Distributed I/O as a service: A data acquisition solution to Industry 4.0

Ricardo P. Pontarolli *, Jeferson A. Bigheti, Felipe O. Domingues, Lucas B.R. de Sá, Eduardo P. Godoy

Institute of Science and Technology, São Paulo State University (Unesp), PO Box 18087-180, Sorocaba, SP, Brazil

Abstract

Data acquisition is a required step for any industrial application. Industrial automation traditionally relies on Supervisory Control and Data Acquisition (SCADA), and remote I/O is an important element of this infrastructure. The remote I/O is responsible for the data acquisition of sensors and actuators of industrial processes. The distributed I/O represents an evolution of the remote I/O to include processing and support network communication. With the adoption of recent technologies within the industry 4.0, automation solutions are facing servitization and being develop as a service. This paper describes the development of an open-source and low-cost distributed I/O as a service for industrial automation. The solution is based on a single board computer with I/O expansion modules, a microservice framework and a customized software to enable the I/O data acquisition to function in a networked, standardized, and scalable manner. Some application examples of the distributed I/O service are presented such as the I/O redundancy and historian, process control, and IIoT monitoring.

Introduction

A data acquisition (DAQ) system is a combination of tools that are used to gather, analyze, and record information about some phenomenon as shown in Fig. 1. Data acquisition is the initial step for any kind of science application. To understand a problem, it is necessary to collect, process and explore the information from signals obtained from the interested environment. Considering the DAQ hardware, there are three general usage types for the data acquisition: Laboratory, Portable and Remoted or Distributed.

Laboratory and Distributed DAQ are usually static hardware put at specific locations and connected to desktop computers in some way. These systems depend on the computer software to access, process, visualize and analyze data. The difference between Laboratory and Distributed DAQ is the way the hardware is connected to the computer software. In the Laboratory DAQ, usually computer interfaces are used such as PCI boards or serial communication. On the other hand, Distributed DAQ uses a network communication. Finally, Portable DAQ systems are small, lightweight units that are easily carried by hand.

* Corresponding author.
E-mail address: pasquati@ifsp.edu.br (R.P. Pontarolli).

https://doi.org/10.1016/j.ohx.2022.e00355
2468-0672/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
DAQ hardware are commercially traditional products with a relevant cost. And sometimes, it is necessary to acquire specific signals which are not supported by the standard DAQ hardware. As a result, it is common to find in the literature papers developing low-cost and open-source DAQ hardware for several different applications [5,6].

Focusing on the scenario of DAQ systems for industry, in most cases standardized analog and digital signals are used such as 4–20 mA and 0–10 VDC analog signals, and 24 VDC digital signals. An important element in industrial systems for acquiring those signals is the remote I/O (input/output) which is responsible for the data acquisition of the inputs/sensors and outputs/actuators (analog and digital signals) of an industrial process. In the traditional ISA-95 hierarchical automation architecture [7], these remote I/O are named precisely because they are physically in a remote place with a certain distance from the controller. And both are connected and communicate through a physical medium, as can be seen in the Fig. 2 a), in a scenario with only one controller [8]. The distributed I/O is an evolution of the remote I/O, Fig. 2 b) and is also at a certain distance from the controller but it has some computing capacity and shares a network, usually an industrial protocol, with other controllers and distributed I/O modules.

Industry processes as well as many other industrial infrastructures rely on Supervisory Control and Data Acquisition (SCADA) and Distributed Control System (DCS) to perform their complex functionality. The Industrial Internet of Things (IIoT) is complementary to these systems, using information from SCADA and DCS as data sources. SCADA focuses on monitoring and control, while IIoT seeks connectivity to gather and analyze data to improve decision making and increase productivity [9].

The new applications related to the IIoT and Industry 4.0 (I4.0) must evolve in terms of automation solutions to overcome the challenges of decentralization, modularity, and independence of the systems. This means, for example, that new architectures and communication technologies are required to facilitate association of automation solutions such as the distributed I/O, programmable controllers, SCADA, and management systems. As presented in [10], I4.0 technical innovation is based on vertical and horizontal integration of industrial systems, continuous digital engineering throughout the product lifecycle, and decentralization of computational resources. For this to occur, a common model that groups these conditions within a reference architecture was defined for I4.0 [11], called RAMI 4.0 (Reference Architectural Model for Industry 4.0). The RAMI 4.0 is based on a Service-Oriented Architecture (SOA).

The SOA is a recent paradigm for automation solutions with proved potential for providing vertical integration and interoperability for networked systems through the use and sharing of services. The most recent trend in SOA for industrial applications is microservices. Variant of SOA, microservices are an architectural style in which applications are decomposed into small services, offering modularity, making applications easier to develop, test, deploy and, most importantly, change and maintain [12]. Reference [13] defines microservices as small standalone services that work together. These services run on their own processes and communicate through both synchronous (request/response) and asynchronous (request/call-back) mechanisms and usually Web integration through REST.

