Journal of Universal Computer Science, vol. 31, no. 7 (2025), 713-734
submitted: 13/6/2024, accepted: 16/1/2025, appeared: 28/6/2025 CC BY 4.0

CIAS: Catalog of Interoperability Architectural Solutions for Software Systems

Pedro Henrique Dias Valle
(University of São Paulo, São Paulo, Brazil
https://orcid.org/0000-0002-6929-7557, pedrohenriquevalle@usp.br)

Elisa Yumi Nakagawa
(University of São Paulo, São Carlos, Brazil
https://orcid.org/0000-0002-7754-4298, elisa@icmc.usp.br)

Abstract: Context: Software systems have become increasingly large and complex, and are required in several critical domains, including Industry 4.0, the military, smart cities, and transportation. Consequently, the architectural design of these systems becomes considerably complicated, in addition to requiring interoperability among diverse systems that sometimes comprise them. Problem: Although many interoperability architectural solutions exist, software architects have struggled to comprehend, analyze, and select the most suitable ones to solve their problems. Objective: This work provides a catalog of the main interoperability architectural solutions (patterns, styles, tactics, and approaches) for addressing the four levels of interoperability (namely, technical, syntactic, semantic, and organizational) to resolve interoperability issues in software systems. Method: 65 studies found systematically in the scientific literature were deeply examined and provided evidence to define our catalog, which comprises interoperability issues and architectural solutions to address these problems. Results: As a contribution, this catalog could help software architects better decide which architectural solutions could solve each interoperability issue in their integration projects.

Keywords: Architectural Solution, Interoperability, Software Systems
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1
DOI: 10.3897/jucs.129692

1 Introduction

In the scientific literature, some reports show that researchers and practitioners have investigated how software systems can communicate among them [Valle et al. 2021, Maciel et al. 2024]. In particular, this concern is even more significant in software systems, due to the complexity of this class of systems [Baldwin et al. 2017, Bouziat et al. 2018]. Software systems are present in connected health, Industry 4.0, smart cities, the military, education, and others.

In particular, software systems are often developed, operated, managed, and evolved independently. These independent systems can exchange information among themselves to achieve their goal. Through that interaction, they may form a new kind of complex system, in other words, a system composed of systems [Bouziat et al. 2018]. In short, software-intensive systems are composed of individual systems that need to exchange information to fulfill a specific purpose. Their operations depend on interactions among other heterogeneous, distributed systems, and sometimes independent, large-scale, and software-intensive [Bouziat et al. 2018]. The individual systems that participate in larger systems are developed in a distributed way and, consequently, have different life cycles, environments, programming languages, platforms, and business interests [Madni and Sievers 2014]. This is because, in general, these systems are created and managed by different organizations. Usually, the development of these systems includes a diversity of stakeholders and multidisciplinary practitioners, who are involved during their whole life cycle, e.g., requirements specification, architectural design, development, testing, deployment, maintenance, and evolution [Valle et al. 2019, Valle et al. 2025].

In this perspective, software engineers have faced difficulties developing software systems due to collaborations among their systems [Abukwaik and Rombach 2017]. Therefore, architects and developers of these systems are encouraged to adopt solutions to integrate heterogeneous, distributed, and independent systems [Kubicek et al. 2011]. In this sense, interoperability is essential in composing complex software 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 and Sievers 2014]. IEEE 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*" [IEEE 2000].

Interoperability has been widely classified into different levels [Rezaei et al. 2014]. For this paper, we explore the four interoperability levels [Ibrahim and bin Hassan 2010]: technical, syntactic, semantic, and organizational interoperability. Interoperability is a complex task because it requires the organizations involved to employ some degree of compatible semantics and common interpretations of the information exchanged [Abukwaik and Rombach 2017]. In short, when a system needs to exchange information, it must 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 and Sievers 2014]. Therefore, interoperability has been claimed to be essential for successfully establishing software systems. Practitioners have faced several interoperability challenges in their projects [Valle et al. 2021] . 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; 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 and means to consider interoperability in software systems at the architectural level [Garcés et al. 2021].

Software architectures are essential because decisions made at the architectural level directly enable, facilitate, or interfere in achieving business goals and functional and quality requirements [Bass 2013, Valle et al. 2021b]. Therefore, different architectural strategies have been proposed to promote the reuse of good architectural designs for different quality requirements [Muketha et al. 2014]. Furthermore, architectural solutions (as patterns) have been proposed for interoperability requirements to address this quality attribute at the architectural level. Despite the critical role of interoperability in the composition and execution of software systems, architects still face challenges that hamper the building and operation of those systems effectively and efficiently. We identified challenges faced by practitioners in their projects: (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 regarding the vocabulary of the data exchanged among systems. In this sense, the main problem addressed in this paper is the difficulty software architects face in considering interoperability in the software architectures of software systems that need to exchange useful data among them.

Based on this scenario, the main contribution of this paper is to provide architects with means to decide how to mitigate interoperability issues in software systems considering architectural solutions available in the scientific literature and industry. For this, we defined a catalog comprising the main issues of each interoperability level and their respective solutions. We named it CIAS (**C**atalog of **I**nteroperability **A**rchitectural **S**olutions for Software Systems). CIAS could help architects decide which architectural solutions can be considered to solve the issues in their integration projects.

The paper is structured as follows: Section 2 presents the main concepts related to this study and the related work. Section 3 provides the research method. Section 4 presents CIAS. Section 5 comprises the discussion of the preliminary results of our work and the main threat to the validity of our work. Section 6 shows possible threats to validity and actions performed to mitigate them. Finally, Section 7 provides the final considerations and future research perspectives.

2 Background and Related Work

This section presents the definition of interoperability, considering the four levels (technical, semantic, syntactic, and organizational). Moreover, software architecture concepts and architectural solutions are described, as well as the related work to this study.

2.1 Interoperability

Software systems interoperability can be defined as the "ability of two or more systems or components to exchange and understand information and to use information that has been exchanged correctly" [IEEE 2000]. In the literature, interoperability has been extensively classified into several types [Rezaei et al. 2014]. For this research, we intend to explore the four types of interoperability:

Technical interoperability concerns data transmission between several components or systems, encompassing the hardware and software elements, networks, and equipment that facilitate the communication between machines. Key aspects include open interfaces, data integration, middleware, data presentation, data exchange, accessibility, and security considerations [Ibrahim and bin Hassan 2010].

