- PRESSMAN, R. S.; MAXIM, D. B. R. **Software engineering: a practitioner’s approach**. New York, USA: Palgrave macmillan, 2014. Citations on pages 18, 19, 20, and 27.

RAHMAN, H.; HUSSAIN, M. I. A comprehensive survey on semantic interoperability for internet of things: State-of-the-art and research challenges. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 31, n. 12, 2020. Citation on page 81.

REPOSITÓRIO. **Repositório Online de Padrões: Enterprise Integration Patterns**. Accessed in 26/03/2021, 2021. Available: <<http://www.enterpriseintegrationpatterns.com>>. Citations on pages 61, 62, 64, 65, 77, and 78.

REPOSITÓRIO Online de Padrões: Architectural Pattern. 2019. Accessed in 26/03/2021. Available: <<http://software-pattern.org/Category/5>>. Citations on pages 64 and 65.

REZAEI, R.; CHIEW, T. K.; LEE, S. P. An interoperability model for ultra large scale systems. **Advances in Engineering Software**, Elsevier, v. 67, p. 22–46, 2014. Citations on pages 2, 3, 27, 61, 62, 63, 77, 78, and 81.

ROTHENBERG, J. Interoperability as a semantic cross-cutting concern. **Interoperabiliteit: Eerlijk zullen we alles delen**, Forum Standaardisatie Den Haag, v. 1, 2008. Citation on page 15.

SOMMERVILLE, I. **Engenharia De Software**. 10rd. ed. São Paulo: PEARSON BRASIL, 2019. Citation on page 14.

STADEN, S. V.; MBALE, J. Levels of information systems interoperability (lisi). **Information Engineering and Electronic Business**, Windhoek, Namibi, 1998. Citation on page 2.

TRACZ, W. Dssa (domain-specific software architecture): Pedagogical example. **SIGSOFT Software Engineering Notes**, ACM, New York, USA, v. 20, 1995. Citation on page 11.

VALLE, P. H. D.; GARCÉS, L.; NAKAGAWA, E. Y. A typology of architectural strategies for interoperability. In: **Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARs)**. Salvador, Brazil: ACM, 2019. p. 3–12. Citations on pages 2, 6, 7, 10, 11, 15, 29, 31, 33, 36, 54, 60, 76, 81, 83, 89, and 90.

VALLE, P. H. D.; GARCÉS, L.; NAKAGAWA, E. Y. Architectural strategies for interoperability of software-intensive systems: Practitioners' perspective. In: **ACM Symposium on Applied Computing, Track Software Architecture: Theory, Technology, and Applications (SAC/SATTA 2021)**. Virtual Conference: ACM, 2021. p. 1–10. Citations on pages 1, 3, 4, 6, 7, 10, 11, 43, 44, 45, 76, 81, 83, 89, 90, and 91.

VALLE, P. H. D.; GARCÉS, L.; VOLPATO, T.; MARTÍNEZ-FERNÁNDEZ, S.; NAKAGAWA, E. Y. Towards suitable description of reference architectures. **PeerJ Computer Science**, p. 1–26, 2021. Citation on page 1.

VALLE, P. H. D.; GARCÉS, L.; GUESSI, M.; MARTÍNEZ-FERNÁNDEZ, S.; NAKAGAWA, E. Y. Approaches for describing reference architectures: A systematic mapping study. In: **XXIII Iberoamerican Conference on Software Engineering (CIbSE)**. Curitiba, Brazil: Springer, 2020. p. 1–14. Citation on page 1.

- VARGAS, I. G.; GOTTARDI, T.; BRAGA, R. T. V. Approaches for integration in system of systems: a systematic review. In: **International Workshop on Software Engineering for Systems-of-Systems (SESoS)**. Austin, USA: IEEE, 2016. p. 32–38. Citations on pages 36, 54, and 58.
- VEER, H. van der; WILES, A. Achieving technical interoperability. **European telecommunications standards institute**, v. 1, 2008. Citations on pages 2, 16, 61, 75, 76, and 78.
- VELTMAN, K. H. Syntactic and semantic interoperability: new approaches to knowledge and the semantic web. **New Review of Information Networking**, Taylor & Francis, v. 7, n. 1, p. 159–183, 2001. Citations on pages 16 and 79.
- VERNADAT, F. B. Technical, semantic and organizational issues of enterprise interoperability and networking. **Annual Reviews in Control**, v. 42, n. 4, p. 728–733, 2009. ISSN 1474-6670. Citations on pages 77, 79, 81, 82, 83, and 84.
- WEGNER, P. Interoperability. **ACM Computing Survey**, ACM, New York, USA, v. 28, n. 1, p. 285–287, 1996. Citation on page 58.
- WELLHAUSEN, T.; FIESSER, A. How to write a pattern? a rough guide for first-time pattern authors. In: **European Conference on Pattern Languages of Programs (EuroPLOP)**. Irsee, Germany: ACM, 2011. p. 1–9. Citation on page 22.
- Wu, J.; Liu, H.; Wu, F. A recommender system based on car pairwise comparisons on a mobile application using association rules. In: **International Conference on Industrial Technology (ICIT)**. Taipei, Taiwan: IEEE, 2016. p. 1344–1346. Citation on page 46.
- ZARKO, I. P.; SOURSOS, S.; GOJMERAC, I.; OSTERMANN, E. G.; INSOLVIBILE, G.; PLOCIENNIK, M.; REICHL, P.; BIANCHI, G. Towards an iot framework for semantic and organizational interoperability. In: **Global Internet of Things Summit**. [S.l.: s.n.], 2017. p. 1–6. Citation on page 84.
- ZHU, W. Semantic mediation bus: An ontology-based runtime infrastructure for service interoperability. In: IEEE. **2012 IEEE 16th International Enterprise Distributed Object Computing Conference Workshops**. [S.l.], 2012. p. 140–145. Citation on page 82.
- ZIMMERMANN, O. Microservices tenets: Agile approach to service development and deployment. **Computer Science-Research and Development**, Springer, v. 32, n. 3-4, p. 301–310, 2017. Citations on pages 62, 79, and 80.
- ZOPPI, T.; CECCARELLI, A.; BONDAVALLI, A. Exploring anomaly detection in systems of systems. In: **Symposium on Applied Computing (SAC)**. Marrakech, Morocco: ACM, 2017. Citation on page 2.