**Fig. 1.** Block Diagram of Data Acquisition System.

**Fig. 2.** Representation of I/O devices in industrial automation applications (Adapted from [8]).
Microservices are deployed and scaled independently and have well-defined boundaries or boundaries and can be written in different languages and use different resources for data storage. A comparison between monolithic architectures and microservices is presented in Fig. 3 [14]. As can be seen, the biggest gain in the adoption of microservices is scalability. It is observed on the left side that an environment built amidst the monolithic approach hinders the scalability of specific services, while on the right side the services can be cloned to facilitate the increase of access to them by decreasing the load of requests on the same server, as well as being able to deal with failures. Resources can be better leveraged, avoiding resource waste with unused or underutilized service allocation.

The Architecture for Service-Oriented Process (IMC-AESOP) [15] project firstly develop a cloud SOA for industrial automation applications focused primarily on process control applications, SCADA and DCS systems. The great advance of the SOA architecture was the availability of services created in a Cloud Computing infrastructure, which increased the architecture availability and enabled it to adapt to variations in applications due to replication of, or creation of new, cloud services. Several services have been developed to support all the functionalities proposed by the architecture such as monitoring, control, alarm, simulation, and supervision services [16].

In this project traditional distributed I/O were used within a mediator service created to get the data from the legacy distributed I/O and replicate them to the SOA services. Reference [17] presents that DAQ structure as can be seen in Fig. 4. The Gateway in this case controls a set of lower-layer non-service-enabled devices, each of which is exposed through the Gateway as a service-enabled device. It allows them to gradually replace legacy devices to web service enabled devices natively without impacting the applications that use those devices.

![Fig. 3. Comparison of Monolithic Structures and Microservices [14].](image1)

![Fig. 4. Gateway vs Service Mediator Concepts [17].](image2)
Still in the same project, a mediator may contain multiple data sources, which may or may not be processed before making them available as a service. Reference [17] used the mediator to aggregate several non-service enabled devices, where applications in the higher layers can communicate with the mediator instead of communicating with proprietary interfaces.

A Microservice-Oriented Architecture (MOA) for automation and control in Industry 4.0 scenarios have been developed in [18,19] by the authors which focused on the development of the microservices for process control. Following this trend for using microservices in industrial automation and Industry 4.0 applications, this paper describes the development of an open-source distributed I/O as a microservice. On the contrary of developing traditional remote I/O [20,21] (Fig. 2), or a mediator service [17] (Fig. 4) for industrial DAQ, our proposal deploys a distributed I/O directly connected to the service-oriented network as shown in Fig. 5. As a result, all information acquired from the distributed I/O is automatically available for use by any other service or application in a standardized and interoperable manner.

**Moleculer framework**

For the creation of the Distributed I/O microservice, we used the Moleculer Framework. A framework represents a base structure or a development platform, containing tools, libraries, systems, and components that supports the process of developing a specific solution. The Moleculer framework is a development framework with the JavaScript language for developing applications oriented to microservices [22]. The framework is open source and runs on the Node.js platform, in which ser-
vices are executed on individual nodes that automatically communicate via the communication service (Transporter). In this distributed architecture, services can be easily replicated to provide resilience and fault tolerance.

In the Moleculer architecture, Fig. 6, all microservices are the same, with no hierarchy or priority. A major advantage of this framework is that all developed microservices have an automatic registration and discovery feature available. In this way, all existing services are informed after the creation of a new service or the availability of new functionality in a service. Another important feature is automatic load balancing, which has the function of dynamically distributing the communication load between replicated microservices. Each node can contain one or more microservices and is responsible for managing its data and its own database. The microservices can offer and perform different tasks that are called actions.

At Moleculer, the Transporter service automatically manages the communication among microservices. The great benefit is the decoupling between producers and consumers of events and data. Decoupling simplifies development and increases availability compared to using distributed transactions. If no consumer is available to process an event, the Transporter will queue the event until the message can be processed.

The Transporter service solves the communication between completely different systems in a reliable way. Message broker makes it possible to integrate such systems so that they can exchange data in an uncoupled manner. The exchange of information between microservices using the Transporter is carried out by the Serializers module (for example JSON), responsible for the serialization of the message data.