Syntactic interoperability involves the regulations governing the structure of exchanged data, delineating how data is organized and its sequential arrangement [van der Veer and Wiles 2008, Muketha et al. 2014]. This interoperability type discerns the elements and rules dictating data structure, encompassing aspects such as data structure, mapping, bridging, and well-defined syntax (e.g., message content structure, size of headers, size of the message body, fields contained in a message), and navigation among equivalent elements [Muketha et al. 2014].

Semantic interoperability focuses on the exact meaning of exchanged data. In this interoperability type, data is treated as information meant to be shared, processed, and understood without ambiguity by systems [Muketha et al. 2014]. Hence, semantic interoperability ensures that the exact meaning of exchanged information is understood by any other system not initially designed for this purpose.

Organizational interoperability pertains to effectively coordinating distributed workflows and activities, a comprehension shared by systems, organizations, or individuals involved in business processes [Muketha et al. 2014]. This type of interoperability pertains to the ability of two or more components/systems to provide and accept services from other components/systems, allowing them to operate together effectively [Muketha et al. 2014].

2.2 Software Architecture

Software architectures have an important role in software development [Garlan 2007, Valle et al. 2020] because they provide a bridge between requirements and code, impacting the understanding of large systems, reuse of components and frameworks, construction of partial views for development, the evolution of systems, analysis of descriptions, management of software development process, and communication among stakeholders [Clements et al. 2002]. In particular, software architectures aim to cover, at design time, important quality attribute requirements, for instance, performance, maintainability, or interoperability.

In particular, architectural synthesis is a *core* activity during the architectural process of a software system. This activity is quite challenging, mainly due to the necessity of defining a *good* architectural design that addresses essential quality attributes requirements in a system. In this sense, different architectural solutions have been proposed to promote the reuse of good architectural designs for additional quality attribute requirements. Architectural patterns capture well-proven design structures, facilitating reuse. These patterns can be considered as a package of design decisions that are found repeatedly in practice [Muketha 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].

2.3 Related Work

Regarding the related work, some studies have reported architectural solutions to achieve interoperability in integration projects. In particular, Spalazesse (2010) described an architectural pattern called Mediating Connector, which is the key enabler for the interoperability of software systems. In addition, they presented a set of Basic Mediator Patterns that describe the basic mismatches that can occur when components try to interact and their corresponding solutions.

From another perspective, Burns et al. (2019) examined the progress made to establish interoperability across a diverse set of systems and identified the challenges in establishing this level of interoperability. To achieve this, the authors carried out a literature review of current Industry 4.0 technologies and current interoperability standards to identify and categorize potential frameworks capable of providing an Industry 4.0 global interoperability standard. [Zhang et al. 2017] discussed the potential of blockchain technology to address healthcare interoperability issues and improve medical practice workflow. The authors presented a case study of a blockchain-based healthcare app called DASH and identified implementation challenges such as tightly coupled designs, duplicated resources, and lack of scalability. The paper proposes using four software patterns - Abstract Factory, Flyweight, Proxy, and Publisher-Subscriber - to address these challenges and improve the design and functionality of DASH. The authors conclude that combining good software design practices with the unique properties of blockchain can create more modular and easily maintainable smart contracts.

Despite some initiatives on using interoperability architectural patterns for software systems, there is still a lack of studies demonstrating how architectural patterns can be used to solve interoperability issues, showing which patterns are more suitable for the main types of existing issues for interoperability. In this sense, the existence of the catalog proposed in this work is justified.

3 Research Method

We defined a research process to define the catalog of interoperability issues and their possible architectural solutions for software systems. Figure 1 presents the overview of this process, which was divided into five steps and can be applied interactively, as presented in the following:

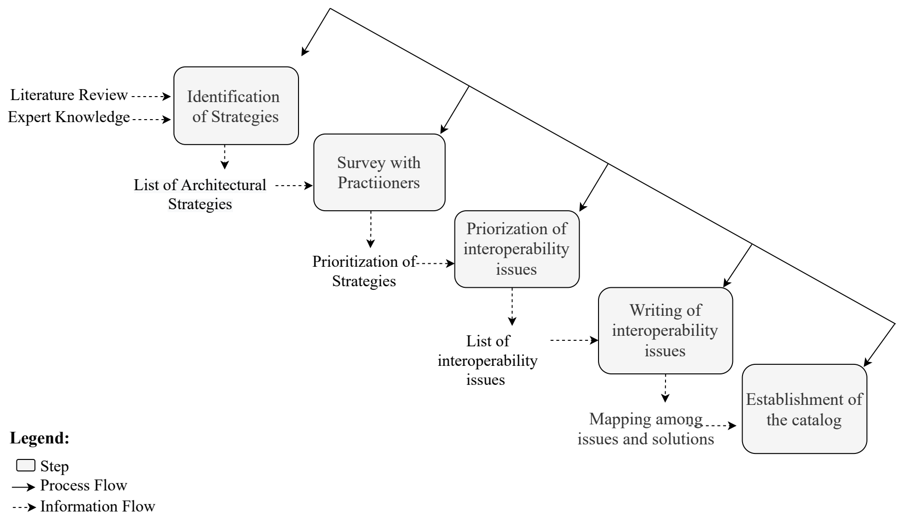
Literature Review
Expert Knowledge
Identification of Strategies
List of Architectural Strategies
Survey with Practiioners
Prioritization of Strategies
Priorization of interoperability issues
List of interoperability issues
Writing of interoperability issues
Mapping among issues and solutions
Establishment of the catalog
Legend:
Step
Process Flow
Information Flow

Figure 1: Process to build the catalog of interoperability architectural solutions

– **Step 1 - Identification of architectural strategies**: Architectural strategies are usually reusable solutions to common issues in software architecture in a given context. Furthermore, these solutions are documented according to the experience and practice of practitioners or experts. The starting point for identifying architectural strategies was to search publications available in the literature. Hence, we first conducted a literature review in two main databases (Google Scholar and Scopus) and searches in architectural solutions repositories[1][2][3][4][5]. As a result, we identified

[1] http://software-pattern.org/Category/5
[2] https://www.microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/
[3] https://learn.microsoft.com/en-us/azure/architecture/patterns/materialized-view
[4] https://www.enterpriseintegrationpatterns.com/
[5] https://wayback.archive-it.org/12090/20200210174143/https://ec.europa.eu/idabc/?id=19529

22 strategies for interoperability at the architectural level, as reported in [Valle et al. 2019].

