Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 833–842

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

Architecture for Data Acquisition in Research and Teaching Laboratories

Walter Quadrini[a]*, Simone Galparoli[a], Domenico Daniele Nucera[a], Luca Fumagalli[a], Elisa Negri[a]

[a]*Department of Management, Economics and Industrial Engineering, Politecnico di Milano, p.zza Leonardo da Vinci, 32, 20133 Milan Italy*

Abstract

In the recent years, several research activities focused on the design, development and deployment of software architectures addressed to the monitoring, management and control of industrial plants, since the ongoing digitalisation of manufacturing companies demanded guidelines and cutting-edge solutions to the research community. In this context, more and more research centres' laboratories have been hence asked to run in parallel different architectures, relying on different research activities, whose feeds consist of the same signals from the machine pool. When these laboratories belong to universities, the request for data from students and trainees is added to the request for data from research activities. this often results in a network congestion with severe implications on the data integrity. For this reason, the authors have designed and implemented an open modular software architecture able to minimise the requests to the machines and to embody Service Oriented Architecture functionalities. The architecture has been named SHIELD, indeed SHIELD Has Integrated Existing Laboratory Data.

© 2021 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Industry 4.0 and Smart Manufacturing

Keywords: CPS; SOA; open architecture; modular architecture; vertical integration; industrial sofware architecture

1. Introduction

The introduction of the guidelines of the so-called "Industry4.0"[1] in 2011 paved the ground to a wide variety of Research and Development projects oriented to formalise the "digitalisation of processes" arisen thanks to the accessibility of the digital technologies that allow an easier integration of the components. These projects were

* Corresponding author. Tel.: +39 02-23999269
E-mail address: walter.quadrini@polimi.it

1877-0509 © 2021 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Industry 4.0 and Smart Manufacturing
10.1016/j.procs.2021.01.333

motivated by the demand of manufacturing companies to make the data from their assets available to different software applications and platforms in turn devoted to the production control, to the energy consumption monitoring, to the warehouse management and to other digital services companies may integrate in the perspective of the so-called "factory digitalisation"[2]. On this behalf, the research laboratories active in the field of industrial engineering have often been involved as testing facilities to deploy and benchmark the software platforms developed from time to time[3]. The software architectures of the different developed platforms are developed mainly from scratch for every research activity (according to the specific requirements) and their designs range from the classical "pipelines" (derived from ISA-95 directives[4]) to the so-called integrated Platforms as a Service (iPaaS) with countless different approaches, mainly relying either on middleware solutions, or on end-to-end connections, or on shared buses[5][6]. The perspective of a laboratory working on the deployment of these kind of software platforms implies, in the first instance, the effort to connect every single stack to the laboratory assets and, as a second but technologically remarkable effect, the overload of the communication interfaces (e.g., the OPC UA servers integrated in machine Programmable Logic Controllers (PLCs)[7]), which have to satisfy information requests from a wide set of software applications and cannot guarantee the on-time delivery and the data integrity. Furthermore, all the industrial engineering research facilities which can be framed in the set of Learning Factories [8] face also the requests for data coming from the students, which result in additional computation and communication loads for the machine pool.

For these needs, the Manufacturing Group of the Department of Management, Economics and Industrial Engineering in Politecnico di Milano has developed an integrated software architecture to allow simultaneous data requests from different services, hiding them to the assets on board communication devices.

This paper is hence structured as follows: Section 2 describes the background of existing solutions for manufacturing environments, Section 3 frames the research design, in terms of requirements and objectives, Section 4 shows the design of the proposed solution, Section 5 describes the technological tools and solutions adopted for the implementation, Section 6 illustrates the solution's application cases, Section 7 reports some discussions and final conclusions and opens to further developments and integration of new tools and technologies.

2. Background

The evolution of the informative systems of manufacturing companies relies often on a hierarchical structure, which reflects the overall enterprise model of the Purdue Enterprise Reference Architecture (PERA)[9]: this management-like structure has been transposed into an industrial automation perspective through the standardisation outcome of the well-known ANSI/ISA 95[4]. Subsequent refinements in the aforementioned standards led to release a new object-oriented model under the standard IEC 62264[10], which is often associated with the so-called "automation pyramid", which again implies a hierarchical structure where the information from the base of the pyramid (the machine pool) can reach the top level (usually ERP-like software) passing through the intermediate levels, where it is supposed to be filtered, parsed, wrapped and enriched according to the client specifications[11].