**Hardware description**

Considering the current migration of industrial automation to distributed architectures such as SOA and MOA, the contribution of this paper is detailing the development of distributed I/O as a microservice for industrial DAQ. The solution is composed by a single board computer (Raspberry Pi) with an industrial I/O expansion module (MegaIO industrial card), a microservice framework (Moleculer) and a customized software embedded in the hardware (see Table 1). The distributed I/O is open-source and low-cost and simplifies the I/O data acquisition for industrial applications. The network communication with the distributed I/O is transparent and automatic (managed by the Transporter), which means that no network coding is needed. This standardized communication provides interoperability and scalability for any industrial automation applicational which needs to use data acquired by the distributed I/O. In addition, the distributed I/O also enables IIoT applications as it automatically provides web access through an HTTP(s) API to all the I/O available. Some of the uses that can be given to this equipment are:

a) Collect analog and digital input signals from sensors;
b) Update analog and digital output signals to actuators;
c) Enable automatic I/O data acquisition redundancy;
d) Standardize network remote communication with devices connected to the I/O;
e) Enable web access to the I/O through API Gateway;
f) Provide industrial I/O data to automation applications (for example a I/O data logging in a process historian, a process control loop, a data acquisition to a SCADA).

| Specifications of distributed I/O. |
|-----------------------------------|
| Specifications table Distributed I/O |
| Hardware name | Distributed I/O |
| Subject area | Engineering and automation  
Industry measurements and data acquisition  
Educational tools and open-source alternatives to existing infrastructure  
Instrumentation |
| Hardware type | Field measurements and communication |
| Closest commercial analog | UniPi [1]  
Revolution Pi [2]  
Iono Pi [3]  
Monarco HAT [4] |
| Open-source license | GNU General Public License (GPL) 3.0. |
| Cost of hardware | $152.47 |
| Source file repository | https://doi.org/10.17605/OSF.IO/EC9FU |

**Design files summary**

1. Hardware schematics Industrial Automation 8-Layer Stackable HAT for Raspberry Pi (see Table 2).
2. Microservice in JavaScript language made used the framework Moleculer with framework to access the functionalities of industrial automation 8-layer stackable hat for raspberry pi hardware in a distributed way.
3 Microservice in JavaScript language made used the framework Moleculer with the function of publishing the microservices and can be accessed through http requests by the browser for example.

4 Messaging server, transporter, responsible for exchanging messages between microservices.

5 Operational system for the Raspberry Pi single board computer;

6 Open-source software for running JavaScript applications.

Bill of materials

Build instructions for the distributed I/O

The hardware part of the distributed I/O is commercially available in the market (see Table 3). It is composed of a single board computer Raspberry Pi and an industrial automation I/O card for Raspberry Pi [23]. The mechanical part of construction is very simple, in which the Industrial automation I/O (megaind) card kit (Section “Industrial automation I/O (megaind) card kit”) is assembled on top of the Raspberry Pi as shown in Fig. 7.

Table 2
Design files summary.

| ID | Design file name          | File type | Open-source license | Location of the file |
|----|---------------------------|-----------|---------------------|----------------------|
| 1  | megaind-rpi.pdf          | Schematics| GPL 3.0             | https://osf.io/v5tz3/ |
| 2  | megaind-rpi.service.js    | Software  | GPL 3.0             | https://osf.io/gd2bf/ |
| 3  | api.service.js            | Software  | MIT                 | https://osf.io/zg86k/ |
| 4  | nats 2.1.7                | Software  | Apache 2.0          | https://hub.docker.com/_/nats |
| 5  | Raspberry Pi OS          | Software  | GPL                 | https://www.raspberrypi.org/software/ |
| 6  | Node.js                   | Software  | MIT                 | https://nodejs.org/en/download/ |

Table 3
Bill of materials.

| Component              | N. | Unit  | Source of materials                                      | Material type |
|------------------------|----|-------|----------------------------------------------------------|---------------|
| Raspberry Pi 3         | 1  | $35.00| https://www.raspberrypi.com/products                     | Electronic    |
| SanDisk 16 GB          | 1  | $6.59 | https://www.amazon.com/SanDisk-Ultra-SDSQUNS-016G-GN3MN-UHS-I-microSDHC/dp/B0146IAXYO | Electronic    |
| Power Supply 24 V      | 1  | $18.98| https://www.amazon.com/AVAWO-Switching-Transformer-Regulated-Computer/dp/B074B4P7KD | Electronic    |
| Megaind Card           | 1  | $90.00| https://sequentmicrosystems.com/collections/industrial-automation/products/industrial-raspberry-pi | Electronic    |
| CR2450                 | 1  | $1.90 | https://www.amazon.com/GloFX-CR2450-Lithium-Battery-Pack/dp/B07ZQNHKIZC | Electronic    |
| Total cost             |    | $152.47|                                                          |               |

Fig. 7. Hardware of the Distributed I/O [23].
Industrial automation I/O (megaind) card kit

The Industrial automation I/O (megaind) card, Fig. 8, is composed by the following items in Fig. 9 and Fig. 10.

- Four M2.5 × 18 mm male–female brass standoffs
- Four M2.5 × 5 mm brass screws
- Four M2.5 brass nuts
- Mounting Hardware
- Two stack level jumpers
- Four 8-pin female mating connector plugs for IO’s

Fig. 8. Industrial automation card for Raspberry Pi [23].

Fig. 9. Mounting hardware and jumpers [23].

Fig. 10. Connector plugs for all inputs and outputs [23].

