eng

MDPI

Article

Microservice-Oriented Architecture for Industry 4.0

Ricardo Pasquati Pontarolli [1,*], Jeferson André Bigheti [2], Lucas Borges Rodrigues de Sá [3] and Eduardo Paciencia Godoy [3,*]

[1] Federal Institute of Education, Science and Technology of São Paulo (IFSP), Boituva 18552-252, Brazil
[2] National Service of Industrial Training (SENAI), Lencois Paulista 18685-730, Brazil
[3] São Paulo State University (UNESP), Sorocaba 18087-180, Brazil
[*] Correspondence: pasquati@ifsp.edu.br (R.P.P.); eduardo.godoy@unesp.br (E.P.G.)

Abstract: Industry 4.0 (I4.0) is characterized by the integration of digital technologies into manufacturing processes and highlights new requirements for industrial systems such as greater interoperability, decentralization, modularization, and independence. The traditional hierarchical architecture of Industrial Automation Systems (IAS) does not fulfill these requirements and is evolving to incorporate information technologies in order to support I4.0 applications. The integration among these technologies, equipment, and systems at different industry levels requires a migration from the legacy vertical architecture to a flat architecture based on services. Service-oriented architecture (SOA) and, more recently, microservices play a critical role in I4.0 by providing a framework for integrating complex systems and meeting those requirements. This paper presents the development of a Microservice-Oriented Architecture for Industry 4.0 (MOAI), initially focused on evolving IAS to the I4.0. The objective is to describe the development, deployment, and testing of an IAS architecture based on microservices prepared for I4.0 applications. On the contrary to developing the whole software for the industrial SOA, the MOAI was developed on top of the Moleculer framework, which allowed focusing on creating services and applications for the automation and process control industry context. The development of several microservices and security mechanisms for the MOAI is presented, as is the deployment of IAS applications as services such as process control, SCADA, discrete automation, among others. The MOAI was implemented in a process control pilot plant for experimentation. Experimental results of the MOAI for IAS applications are investigated, the microservice communication performance is evaluated, and the pros and cons of microservices for I4.0 are discussed.

Keywords: service-oriented architecture; industrial automation system; industrial internet of things; Moleculer framework

check for updates

Citation: Pontarolli, R.P.; Bigheti, J.A.; de Sá, L.B.R.; Godoy, E.P. Microservice-Oriented Architecture for Industry 4.0. *Eng* **2023**, *4*, 1179–1197. https://doi.org/10.3390/eng4020069

Academic Editors: Antonio Gil Bravo and Lech Bolesław Dobrzański

Received: 27 January 2023
Revised: 28 March 2023
Accepted: 13 April 2023
Published: 19 April 2023

1. Introduction

The goal of Industry 4.0 (I4.0) is to create a more connected, digitalized, intelligent, and autonomous manufacturing system that can improve efficiency, reduce costs, and increase quality. Most of the technologies needed for the implementation of I4.0 already exist, such as augmented reality, IPV6, wireless networks, artificial intelligence, robotics, radio frequency identification, the Industrial Internet of Things (IIoT), cloud computing, cyber-physical systems, and big data. Therefore, the biggest obstacle relies on the integration and joint use of these technologies to obtain a new manufacturing reality, where all the participants involved will be connected so that the best production decisions, safety, and values are made on demand [1–3].

The requirements for industrial systems in the context of IIoT and I4.0 must go through the recent demand for greater decentralization, modularity, and autonomy of the systems. The traditional Industrial Automation Systems (IAS) do not fulfill these requirements and need to advance to promote cooperation among different technologies, equipment, and systems at different industry levels [4,5]. The paradigm of collaborative automation has emerged as a solution to provide it through a networked architecture based on the use and sharing of services [6,7].

Operational Technology and Information Technologies (OT-IT) convergence in IAS was not possible decades ago, starting with the mainframes and direct digital control in Figure 1. The development of industrial networks and programable controllers enabled OT-IT convergence through hierarchical architectures such as the ISA-95 pyramid model. The need for data interoperability among vertical layers was met through a parallel link with the NAMUR Open Architecture (NOA). Nowadays, greater scalability and total integration can be achieved using cloud-based services or Microservice-Oriented Architectures (MOA), as shown in Figure 1 [8].

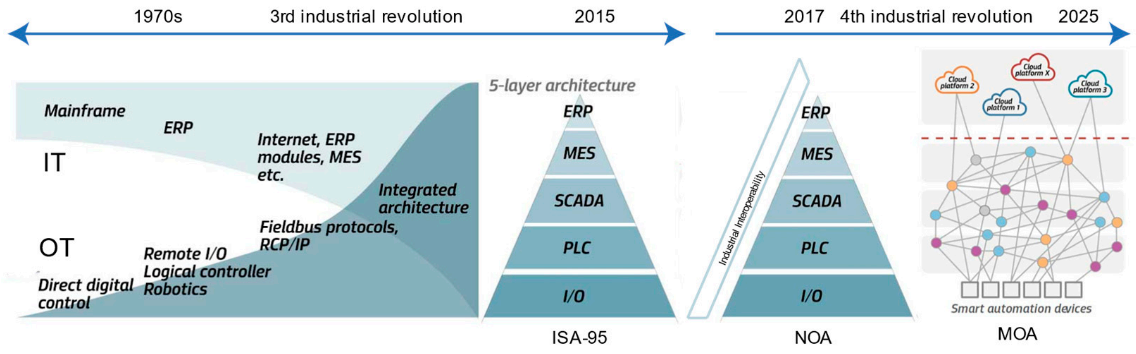

Figure 1. Convergence of OT and IT architectures adapted from [8].

References [9,10] present the NAMUR Open Architecture (NOA) to enable innovative solutions for existing process industries and adapt them to I4.0. The process control core remains largely unchanged. The basic idea is an open interface, for example OPC UA, between the existing layers of the main process control domain and the optimization monitoring domain [11]. Alternatively, a second communication channel, shown in Figure 1 (industrial interoperability), can provide direct and standard access to the existing solutions.

With I4.0, industries are using technology to create highly interconnected systems, digitalized processes, and data-driven decision-making. However, these systems often involve multiple applications and devices that need to communicate with each other seamlessly. This is where Service Oriented Architecture (SOA) comes in, as it allows for the development of modular, loosely coupled services that can be combined to create larger systems. This approach reduces complexity and increases flexibility, making it easier to adapt to changing business needs. SOA enables cross-layer integration and interoperability for decentralized systems by providing cloud services to various devices, gateways, or systems and standardizing the communication among them [7,12].

Considering the SOA application in I4.0, there is a new trend toward using microservices [13]. Microservice Oriented Architecture (MOA) represents an evolution of SOA and involves breaking up traditional applications into smaller, independent services that work together to provide the original functionality. Microservices are more independent, simple, and lightweight services with improved network communication and greater granularity (functionality) than in SOA, increasing the modularity and scalability of the application [13,14]. MOA also facilitates the reuse, redundancy, and composition of services, which can reduce development time and cost. Overall, it is a critical enabler of the principles at the core of I4.0.

The main principles of I4.0 are interoperability, virtualization, decentralization, service orientation, and modularity [15,16]. This paper presents the development of a Microservice Oriented Architecture for Industry 4.0 (MOAI), initially focused on evolving the IAS to properly support I4.0 applications. The objective is to describe the development, deployment, and testing of an IAS architecture based on microservices and discuss how it covers those principles and is prepared to support I4.0 applications.

This paper is organized as follows: This introduction presented an overview of the evolution of industrial architectures to support I4.0 and IIoT applications, with a focus on the adoption of service orientation. Section 2 presents a literature review about industrial SOA and MOA architectures and their differences from the MOAI proposal. A detailed explanation of service-oriented concepts and the Moleculer framework is given in Section 3. Section 4 describes the MOAI approach, microservices, networking, and security mechanisms. An application example of the MOAI is presented in Section 5 along with results discussions. Finally, Section 7 summarizes the conclusions of the paper.

2. Related Works

The IAS architecture has evolved to enable the integration of OT and IT technologies and support I4.0. Industrial-Ethernet networking allows integration of IAS with standard IT services. IAS covers some industry levels such as field level (devices, data acquisition), control level (programmable equipment, process continuous control, and discrete automation), supervisory level (monitoring, historian, and process management), management level (production and execution systems), and enterprise level (resources and planning systems). With the SOA's adoption, all elements (technologies, equipment, and systems) in these levels are able to communicate with each other without restrictions or hierarchical concerns [17]. These elements are services with standardized communication among them and are able to be combined, or used together, in order to create an industrial application [6].

Even with all the benefits provided by SOA for industry applications, the biggest bottleneck relies on how to enable and standardize communication and the discovery of services to integrate heterogeneous equipment and systems. Some technologies have already been applied for SOA in I4.0, such as Open Platform Communication Unified Architecture (OPC UA), Device Profile for Web Services (DPWS), and Representational State Transfer (REST) [5,7].