The major criticism levelled against this model lays in the hierarchy itself, which, for example occurs whenever an external entity requires the access to raw data coming from the machines (e.g. remote services for maintenance [12]), or forces the ERP to pass a request through all the levels to access eventual raw data from the PLCs (with subsequent delays between the request and the receiving of the information.

To overcome these issues, new modelling approaches have been formalised, in order to deal with the increasing spread of the so-called Cyber Physical Systems (CPS) and to make the software modules framed in the architectures capable to operate in distributed networks. In recent years, IEC 61499[13] has been formalised and is being adopted in order to address these issues by exploiting the CPS capability[14]: in particular, the aptitude of modern devices to directly handle complex communication protocols paved the ground to a paradigm where such devices can be directly queried by modules of every abstraction layer[15].

The adoption of this policy in architectural design implies technological choices over the communication policies devoted to implement the actual data exchange. These choices often lead to the deployment of a middleware[16] which, with a certain degree of confidentiality, can be defined as a buffer platform where the data are queued and formatted according to a certain lingua franca. This solution fits well for manufacturing environment, since, once the middleware is chosen the specific format is straightforward and every new module connected to the architecture is supposed to be interfaced with the data model of the exchanged information adapts to the chosen middleware

format. However, the multitude of architectures, tools and applications tested in a research laboratory often requires different datasets, with different update frequencies and cannot rely on the strict structures typical of industrial applications. For this reason, industry4.0Lab has designed SHIELD (SHIELD Has Integrated Existing Laboratory Data), a Service Oriented Architecture (SOA) able to provide to every connected consumer only the required data.

3. Research design

3.1. Research objectives

One of the important aspects to investigate is the scenario the SHIELD architecture is designed for, since the main application framework is the Research laboratory one. The laboratory environment differs from the industrial one for several aspects and some of them reflect in different requirements for a data acquisition architecture. The first aspect is the time horizon a user has in the development of a solution based on the data acquired: in a laboratory, typical users work on time-limited projects. They need to acquire data for a limited amount of time and, after the project has ended, the solutions are no more used, as the data acquisition pipeline. Beside of this, the typical student does not have by definition the complete knowledge on the assets he is going to work on and the information required to extract data should hence be minimized to help the user and ease the learning process. Another consideration lays in the fact that most of projects developed in a laboratory are proofs of concept and prototypes, so requirements such as reconfigurability, fast deployment, modularity, scalability are valued from users more than standard industrial requirements, often based on the need of immanent solutions almost maintenance-free.

The aim of this work is therefore to design, develop and document an architecture for industrial data acquisition developed specifically for a laboratory environment, whose general requirements are listed below:

- *Service oriented*: data flows and data manipulations should be enabled only as response to a specific user request, to increase efficiency and guaranteeing data control.
- *Distributed*: services should be provided by modules individually developed, deployed and connected through the architecture itself.
- *Self-documented*: the architecture should contain, and be capable of providing, all the information a user needs to access the available services

Another important aspect taken into consideration is the data access mechanism. In industrial environment the sensor reading procedure, as first step of the acquisition chain, is provided from the asset producer and there are no guarantees on the performances. This causes a loss of flexibility when multiple users and applications want to access data and can cause a bottleneck limiting the performance of the whole system[17]. For this reason, one of the focuses in the design phase has been the minimization of interactions with data sources and the transfer of computational complexity (e.g. data duplication, user specific data formatting) inside the core modules of the architecture, in order to have them at a level where scalability and native distribution on multiple hardware resources are granted.

3.2. Core modules identification

Due to the distributed nature of the architecture itself and the service-oriented design, the most important aspect and starting point for the design is the definition of the core services and of the core modules providing them.

With “core services” is intended the minimum set of internal services needed to grant the data to the users.

The core modules and their services are here stated:

- Connector
  - Data acquisition from source and data import into middleware
- Orchestrator
  - Acquisition of user’s request for data
  - Internal module orchestration
    - Data acquisition management
    - Garbage collection
    - Ontological translation
    - Specific messages definition
- Topic Manager

  - Creation of user specific messages from middleware data
  - Delivery of data to the user
- Static Database
  - High level description of the asset/sensors/signal provided to the user
  - Translation for modules from user languages to architecture internal one
- Middleware
  - Data accessibility and uniformity
  - Data persistence and guarantees against loss of data

4. Design of architecture

Starting from the requirements stated in previous section, the modules identified have been designed together with the interfaces connecting them one to the other and to the external environment.