Fig. 11. Industrial automation card for Raspberry Pi card layout and mechanical specifications [23].
• One 4-pin female mating connector plug for RS465 ports.
• One 2-pin female mating connector plug for power.

**Industrial automation I/O (megaind) card features**

The Industrial automation I/O (megaind) card layout and mechanical specifications are shown in Fig. 11.

The industrial automation I/O (megaind) card main features are:

• Four Optically Isolated Digital Inputs with status LEDs;
• Four 0–10 V or ± 10 V 12 bits Analog Inputs;
• Four Optically Isolated 4–20 mA 12 bits Inputs;
• Four Optically Isolated Open Drain Outputs, 24 V/4A;
• Four 0–10 V 12 bits Analog Outputs;
• Four Optically Isolated 4–20 mA 12 bits Analog Outputs;
• Four General Purpose LEDs;
• TVS Protection on all inputs;
• On Board Hardware Watchdog;
• On Board Resettable Fuse;
• Real Time Clock with Battery Backup.

**Additional information:**

• Communication: connect the card to other Industrial Automation systems using RS485/Modbus and CAN communication ports;
• Compatibility: the industrial automation I/O card shares the I2C bus using only two of the Raspberry Pi’s GPIO pins to manage all eight cards. This feature leaves the remaining 24 GPIOs available for the user. It is compatible with all Raspberry Pi versions from Zero to 4 and has all the necessary I/Os for your Industrial Automation projects.
• Power requirements: The Industrial Automation card needs 24VDC to operate and must be powered from its own pluggable connector. The card supplies 5 V and up to 3A to the Raspberry Pi on the GPIO bus.
• Pluggable connectors: All the IO’s are connected to heavy duty (8A), 3.5 mm pitch pluggable connectors which make field wiring very convenient for installation and debugging.
• Stacking multiple cards: Up to eight Industrial Automation cards can be stacked on a Raspberry Pi. Cards can be installed in any order. Each card is identified by jumpers needed to indicate the level in the stack.

**Operational instructions and testing**

In order to explain the operation instructions and initial testing, as well as the usage learning of the distributed I/O microservice, a simple web application scenario is shown in Fig. 12. In this scenario, the I/O data from the distributed I/O is remotely accessed from the web (Internet browser) using API requests (HTTP method GET). All microservices (API Gateway, Transporter and Distributed I/O) were deployed in only one hardware (Raspberry Pi) for simplifying the operational instructions and the resources needed. However, it would be possible to distribute these microservices in three different hardware (for example with each service in a distinct Raspberry Pi).

In this scenario, the distributed I/O microservice provides access to the I/O (sensors/actuators) connected to the automation card. The Transporter microservice provides network communication with other microservices (e.g. distributed I/O). And the API gateway microservice provides an HTTP(s) web access to the I/O data of the distributed I/O. A summary list containing all the available actions of the megaind-rpi service for I/O data acquisition of the megaind card is presented in Table 4.
For each one of these actions, there is a standardized HTTP(s) request. For example, the `setOutputVoltage` action is responsible for writing (updating) an analogic value from 0 to 10 V to a selected channel of the megaind card. A generic request to access these actions is standardized in the following format (ip-address 1:port2).

https://ip-address:port/megaind-rpi.action?

**Header** {"Content-Type": "application/json"}.

**Body** {"param-1": value, "param-2": value}.

Hardware

1. Plug your card on top of your Raspberry Pi and power up the system, Fig. 7.
2. Enable I2C communication on Raspberry Pi using `raspi-config`.

Software

Prepare the Raspberry Pi with the following software below. For detailed information, visit the link available in the Section 3 Design files summary.

- Raspberry Pi OS
- Node.js
- NATS transporter
- api.service.js
- megaind-rpi.service.js

Usage

The usage procedure (step-by-step) is available at [https://github.com/pontarolli/megaind-rpi](https://github.com/pontarolli/megaind-rpi). Here it is shown a simple example of writing a new value (update) to an analog output channel of the distributed I/O. For example, the output of one analog voltage output of the distributed I/O is set to 2.5 V (25 % of the scale) using the `setOutputVoltage` request in Fig. 13. After that, it is possible to confirm the output value with a multimeter as shown in Fig. 14.

---

1. Hardware (Raspberry Pi) in which API gateway service is running.
2. TCP port in which the API Gateway service is listening.