The OPC UA protocol is service-oriented and provides in its part four (Services) a group of essential services for networking [11]. Even though OPC UA has been widely used in I4.0 applications, the complete development of application-oriented services is required to complement the basic ones provided by the standard [13]. Reference [18] proposes an OPC UA-based SOA with three elements: a group of services for device integration, a group of application-oriented services with the necessary functionalities for the application, and a composition service in charge of controlling the sequence of service execution necessary to create an application.

The SOA development in IAS and I4.0 applications has been investigated in several projects sponsored by the European Union [12,19–21], such as the IMC-AESOP, PerFORM, and Arrowhead. In these projects, the whole software framework and network protocol to support SOA have been developed. The service development covered IAS applications ranging from the first level of devices to the level of control and supervision. DPWS communication and integration of services were used, and case studies applied to factory IAS were used to validate the projects [19,20]. The Simple Object Access Protocol (SOAP) in DPWS had undesirable performance due to protocol verbosity, even though it provided standard and secure communication and discovery capabilities for SOA. In order to overcome that, the REST standard was used in communication and service integration, resulting in better network performance (response time and jitter) for SOA [22].

The Arrowhead project [21] was based on the results of the IMC-AESOP project. It developed the concept of an on-premises automation cloud, in which IAS components and devices were made available as services, providing full interoperability for I4.0 [23]. In this project, the entire integration and communication infrastructure used SOA, structured in the format of a framework. Usually, industrial SOA architectures are based on service orchestration for the composition of applications and ease of deployment. In [24], the communication between devices using the Arrowhead framework through service choreography was presented and investigated.

The MOA has also been recently developed and tested for industry applications. Two European projects are focusing on the development of MOA for Cyber-Physical Systems (CPS), IIoT, and I4.0 applications. Reference [25] presents a MOA proposal to foster the implementation of the digital factory concept. This architecture is part of the MAYA (Multi-Disciplinary Integrated Simulation and Forecasting Tools) project, whose focus is the deployment and testing of CPS and I4.0 applications with support for the conception of digital twins [26]. In the architecture, five main groups of services are proposed that communicate via the REST standard with HTTP and WebSocket via TCP/IP. A highlight can be given to the orchestrator and scheduler microservices, which coordinate and organize the other services to enable the composition and creation of high-level services and process applications. A cloud-based MOA proposal to build a collaborative platform for I4.0 and IoT that enables real-time monitoring and optimization in manufacturing was presented in [27]. This architecture is part of the NIMBLE project [28].

Considering the literature review, the majority of SOA projects initially focused on developing the SOA software and communication protocol for industrial applications. In addition, the services and applications focused on adapting or migrating legacy IAS to the I4.0 requirements. The MOA projects are more recent and are also focusing on supporting I4.0 applications in addition to providing the required IAS functionalities. However, most of the development results of these projects are complex and even proprietary, hindering their usage. This is the main difference between them and the MOAI proposal. The MOAI was built on top of the Moleculer framework for microservices, which is open source, leveraging its development and simplifying its usage. Therefore, the first contribution of the paper is investigating the use of a non-industrial framework for microservices in industrial applications. The second difference is related to the MOAI proposal. Even though the MOAI initially covers the same required IAS functionalities as other projects in the literature, the idea was not to compare the architectures. Nevertheless, the second contribution of the paper is describing how those IAS functionalities and applications were developed using microservices and what the pros and cons of that proposal are. Considering this, this research collaborates with a MOAI, presenting the development and results of microservices initially applied to IAS [29–33].

3. Service Oriented Paradigm

In the computational context, services refer to software components that perform specific functions and are designed to be accessed by other software applications over a network through standard protocols. Examples of computational services include web services, which allow different software applications to communicate with each other over the Internet, and cloud services, which provide on-demand access to computing resources such as storage, processing power, and databases. Services are often designed to be modular and composable, allowing them to be combined with other services to build more complex applications and systems [34].

Microservices represent the evolution of the service-oriented paradigm. It refers to a style of software architecture where applications are built as a collection of small, independent, and loosely coupled services. Each microservice is designed to perform specific functionality and can be developed, deployed, and scaled independently. Microservices typically communicate with each other through well-defined, standardized, and lightweight protocols. They can be developed using different programming languages, databases, and technology stacks, as long as they conform to a common set of interfaces and standards [14].

3.1. Composition of Services

The composition of services refers to the process of integrating multiple individual services to create a more complex application or system that provides higher-level functionality. Service composition involves defining the interactions among the services and the sequence of executing them to achieve a specific goal or business process [34].

The composition of services can be achieved using different approaches, including orchestration, choreography, and an API gateway. The specific approach chosen depends on the requirements of the application or system being developed. The orchestration approach involves using a central service or middleware component to coordinate the interactions between different services. In orchestration, there is a master (composer), such as a maestro in an orchestra, that can be a service or an application that coordinates the requests of other services to make up a more complex function. The services that are requested are unaware of the complete composition, and the only one that holds this information is the master.

The choreography approach involves allowing the services to communicate with each other directly, without the need for a central orchestrator. Each service is responsible for managing its own interactions with other services based on events and messages received and predefined rules and workflows. In this type of approach, intelligence is distributed among the services involved, with each member having a part of the application's knowledge. Because the output of a service is used in the subsequent service, it is possible to use more than one means of communication.

3.2. Moleculer Framework

The Moleculer framework is an open-source and fast microservices framework for Node.js, designed to simplify the development of scalable and fault-tolerant microservices-based applications. It provides a set of features and tools to help developers build distributed systems. Microservices are developed in individual nodes with no hierarchy or priority and are designed to be agnostic to the underlying transport layer, as Moleculer supports various network protocols with the communication being automatic and transparent (no coding is needed). Moleculer also provides built-in support for service discovery, load balancing, and circuit breakers, which helps to ensure the availability and reliability of the system through the reuse and replication of microservices [35]. The microservices can provide one or multiple actions, which are specific functionalities developed that can be requested by other microservices or applications.

4. Microservice-Oriented Architecture for Industry 4.0 (MOAI)

This section presents the development of the MOAI, presented in Figure 2, describing the microservices created, security mechanisms, and microservices deployment and management. MOAI is built on the of the Moleculer framework and supports IAS functionalities in I4.0. The microservices are separated at abstract industry levels for organization and isolation of functionalities because the architecture itself is flat (without level hierarchy). Until now, microservices have been developed up to the third level (I/O, Control, and SCADA). However, future work may develop some for the top industry levels (MES, ERP).

The basic architecture infrastructure is composed of two types of components: microservices and applications. Infrastructure microservices provide functionalities for high-level composed services, and the following are available: Transporter (networking), Machine to Machine (M2M), and Data Acquisition (DAQ). In addition, microservices responsible for business or processes provide high-level composite resources such as supervision and control, and the following are available: Database, Process Control (PID4.0), and Programmable Logic Controller (PLC). IAS functionalities and applications can be created using and composing the Infrastructure and Process microservices. Both orchestration and choreography may be used for the microservice composition.

In the MOAI (Figure 2), the applications are composed of other microservices and can be Internal (blue square) or External (purple square). The Internal is based on platforms supported by Moleculer (Python, JavaScript, etc.), where the composition of microservices occurs only through the Transporter. The external is compatible with any platforms that support REST communication, which is interesting for the integration of industrial software and systems. The composition of microservices occurs initially through the API gateway and next through the Transporter for microservice communication.

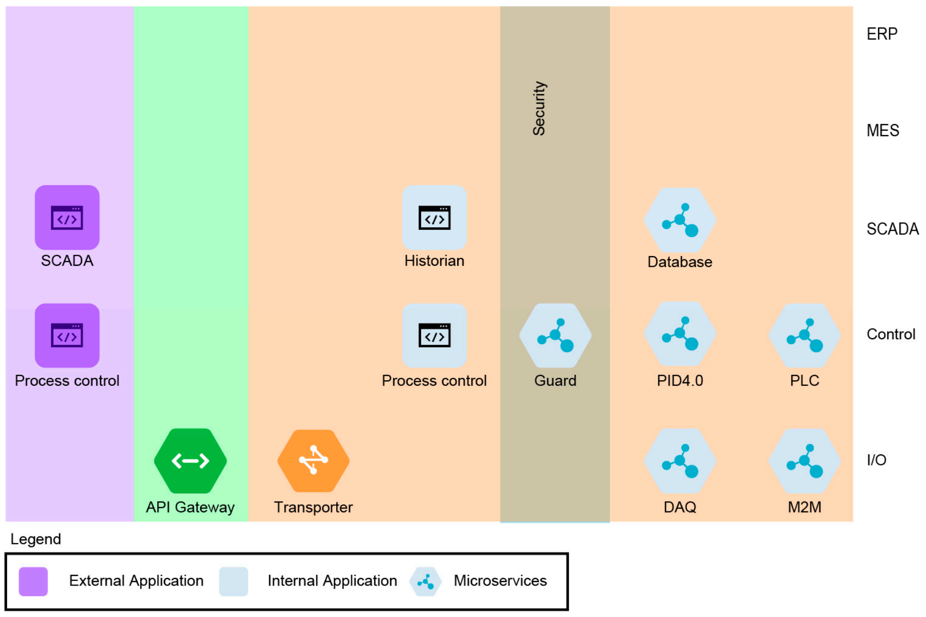

Figure 2. Microservice-Oriented Architecture for Industry 4.0.