- **Step 2 - Survey with practitioners:** We conducted a survey with practitioners from different countries to understand how the 22 strategies for interoperability (which we collected previously) have been applied in their integration projects, also helping us to validate if such strategies are correct. For this, we elaborated a questionnaire with the following research questions (RQ): RQ1: Which architectural solutions have practitioners used to resolve interoperability issues in software systems?; RQ1.1: What is the difficulty level of applying such strategies in integration projects?; RQ1.2: What is the impact level over the quality attributes of using such strategies in integration projects?; RQ2: How could the process of integrating software systems be improved? (e.g., in terms of agility, simplicity, and documentation); RQ3: Which are the technologies used in integration projects?; and RQ: Which are the main challenges perceived by practitioners during the software integration?. As a result, we received 33 responses from participants that helped us understand how these strategies could be applied at each interoperability level. Regarding the characterization of participants, South America is the localization where more respondents work (45%), followed by Europe (33%), North America (12%), and Asia (9%). The respondents have performed in more than one role in their organization. 61% of practitioners also identified themselves as researchers in their companies; 27% are architects, 15% are project managers, 6% are developers, and 3% are product owners, testers, network analysts, or system analysts. Regarding the experience in integration projects, 85% had more than two years of experience, from which 55% accumulated more than eight years participating in projects for software integration and leading with interoperability issues. Finally, the most representative domains are healthcare and smart cities, with 15% of practitioners working on them. Other representative domains are manufacturing, government, IoT (Internet of Things), automation technology, and education, as reported in [Valle et al. 2021, Valle 2021].
- **Step 3 - Prioritization of interoperability issues:** Based on the information collected from the two sources—scientific literature and survey —we prioritized the main interoperability issues at each interoperability level. These issues were reported both in the scientific literature and by the industry through the survey. The prioritization process was made based on the experience of practitioners who participated in our survey, ensuring that the most critical and relevant issues were emphasized. As a result of this process, we identified 11 interoperability issues that software architects faced when they needed to ensure interoperability. Addressing these prioritized issues is essential for improving system interoperability and communication.
- **Step 4 - Writing of Interoperability Issues:** In this phase of the process, we focused on a comprehensive writing of each interoperability issue that had been previously selected. For each identified issue, we provided a thorough description outlining the specific challenges that architects and developers faced within the industry. This step plays a significant role in enhancing research on interoperability by presenting a consolidated overview of the primary challenges affecting the architectural level of systems. This overview could be considered a valuable resource for both academic researchers and industry practitioners, guiding future studies to lead with interoperability issues effectively.
- **Step 5 - Establishment of the catalog:** Finally, we mapped issues and solutions,

showing which architectural solutions can mitigate or even solve interoperability issues. In this context, the practitioners can analyze which architectural solutions they must consider in their projects.

It is important to highlight that Steps 1 and 2 were reported in previous works [Valle et al. 2021, Valle et al. 2019]. We intend to provide data on the identified architectural strategies, the interoperability problems, and the validations carried out through the survey.

4 Catalog of Interoperability Architectural Solutions

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

4.1 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 hardware and software components, networks, and equipment enabling machine-to-machine communication, including open interfaces, connectivity, data integration, accessibility, and security issues [van der Veer and Wiles 2008, Ibrahim and bin Hassan 2010]. In the following, we describe three main interoperability issues for the technical level that the architects have faced:

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 low 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:** It can make it difficult for heterogeneous systems to achieve interoperability, making 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 [Ibrahim and bin Hassan 2010]

3. **Different communication protocols:** in the context of the software systems, the individual systems can use different protocols to communicate with other 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 and bin Hassan 2010, Farooq et al. 2020]. They can be implemented by hardware, software, or a combination. 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].

For each identified issue, we suggested architectural solutions to solve or mitigate the identified interoperability issue at the technical level. The suggested architectural solutions are based on the information provided in the literature and the survey we conducted [Valle et al. 2019, Valle et al. 2021]. Table 1 summarizes the main issues and solutions.

Issues	**Solutions**	**Description**	**Ref.**
Peer-to-Peer	Share Database	It stores the data of the system in a common shared database/repository, where then they are consulted by other systems that need to interoperate.	[Guo et al. 2011, Repositorio 2021]
	File Transfer	It produces files containing the information required by the other systems for each system.	[Kubicek et al. 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 is widely used in Web-based systems, providing a set of restrictions for communication and data format.	[Al-Zoubi and Wainer 2010, Rezaei et al. 2014]
	Messaging	It transfers packets of data, immediately, reliably, and asynchronously and it helps in the definition of infrastructure for systems to interoperate.	[Chen et al. 2008, Repositorio 2021]
Different Communication Protocols	Adapter	It transforms protocols used by heterogeneous systems to a common standard.	[Keshav and Gamble 1998, Harrer et al. 2008]

Table 1: Architectural solutions to solve interoperability issues at the technical level

The first issue is 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 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* can be used to solve peer-to-peer communication issues. This architectural solution makes it possible for heterogeneous systems to be integrated, storing their data in a common shared database/repository, where then they are consulted by other systems that need to interoperate [Guo et al. 2011]. This solution is most common in extensive data systems because data produced by different entities are homogenized and stored in a central repository [Repositorio 2021].

File Transfer is another possible solution for this issue. For each system, it produces files, containing the information required by the other systems [Kubicek et al. 2011]. A benefit is less risk of misplaced or misdirected files because the production can automatically route sensitive files to and from the correct locations and follow proper workflows [Repositorio 2021]. *RPI* (Remote Procedure Invocation) uses the principle of encapsulation to interoperate. For this, it is required that systems be developed as large-scale objects or components with encapsulated data [Maybee et al. 1996]. Thus, it is possible to provide an interface to allow other systems to interact with the running systems. For example, when a system needs data 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. This issue concerns the software and hardware that communicate among heterogeneous systems. *REST (Representational State Transfer)* has been used to mitigate this issue. It is widely used in web services and microservices, and its correct implementation is denominated RESTful [Al-Zoubi and Wainer 2010]. In this sense, a set of communication and data format restrictions should be defined so that there is communication among heterogeneous systems [Rezaei et al. 2014, Repositorio 2021].

Messaging can transfer data packets 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 architectural solution also helps define infrastructure for the heterogeneous systems to interoperate [Chen et al. 2008]. The infrastructure can be established through a message bus that transfers data packets among systems asynchronously using customizable formats [Repositorio 2021].

It is worth highlighting that the two interoperability issues aforementioned have differences. In particular, peer-to-peer communication refers to the method by which devices communicate directly without needing a central server. The focus is on the communication model and implementation, while lack of infrastructure refers to the absence or inadequacy of the fundamental systems required to support specific operations, including communication. This issue concerns the physical or organizational resources needed to establish a network. Both issues can affect connectivity and the ability to share data seamlessly. Lack of infrastructure can prevent effective peer-to-peer communication due to insufficient supporting systems. Therefore, the difference between these issues is that the former concerns the communication style, while the latter concerns the availability of resources to support any communication model.