---

### Table 4
Summary of the Distributed I/O microservice actions for I/O data acquisition.

| N. | Action             | Description                              | params                                                                 | Return                        |
|----|--------------------|------------------------------------------|------------------------------------------------------------------------|-------------------------------|
| 1  | `setOutputVoltage` | Write 0–10 V output voltage value (V)    | Stack,\(^4\) channel,\(^5\) value                                    | 0 (Success) or −1 (Error)    |
| 2  | `getOutputVoltage` | Read 0–10 V output voltage value (V)     | stack, channel                                                        | Percentage of 0 to 10 V [0..100] |
| 3  | `getInputVoltage`  | Read 0–10 V input voltage value (V)      | stack, channel                                                        | Percentage of 0 to 10 V [0..100] |
| 4  | `setOutputCurrent` | Write 4–20 mA output value (mA)          | stack, channel, value\(^6\)                                          | 0 (Success) or −1 (Error)    |
| 5  | `getOutputCurrent` | Read 4–20 mA output current value (mA)   | stack, channel                                                        | Percentage of 4 to 20 mA [0..100] |
| 6  | `getInputCurrent`  | Read 4–20 mA input value (mA)            | stack, channel                                                        | Percentage of 4 to 20 mA [0..100] |
| 7  | `setOutputOpenDrain` | Write open-drain output PWM value (0..100 %) | stack, channel, value\(^7\)                      | 0 (Success) or −1 (Error)    |
| 8  | `getOutputOpenDrain` | Read open-drain Output PWM value (0..100 %) | stack, channel                                                        | Percentage of 0 to 24 V PWM [0..100] |
| 9  | `getInputOpto`     | Read opto status                         | stack, channel                                                        | 0 (OFF) or 1 (ON)             |
| 10 | `getBoardDiagnose` | Display info about the card, firmware version, power supply and temperature | Stack | firmware, version, power: voltage (V), temperature: CPU (°C) |

---

\(^3\) Stack level of the megaind card (selectable from address jumpers [0..7]). The megaind card is stackable to up to 8 cards in the same Raspberry Pi.

\(^4\) Selected channel number [1..4] of the megaind card I/O.

\(^5\) Percentage of 0 to 10 V [0..100].

\(^6\) Percentage of 4 to 20 mA [0..100].

\(^7\) Percentage 0 to 24 V PWM [0..100].
http://localhost:3000/megaind-rpi.setOutputVoltage?
Header {"Content-Type": "application/json"},
Body {"stack": 0, "channel": 2, "value": 25}.
Return: 0 (Success) or -1 (Error).

It is also possible to do a self-test by connecting the voltage outputs of distributed I/O in their own voltage inputs as shown in Fig. 14 (schematics available in https://github.com/pontarolli/megaind-rpi/tree/master/schematics, page 4/16). After setting the output voltages, it would be possible to get the value in the inputs using the getInputVoltage action. The self-test facilitates testing because no extra hardware (sensors/actuators) is needed.

In case it is required to remotely access the inputs and outputs of the distributed I/O, for example using a Web application or through the Internet, the API Gateway service is used. As explained, the API gateway service automatically creates an HTTP (s) route for requesting any of the I/O resources available on the distributed I/O service. It is possible to send HTTP requests
with any application, such as the Postman shown in Fig. 15, which shows a request for reading analog voltage inputs of the distributed I/O.

https://localhost:3000/megaind-rpi.getInputVoltage?
Header {"Content-Type": "application/json"}.
Body {"stack": 0, "channel": 2}.
Return: Percentage of 0 to 10 V [0..100].

The sequence of execution of this request by the web application can be seen in the sequence diagram below in Fig. 16. The GET api/megaind-rpi request firstly passes first through the API Gateway service, then through the Transporter service, until it gets to the distributed I/O service and then return back.

The average total request time was 22.9 ms, as well as the time in each part of the communication route, which are shown in the table of the Table 5.

Table 5
Communication times (T in Fig. 16) for an API request of the Distributed I/O.

|   | Action            | Average (ms) | Minimum (ms) | Maximum (ms) | Standard deviation (ms) | Successful rate (%) |
|---|-------------------|--------------|--------------|--------------|-------------------------|---------------------|
| 1 | Web application   | 22.9         | 16.3         | 111.5        | 22.3                    | 99.2                |
| 2 | API Gateway       | 19.0         | 13.5         | 346.6        | 41.7                    |                     |
| 3 | megaind-rpi       | 4.8          | 2.7          | 22.7         | 1.2                     |                     |
Application examples

In the distributed I/O of the Fig. 17, the megaind-rpi service is represented by a blue hexagon. This service implements the coding for industrial I/O DAQ using the Megaind automation card. The interface between the Raspberry Pi running the service and the Megaind card is via I2C (Inter Integrated Circuit) protocol.

Microservices are like LEGO pieces and creating an application is like composing microservices or using different microservices according to a sequence needed. As a result, the distributed I/O as microservice developed in this paper can be used as a basis for any other science or research application that needs to acquire data. In the industry scenario focused in this paper, we will present the following examples such as a process historian, a closed-loop process controller or even an IIoT application accessing data from the distributed I/O.

Example 1 – Internal application

There are two types of applications that can be created using the Moleculer architecture: internal application and external application. An application is a high-level functionality that uses microservices. The internal applications are all programmed using JavaScript using the Node.js, in which services communicate among them using the Transporter represented by the orange hexagon on Fig. 18. In this case an application service represented by a purple hexagon can request the data of an I/O of the distributed I/O using the Transporter. The Transporter locates the distributed I/O (megaind-rpi service) and returns the requested value (a sensor data for example).