4.1. Microservices Description

This subsection describes the microservices of the MOAI. The DAQ is in charge of accessing data from process variables through I/O hardware and has two main actions: reading the inputs (sensors) and updating the outputs (actuators). Because these actions are made transparently available by Moleculer discovery, registration, and networking, there is no need to conform to the hardware and software (coding) required to obtain the input and output data. This information is simply requested over the network.

The M2M performs the same functionalities as the DAQ but allows network communication with legacy devices, being middleware that allows obtaining information from other protocols (Modbus TCP/IP, CoAP, etc.) and making it accessible to the MOAI microservices and applications.

The PLC enables the functionalities of a programmable logic controller (PLC) at the MOAI. Actually, it comprises in a microservice the functionalities of softPLC on an open hardware and software platform. This microservice allows the execution of PLC programs (IEC 61131-3) and monitoring of input and output data and is built on top of the OpenPLC project [36].

The PID4.0 is in charge of controlling processes and is able to implement redundant controllers (running replicas of the same microservice). As a high-level microservice, it needs to be composed with other low-level microservices such as DAQ or M2M in order to obtain the input and output data. This microservice is based on a modified version of the PIDPlus control algorithm developed for network control [37]. However, it was changed to run as a microservice with redundancy for the MOAI.

The database is a microservice adapter for the InfluxDB database, which is an open-source time series database used to record historical industrial process data. The data can be requested for creating any necessary applications. IAS-based applications (internal or external) were developed through the composition of microservices such as Process Control, SCADA (supervision), and Process Historian.

The microservices in MOAI are able to be hosted on different platforms as necessary, such as embedded systems, computers, local servers, or Platform as a Service (PaaS), with different Windows, Linux, or MAC-based operating systems. All the microservices have the ServiceBroker, which is the core of the microservices and is in charge of various parameterizations as well as networking configuration (Transporter and API gateway) for communication between microservices.

4.2. Networking

The digitalization of processes and cross-layer integration demanded by I4.0 can be provided by the powerful networking mechanisms available in SOA and MOA. In the case of the MOAI, it is carried out by the networking microservice that is called Transporter (orange hexagon) and by the API Gateway (green hexagon) in Figure 2. Internal calls for microservices are handled by the Transporter and external calls are handled by the API Gateway.

The API Gateway is responsible for interfacing the MOAI microservices with external applications through HTTP(s). All actions (functionalities) from all microservices are automatically mapped to a RESTful API, which standardizes the external communication with the MOAI. It is an important characteristic for enabling integration with other IT-OT solutions. The Transporter is responsible for several networking functions, such as message communication (events, requests, and responses) among services, microservice discovery and registry, microservice load balancing, and status checking. In the MOAI, the Transporter can be seen as a broker element in a producer-consumer communication architecture.

The Transporter provides different protocols for message communication between microservices (TCP/IP, NATS, MQTT, AMQP, etc.), which are mapped to the Moleculer protocol. There is also the possibility of creating customized protocols or adding other existing protocols to them. The Transporter is defined and configured in the microservices. After configuring it, the communication between microservices and the communication between microservices and the API Gateway occurs transparently. It is also possible to change between transporters. Therefore, no matter what communication protocol is used, no additional configuration or coding is required for all message communication in the MOAI. It is an important characteristic for standardizing communication and enabling flexibility. As communication is transparent, only changes in microservices and actions are required in order to provide other functionalities that are automatically made available on the network.

Service discovery is the process of automatically detecting and locating available services in a distributed system and is an important subject in SOA and MOA. The MOAI service registry and discovery are also carried out by the Transporter. Microservice discovery is dynamic, meaning that a microservice or application does not need to know anything about other ones. When it starts, it will announce its presence to the Transporter and consequently to all the other microservices for service registry. In the event of a microservice failure (or stop), it is detected and removed from the service registry. As a result, Transporter can route calls or requests to live microservices.

Another important characteristic of the Transporter is the load balancing mechanism, as it provides availability and microservice redundancy. In the MOAI, microservices can be reused or even replicated (copies). If multiple instances of a microservice are running on different nodes (location/hardware), the Transporter balances the calls/requests between them according to the defined strategy. Available strategies are Round-Robin, Random, CPU (it is requested the microservice with the lowest CPU usage), and Latency (it is requested the microservice with the lowest network latency).

4.3. Security Mechanisms

In this section, the security mechanisms developed for the MOAI are described [31]. Figure 3 summarizes the security layers developed in the MOAI. An explanation of the security mechanism is given for each of the three layers presented: API Gateway HTTPs and Authentication, Transporter Authentication, and Microservice Authorization. In layer 1 (green), the API Gateway security is guaranteed by using authentication through the URL and encryption with HTTPs. In layer 2 (orange), the Transporter also uses authentication for microservices connections. In layer 3 (blue), the access control of microservices is guaranteed using the Guard microservice.

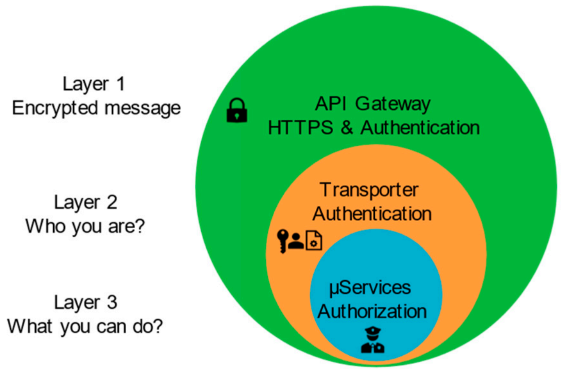

Figure 3. Security layers.

1. API Gateway (Layer 1)

Hyper Text Transfer Protocol Secure (HTTPS) is a security-enhanced version of the HTTP protocol that adds an extra layer of security using the Secure Socket Layer (SSL)/Transport Layer Security (TLS) protocols. This allows for the secure transmission of data over an encrypted connection and ensures that the identities of both the server and client are verified through the use of digital certificates.

This layer in MOAI ensures security in the access of external applications, preventing attacks (man in the middle) and the capture of the information contained in the packages. An authentication procedure has also been developed, where it is required that the client enter a token together with the URL address of the requested microservice.

2. Transporter (Layer 2)

Authentication is a security procedure in which a client's connection to a server is allowed by checking an id (ID) and password. Without this procedure in the MOAI, any microservice could use the Transporter to communicate with or access other microservices without any security. The Transporter authentication mechanism was implemented in order to enable the microservice id/password verification against the Transporter registered list.

As a result, only registered microservices will be allowed to connect to the Transporter and be part of the MOAI. Connection requests not allowed by the Transporter will be triggered, indicating the attempted security breach, and registered in a log. This security layer enables controlling the connections to the Transporter and the access to the MOAI microservices.

3. Microservice Guard (Layer 3)

Authorization is a security mechanism that determines whether a user has permission to access a particular resource, perform an action, or execute a specific operation within a system or application. It is a process that verifies if a user has the necessary privileges or permissions to perform a specific action or access a particular resource. In the MOAI, the Guard microservice is responsible for authorization.

The Guard microservice acts as a surveillance system for messages exchanged between microservices. It intercepts messages being sent before they reach their intended microservice. The authorization procedure uses a JSON Web Token (JWT) to safeguard the actions of the microservices. This involves generating an encoded token using a secret or private key that is internally defined in the system. Once the key is validated and accepted, the encoded token is returned. This process is repeated for each microservice, with each one having its own unique token.

If a microservice or application requests an action from another microservice, the Guard microservice verifies whether the client token is valid or not, controlling the client's access to this action. If it is valid, the action will be accessed normally through the Transporter, if not, the desired action will be blocked by returning an error.

4.4. Microservices Management

Managing microservice-based architectures may be a challenging task, mainly when considering the conservative industrial area. Considering that, a management structure for the MOAI was proposed. The management systems are based on container images and network-based deployment and monitoring tools. The container is an application-level abstraction that wraps code and dependencies together, illustrated as a generic microservice (blue hexagon in server 3) in Figure 4, showing the layers that this service encapsulation contains (Container layers). Containers can be allocated on the same server, sharing resources with other containers running autonomous processes, occupying reduced storage spaces relative to virtual machines (VMs), and dealing with a larger number of applications without requiring much from the operating system [38–40].