Survey Questionnaire

A.1 Initial Considerations

This appendix presents the questionnaire used to carry out the survey on how architectural strategies for interoperability of software-intensive systems have been considered by practitioners in their integration projects in the industry context.

This appendix is organized as follows: Section [A.2](#) presents the consent form for the conduct of the research study. Section [A.3](#) outlines the questions about participant's attributes, such as demographic characteristics, profession, and experience in integration projects. Section [A.4](#) shows the questions used to investigate the reasons for adopting architectural styles in integration projects. Section [A.5](#) describes the questions used to investigate the reasons for adopting architectural patterns in integration projects. Section [A.6](#) presents the questions used to investigate which is the difficulty level for applying each architectural strategy. Section [A.7](#) outlines the qualitative questions about the agreement level of the participant regarding the architectural strategies. Section [A.8](#) presents the questions to investigate the main challenges observed by practitioners in their integration projects. Finally, Section [A.9](#) describes the questions about name, e-mail, whether the participant is interested in receiving a digital certificate, and also the results of the survey.

A.2 Consent for the Research Study

You are being asked to take part in a research study. This study is being conducted to validate a typology of architectural strategies for interoperability in the industrial context. Participation in the study typically takes 20 minutes and is strictly anonymous.

Participants begin by answering questions about their attributes, such as demographic characteristics, profession, and experience in integration/interoperability project problems. Following, participants answer questions about the use of architectural strategies for interoperability in their integration projects. Finally, participants answer questions about the challenges faced during the integration of software. All responses are treated as confidential, and in no case will responses from individual participants be identified.

All data will be coded so that your anonymity will be protected in any research papers and presentations that result from this work. Participation is voluntary, refusal to take part in the study involves no penalty or loss of benefits to which participants are otherwise entitled, and participants may withdraw from the study at any time.

We will provide a digital CERTIFICATION for participants who require it.

If participants have further questions about this study or their rights, or if they wish to lodge a complaint or concern, they may contact the main researchers.

If you understand the statements above, and freely consent to participate in the study, click on the “I Agree” button to begin the experiment.

To exit, simply close the browser window.

A.3 Participants

Questions about participant’s attributes, such as demographic characteristics, profession, and experience in integration projects.

Localization

- Europe
- Asia
- North America
- South America
- Africa
- Oceania
- Antarctica

Your current role

- Developer
- System analyst
- Architect
- Tester
- Requirements engineer
- Project manager
- Researcher
- Other

Domains of integration projects that you worked (e.g., healthcare, embedded systems, smart cities, among others)

Size of the organization that you work

- Small (< 50 employees)
- Medium (51- 250 employees)
- Large (251 - 999 employees)
- Extra-large (> 1000 employees)

Kind of organization that you work

- University
- Research center
- Enterprise/Company
- Other

Your experience in integration projects

- < 2 years
- 2 - 5 years
- 5 - 8 years
- > 8 years

Amount of integration projects that have you worked

- < 2 projects
- 2 - 5 projects
- 5 - 8 projects
- > 8 projects

A.4 Architectural Style

This section aims to investigate the reasons for adopting architectural styles in integration projects. Please, select the best option(s) that describe why you have used (or not) the following styles. Choose at least one option for each style.

Microservices

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

SOA

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability

- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

Layer

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

REST

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

File Transfer

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

Remote Procedure Invocation

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

Messaging

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability

- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

Shared DataBase

- I've used the style for addressing Technical Interoperability
- I've used the style for addressing Syntactic Interoperability
- I've used the style for addressing Semantic Interoperability
- I've used the style for addressing Organizational Interoperability
- I haven't used the style because the added cost
- I haven't used the style because the learning cost
- I haven't used the style because the lack of documentation
- I haven't used the style because the lack of support tools
- I haven't used the style because it wasn't considered as necessary

A.5 Architectural Pattern

This section aims to investigate the reasons for adopting architectural patterns in integration projects. Please, select the best option(s) that describe why you have used (or not) the following patterns. Choose at least one option for each pattern.