Finally, the last interoperability issue addressed at the technical level is using **different communication protocols** by the heterogeneous systems. *Adapter* 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 architectural solution can also act as a messaging client to the messaging system and invoke systems functions via a system-supplied interface. So, any system can connect to the messaging system and be integrated with other systems [Keshav and Gamble 1998].

4.2 Syntactic Interoperability

Syntactic interoperability is based on data coding, using markup languages to develop systems, document management models, electronic records, and formats of information presentation [van der Veer and Wiles 2008, Muketha et al. 2014]. Issues at the syntactic 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 (Extensible Markup Language)-based standards [Maciel et al. 2017]. We identified two syntactic interoperability issues that have occurred in integration projects according to evidence obtained in the scientific literature and reports provided by architects and developers, as described below:

1. **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 [Diván and Sánchez Reynoso 2018]. These systems use different data formats to represent and store their information, making it challenging to achieve interoperability at the syntactic level [Muketha et al. 2014]. Therefore, the different data formats challenge building a structure and syntax that enables the information to be ascertainable by both systems that must communicate [Veltman 2001, Muketha et al. 2014].

2. **Encoding incompatible rules:** Different organizations often develop software systems 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 (Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure), SMTP (Simple Mail Transfer Protocol), MIME (Multipurpose Internet Mail Extensions), JMS (Java Message Service) or SOAP (Simple Object Access Protocol) over TCP/IP (Transmission Control Protocol/Internet Protocol) [Vernadat 2009]. Despite that, many systems have problems with incompatible encoding rules. It has hampered communication among heterogeneous systems, causing difficulty in achieving interoperability at the syntactic level. Therefore, identifying elements and rules for structuring, mapping, and crosswalks among equivalent elements should be established [Veltman 2001, Muketha et al. 2014].

As we listed above, we defined solutions that could be used in integration projects to solve each identified issue. In short, Table 2 shows an overview of architectural solutions proposed for each identified interoperability syntactic issue.

The **different data formats** are one of the main challenges when software systems must interoperate at the syntactic level. From this perspective, architectural solutions have been proposed to mitigate this issue. The *SOA* (Service-Oriented Architecture) has been used to help the publication of capabilities using services with well-defined formats and interfaces [Zimmermann 2017]. The *SOA* represents the strategy most adopted to support syntactic interoperability among multichannel and composite real-time applications implemented within large-scale distributed environments [Gazzarata et al. 2017]. This architectural solution is formed by network accessories that can encapsulate the functionality and information of software systems. It provides them through well-defined interfaces, facilitating interoperability among different systems [Gazzarata et al. 2017].

The *Microservice* has also been used to support software systems to achieve interoperability when there are software systems with different data formats [Newman 2015, Zimmermann 2017]. This architectural solution can help standardize the data format of software systems that need to communicate using services through interface specifications and continuous integration [Newman 2015, Zimmermann 2017]. The

Issues	Solutions	Description	Ref.
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 et al. 2017, Zimmermann 2017]
	Microservice	It helps in standardizing 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 incompatible rules	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 and Ramesh 2019, Aydin and Aydin 2020]
	Adapter	It supports transforming data formats and adapts the coding rules that are incompatible.	[Keshav and Gamble 1998]

Table 2: Architectural solutions to solve interoperability issues at the syntactic level

Adapter encodes information in a specific message format to be understood by other software systems [Garcés 2018]. This architectural solution aggregates new data types in a message to facilitate other software systems' understanding of these messages. The *Adapter* is a fundamental strategy to transform message formats, for example, XML to JSON (JavaScript Object Notation) message specification [Keshav and Gamble 1998].

Some solutions have been proposed regarding the issue of **encoding incompatible rules** among software systems. This challenge is related to elements and rules for structuring communication. In this sense, the *Layers* can be considered to mitigate this issue in integration projects [Daliya and Ramesh 2019]. This architectural solution separates the type of information embedded in message packages, helping to identify elements and rules for structuring the information exchanged among software systems [Aydin and Aydin 2020]. Another architectural solution for this interoperability issue is the *Adapter*. Besides supporting transforming data formats, this architectural solution also helps to adapt coding rules that are incompatible [Keshav and Gamble 1998].

Other techniques can be considered to support syntactic interoperability [Rezaei et al. 2014]. So are examples: (i) BPEL (Business Process Execution Language) for easier and more flexible 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 (Yet Another Markup Language), XML.

It is important to clarify the difference between two interoperability issues at the syntactic level. "Encoding incompatible rules" and "different data formats" refer to distinct interoperability issues. Encoding incompatible rules involves the conventions and regulations that dictate how communication is structured and encoded between systems. This issue addresses the syntactic aspects of data exchange, where differing encoding standards or structural conventions can lead to compatibility. An example of this issue is that special characters, such as accented letters, may be represented differently depending on the encoding used. A system that does not support UTF-8 may not interpret these characters correctly, resulting in corrupted characters. Solutions to mitigate this issue include using Layers to help identify and separate different types of information within message packages. Additionally, Adapters can adjust these incompatible encoding rules to ensure seamless communication.

On the other hand, “different data formats” issue focuses on the syntactic representation of the data itself. This involves differences in how data is formatted, such as the choice between XML and JSON, which can obstruct data processing if not standardized, or CSV (Comma-Separated Values) files and Excel (.xlsx) files used to store tabular data. A system that exports reports in CSV may not be compatible with a system that expects data in Excel format, requiring conversion from one format to the other. To address this issue, architectural solutions like Service-Oriented Architecture (SOA) and microservices can be employed to facilitate the standardization of data formats. Adapters also play a crucial role in transforming data formats to ensure interoperability. In summary, the ”encoding incompatible rules” issue refers to different means by which data are represented, which may depend on the world’s regions, countries, legislations, and so on. Otherwise, the ”different data formats” issue refers to different computational formats adopted to represent data.

4.3 Semantic Interoperability

Semantic interoperability aims to provide systems with a way to interpret the meaning of data, information, or knowledge. However, providing 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 themselves, exchanging data and having this data interpreted by the receiving system with the same meaning as provided by the sending system [Rahman and Hussain 2020]. We identified three main interoperability issues for the semantic level. 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, as described below:

1. **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 conflicting information.