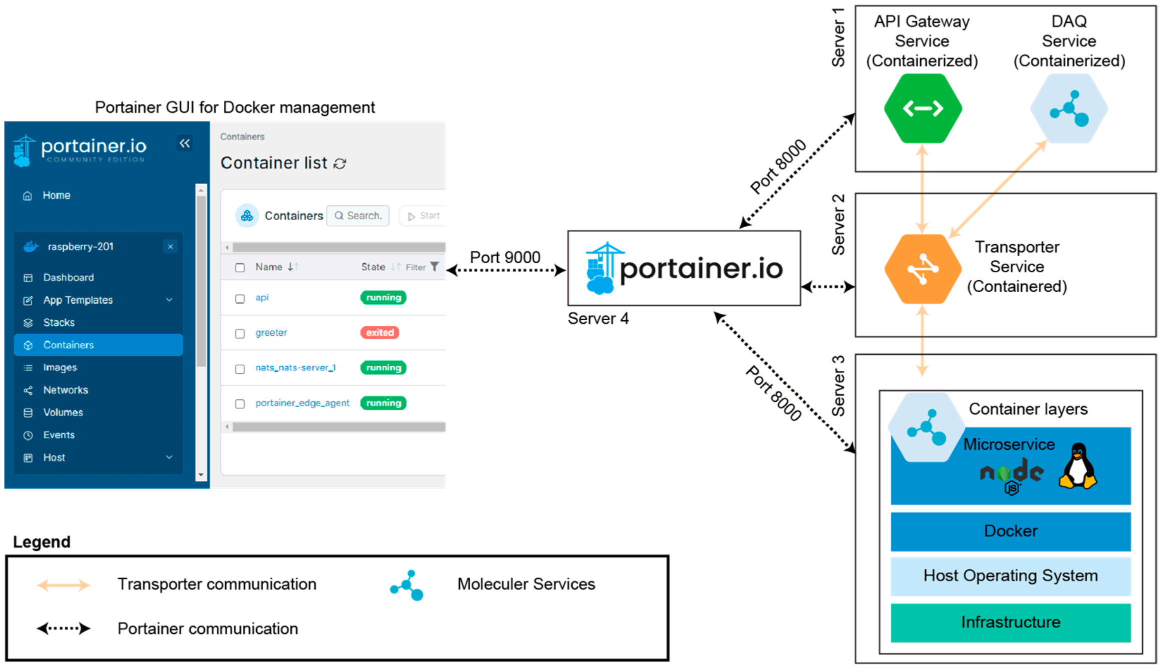

Figure 4. Services management through Portainer GUI in the MOAI.

A container can contain multiple applications (App A, App B, and App C) functioning in a monolithic manner, or just one application (App A) that contains a simple function of a more granularized (microservice) functioning as a service, as represented in Server 3 in Figure 4. Dockerfile is a simple text file with a set of commands or instructions to support the creation of a container in Docker [38]. For example, it informs that a Moleculer framework microservice developed in JavaScript will run in Node.js and all project dependencies must be installed. These commands are then executed for the construction (build) of an image (a Docker Image) for multiple CPU architectures. This static image can be stored in a local or remote registry (Docker Hub) that functions as a library of images available simply by downloading it and running it (run) on the desired host [39].

Multiple microservices (Transporter, API Gateway, and DAQ) in Figure 4 were created and distributed in different environments (Server 1, Server 2, and Server 3) on the same network, making the management task arduous. Portainer's graphical interface, together with Portainer Edge Agent, facilitates this task by abstracting and removing the need to use the Command-Line Interface (CLI) [40]. The remote environments (Server 1, Server 2, and Server 3) in Figure 4 need to be able to access the Portainer Server (Server 4). This communication is performed by an encrypted TLS tunnel (Port 8000) with keys between the Portainer Edge Agent of each remote environment and the Portainer Server.

To manage these environments, the Portainer Server GUI is accessed through a browser (Port 9000). This GUI enables controlling the operation of containers (microservices), in addition to loading environment variables and new services directly from the Docker HUB registry according to the application. As a result, it simplifies the management into just one dashboard (Portainer GUI for Docker management) in Figure 4, which contains all microservices and applications.

5. MOAI Application Examples

The MOAI was developed and implemented in a real pilot plant focused on process control. The difference of this pilot plant is that the MOAI is used for all activities. Different IAS applications were developed to validate the MOAI and demonstrate its flexibility and advantages. To simplify the presentation of the developments, it was decided to describe in detail one of these applications, which is closed-loop process control (number 1). This explanation and comprehension provide the necessary information to understand how the MOAI works and can be further applied to understand any application (including the other examples from numbers 2 to 5 briefly presented).

An industrial pilot plant for process control in the laboratory was used to test the MOAI (Figure 5). The P&ID of the plant is shown in Figure 6, comprising a tank (TQ02) at the bottom responsible for storing the fluid, which can be pumped to the upper tank (TQ01) and to the reservoir (R01) through the pumps (P2 and P1). The frequency inverters (PZ and UZ) and the solenoid valve (LV 122) represent the manipulated variables of the control loop. The process or controlled variables are the control (LIT125), the pipe flow (FIT116), the pipe pressure (PIT118), and the reservoir pressure (PIT129).

1. Closed-loop Process Controller.

Closed-loop control is the basis of any IAS. The necessary microservices to develop this external application are API Gateway, Transporter, DAQ, PID4.0, and Guard (security) in Figure 2. The orchestration of each of the plant's microservices can be seen in the sequence diagram of Figure 7.

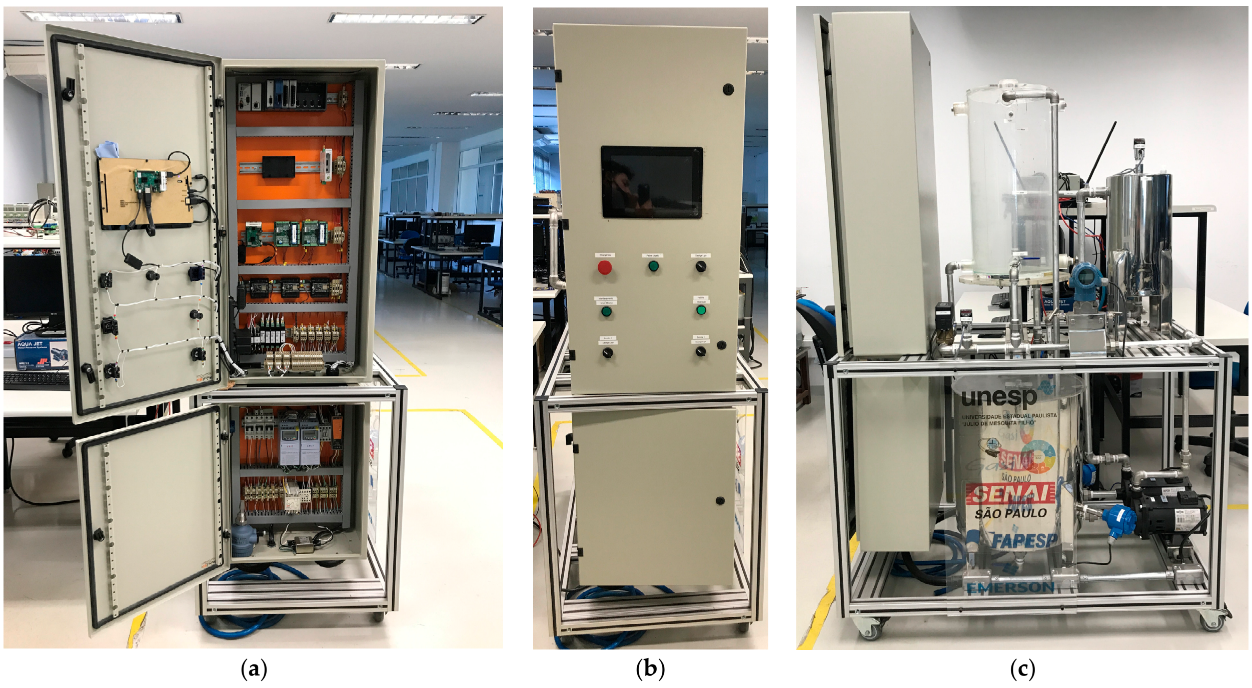

Figure 5. Industrial pilot plant: (**a**) Front part with open panel; (**b**) front part with the panel closed; (**c**) right side.