Example 2 – External application and Replica

The external applications can be implemented in any type of platform, and the communication with the internal services uses HTTP (REST) communication in conjunction with the API Gateway service represented by a green hexagon in the Fig. 19. The API gateway bridges data from the outside to the inside of the Moleculer microservices.

In this case the data of an I/O of the distributed I/O is requested via an HTTP GET method from a web application. The API Gateway request is sent to the Transporter, which locates the distributed I/O (megaind-rpi service) and returns the requested value (a sensor data for example).

The example in Fig. 19 also presents a scenario in which redundancy of industrial I/O DAQ is needed. The distributed I/O as a microservice enables the redundancy of I/O data acquisition by the replication of the hardware and microservice. In this
redundancy scenario, there are more than one Distributed I/O connected to the same I/O. The replicated microservices (megaind-rpi) automatically enable the redundancy, when if one of the distributed I/O fails, other will be available to reply.

Examples of industrial I/O DAQ redundancy can be found in large automation manufacturers as shown in [24], Fig. 20, where the commercial distributed I/O (ET200M model) is instantiated twice for the same controller (S7-400H). In that configuration, if one of the distributed I/O fails, the other one is available for acquiring the sensor value. However, mandatory network coding is needed in order to implement the redundancy. Besides that, these commercial solutions for DAQ redundancy cost a high value in addition to not being open-source. However, for research and development projects, the solution proposed in this paper would be more viable in relation to costs, providing the same redundancy functionality with simplified deployment.

**Fig. 19.** External Application and Redundancy Examples using the Distributed I/O.

**Fig. 20.** Commercial Distributed Redundant I/O [24].
Example 3 – Closed-loop process controller

In order to demonstrate a closed-loop process controller example, a composition service was created and is represented by a purple hexagon in Fig. 21. This composition cyclically and sequentially requests the following microservices: distributed I/O (megaind-rpi service: sensor input), Control service (control algorithm), and distributed I/O (megaind-rpi service: actuator output). The composition service defines the setpoint value or profile as well as the cyclic rate. This closed-loop control example was presented by the authors in [19], in which a real application using the composition of microservices for process control using an industrial process pilot plant is investigated.

This example presents a research application of the distributed I/O service developed in this paper. The industrial pilot plant shown in Fig. 22 was developed by the authors together with the Emerson Automation Solutions company for enabling the research on microservice applications for automation and control. The pilot plant has a main tank in stainless steel at the bottom of the plant for water storage, the liquid can be pumped to the top of the plant in two different routes: for the acrylic

![Fig. 21. Application Composition Example using the Distributed I/O: Process Control Loop.](image)

![Fig. 22. Industrial Process Pilot Plant and the Distributed I/O highlighted in the electrical panel.](image)
tank or for the stainless-steel reservoir. The distributed I/O as microservice developed is used for data acquisition from the sensors (level, flow rate, line pressure and reservoir pressure) and to command the actuators (pumps with frequency inverter and proportional valve). Fig. 22 shows the distributed I/O, megaind-rpi service, deployed in the electrical panel of the industrial pilot plant. Fig. 23 shows an example of the pipe pressure control loop response during the operation of the industrial pilot plant.

Fig. 23. Example of a Control Loop Response using the Distributed I/O.

Fig. 24. Historian Application Composition Example using the Distributed I/O.

Fig. 25. Grafana Dashboard of the Process Historian service.
Example 4 – Process Historian

The next example also exemplifies a real research application within the pilot plant. Process historians are industrial applications that are used to store and analyze vital process and industrial data. It is essentially composed by a time-series database with an IHM software for data visualization (dashboard).

The distributed I/O can be used in order to create a process historian application. For this example, shown in Fig. 24, the historian service (purple hexagon) requests data the I/O data from the distributed I/O. In the historian service, the I/O data is cyclically requested and stored in a time series database (InfluxDB [25]), e.g. the I/O data is automatically stored with a timestamp (date and time). InfluxDB is an open-source time series database.

The I/O data obtained from the distributed I/O can be visualized in dashboards using Grafana [26]. Grafana is an open-source analysis and monitoring solution with database type access plugins, simplifying writing/reading procedures in the database for creating HM and historical data interfaces. Fig. 25 presents an example of a dashboard designed to monitor and analyze data from the industrial pilot plant in [19].

This application example emphasizes the potential of using the distributed I/O in a historian setup for IIoT and data analysis. The collected and stored information by the historian can be used in the future for processes diagnosis and KPI based performance analysis as well as data science and condition monitoring in IIoT and Industry 4.0 related applications [27].