2. **Different interpretations of the same concepts:** At this interoperability level, the exchanged message should ensure the same meaning for both the sender and the receiver [Maciel et al. 2017]. Data in both messages have meaning only when 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, similar concepts are divergent, and consequently, different definitions of messages are exchanged. It is related to different interpretations of the same concept for a given application domain.

3. **Naming problems:** heterogeneous systems can work with data that has different nomenclatures [Maciel et al. 2017]. However, the data exchanged among the different systems often have synonyms (i.e., several terms with the same meaning) and homonyms (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.

This section discusses architectural solutions that can be applied to mitigate the identified issues or even solve them. These architectural solutions are well-established solutions in the scientific literature and industry to solve interoperability issues at the semantic level. Table 3 shows an overview of solutions suggested for each identified issue at the semantic interoperability level. The **inconsistent information** exchanged among heterogeneous systems is one of the significant challenges of semantic interoperability. The *Bus* can establish connectivity and transform and route messages between systems to mitigate this problem. Bus enables runtime semantic mediation within traditional SOA infrastructure, creating a semantic mediation bus [Zhu 2012]. *Bus* can also simplify message transformations based on technologies such as the XSLT (Extensible Stylesheet Language Transformation) [Arsanjani et al. 2007]. The *Adapter* can add extra information to information exchanged to solve the problem of **different interpretations of the same concept**. The added extra information helps the systems find the same interpretation for information exchanged [Garcés et al. 2018b].

Issues	**Solutions**	**Description**	**References**
Inconsistent information	Bus	It establishes connectivity and transform and route messages between among systems and enables runtime semantic mediation within traditional SOA infrastructure.	[Zhu 2012, Arsanjani et al. 2007]
A different interpretation of the same concepts	Adapter	It adds extra information to the information exchanged. The extra information added helps the systems to find the same interpretation for information exchanged.	[Garcés et al. 2018b]
Naming problems	Adapter	It standardizes the understanding of data exchanged by analyzing their synonyms and antonyms.	[Keshav and Gamble 1998, Moreira et al. 2018]

Table 3: Architectural solutions to solve interoperability issues at the semantic level

The *Adapter* 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 and Gamble 1998, Moreira et al. 2018]. Besides these architectural solutions, other techniques, such as thesaurus, taxonomies, and ontologies, can support heterogeneous systems to achieve semantic interoperability. These techniques have been used in the health domain to help understand the meaning of the data exchanged among different systems [Noura et al. 2019].

4.4 Organizational Interoperability

Organizational interoperability deals with defining business goals, aligning and coordinating business processes, and bringing collaboration capabilities to organizations that wish to exchange information and have different internal structures and processes [Vernadat 2009]. Therefore, organizational interoperability aims to address the user community's requirements by making services available, easily identifiable, accessible, and user-centric [Vernadat 2009].
We defined the main issues that occur at the organizational level and make it difficult to exchange information among software systems; in other words, they are the main inter-

operability issues. These issues result from research about interoperability and responses provided by practitioners who have faced them in their integration projects.

1. **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, organizations have different legal bases, legislation, and cultures [Vernadat 2009]. As a result, it makes it challenging to coordinate the business processes of cooperating business entities. Therefore, architectural solutions should be made when the individual systems that are part of this class of systems have different legal bases, legislation, culture, and communication needs.

2. **Different business processes :** The individual systems that are part of the software systems frequently have organizational aspects that define business processes [Muketha et al. 2014, Adamo et al. 2018]. These systems need diverse business processes to work together even when different organizations develop them. The collaboration of different business processes can compromise interoperability at the business level [Vernadat 2009].

3. **Different organizational structures:** An organizational structure describes what employees do, whom they report to, and how decisions are made across the business [Muketha et al. 2014]. 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 coordinating and communicating among the different systems that are part of software systems [Vernadat 2009].

This section presents the architectural solutions to mitigate the issues of organizational interoperability in software systems. These solutions can be found in the scientific literature and were validated through surveys with practitioners facing such issues in integration projects.

Table 4 shows the main identified issues for the organizational level and their respective possible solutions. Organizational interoperability coordinates distributed workflows and activities understood by different systems, organizations, or people interacting in business processes [Benson and Grieve 2016]. Individual systems that need to interoperate have faced **different legal bases, legislation, and cultures**. To mitigate this, practitioners, architects and developers have considered some architectural solutions. For example, the *Pipes & Filters* 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 and Gamble 1998].

The *Contract Monitor* can also mitigate the problem of different legal bases, legislation, and cultures. The architectural solution should monitor strategies that indicate precisely how and when a given contract may interact with a system with a different sender system [Ingram et al. 2015]. These architectural solutions capture the implementation requirements and the behavioral guarantees that result when these requirements are satisfied [Hallstrom et al. 2006].

The individual systems can have **different business processes**, making achieving interoperability at the business level difficult. The *SOA* and *Microservices* have been used to mitigate issues when software systems that need to interoperate have different business processes [Bass 2013]. These architectural solutions support describing organizational processes and also standardizing different business processes using languages for this purpose, such as WSDL (Web Services Description Language) and BPMN (Business

Issues	Solutions	Description	References
Different legal bases, legislation, and cultures	Pipes & Filters	It helps to obtain messages from other systems without sending requests for them.	[Keshav and Gamble 1998]
	Contract Monitor	It monitors strategies that indicate precisely how and when a given contract may interact with a system.	[Hallstrom et al. 2006, Ingram et al. 2015]
Different business processes	SOA	It supports the description of organizational processes.	[Bass 2013]
	Microservices	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 and Beqqali 2018]
Different organization structures	Centralized Architecture	It provides a central point of control facilitating proper communication among individual systems.	[Ingram et al. 2015]

Table 4: Architectural solutions to solve interoperability issues at the organization level

Process Model and Notation) [Newman 2015]. The *Orchestration* and *Choreography* also contribute to mitigating issues related to different organizational processes of individual systems that need to interoperate [Newman 2015, Benany and Beqqali 2018].

Architectural solutions have been adopted to mitigate the problem faced by practitioners regarding the **different organizational structures** of individual systems that need interoperability at the business level. For this problem, the *Centralized Architecture* can be considered to provide a central point of control, facilitating proper communication among systems [Ingram et al. 2015].

5 Discussions

Software systems, in general, can communicate with different individual systems, and interoperability has been considered essential for correctly constructing these systems. However, designing these systems is not trivial, once it involves integrating different systems developed by distinct organizations using different business rules. In this context, software architectures can contribute to building such systems, because they help understand how the components should be defined and how the communication among components should be established. Therefore, many architectural solutions have been proposed in the scientific literature and applied in the industry to support architects in developing a great solution to allow correct communication among different systems.