4.1. Connector

The module identified as connector represents a blueprint for a component that can be instantiated multiple times. This module is the one granting the connection with all the CPSs. In this sense, some further requirements for a connector are needed. The most important is related to data acquisition frequency. A connector needs to be designed considering that it must be able to read the data, pre-process them and send them to the middleware at a frequency higher than the original sampling frequency of the signal itself, to avoid losses, to limit aliasing and to guarantee that all the produced data are delivered to the middleware. The acquisition capability depends on several factors, such as hardware and network and cannot be granted only by architectural design. Some aspects can, however, be considered to deal with this challenge and are here reported as a strong starting point to match the aforementioned requirements. These additional requirements are:

- Reduce the number of operations a connector needs to perform to the minimum set needed to grant its services.
- Ensure that the connector needs to publish only useful information into the middleware.
- Guarantee a structure allowing parallelization of connectors between different hardware peripherals.

The minimum set of operations a connector must perform are (i) the reception of signal requests, (ii) the acquisition of the related signals and (iii) the data publication to the middleware, following the architectural communication convention. To ensure the publication of useful information only, the acquisition of a signal from a connector should be started only if required and stopped when not required anymore.

4.2. Orchestrator

The second core module is the Orchestrator. This module provides the main service used from the user to interact with the architecture. The user sends, as service request, a list of signals it wants to acquire and some parameters for the acquisition. The Orchestrator calls and configures all the internal services needed to fulfil the request and answers the user with the instruction on how to gather the required information into the middleware.

This module can be classified as a low frequency one, since it is heavier on a computational and communication point of view with respect to other modules but, since it is performed only on demand, the low latency required to other modules is not mandatory for the Orchestrator.

4.3. Topic Manager

The Topic Manager is the module operating at high frequency and is responsible of reading data from the middleware, creating tailor-made messages and delivering them to the user. This module has additional requirements since it must respect eventual frequency and time requirements stated by the user.

For this reason, a minimum set of operations the Topic Manager has to perform are stated in the following list:

- Receiving the specification for the message as a service request.
- Evaluating when it is time to send a new message.
- Retrieving the required data from the middleware.
- Composing the message for the user.
- Uploading the message into the middleware.

The Topic Manager receives from the Orchestrator the information about data location on the middleware and about the message composition: therefore, the tasks it must perform are simple and can be easily distributed and parallelized. The policies for message delivery can be of three different types: *As fast as possible*, *At a given frequency*, *When at least one signal has changed*. Fig. 1a shows up the internal structure for the Topic Manager, based on a multithread software design able to fulfil the requirements above listed, while maintaining as little as possible the computational overhead.

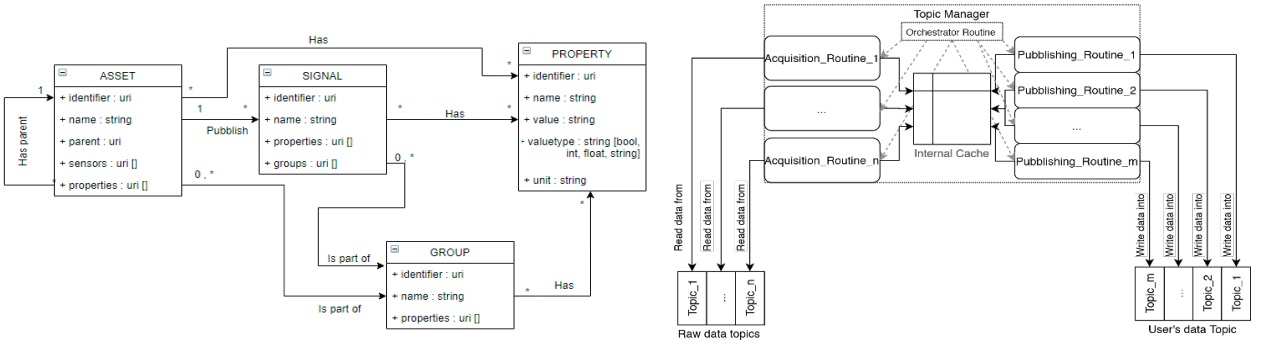

Fig. 1. (a) UML representation of database's classes and relations; (b) Topic Manager internal structure

4.4. Static Database

The Static Database lays in the architecture as one of the core modules to address the self-documentation requirement of the architecture. There are two main objectives the database is used for: the first one is to contain static data describing the CPSs connected to the architecture. The second one is to contain technical information needed for the architecture (e.g. timestamp convention, measurements units).