Conclusions

Data acquisition (DAQ) is a vital step for science applications. Specifically in industry applications, distributed I/O is a traditional DAQ hardware used for remote and networked data acquisition of sensors and actuators. The use of service-oriented architectures in automation applications related to IIoT and I4.0 is a necessary reality for enabling interoperability and vertical interaction of devices and systems, in addition to enabling a distributed and modular architecture. This paper discussed the development and testing of an open-source and low-cost distributed I/O as a microservice.

Microservice enhances the flexibility and modularity of industrial applications, simplifying the network communication among services and applications. The distributed I/O relies on open source solutions for hardware (Raspberry Pi and industrial I/O expansion card) and software (Moleculer framework and Node.js), providing a reliable and affordable solution, mainly when compared to industrial ones. Its operation is similar to a traditional distributed I/O, with the advantage of being programmable, being able to incorporate customized functionality and making the I/O data acquisition interoperable, standardized and scalable.

Several applications examples for the distribute I/O were presented, such as internal and external applications, process historian and process controller. The operation instructions considered the assembly, installation and usage of the distributed I/O together with the other services available on the Moleculer framework such as the API Gateway and transporter. These instructions and examples provide the reader a first step into the recent scenario of service-oriented applications for industrial applications.

Future work is focusing on using microservices with containers like Docker in order to improve its management. Doing that, it will be possible to make them available in a network repository (Docker Hub) and manages them through a graphical interface (Portainer), facilitating its deployment when compared to the traditional method through terminal commands.

CRediT authorship contribution statement

Ricardo P. Pontarolli: Data curation, Writing – original draft. Jeferson A. Bigheti: Visualization, Investigation. Felipe O. Domingues: Supervision. Lucas B.R. de Sá: Software, Validation, Writing – review & editing. Eduardo P. Godoy: Conceptualization, Methodology, Software.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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

References

[1] “UniPi.” https://www.unipi.technology/ (accessed Oct. 12, 2021).
[2] “Analog IO expansion module - Industrial Raspberry Pi.” https://revolutionpi.com/analog-io-module/ (accessed Oct. 12, 2021).
[3] “Iono Pi Max – Industrial Raspberry Pi PLC - I/O UPS RS232 RS485 CAN CE FCC.” https://www.sferalabs.cc/product/iono-pi-max/ (accessed Oct. 12, 2021).
[4] “Monarco HAT - Analog and digital I/O, RS-485 and 1-Wire bus for Raspberry Pi.” https://www.monarco.io/ (accessed Oct. 12, 2021).
[5] J.A. Basson, A. Broekman, S.W. Jacobsz, TD-DAQ: A low-cost data acquisition system monitoring the unsaturated pore pressure regime in tailings dams, HardwareX 10 (Oct. 2021) e00221.

[6] A. Broekman, Willem Jacobsz, H. Louw, E. Kearsley, T. Gaspar, and T. S. da Silva Burke, “Fly-by-Pi: Open source closed-loop control for geotechnical centrifuge testing applications,” HardwareX, vol. 8, p. e00151, Oct. 2020, 10.1016/j.hdx.2020.e00151.

[7] J. Delsing, Local cloud internet of things automation: Technology and business model features of distributed internet of things automation solutions, IEEE Ind. Electron. Mag. 11 (4) (Dec. 2017) 8–21, https://doi.org/10.1109/MIE.2017.2759342.

[8] “How are Remote I/O and Distributed I/O Different? | RealPars.” https://realpars.com/distributed-io/ (accessed Sep. 30, 2021).

[9] “Blog on the Potential of IoT and SCADA Duo in Industry 4.0 | Altizoon.” https://altizoon.com/iot-scada-complementary-technologies/ (accessed Sep. 30, 2021).

[10] Y. Wang, T. Towara, and R. Anderl, “Topological approach for mapping technologies in reference architectural model industrie 4.0 (RAMI 4.0),” in Lecture Notes in Engineering and Computer Science, 2017, vol. 2.

[11] P. Adolphs et al, Structure of the Administration Shell Continuation of the Development of the Reference Model for the Industrie 4.0 Component, ZVEI and VDI (2016).

[12] S. Newman, “Building Microservices - Chapter 1, 4 and 11,” Building Microservices, p. 102, 2015.

[13] C. Richardson and F. Smith, “Microservices design and deployment with NGINX | Free Ebook.” https://www.nginx.com/resources/library/designing-deploying-microservices (accessed Oct. 07, 2020).

[14] M. Fowler, “Microservices.” https://martinfowler.com/articles/microservices.html (accessed Oct. 07, 2020).

[15] “AESOP Project.” https://www.imc-aesop.org/ (accessed Sep. 30, 2021).

[16] S. Karnouskos et al, “The IMC-AESOP architecture for cloud-based industrial Cyber-physical Systems,” in Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach, vol. 9783319056241, Springer International Publishing, 2014, pp. 49–88, 10.1007/978-3-319-05624-1_3.