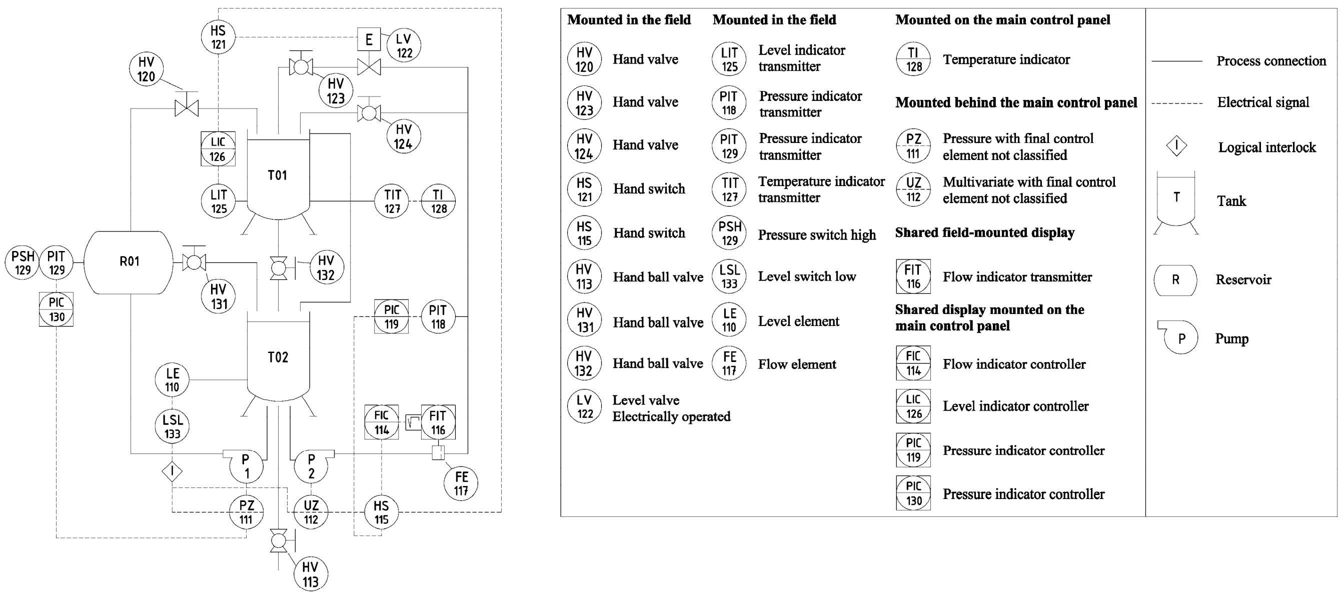
HS 121
E
LV 122
HV 120
HV 123
HV 124
LIC 126
T01
LIT 125
TIT 127
TI 128
PSH 129
PIT 129
R01
HV 132
HV 131
PIC 130
PIC 119
PIT 118
T02
LE 110
FIC 114
FIT 116
LSL 133
P 1
P 2
FE 117
PZ 111
UZ 112
HS 115
HV 113
Mounted in the field
Mounted in the field
Mounted on the main control panel
HV 120 Hand valve
HV 123 Hand valve
HV 124 Hand valve
HS 121 Hand switch
HS 115 Hand switch
HV 113 Hand ball valve
HV 131 Hand ball valve
HV 132 Hand ball valve
LV 122 Level valve Electrically operated
LIT 125 Level indicator transmitter
PIT 118 Pressure indicator transmitter
PIT 129 Pressure indicator transmitter
TIT 127 Temperature indicator transmitter
PSH 129 Pressure switch high
LSL 133 Level switch low
LE 110 Level element
FE 117 Flow element
TI 128 Temperature indicator
Mounted behind the main control panel
PZ 111 Pressure with final control element not classified
UZ 112 Multivariate with final control element not classified
Shared field-mounted display
FIT 116 Flow indicator transmitter
Shared display mounted on the main control panel
FIC 114 Flow indicator controller
LIC 126 Level indicator controller
PIC 119 Pressure indicator controller
PIC 130 Pressure indicator controller
Process connection
Electrical signal
Logical interlock
T Tank
R Reservoir
P Pump
(a)
(b)

Figure 6. Industrial pilot plant: (**a**) P&ID; (**b**) legend.

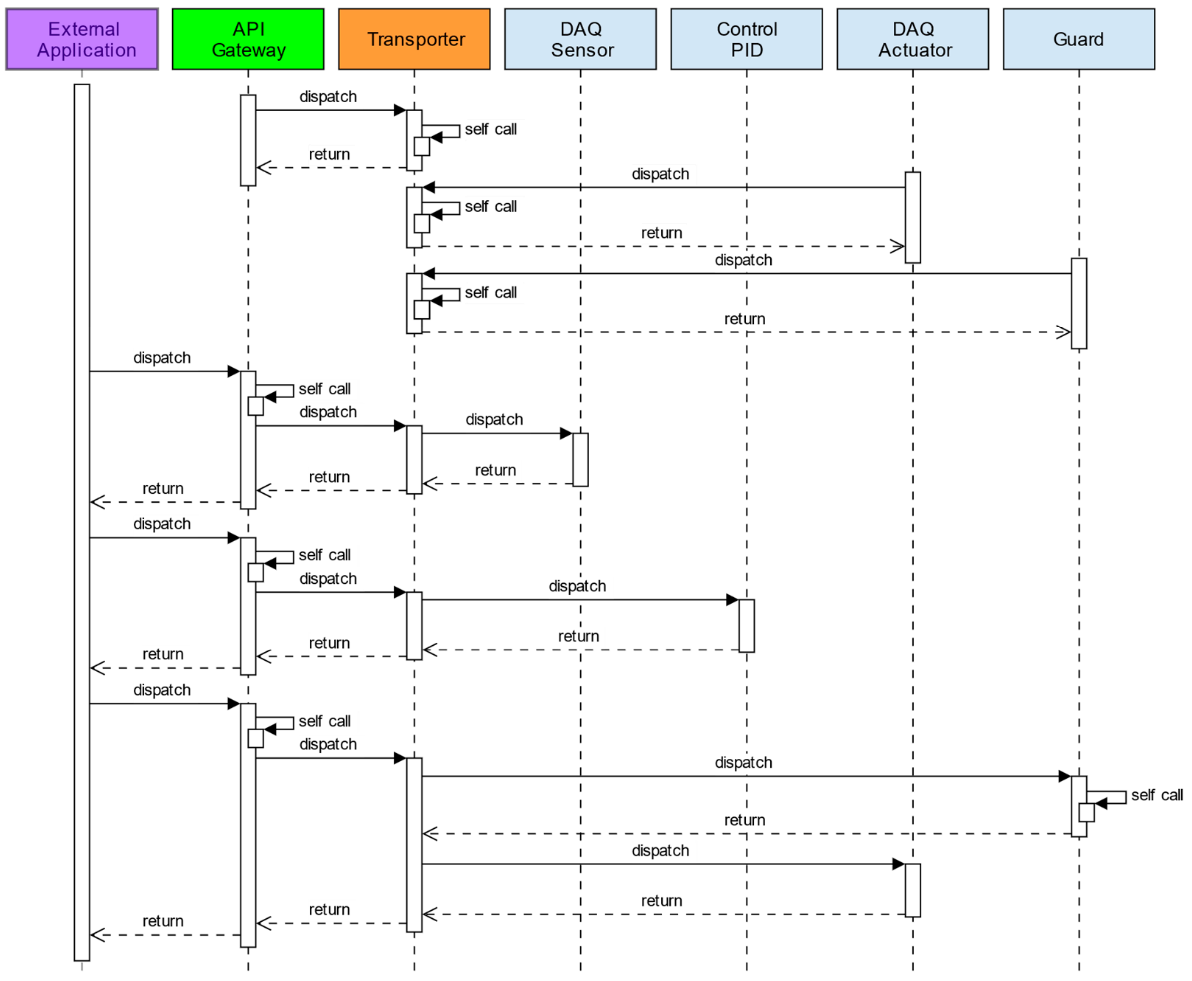

Figure 7. Diagram of the external orchestration sequence with all security mechanisms enabled.

The industrial software (LabVIEW) is represented by the external application (purple). This application orchestrates microservices through the API Gateway (green) and the communication service Transporter (orange). The microservices used are blue. This example also includes the usage of all security mechanisms implemented in the MOAI. This sequence diagram is important as its compression is the basis for understanding how any other application operates in the MOAI.

The orchestration of microservices initiates with an external application: The process control application (LabVIEW), Figure 7, cyclically requests sensor data to the DAQ service via API. If this request is successful, it receives a header with the code "200 OK", and the value of all sensors in the plant in the response. The API gateway service makes an internal call to check the token passed in the request for each request made. The HTTPs encryption steps occur but have been omitted to simplify the figure. All services must authenticate to the Transporter only once; this process was also omitted for better understanding of Figure 7.

The sensor data are sent to the PID4.0 control microservice, which is responsible for calculating the control signal to be applied to the plant. Finally, the control signal is sent to the DAQ microservice, which is responsible for updating the plant's actuator. However, before that happens, this request goes through the guard service. The Guard microservice was used to ensure secure access to the actuators, such as the actuator (output) of the DAQ service. This way, only authorized services will be able to actuate on the plant, thus assuring safety in the three layers of the IAS application.

A compilation of the security mechanisms used in each test can be seen in Table 1, where in the first test no security mechanism was used. The other tests enabled one security mechanism at a time to see its impact individually. In test 2, it only enabled authentication (Authen) in the API gateway. In test 3, it used only HTTPS encryption. In test 4, only the Transporter requires authentication. In test 5, only the Guard check was enabled (Authorization), and in test 6, all security mechanisms were enabled together.

Table 1. Security mechanisms used in each test with statistics of all communication tests performed.