The first type of information should allow a user to approach a data acquisition procedure only via architecture's services, without any need to search through external documentation or previous knowledge.

The database is organized through a classes and relations model, suited for needs of describing sensors, signals and assets strongly interconnected one with the other in terms of common and inherited properties. Fig. 1a describes the internal database structure. The main idea of the database is that every module refers to a common and centralized source of information and it is, of course, used also to allow the user to give a meaning to the raw data published from the architecture.

4.5. Middleware

The last core module is the middleware. The need for a middleware is well documented in several scientific papers[16][18][19] as an important source of homogeneity in information exchange enabling communication between different agent in the industrial environment. In the scope of this architecture other important requirements are stated for the middleware.

The first one is a Pub/Sub architecture based on data stream, because data need to be ordered and directed to specific recipients in the architecture.

The second one is the capability to work as data buffer and data storage to grant a decoupling between heterogeneous source in terms of frequency and common frequency outputs.

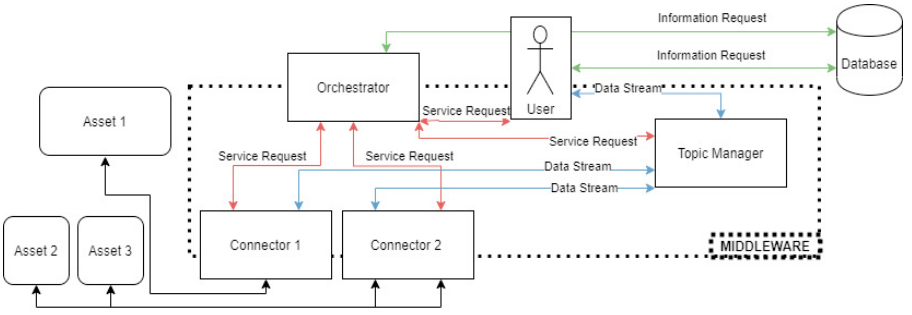

Fig. 2. Communication flows divided by communication type

4.6. Communication

The inner communication can be directed in two ways: to the middleware and to the database. Middleware communication is used for all the communication except for the database querying. Communications are divided into two types: services and data streams as shown in Fig. 2. Services are bilateral communication used to require a specific functionality to a specific module. The communication is bilateral since it implies a request and a response. Data streams are monodirectional communication used to send data from the source to the user. A data flow needs to be started with a service call. Messages in the monodirectional communication are defined in two different ways, depending on the source: for messages coming from a connector, the timestamp is unique for all the data, since they are all from the same source and assumed simultaneously acquired. The data for the final user attach a timestamp to each value since the generation time is supposed non-homogeneous for all the signals in the message. Data are provided to the user following a defined message structure, reported in Table 1 as the following Apache Avro[20] specification.

Table 1: Message for user's definition and example

```
{"type":"array",
  "items":{
     "type":"record",
     "fields":[
         {"name":"Signal_ID",
          "type":"String"},
         {"name":"Timestamp",
          "type":"String"},
         {"Name":"Value",
           "type":["null","int","bool","float","string"]}]}}
```

```
{"items":[
   {"Signlal_ID":"Press_xbg1",
    "Timestamp":"1594996120",
    "Value":56.4},
   {"Signal_ID":"Press_c63",
    "Timestamp":"1594996145",
    "Value":True},
   {"Signlal_ID":"Cobot_3",
    "Timestamp":"1594996215",
    "Value":44}]}
```

An important aspect about communication and resource location is represented from a set of mandatory properties each sensor must possess and that must be stored in the Static Database and that represents the set of information needed from connectors to start the acquisition. These properties are:

- "signal_id": unique identifier of the signal;
- "resource_type": the protocol the signal is provided on (e.g. OPC_UA, ROS, MQTT, …);
- "resource_address": the address of the resource source of data (e.g. the OPC_UA server);
- "signal_address": the signal specific address (e.g. the variable name for OPC_UA or the topic for MQTT).