We group and categorize the existing solutions to help the architects mitigate system interoperability issues. The overview of existing solutions for software architects resulted in the building of CIAS. Therefore, in this work, we present the validation of CIAS using the information provided by experienced practitioners in software architecture and interoperability as a novelty to the community. Thereby, CIAS can help architects choose how to address interoperability in the architecture of software systems and how to combine the existing solutions to achieve the correct data exchanged among different systems.

This work aims to provide software architects with a catalog of the main interoperability issues and possible architectural solutions that can be used in integration projects to mitigate such issues. The catalog is divided according to the four interoperability levels and defined according to the evidence available in the scientific literature. In previous work, we also surveyed practitioners with expertise in interoperability, software architecture, and software systems to validate the suggested architectural solutions in the catalog.

Figure 2 illustrates an overview of this catalog. All information used to define the CIAS was extracted from studies that received peer review. For this, we collected information from previous work from practitioners participating in our survey. Still, 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.

As the main lessons learned and observations this work highlights: (i) the same architectural solutions can solve more than one interoperability issue, and it can also be applied at different interoperability levels; (ii) few studies detail the existing issues at each interoperability level; (iii) few studies report information on the way software architects have addressed interoperability issues in the industry; (iv) there is a lack of guidelines for supporting software architects to address interoperability in their integration projects; and (v) there is little participation of industry practitioners in research on interoperability at the architectural level.

CIAS can be used throughout the software development life cycle, particularly during the architectural design phase. Its value is helping architects make informed decisions before significant resources are invested in inadequate integration strategies. Ideally, its use can be in the requirements analysis and initial stages of system design. Hence, interoperability issues are addressed proactively rather than reactively as problems arise.

Regarding team involvement, the CIAS' intended audience is primarily software architects. While a broader team might be involved in the ultimate implementation of the chosen solutions, the decisions regarding which interoperability solutions to use would ideally be made collaboratively by a small group of experienced architects. This group may include individuals with expertise in different areas (e.g., data modeling, network infrastructure, application development).

In addition, assessing the CIAS' sufficiency for a specific project requires a multipronged approach. Firstly, it is necessary to consider a particular interoperability issue and an interoperability level to be addressed (technical, syntactic, semantic, or organizational). Then, CIAS is checked to determine whether that issue is addressed and which architectural solutions are proposed. Architects should critically review the description of those architectural solutions to check their completeness, clarity, and feasibility in the context of the project's constraints. The review might involve other professionals or researching external resources to evaluate the solutions. In short, the value of CIAS is its ability to provide interoperability solutions that address the project's particular needs and challenges.

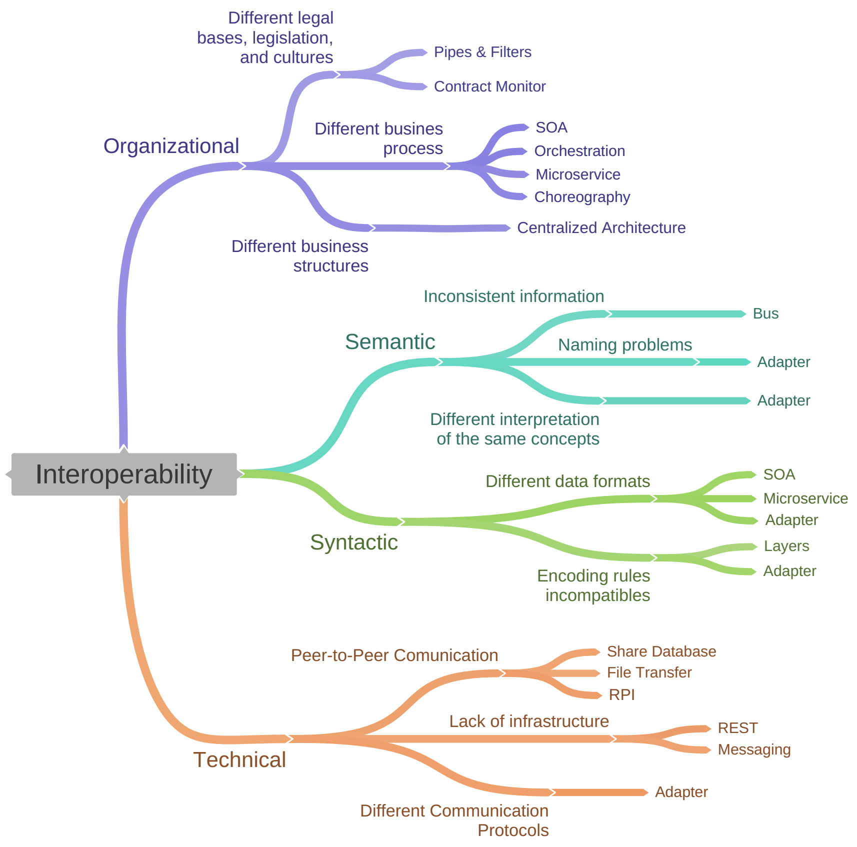

Figure 2: Catalog of architectural solutions to solve interoperability issues

It is important to emphasize the significant contribution of this paper to our scientific community is to show, through a catalog, the interoperability issues that make it difficult for systems to interoperate, classifying them according to the interoperability levels. For each highlighted issue, we also suggested solutions, also known as architectural solutions, that have been reported in the scientific literature and confirmed through a survey of the state of practice to solve these problems. Therefore, architects who face such issues can consult the catalog that will serve as a body of knowledge about interoperability issues and their respective solutions at an architectural level.

6 Threats to Validity

To minimize biases in our work, we present below the potential threats to validity and some actions performed to mitigate them.
Internal Validity: selection bias of architectural solutions - the selection of interoperability solutions and problems might have been influenced by subjective criteria or more well-known architectural solutions, potentially compromising the representativeness

of the catalog. To address this, we analyzed the interoperability architectural solutions repositories to select those relevant to our proposal. However, errors in solutions analysis, such as human errors in identifying, describing, or categorizing architectural solutions, could lead to imprecise conclusions. To mitigate this, we relied on paired analysis by two researchers.

External Validity: sample of studies and repository - research in architectural solutions repositories and studies may have omitted relevant sources, restricting the identification of the considered architectural solutions. For this, we selected databases and repositories of knowledge on computer science that have expertise in this content.