Expt. Number	API Authen.	API HTTPS	Transporter Authn.	Guard Authz.	Mean (ms)	Median (ms)	Mode (ms)	SD	Variance
1	No	No	No	No	68.14	63.97	61.72	14.32	205.19
2	Yes	No	No	No	75.70	71.99	67.57	16.96	287.55
3	No	Yes	No	No	197.27	201.06	202.47	18.23	332.18
4	No	No	Yes	No	74.33	69.01	66.60	17.10	292.31
5	No	No	No	Yes	93.36	91.01	91.44	15.12	228.56
6	Yes	Yes	Yes	Yes	263.31	267.51	274.11	16.86	284.35

In Table 1, in all tests, the standard deviation (SD) ranged from 14 to 18, having a symmetrical distribution profile. The variance of the values was between 200 and 300 and cannot be disregarded in processes where latency is significant. Regarding the control loop orchestration times, the best result obtained was without the use of any security mechanism (Expt. 1), with an average time of 68.14 ms for a closed loop control cycle. The security mechanisms that have the least impact, respectively, are the authorization in the API gateway (Expt. 2) with 75.70 ms, the authentication in the Transporter (Expt. 4) with 74.33 ms, and finally the guard service (Expt. 5) with 93.36 ms.

The communication performance of all experiments (Expt.) was analyzed. The control experiment interval duration was 300 s (600 control cycles). The objective was to measure the communication time of the control loop orchestration (execution sequence of microservices in Figure 7). All times were compiled and presented through statistical data using a boxplot, as shown in Figure 8 and its respective summary table, Table 1.

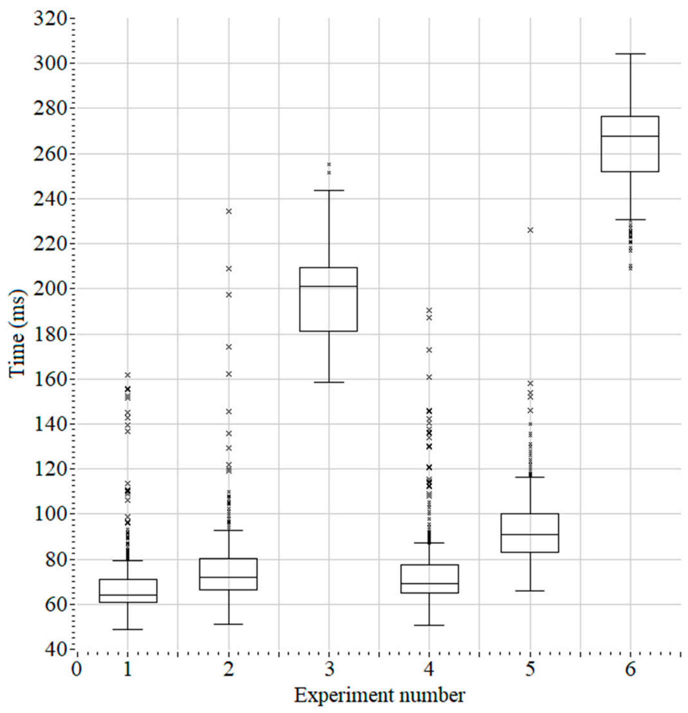

Figure 8. Boxplot statistics of all experiments performed.

The security mechanism that most impacted the communication was enabling HTTPs (Expt. 3), where the total time rose to 197.27 ms. Considering the worst-case scenario with all mechanisms enabled, the control loop orchestration time was 263.31 ms, which means an increase of 4x times to the test without any security. However, even the worst-case result is within the acceptable limits for controlling processes, for which a control cycle of 500 ms (2 Hz) is an acceptable standard in industries today. The communication performance of the MOAI was better (for example, faster than in [22]) or at least similar to the other reviewed SOA/MOA in industrial applications.

During the experiments, the following process variables of the pilot plant were controlled: pipe pressure, reservoir pressure, and pipe flow. The objective was not to evaluate or optimize the control performance, nor to compare the response curves of the processes, because each control loop has its own characteristic dynamics. The objective was to investigate the composition (orchestration) of microservices and their capacity to create a composite application to control different processes. Figure 9 presents the control responses, in which red is the desired value (setpoint) and the process variables are in other colors. The output curves can track the setpoint; in addition, there is no control saturation and the variation is smooth. However, the results show in general that the process control is stable and feasible to perform using the MOAI.

2. Distributed I/O

Data acquisition is a necessary step in any industry application. IAS has traditionally been used for SCADA and remote I/O. Remote I/O is responsible for acquiring data from sensors and actuators in industrial processes. Distributed I/O is an evolution of remote I/O used for networked data acquisition. IIoT and I4.0 applications require interoperability and vertical interaction among devices and systems. To fulfill that, a distributed I/O has been developed as a microservice, and some applications have shown its effectiveness [32].

The necessary microservices to develop the distributed I/O were Transporter and DAQ in Figure 2. The distributed I/O operation is equal to the traditional solution, but the network communication is automatic through the Transporter. Another difference is that it is programmable, so customer-specific functions can be added. A great advantage over the traditional solution is the ease of deployment of distributed I/O redundancy as a result of the replication of microservices. Redundancy in IAS means keeping redundant (duplicate or triplicate) systems in order to ensure the availability of critical processes or devices. The redundancy of distributed I/O is achieved when two or more replicas of the same distributed I/O communicate through the Transporter.

Several use cases were presented demonstrating that the distributed I/O using microservices enabled the I/O data acquisition in IAS to function in a networked, standardized, and scalable manner [32].

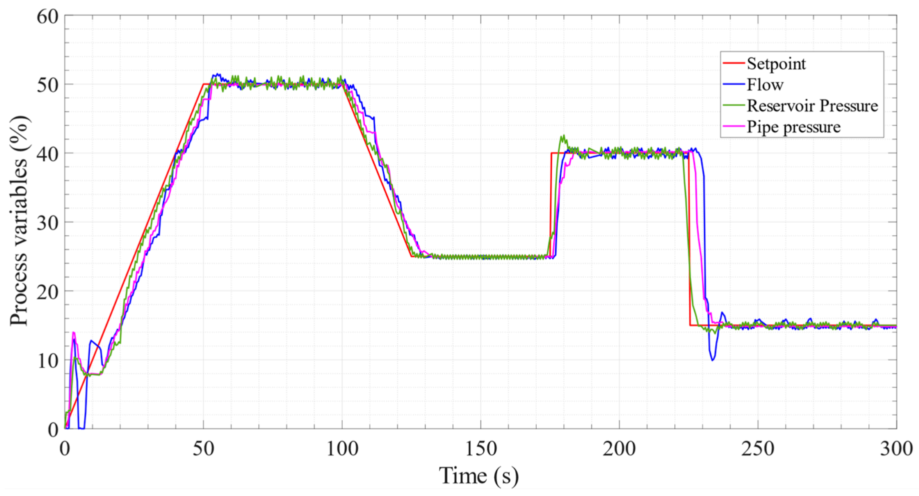

Figure 9. Control response from all pilot plant control loops.

3. Control as a Microservice and Controllers' Redundancy

The I4.0 is also pushing forward process control in IAS. Beyond interoperability and vertical integration, process control needs to be flexible, modular, and easy to deploy. In order to fulfill that, a process controller has been developed as a microservice [34]. As a result, the control algorithm can be developed according to the application's needs and made scalable through the reuse of the microservice. Considering the redundancy scenario cited before, this approach also eases and enables the redundancy of process controllers in the industry.

The necessary microservices to develop a process controller (redundant or not) were the Transporter, PID4.0, and the DAQ or M2M for I/O data acquisition in Figure 2. PID4.0 implements a network-based PID control algorithm (PIDPlus) as a microservice. The difference is that when the controller is made available as a microservice, it can be easily reused and replicated on multiple platforms, generating great hardware savings and ease of deployment. That is interesting because different versions of hardware, OS, and programming languages can be used, which is very different from a traditional PLC with dedicated hardware controllers. As a result, it reduces the overall complexity of implementing redundancy in the IAS.

Another advantage is that the Transporter communication uses load balancing strategies available in the Moleculer framework, reducing CPU overload or latency according to the selected strategy. Several use cases were presented with service compositions demonstrating that the controller as a microservice enabled using and sharing microservices in order to obtain a flexible and distributed control architecture [30,34].

4. PLC with IEC 61131-3 standard

IAS are usually based on programmable logic controllers (PLC) and development software based on IEC 61131-3 PLC programming languages. In order to make the MOAI fully compatible with IAS, a PLC microservice has been developed, and function blocks have been added to the programming IDE to make PLC programs integrated with the Transporter communication [32]. The PLC microservice integrates the OpenPLC project [38] into the Moleculer framework. OpenPLC brings together the functionalities of a PLC with the advantages of an open software and hardware architecture.

The necessary microservices to develop PLC-based automation and process control are Transporter, PLC, and the DAQ or M2M for I/O data acquisition shown in Figure 2. The PLC microservice can perform logic and sequencing of tasks that are performed on conventional PLCs, as well as the acquisition and monitoring of data from a local controller or distributed inputs and outputs (DAQ and M2M microservices). However, it is available as a microservice, which provides the same flexibility advantages cited before, such as reusing and deployment on multiple platforms.

The working logic of the controller process is developed in the OpenPLC Editor, a software that allows to write PLC programs according to IEC 61131-3. Considering the MOAI, it is also possible to create applications using the OpenPLC Editor that orchestrate the microservices available. Several use cases were presented with the PLC microservice usage in discrete automation and process control and with the microservice composition of applications using IEC 61131-3 (Ladder and Function Block Diagram) [33].