All the other information about the data itself, such as the type, the unit of measurement or the full-scale value can be retrieved from the database and are application dependent.

5. Tools

As part of the development of the architecture a complete data pipeline from physical machines to data visualization, storage and analysis has been implemented to test the concepts and the design choices made during the design phase. To implement a real instance some additional decisions on the tools needed for the development have been made and are here documented.

For the middleware module, Apache Kafka has been used. Kafka is a middleware designed for data streams, aiming for a low latency high speed platform[21]. Among all its features, Kafka embodies some specificity that are considered in the choice, since it is natively distributed (allowing to have multiple nodes running the middleware, granting scalability and the possibility to work on multiple hardware simultaneously), it is topic-based and grants an internal data storage capability (allowing the middleware to work as a buffer).

For the modules development two languages have been selected, based on the required characteristics. For the modules that require flexibility and great data manipulation capability Python3 has been used. These modules are the Orchestrator and the User's module.

For the module requiring high performances and the simultaneous access to different resources, compiled languages like C++, Go or Rust were considered. The chosen language was Go due to its concurrent programming

primitives and the direct implementation of goroutines, which grant better scalability than Operative System threads[22][23]. It was thus used for developing the Connector and the Topic Manager.

For the Static Database, the selected tool has been chosen as OrientDB, a NoSQL database based on graph information representation. This is well suited for the typical industrial application where a tree of assets, signals, and properties is natively present[24]. The additional advantage of OrientDB is the capability to use SQL query to explore it together with his own specific query languages. This is an important factor given that the users of a Learning Factory vary from high skilled researchers to students and the possibility to use the well-known and studied SQL languages is helpful for novice users.

6. Application Experiments

The proposed architecture has been implemented and tested in the Industry4.0Lab in the Department of Management, Economics and Industrial Engineering at Politecnico di Milano[25]. The laboratory represents a specific environment, since it is both a Learning Factory and a research lab, so the users of the architecture are supposed to be students, researchers and companies that want to use it as a testbed develop specific Industry4.0 solutions. Given the positive results shown by laboratory tests, an additional test has been performed using the SHIELD architecture to gather data form an industrial machine. This second test, performed with an industrial asset, provided by Balance Systems, an Italian company designing and manufacturing automatic balancing machines, allowed us to obtain insights on the deployment criticalities and performance in industrial scenarios.

6.1. Assets

The experiment setup located in the laboratory and connected with the architecture is constituted by an assembly line composed by five automated assembly station, one automated robotic cell and one manual station. Each station is equipped with two PLCs: the operational one, controlling the station's sensors and actuators, and the energetic one, monitoring the station energy consumption. Both the PLCs are equipped with an OPC_UA server, publishing data in an internal network. This creates a set of 14 OPC_UA servers with 624 signals to be acquired. The operational PLCs provide data on sensor states, such as presence of pieces to be assembled or operational state of the station, while the energetic PLCs provide data on electrical and pneumatical consumption of the station.

In Balance Systems, the architecture has been used to gather data from BMK6, a machine for automatic, high production volume, balancing for rotating masses in electric motors. The machine provides signals for power consumption of its three main motors and for the pneumatic air flow by means of 10Hz MQTT signals.

6.2. Hardware deployment

For the Laboratory test the architecture has been deployed on two different machines, one acting as server and running the architecture and the other as user. The server machine has been equipped with an Intel i7 6700K CPU, 24GB DDR4 RAM, 1TB HDD, running Ubuntu 18.10LTS; as user machine, a simple laptop has been used, but since it was used only to gather the produced data, hardware details have not been reported. In the industrial test the server and user were on the same machine equipped with an Intel i5 7200U processor, 12GB DDR4 RAM, 254GB SSD, running Ubuntu 20.04.

6.3. Experiment results

The architecture, once deployed, has been tested to evaluate if the requirements had been matched and the desired functionalities for the architecture had been granted. Two different test typologies have been done for both the testing environments: functional and performative. Successfully passing the functional tests is considered a necessary condition to consider meet the requirements stated for the architecture.