Conclusion Validity: causal inference - as the study is based on a retrospective analysis of existing architectural solutions, it is impossible to establish causal relationships between the application of these architectural solutions and the effective resolution of interoperability problems; and lack of comparison - the absence of a control or comparison group might limit the ability to identify the direct impact of architectural solutions on resolving interoperability problems.

7 Final Remarks

We proposed a catalog of interoperability solutions designed to address integration challenges at the architectural level effectively. By offering software architects and developers a comprehensive overview of existing architectural solutions, our catalog successfully mitigates the 11 interoperability issues. These issues have been thoughtfully organized into four interoperability levels, streamlining the decision-making process for architects in their integration projects. The catalog's knowledge, categorization, and prioritization empower architects to make well-informed choices on approaching each interoperability level in their specific integration endeavors. It enhances the overall efficiency of the integration process and ensures a more seamless and robust architecture.

It is worth noting that defining the pros, cons, challenges, and limitations of each architectural solution is a complex task. Such a definition may depend on the context-specific elements of each software project, such as the application domain, budget, development team's expertise, project deadlines, technology compatibility, and legal compliance. Hence, a specific analysis of each project is necessary to select the best architectural solution.

We will thoroughly evaluate the proposed catalog in future work to showcase its practical utility in resolving architectural-level integration problems. By doing so, we aim to reinforce the catalog's significance as a powerful tool for software architects in real-world scenarios. Moreover, we plan to develop a comprehensive pattern language of interoperability explicitly tailored for software systems. This expanded framework will further enrich the software architecture community, providing a more specialized and practical approach to handling interoperability challenges in complex and dynamic software environments.

Acknowledgements

This work was supported financially by CNPq (Grants: 141602/2017-1, 313245/2021-5) and FAPESP (Grant: 2023/00488-5).

References

[Abukwaik and Rombach 2017] Abukwaik, H., Rombach, D.: "Software interoperability analysis in practice: A survey"; International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM (2017), 12–20.

[Adamo et al. 2018] Adamo, G., Borgo, S., Di Francescomarino, C., Ghidini, C., Guarino, N.: "On the notion of goal in business process models"; International Conference of the Italian Association for Artificial Intelligence, Springer (2018), 139–151.

[Al-Zoubi and Wainer 2010] Al-Zoubi, K., Wainer, G.: "Rise: Rest-ing heterogeneous simulations interoperability"; Proceedings of the 2010 Winter Simulation Conference (2010), 2968-2980.

[Arsanjani et al. 2007] Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., Channabasavaiah, K.: "S3: A service-oriented reference architecture"; IT professional, 9 (2007), 10–17.

[Aydin and Aydin 2020] 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, 10, 13 (2020), 4460.

[Baldwin et al. 2017] Baldwin, W. C., Sauser, B. J., Boardman, J.: "Revisiting "The Meaning of Of" as a Theory for Collaborative System of Systems"; IEEE Systems Journal, 11, 4 (2017), 2215-2226.

[Bass 2013] Bass, L.: "Software architecture in practice"; Addison-Wesley, Massachusetts, USA (2013).

[Benany and Beqqali 2018] Benany, E., Beqqali, E.: "Choreography for interoperability in the e-Government applications"; International Conference on Intelligent Systems and Computer Vision (ISCV), IEEE (2018), 1–4.

[Benson and Grieve 2016] Benson, T., Grieve, G.: "Principles of health interoperability: SNOMED CT, HL7 and FHIR"; Springer, London, UK (2016).

[Bicer et al. 2005] 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, 34, 3 (2005).

[Bouziat et al. 2018] Bouziat, T., Camps, V., Combettes, S.: "A Cooperative SoS Architecting Approach Based on Adaptive Multi-agent Systems"; International Workshop on Software Engineering for Systems-of-Systems (SESoS), ACM (2018), 8–16.

[Burns et al. 2019] Burns, T., Cosgrove, J., Doyle, F.: "A Review of Interoperability Standards for Industry 4.0."; Procedia Manufacturing, 38 (2019), 646–653.

[Chainho et al. 2017] 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"; 2017 European Conference on Networks and Communications (EuCNC) (2017), 1-5.

[Chen 2018] Chen, J.: "Devify: Decentralized Internet of Things Software Framework for a Peer-to-Peer and Interoperable IoT Device"; ACM, New York, USA, 15, 2 (2018).

[Chen et al. 2008] Chen, D., Doumeingts, G., Vernadat, F.: "Architectures for enterprise integration and interoperability: Past, present and future"; Computers in Industry, 59, 7 (2008), 647–659.

[Clements et al. 2002] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., Little, R.: "Documenting software architectures: views and beyond"; Pearson Education (2002).

[Daliya and Ramesh 2019] Daliya, V. K., Ramesh, T. K.: "Data Interoperability Enhancement of Electronic Health Record data using a hybrid model"; International Conference on Smart Systems and Inventive Technology (2019), 318-322.

[Diván and Sánchez Reynoso 2018] Diván, M., Sánchez Reynoso, M. L.: "Fostering the Interoperability of the Measurement and Evaluation Project Definitions in PAbMM"; International

Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (2018), 231-238.

[eHGI 2017] eHealth Governance Initiative eHGI: "Discussion paper on semantic and technical interoperability" (2017).

[Farooq et al. 2020] Farooq, M. O., Wheelock, I., Pesch, D.: "IoT-Connect: An Interoperability Framework for Smart Home Communication Protocols"; IEEE Consumer Electronics Magazine, 9, 1 (2020), 22-29.

[Garcés 2018] Garcés, L.: "A Reference Architecture for Healthcare Supportive Home (HSH) systems"; Universidade de São Paulo (2018).

[Garcés et al. 2018b] Garcés, L., Oquendo, F., Nakagawa, E.: "Towards a Taxonomy of Software Mediators for Systems-of-Systems"; Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS) (2018), 53–62.

[Garcés et al. 2021] Garcés, L., Martínez-Fernández, S., Oliveira, L., Valle, P., Ayala, C., Franch, X., Nakagawa, E. Y.: "Three decades of software reference architectures: A systematic mapping study"; Journal of Systems and Software, 179 (2021), 111004.

[Garlan 2007] Garlan, D.: "Software architecture"; Wiley Encyclopedia of Computer Science and Engineering (2007).

[Gazzarata et al. 2017] Gazzarata, R., Giannini, B., Giacomini, M.: "A SOA-based platform to support clinical data sharing"; Journal of healthcare engineering, 2017 (2017).

[Guo et al. 2011] Guo, Y., Hu, Y., Afzal, J., Bai, G.: "Using P2P technology to achieve eHealth interoperability"; International Conference on Service Systems and Service Management (2011), 1-5.