Pipes and Filters

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability

- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Scatter/Gather

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Centralized Architecture

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools

- I haven't used the pattern because it wasn't considered as necessary

Mediator

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Contract Monitor

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Bus

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability

- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Adapter

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Wrapper

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation

- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Broker

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Facade

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Orchestration

- I've used the pattern for addressing Technical Interoperability

- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Coreography

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Discover Service

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost

- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

Tailor Interface

- I've used the pattern for addressing Technical Interoperability
- I've used the pattern for addressing Syntactic Interoperability
- I've used the pattern for addressing Semantic Interoperability
- I've used the pattern for addressing Organizational Interoperability
- I haven't used the pattern because the added cost
- I haven't used the pattern because the learning cost
- I haven't used the pattern because the lack of documentation
- I haven't used the pattern because the lack of support tools
- I haven't used the pattern because it wasn't considered as necessary

A.6 Difficulty Level

Microservices

Which is the difficulty level for applying microservices in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Select the level of impact over the next quality attributes that you have observed at using microservices in integration projects.

SOA

Which is the difficulty level for applying SOA in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using SOA in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Layers

Which is the difficulty level for applying Layers in integration projects?

- Very low

- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Layers in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

REST

Which is the difficulty level for applying REST in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using REST in integration projects.

File Transfer

Which is the difficulty level for applying File Transfer in integration projects?

- Very low

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using File Transfer in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Shared Database

Which is the difficulty level for applying Shared Database in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Shared Database in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

RPI

Which is the difficulty level for applying RPI in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using RPI in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Messaging

Which is the difficulty level for applying Messaging in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Messaging in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Pipes and Filters

Which is the difficulty level for applying Pipes and Filters in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Pipes in integration projects.

Scatter/Gather

Which is the difficulty level for applying Scatter/Gather in integration projects?

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Scatter/Gather in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Centralized Architecture

Which is the difficulty level for applying Centralized Architecture in integration projects?

- Very low
- Low
- Medium
- High

- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Centralized Architecture in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Mediator

Which is the difficulty level for applying Mediator in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Mediator in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Contract Monitor

Which is the difficulty level for applying Contract Monitor in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Contract Monitor in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Bus

Which is the difficulty level for applying Bus in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Select the level of impact over the next quality attributes that you have observed at using Bus in integration projects.

Adapter

Which is the difficulty level for applying Adapter in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Adapter in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Wrapper

Which is the difficulty level for applying Wrapper in integration projects?

- Very low

- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Wrapper in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Broker

Which is the difficulty level for applying Broker in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Broker in integration projects.

Facade

Which is the difficulty level for applying Facade in integration projects?

- Very low

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Facade in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Orchestration

Which is the difficulty level for applying Orchestration in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Orchestration in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Coreography

Which is the difficulty level for applying Coreography in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Coreography in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Discover Service

Which is the difficulty level for applying Discover Service in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Discover Service in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

Tailor Interface

Which is the difficulty level for applying Tailor Interface in integration projects?

- Very low
- Low
- Medium
- High
- Very high

If high/very high, why?

Select the level of impact over the next quality attributes that you have observed at using Tailor Interface in integration projects.

	Highly negative	Lowly negative	None impact	Lowly positive	I don't know
Security					
Performance					
Functionality					
Reliability					
Usability					
Portability					
Maintainability					

A.7 Background

Qualitative questions about the agreement level of the participant regarding the architectural strategies.

1. Which technologies have you worked with in integration projects?
2. Have you used reference architectures in integration projects?
 - Yes
 - No
 - I'm not familiar with the term of reference architecture
3. Please, describe some benefits you have observed at using reference architectures in integration projects?
4. Name or link to consult this architecture

A.8 Challenges

1. Which are the main challenges you have observed during the integration of software?
2. How the process of integrating software systems could be improved? (e.g, in terms of, agility, simplicity, documentation, or among other).
3. Additional comments to authors

A.9 Contact

1. Are you interested in receiving a digital certificate for your kindly participation in this survey?
 - Yes
 - No
2. Name to appear in the certificate
3. E-mail to send the certificate
4. If necessary, could we contact you for resolving possible doubts?
 - Yes
 - No
5. Contact E-mail
6. Are you interested in receiving the results of this survey?
 - Yes
 - No

A.10 Final Considerations

This questionnaire supported the conduction of the survey to investigate the way that practitioners have been tackled interoperability on software-intensive systems. This questionnaire was implemented using LimeSurvey, and it was available for three months. In this period, we sent three times the invitation to the target audience of study using the selected e-mail list and professional and social networks, such as LinkedIn and Facebook.