The performed tests have been, in order:

- Single signal test: one signal has been required by a user and provided to him/her.
- Multiple signals from single resource address: multiple signals from the same resource address have been required by a user and provided to him/her.
- Multiple signals from multiple resource addresses: multiple signals from multiple resources have been required by user and provided to him/her.
- Service test: signals to be acquired have been added and removed multiple times by different users.

All tests successfully passed, granting the architecture compliance with the initial requirement and the second performative tests followed.

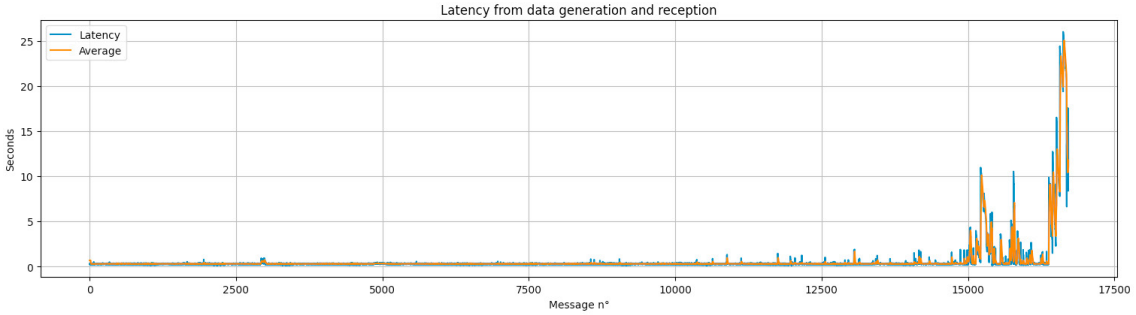

Fig.3. Experimental latency data from laboratory stress test

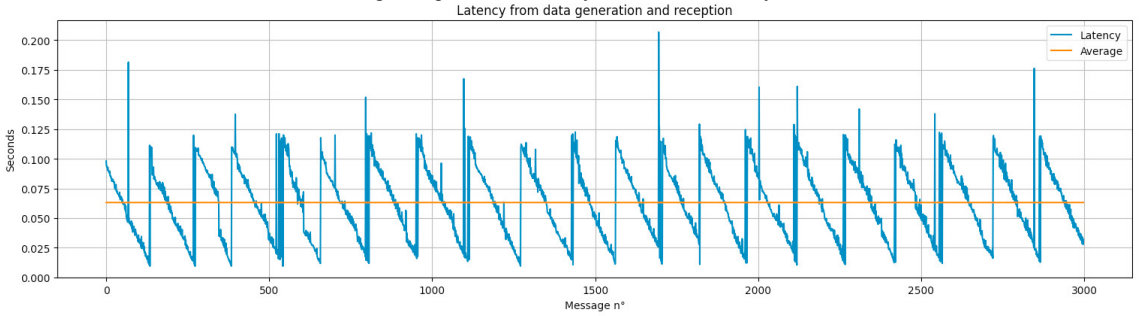

Fig. 4. Experimental latency data from industrial test

The first performative test, conducted in the laboratory, has been a stress test aimed at evaluating the performance in nominal operating conditions and the limits of the architecture in its implementation. For this test two parameters have been evaluated: the time interval between receiving two messages from the user and the latency, intended as the time delay between the data acquisition and the availability to the user. Both these parameters contain a contribution from the Network latency and from the computational time for the data to be processed by the architecture modules. During the test, all the 624 signals have been required by the users, the load on the architecture has been gradually increased by adding new users requiring a new topic with all the signals. The tests have started with 10 user (6240 signals required) and have ended at 80 users (49920 signals). All the signals have been required at 10Hz.

An overview of the stress test is reported in Fig 3. Based on the latency value between the data generation by the connector and the data reception by the user, the architecture has been considered stable when the latency moving average (10 samples) values are constantly under the mean $+3\sigma$ value. This is true up to 78 topics (48672 signals required). Based on the result of this test, repeated several times in similar conditions, it can be stated that the architecture satisfies a real-like laboratory condition. The average latency in the stable region of the test is around 0.30 seconds.