[Hallstrom et al. 2006] Hallstrom, J. O., Dalton, A. R., Soundarajan, N.: "Parallel Monitoring of Design Pattern Contracts."; SEKE, Citeseer (2006), 236–241.

[Harrer et al. 2008] 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, 1, 2 (2008), 131-143.

[Ibrahim and bin Hassan 2010] Ibrahim, N., bin Hassan, M.: "A survey on different interoperability frameworks of SOA systems towards seamless interoperability"; International Symposium in Information Technology (ITSim), IEEE (2010), 1119–1123.

[IEEE 2000] IEEE: "The Authoritative Dictionary of IEEE Standards Terms"; IEEE Std 100, 2000 (2000), 1–1362.

[Ingram et al. 2015] Ingram, C., Payne, R., Fitzgerald, J.: "Architectural Modelling Patterns for Systems of Systems"; Annual International Council on Systems Engineering (INCOSE), Wiley Online Library (2015), 1177–1192.

[Keshav and Gamble 1998] Keshav, R., Gamble, R.: "Towards a taxonomy of architecture integration strategies"; International Workshop on Software Architecture (ISAW), ACM (1998), 89–92.

[Kubicek et al. 2011] Kubicek, H., Cimander, R., Scholl, H. J.: "Chapter 7 - Layers of interoperability"; Organizational Interoperability in E-Government (ICSOC), Springer (2011), 85–96.

[Maciel et al. 2017] 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).

[Maciel et al. 2024] Maciel, R., Valle, P. H. D., Santos, K., Nakagawa, E. Y.: "Systems Interoperability Types: A Tertiary Study"; ACM Computing Survey, 56, 10 (2024), 1–37.

[Madni and Sievers 2014] Madni, A. M., Sievers, M.: "System of systems integration: Key considerations and challenges"; Systems Engineering, 17, 3 (2014), 330–347.

[Maybee et al. 1996] Maybee, M. J., Heimbigner, D. M., Osterweil, L. J.: "Multilanguage interoperability in distributed systems. Experience report"; International Conference on Software Engineering (1996), 451-463.

[Moreira et al. 2018] Moreira, M. W. L., Rodrigues, J. J. P. C., 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 **89** (2018), 137–147.

[Muketha et al. 2014] Muketha, G. M., Wamocho, L., Micheni, E.: "A Review of Agent Based Interoperability Frameworks and Interoperability Assessment Models"; Scholars Journal of Engineering and Technology (SJET), 2 (2014).

[Newman 2015] Newman, S.: Building microservices: designing fine-grained systems";O'Reilly Media, Inc.", New York, USA (2015).

[Noura et al. 2019] Noura, M., Atiquzzaman, M., Gaedke, M.: "Interoperability in internet of things: Taxonomies and open challenges"; Mobile Networks and Applications, 24, 3 (2019), 796–809.

[Pang et al. 2015] Pang, L. Y., Zhong, R. Y., Fang, J., Huang, G. Q.: "Data-source interoperability service for heterogeneous information integration in ubiquitous enterprises"; Advanced Engineering Informatics, 29, 3 (2015), 549–561.

[Rahman and Hussain 2020] 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, 31, 12 (2020).

[Repositorio 2021] Repositorio: "Repositório Online de Padrões: Enterprise Integration Patterns"; http://www.enterpriseintegrationpatterns.com (2021).

[Rezaei et al. 2014] Rezaei, R., Chiew, T., Lee, S. P.: "An interoperability model for ultra large scale systems"; Advances in Engineering Software, 67 (2014), 22–46.

[Spalazzese and Inverardi 2010] Spalazzese, R., Inverardi, P.: "Mediating connector patterns for components interoperability"; 4th European Conference on Software Architecture (ECSA), Springer (2010), 335–343.

[Valle 2021] Valle, P. H. D.: "Architectural decision-making on interoperability in software-intensive systems"; Universidade de São Paulo (2021).

[Valle et al. 2019] Valle, P., Garcés, L., Nakagawa, E.: "A Typology of Architectural Strategies for Interoperability"; 13th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS) (2019), 3-12.

[Valle et al. 2020] Valle, P. H. D., Garcéss, L., Guessi, M., Martínez-Fernández, S., Nakagawa, E. Y.: "Approaches for Describing Reference Architectures: A Systematic Mapping Study"; XXIII Iberoamerican Conference on Software Engineering (CIbSE), Springer (2020), 1–14.

[Valle et al. 2021] Valle, P. H. D., Garcés, L., Nakagawa, E. Y.: "Architectural Strategies for Interoperability of Software-Intensive Systems: Practitioners' Perspective"; ACM Symposium on Applied Computing, Track Software Architecture: Theory, Technology, and Applications (SAC/SATTA 2021), ACM (2021), 1–10.

[Valle et al. 2021b] Valle, P. H. D., Garcés, L., Volpato, T., Martínez-Fernández, S., Nakagawa, E. Y.: "Towards Suitable Description of Reference Architectures"; PeerJ Computer Science (2021), 1-26.

[Valle et al. 2025] Valle, P. H. D., Tonon, V. R., Garcés, L., Rezende, S. O., Nakagawa, E. Y.: "TASIS: A typology of architectural strategies for interoperability in software-intensive systems"; Computer Standards amp; Interfaces, 91 (2025), 103874.

[van der Veer and Wiles 2008] van der Veer, H., Wiles, A.: "Achieving technical interoperability"; European telecommunications standards institute, 1 (2008).

[Veltman 2001] Veltman, K. H.: “Syntactic and semantic interoperability: new approaches to knowledge and the semantic web”; New Review of Information Networking, 7, 1 (2001), 159–183.

[Vernadat 2009] Vernadat, F. B.: “Technical, Semantic and Organizational Issues of Enterprise Interoperability and Networking”; Annual Reviews in Control, 42, 4 (2009), 728-733.

[Zhang et al. 2017] Zhang, P., White, J., Schmidt, D. C., Lenz, G.: “Applying software patterns to address interoperability in blockchain-based healthcare apps”; arXiv preprint arXiv:1706.03700 (2017).

[Zhu 2012] Zhu, W.: “Semantic mediation bus: An ontology-based runtime infrastructure for service interoperability”; 2012 IEEE 16th International Enterprise Distributed Object Computing Conference Workshops, IEEE (2012), 140–145.

[Zimmermann 2017] Zimmermann, O.: “Microservices tenets: Agile approach to service development and deployment”; Computer Science-Research and Development, 32, 3-4 (2017), 301–310.