[17] S. Karnouskos, T. Bangemann, C. Diedrich, Integration of Legacy Devices in the Future SOA-based Factory, IFAC Proceedings Volumes 42 (4) (Jan. 2009) 2113–2118, https://doi.org/10.3182/20090603-3-RU-2001.0487.

[18] J. A. Bigheti, M. M. Fernandes, and E. Paciencia. Godoy, “Control as a Service: A Microservice Approach to Industry 4.0,” in 2019 IEEE International Workshop on Metrology for Industry 4.0 and IoT, Metrolind 4.0 and IoT 2019 - Proceedings, Jun. 2019, pp. 438–443, 10.1109/METROL4.2019.8792918.

[19] R. P. Pontarolli, J. A. Bigheti, M. M. Fernandes, F. O. Domingues, S. L. Risso, and E. P. Godoy, “Microservice Orchestration for Process Control in Industry 4.0,” in 2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT, Metrolind 4.0 and IoT 2020 - Proceedings, Jun. 2020, pp. 245–249, 10.1109/Metrolind4.0.IoT48571.2020.9138228.

[20] M. Haizad, R. Ibrahim, A. Adnan, T. D. Chung, and S. M. Hassan, “Development of low-cost real-time data acquisition system for process automation and control,” 2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation, ROMA 2016, Feb. 2017, 10.1109/ROMA.2016.7847826.

[21] M. Abdallah and O. Elkeelany, “A survey on data acquisition systems DAQ,” ICC2009 - International Conference of Computing in Engineering, Science and Information, pp. 240–243, 2009, 10.1109/ICC.2009.24.

[22] “Molecular - Progressive microservices framework for Node.js.” https://molecular.services/ (accessed Jan. 28, 2020).

[23] “Raspberry Pi Stackable Card for Industrial Automation.” https://sequentmicrosystems.com/index.php?route=product/product&product_id=52 (accessed Jun. 19, 2020).

[24] “Configuration of Redundant I/O Modules in SIMATIC PCS 7 - ID: 28430682 - Industry Support Siemens.” https://support.industry.siemens.com/cms/document/28430682/configuration-of-redundant-i-o-modules-in-simatic-pcs-7?dl=0&lc=en-WW (accessed Nov. 18, 2021).

[25] “InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData.” https://www.influxdata.com/ (accessed Oct. 12, 2021).

[26] “Grafana: The open observability platform | Grafana Labs.” https://grafana.com/ (accessed Oct. 12, 2021).

[27] X. Gao, F. Yang, C. Shang, D. Huang, A review of control loop monitoring and diagnosis: Prospects of controller maintenance in big data era, Chin. J. Chem. Eng. 24 (8) (2016) 952–962, https://doi.org/10.1016/J.CJCHE.2016.05.039.

Eduardo Paciencia Godoy received the B.Eng. degree in Control and Automation Engineering at Itajubá Federal University (MG-Brazil) in 2003 and the M.Sc. and Ph.D. degrees in Mechanical Engineering at University of São Paulo at São Carlos (SP-Brazil) in 2007 and 2011, respectively. Currently he is an Associate Professor of the São Paulo State University (Unesp) at Sorocaba (SP-Brazil). His research interests include industrial networks and automation, networked control systems (NCS), wireless networks and telemetry, embedded electronics, Internet of Things and Industry 4.0.

Ricardo Pasquati Pontarolli received the B.Tech. degree in Industrial Automation from Technology College (FATEC) (SP-Brazil) in 2011 and the M.Sc. in Electrical Engineering from the São Paulo State University (Unesp) (SP-Brazil) in 2020. He is currently a Ph.D. student at Unesp and laboratory technician in industrial automation at the Federal Institute of São Paulo (IFSP) in Boituva (SP-Brazil). His research interests include industrial automation, supervision of industrial systems, Internet of things, embedded electronics, Industry 4.0, and Security mechanisms.
Jeferson André Bigheti received the B.Tech. degree in Electrical Systems in 1990, the Academic specialization in Mechatronic Engineering in 2007 and the M.Sc. and Ph.D. in Electrical Engineering in 2011 and 2020, from the São Paulo State University (Unesp) at Bauru (SP-Brazil). Currently he is a professor of the National Service for Industrial Learning (SENAI) at Lençóis Paulista (SP-Brazil). His research interests include automation, supervision of industrial systems, Internet of Things and Industry 4.0.

Felipe de Oliveira Domingues is currently a Control and Automation Engineering student at the São Paulo State University (Unesp) in Sorocaba (SP-Brazil). His research interests include industrial automation and Industry 4.0.

Lucas Borges Rodrigues de Sá received the B.Eng. degree in Electronic Engineering from National Institute of Telecommunications (Inatel) (MG-Brazil) in 2003. He is currently studying M.S. in Electrical Engineering at São Paulo State University (Unesp) – Sorocaba (SP). His research interests include industrial automation, supervision of industrial systems, Internet of things, embedded electronics and Industry 4.0.