Furthermore, given the achieved results, the architecture has been tested also in a non-laboratory context to verify its industrial applicability[1]. The architecture has been hence stress-tested in Balance Systems, where the latency between data generation and consumption has dropped to 0.05 seconds due to the better quality of the data acquisition hardware and the data flow has remained stable, as reported in Fig.4, without losses and without significant changes in latency. The latency value is aligned with the one obtained through laboratory experiments, since in the Balance Systems test the sampling frequency was guaranteed by the data source, while in the laboratory

[1] Datasets produced during the performative tests are available at https://github.com/simogalpa/shield.git

test the UPC_UA servers have been queried at a 2Hz frequency and the Topic Manager has replicated the same data (with the same timestamp) 5 times, leading to an average bias on the computed latency of 0.25 seconds.

It can therefore be stated that, despite the architecture designed for laboratory and research field, there are good prospects for developing and exploiting it also for low frequency monitoring, measurement, maintenance and control applications in industrial contexts.

7. Conclusions and future works

The work illustrated in this paper proposes an architecture for the acquisition of data from industrial assets, following architectural requirements specifically developed for a research and didactic laboratory. The designed architecture has been implemented in a laboratory environment to validate the design concept and to investigate the research directions for future works and in industrial environment to test the applicability also in this context.

The tests conducted have shown the capability to fulfil the task it has been designed for with good performance and stability. The requirements stated before the design phase guided the design itself in directions capable of granting that the architecture, once implemented, accomplishes its tasks under policies aligned with the researchers' expectance when formulating the requirements. More in details the architecture:

- Can be used for a current time acquisition of data from industrial assets,
- Represents a homogeneous data layer, granting data consistency starting from non-homogeneous data sources,
- Allows multiple users to access the same data source without creating additional loads on the data sources,
- Moves the additional load needed to distribute data to different users and create custom information flows to a structure that is natively distributed and can scale up with the addition of new hardware and software nodes to keep up with computational requests.

The main criticalities in the work are represented from the limitations on the testing scenario that can hide bottlenecks or criticalities in the case of complex, multiuser and multisource scenarios. To cope with this, immediate future works will integrate in the proposed architecture several existing services and research solutions, such as an Automated Guided Vehicles (AGVs) fleet management system[26] and a real time monitoring tool for predictive maintenance[12]. Furthermore, the acquisition of feedbacks from researchers and students using the architecture in their daily activities will help identifying criticalities and implementing upgrades in a continuous improvement perspective guided by the usage of the SHIELD architecture itself. The so-achieved robustness will allow SHIELD deployment in real factories as a central collector for the field-generated data.

Acknowledgements

This work is supported by European Union funded project DIMOFAC (GA 870092) and Lombardy funded project SMART4CPPS (ID: 236789 CUP: E19I18000000009)

References

[1] Schweichhart K. Reference Architectural Model Industrie 4.0 (RAMI 4.0) - An Introduction. Plattf Ind 40 2016.

[2] De Carolis A, MacChi M, Negri E, Terzi S. Guiding manufacturing companies towards digitalization a methodology for supporting manufacturing companies in defining their digitalization roadmap. 2017 Int. Conf. Eng. Technol. Innov. Eng. Technol. Innov. Manag. Beyond 2020 New Challenges, New Approaches, ICE/ITMC 2017 - Proc., 2018. https://doi.org/10.1109/ICE.2017.8279925.

[3] Cimino C, Negri E, Fumagalli L. Review of digital twin applications in manufacturing. Comput Ind 2019. https://doi.org/10.1016/j.compind.2019.103130.

[4] International Society of Automation. Enterprise-Control System Integration Part 2 : Object Model Attributes. 2001.

[5] Mohammed WM, Ramis Ferrer B, Iarovyi S, Negri E, Fumagalli L, Lobov A, et al. Generic platform for manufacturing execution system functions in knowledge-driven manufacturing systems. Int J Comput Integr Manuf 2017. https://doi.org/10.1080/0951192X.2017.1407874.

[6] Fumagalli L, Negri E, Severa O, Balda P, Rondi E. Distributed control via modularized CPS architecture Lessons learnt from an industrial case study. IFAC-PapersOnLine 2018. https://doi.org/10.1016/j.ifacol.2018.08.417.

[7] Zezulka F, Marcon P, Bradac Z, Arm J, Benesl T, Vesely I. Communication Systems for Industry 4.0 and the IIoT. IFAC-PapersOnLine 2018;51:150–5. https://doi.org/10.1016/j.ifacol.2018.07.145.