5. Process Historian and SCADA

Monitoring and understanding the behavior of industrial processes is fundamental to increasing efficiency in the production line and enabling management decision-making. Process historians or operational historians are being applied to store increasing amounts of data obtained from a broader variety of sources, including control and monitoring, enterprise resource planning (ERP), and asset management systems. With that in mind, a SCADA application was developed with a historian service for the acquisition, storage, and analysis of structured data from industrial processes [32].

The necessary microservices to develop the SCADA and historian were the Transporter, Database, and DAQ or M2M for I/O data acquisition in Figure 2. The solution is based on using the Transporter for automatically acquiring data through the DAQ or M2M, storing it in a time-series database (InfluxDB), and feeding the process historian (Grafana). Several IAS use cases were presented for SCADA and historian usage [32].

6. Discussions and Final Remarks

The MOAI enabled the use and sharing of microservices for the design of a scalable, flexible, interoperable, and distributed industrial architecture for IAS. The implementation of IAS as microservices contributes to a new model of interactions between different industrial systems, equipment, users, and applications that meets the changes and fulfills the requirements of I4.0 applications.

On the contrary of the SOA/MOA for industrial applications available in the literature, the MOAI was built on top of the Moleculer framework for microservices. It was a very positive development choice for the MOAI. Many mandatory development steps have been avoided, such as service-oriented software, service communication, service registry, and discovery, because the framework already provides them. The functionalities of the MOAI and other architectures in the literature are very similar, as they were developed for the same proposal of evolving IAS to support I4.0 applications. As a result, a simple and direct comparison among them is not fair, but rather some aspects and functionalities can be confronted. Most of the services and applications cover the same tasks. Service composition by orchestration is most commonly used in industrial applications and is supported by all architectures in the literature as it provides better execution control and management of the application. The MOAI, as also described in [22], additionally supports choreography, which can improve decentralization and aims to keep services loosely coupled and more autonomous.

The great difference between the MOAI and the other architectures is the service communication mechanism (Transporter). In addition to providing the basic microservice networking functionalities, such as communication, discovery and registry, and load balancing and status checking, it also enables the unique functionality of microservice redundancy. Microservice redundancy (through replication of the same service) is an important feature for IAS, as described in the distributed I/O and controller redundancy examples. In this case, the Transporter verifies if there are two or more (redundant) instances of the same microservice, and when there is a request for that service, it automatically chooses the best instance of them for execution at that time, enabling the redundancy. As the Transporter itself is also a microservice in the MOAI, it is also possible to make it redundant. In this case, the microservice networking will change from one Transporter (active) to another (redundant) in case of failure.

Considering the pros and cons of the MOAI, although there are disadvantages, the advantages are greater and more relevant. The MOAI is scalable because the microservices are modular and independently deployable and manageable, allowing them to be easily changed on demand. In addition, microservices can be reused for scaling applications (ex: a set of DAQ-sensor, PID4.0, and DAQ-actuator microservices reused for each control loop of the plant) and replicated in order to provide redundancy or availability (ex: two or more redundant PID4.0 microservices for the same control loop or redundant DAQ microservices for data acquisition as described in the distributed I/O example).

The MOAI is flexible as it is able to adapt to changing requirements and can be modified or extended to meet new needs without requiring significant changes to the architecture. New microservices (functionalities) can be easily created, as well as new applications that can be composed using the microservices.

The MOAI is interoperable because the networking functionalities of automatic, standardized, and transparent communication between microservices and applications overcome data integration problems between different hierarchical industry levels. The MOAI is fundamentally distributed as microservices are deployed in individual nodes that communicate through the network (Transporter). It enables decentralization of control tasks or decision-making, as each node or microservice has its own functionalities and responsibilities. In addition, it provides resilience and fault tolerance as the MOAI continues to operate in the case of individual microservice failures and failures of replicated microservices.

Microservices are a different paradigm from traditional IAS monolithic architectures in terms of development, maintenance, and commissioning/deployment. The disadvantages of the MOAI or difficulties are more related to learning new concepts, tools, and development environments and their development in the industry than to the complexity of using microservices. The development of microservices, such as those in the MOAI, must be based on standardized structures (containers) and management tools. Maintaining the applications and commissioning microservices require a repository structure for better version management and operational control, which is not common in IAS. Another point to be highlighted is the complexity of management in large applications containing many microservices. Due to the networked characteristics of MOAI, fault maintenance or error checking is more complex. On the other hand, these activities can be carried out entirely online and remotely over the network on the MOAI.

A limitation of the MOAI is related to the network protocols available for communication. As the Moleculer is a computing framework, the supported protocols are not industrial standards. As a result, direct communication with legacy industrial devices is not possible, and an intermediary service is necessary for bridging them to the Transporter. However, it is a known issue of SOA/MOA in industrial applications that was handled in this research with the M2M microservice, using the same idea as other approaches in the literature as the mediator service in [20] and the translation and gateway services in [22].

In terms of security, microservices introduce additional concerns, as each service's access needs to be secured and there needs to be secure communication between services. The common security mechanisms available in other SOAs/MOAs, such as encryption, authentication, and authorization, have also been developed in the MOAI. As an additional security layer in the MOAI, the Guard microservice mechanism checks all microservice requests before they reach their recipient, verifying permissions and blocking unallowed requests.

7. Conclusions

Even though traditional IAS architecture is still present in the industry, great efforts are being made to evolve it, especially using service-based architectures. SOA and MOA overcome the main problems of interoperability and vertical integration of heterogeneous systems in the IAS, fulfilling the requirements for I4.0 applications.

The paper described the development, deployment, and testing of a Microservice Oriented Architecture for Industry 4.0 (MOAI) based on the Moleculer framework. The first contribution was investigating and demonstrating that a non-industrial framework can be applied to IAS and I4.0 applications. In addition to simplifying MOAI development, it provided a flexible, interoperable, and distributed architecture.

The functionalities, pros, and cons of the MOAI were discussed. The highlight and differential point of the MOAI is the networking microservice (Transporter). In addition to offering fundamental microservice networking capabilities, it enabled the distinctive feature of microservice redundancy. Operational details of the architecture were explained, as well as the communication and service composition through orchestration and choreography. The development of microservices and applications as well as security mechanisms were also discussed.

Considering the MOAI microservices, some can be highlighted. The PLC microservice can be applied in applications where determinism is not required and is advantageous because it is not necessary to deploy dedicated physical controllers in the plant. The DAQ microservice resembles the operation of a networked remote I/O, with the advantage of programing capability and incorporating customized functionality. The PID4.0 microservice evolved traditional PID control into a network-based, redundant-ready controller for IAS.

Application examples of the MOAI were discussed, and the results demonstrated the feasibility of supporting IAS in the context of the I4.0. The MOAI proved to be valuable both at the device and application levels by providing a high level of loose coupling between the various components of the system.

Future work will focus on expanding the experiments to evaluate the benefits and drawbacks of the MOAI developed. It is expected to automate the composition of services by taking advantage of microservice discovery and using the concept of Plug and Play.

Author Contributions: Conceptualization, J.A.B. and E.P.G.; Methodology, R.P.P. and L.B.R.d.S.; Software, R.P.P., J.A.B. and L.B.R.d.S.; Validation, R.P.P.; Investigation, R.P.P.; Writing–original draft, R.P.P.; Writing–review & editing, E.P.G.; Supervision, E.P.G.; Funding acquisition, E.P.G. All authors have read and agreed to the published version of the manuscript.

Funding: Research supported by grant 2018/19984-4, São Paulo Research Foundation (FAPESP).

Data Availability Statement: There is no research data available in this article.

Acknowledgments: We are thankful for the participation of Sergio L. Risso, Douglas P. Fernandez, João P. da Silva Cavalcante, João V. Rocha Santana, Vitor I. Guermandi, Felipe O. Domingues, and Michel M. Fernandes in the development of this project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lu, Y. Industry 4.0: A Survey on Technologies, Applications and Open Research Issues. *J. Ind. Inf. Integr.* **2017**, *6*, 1–10. [CrossRef]
2. Borangiu, T.; Trentesaux, D.; Thomas, A.; Leitão, P.; Barata, J. Digital Transformation of Manufacturing through Cloud Services and Resource Virtualization. *Comput. Ind.* **2019**, *108*, 150–162. [CrossRef]
3. Sisinni, E.; Saifullah, A.; Han, S.; Jennehag, U.; Gidlund, M. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Trans. Ind. Inf.* **2018**, *14*, 4724–4734. [CrossRef]
4. Colombo, A.W.; Karnouskos, S.; Bangemann, T. Towards the next Generation of Industrial Cyber-Physical Systems. In *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; Volume 9783319056241, pp. 1–22. ISBN 9783319056241.
5. Wollschlaeger, M.; Sauter, T.; Jasperneite, J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Ind. Electron. Mag.* **2017**, *11*, 17–27. [CrossRef]