[8] Abele E, Chryssolouris G, Sihn W, Metternich J, ElMaraghy H, Seliger G, et al. Learning factories for future oriented research and education in manufacturing. CIRP Ann - Manuf Technol 2017. https://doi.org/10.1016/j.cirp.2017.05.005.

[9] Bernus P, Nemes L, Schmidt G. Handbook on Enterprise Architecture. Strategy 2003. https://doi.org/10.1007/978-3-540-24744-9.

[10] IEC. IEC 62264-2: Enterprise-control system integration - Part 2: Objects and attributes for enterprise-control system integration". IEC 62264-2 Stand., 2015.

[11] Fumagalli L, Pala S, Garetti M, Negri E. Ontology-Based Modeling of Manufacturing and Logistics Systems for a New MES Architecture. IFIP Adv. Inf. Commun. Technol., 2014. https://doi.org/10.1007/978-3-662-44739-0_24.

[12] Tavola G, Caielli A, Taisch M. An “additive” architecture for industry 4.0 transition of existing production systems. Stud. Comput. Intell., 2020. https://doi.org/10.1007/978-3-030-27477-1_20.

[13] Vyatkin V. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. IEEE Trans Ind Informatics 2011. https://doi.org/10.1109/TII.2011.2166785.

[14] Barni A, Brusaferri A, Cavadini FA, Landolfi G, Patil S, Piga D, et al. Fostering the creation of a digital ecosystem by a distributed IEC-61499 based automation platform. IEEE Int. Conf. Ind. Informatics, 2019. https://doi.org/10.1109/INDIN41052.2019.8972293.

[15] Garetti M, Fumagalli L, Negri E. Role of Ontologies for CPS Implementation in Manufacturing. MPER - Manag Prod Eng Rev 2015;6:26–32. https://doi.org/10.1515/mper-2015-0033.

[16] Navet N, Song Y, Simonot-Lion F, Wilwert C. Trends in automotive communication systems. Proc. IEEE, 2005. https://doi.org/10.1109/JPROC.2005.849725.

[17] Raza M, Aslam N, Le-Minh H, Hussain S, Cao Y, Khan NM. A Critical Analysis of Research Potential, Challenges, and Future Directives in Industrial Wireless Sensor Networks. IEEE Commun Surv Tutorials 2018. https://doi.org/10.1109/COMST.2017.2759725.

[18] Rellermeyer JS, Alonso G, Roscoe T. R-OSGi: Distributed applications through software modularization. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2007. https://doi.org/10.1007/978-3-540-76778-7_1.

[19] He W, Xu L Da. Integration of distributed enterprise applications: A survey. IEEE Trans Ind Informatics 2014;10:35–42. https://doi.org/10.1109/TII.2012.2189221.

[20] Barba-González C, Nebro AJ, Benítez-Hidalgo A, García-Nieto J, Aldana-Montes JF. On the design of a framework integrating an optimization engine with streaming technologies. Futur Gener Comput Syst 2020. https://doi.org/10.1016/j.future.2020.02.020.

[21] Le Noac’h P, Costan A, Bougé L. A performance evaluation of Apache Kafka in support of big data streaming applications. Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017, 2017. https://doi.org/10.1109/BigData.2017.8258548.

[22] Whitney J, Gifford C, Pantoja M. Distributed execution of communicating sequential process-style concurrency: Golang case study. J Supercomput 2019;75:1396–409. https://doi.org/10.1007/s11227-018-2649-2.

[23] Jo H, Ha J, Jeong M. Light-Weight Service Lifecycle Management for Edge Devices in I-IoT Domain. 9th Int Conf Inf Commun Technol Converg ICT Converg Powered by Smart Intell ICTC 2018 2018:1380–2. https://doi.org/10.1109/ICTC.2018.8539579.

[24] Barmpis K, Kolovos DS. Evaluation of contemporary graph databases for ecient persistence of large-scale models. J Object Technol 2014. https://doi.org/10.5381/jot.2014.13.3.a3.

[25] Fumagalli L, Macchi M, Pozzetti A, Taisch M, Tavola G, Terzi S. New methodology for smart manufacturing research and education: The lab approach. Proc. Summer Sch. Fr. Turco, 2016, p. 42–7.

[26] Quadrini W, Negri E, Fumagalli L. Open interfaces for connecting automated guided vehicles to a fleet management system. Procedia Manuf 2020;42:406–13. https://doi.org/10.1016/j.promfg.2020.02.055.