6. Givehchi, O.; Landsdorf, K.; Simoens, P.; Colombo, A.W. Interoperability for Industrial Cyber-Physical Systems: An Approach for Legacy Systems. *IEEE Trans. Ind. Inf.* **2017**, *13*, 3370–3378. [CrossRef]
7. Jammes, F.; Karnouskos, S.; Bony, B.; Nappey, P.; Colombo, A.W.; Delsing, J.; Eliasson, J.; Kyusakov, R.; Stluka, P.; Tilly, M.; et al. Promising Technologies for SOA-Based Industrial Automation Systems. In *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*; Springer: Cham, Switzerland, 2014; pp. 89–109. ISBN 978-3-319-05623-4. [CrossRef]
8. ISA beyond the Pyramid: Using ISA95 for Industry 4.0 and Smart Manufacturing. Available online: https://www.isa.org/intech-home/2021/october-2021/features/beyond-the-pyramid-using-isa95-for-industry-4-0-an (accessed on 10 January 2023).
9. Klettner, C.; Tauchnitz, T.; Epple, U.; Nothdurft, L.; Diedrich, C.; Schröder, T.; Großmann, D.; Banerjee, S.; Krauß, M.; Iatrou, C.; et al. Namur Open Architecture: Die Namur-Pyramide wird geöffnet für Industrie 4.0. *Atp Mag.* **2017**, *59*, 20–37. [CrossRef]
10. de Caigny, J.; Tauchnitz, T.; Becker, R.; Diedrich, C.; Schröder, T.; Großmann, D.; Banerjee, S.; Graube, M.; Urbas, L. NOA–Von Demonstratoren Zu Pilotanwendungen. *Atp Mag.* **2019**, *61*, 44–55. [CrossRef]
11. OPC Foundation Unified Architecture—OPC Foundation. Available online: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/ (accessed on 21 December 2022).
12. Leitão, P.; Colombo, A.W.; Karnouskos, S. Industrial Automation Based on Cyber-Physical Systems Technologies: Prototype Implementations and Challenges. *Comput. Ind.* **2016**, *81*, 11–25. [CrossRef]
13. di Francesco, P.; Malavolta, I.; Lago, P. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In Proceedings of the 2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, 5–7 April 2017; pp. 21–30. [CrossRef]
14. Xiao, Z.; Wijegunaratne, I.; Qiang, X. Reflections on SOA and Microservices. In Proceedings of the 4th International Conference on Enterprise Systems: Advances in Enterprise Systems, ES 2016, Online, 2–3 November 2016; IEEE: Piscataway, NJ, USA, 2017; pp. 60–67.
15. da Xu, L.; Xu, E.L.; Li, L. Industry 4.0: State of the Art and Future Trends. *Int. J. Prod. Res.* **2018**, *56*, 2941–2962. [CrossRef]
16. Mrugalska, B.; Wyrwicka, M.K. Towards Lean Production in Industry 4.0. *Procedia. Eng.* **2017**, *182*, 466–473. [CrossRef]
17. Delsing, J.; Rosenqvist, F.; Carlsson, O.; Colombo, A.W.; Bangemann, T. Migration of Industrial Process Control Systems into Service Oriented Architecture. In Proceedings of the IECON 2012—38th Annual Conference on IEEE Industrial Electronics Society 2012, Montreal, QC, Canada, 25–28 October 2012; pp. 5786–5792. [CrossRef]
18. Girbea, A.; Suciu, C.; Nechifor, S.; Sisak, F. Design and Implementation of a Service-Oriented Architecture for the Optimization of Industrial Applications. *IEEE Trans. Ind. Inf.* **2014**, *10*, 185–196. [CrossRef]
19. Cannata, A.; Gerosa, M.; Taisch, M. SOCRADES: A Framework for Developing Intelligent Systems in Manufacturing. In Proceedings of the 2008 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2008, Singapore, 8–11 December 2008; pp. 1904–1908. [CrossRef]
20. Karnouskos, S.; Colombo, A.W.; Bangemann, T.; Manninen, K.; Camp, R.; Tilly, M.; Sikora, M.; Jammes, F.; Delsing, J.; Eliasson, J.; et al. The IMC-AESOP Architecture for Cloud-Based Industrial Cyber-Physical Systems. In *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; Volume 9783319056241, pp. 49–88. ISBN 9783319056241.
21. ARROWHEAD Eclipse Arrowhead—Eclipse Arrowhead Framework and Implementation Platform. Available online: https://arrowhead.eu/ (accessed on 10 January 2023).
22. Espí-Beltrán, J.V.; Gilart-Iglesias, V.; Ruiz-Fernández, D. Enabling Distributed Manufacturing Resources through SOA: The REST Approach. *Robot. Comput. Integr. Manuf.* **2017**, *46*, 156–165. [CrossRef]
23. Delsing, J. Local Cloud Internet of Things Automation: Technology and Business Model Features of Distributed Internet of Things Automation Solutions. *IEEE Ind. Electron. Mag.* **2017**, *11*, 8–21. [CrossRef]
24. Paniagua, C.; Eliasson, J.; Delsing, J. Efficient Device-to-Device Service Invocation Using Arrowhead Orchestration. *IEEE Internet Things J.* **2020**, *7*, 429–439. [CrossRef]
25. Ciavotta, M.; Alge, M.; Menato, S.; Rovere, D.; Pedrazzoli, P. A Microservice-Based Middleware for the Digital Factory. *Procedia. Manuf.* **2017**, *11*, 931–938. [CrossRef]
26. MAYA Multi-DisciplinArY Integrated SimulAtion and Forecasting Tools, Empowered by Digital Continuity and Continuous Real-World Synchronization, towards Reduced Time to Production and Optimization | MAYA Project | Fact Sheet | H2020 | CORDIS | European Commission. Available online: https://cordis.europa.eu/project/id/678556 (accessed on 19 January 2023).
27. Innerbichler, J.; Gonul, S.; Damjanovic-Behrendt, V.; Mandler, B.; Strohmeier, F. NIMBLE Collaborative Platform: Microservice Architectural Approach to Federated IoT. In Proceedings of the GIoTS 2017—Global Internet of Things Summit, Geneva, Switzerland, 6–9 June 2017; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2017.
28. NIMBLE The Novel, Federated Approach for Industry B2B Platforms—Nimble Project—The Novel, Federated Approach for Industry B2B Platforms. Available online: https://www.nimble-project.org/ (accessed on 10 January 2023).
29. Pontarolli, R.P.; Bigheti, J.A.; Fernandes, M.M.; Domingues, F.O.; Risso, S.L.; Godoy, E.P. Microservice Orchestration for Process Control in Industry 4.0. In Proceedings of the 2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2020, Roma, Italy, 3–5 June 2020; pp. 245–249. [CrossRef]
30. Pontarolli, R.P.; Bigheti, J.A.; de Sá, L.B.R.; Godoy, E.P. Towards Security Mechanisms for an Industrial Microservice-Oriented Architecture. In Proceedings of the 2021 14th IEEE International Conference on Industry Applications, INDUSCON 2021, São Paulo, Brazil, 15–18 August 2021; pp. 679–685. [CrossRef]

31. Pontarolli, R.P.; Bigheti, J.A.; Domingues, F.O.; de Sá, L.B.R.; Godoy, E.P. Distributed I/O as a Service: A Data Acquisition Solution to Industry 4.0. *HardwareX* **2022**, *12*, e00355. [CrossRef] [PubMed]
32. Fernandes, M.D.M.; Bigheti, J.A.; Pontarolli, R.P.; Godoy, E.P. Industrial Automation as a Service: A New Application to Industry 4.0. *IEEE Lat. Am. Trans.* **2021**, *19*, 2046–2053. [CrossRef]
33. Bigheti, J.A.; Fernandes, M.M.; Godoy, E. Paciencia. Control as a Service: A Microservice Approach to Industry 4.0. In Proceedings of the 2019 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2019, Naples, Italy, 4–6 June 2019; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2019; pp. 438–443.
34. Dustdar, S.; Papazoglou, M.P. Services and Service Composition-An Introduction. *IT Inf. Technol.* **2008**, *50*, 86–92.
35. Moleculer Services | Moleculer—Progressive Microservices Framework for Node.Js. Available online: https://moleculer.services/docs/0.14/services.html#Internal-Services (accessed on 17 October 2022).
36. OpenPLC OpenPLC—Open-Source PLC Software. Available online: https://openplcproject.com/ (accessed on 30 December 2022).
37. Song, J.; Mok, A.K.; Chen, D.; Nixon, M.; Blevins, T.; Wojsznis, W. Improving PID Control with Unreliable Communications. In Proceedings of the ISA Expo Technical Conference, Houston, TX, USA, 17–19 October 2006; pp. 105–116.
38. Docker. Docker What Is a Container? Available online: https://www.docker.com/resources/what-container/ (accessed on 28 December 2022).
39. Moleculer Deploying. Moleculer—Progressive Microservices Framework for Node.Js. Available online: https://moleculer.services/docs/0.14/deploying.html (accessed on 28 December 2022).
40. Portainer Portainer Architecture—Portainer Documentation. Available online: https://docs.portainer.io/start/architecture (accessed on 28 December 2022).