ONE4ALL - Agile and modular cyber-physical technologies supported by data-driven digital tools to reinforce manufacturing resilience

Project nr: **101091877**

D3.1 Technology scouting and Intelligent Orchestration Platform specifications

Version: 1.0

Funded by the European Union

ONE4ALL project is funded by the European Union under the Grant Agreement No 101091877. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Characteristics of deliverable	
Title	ONE4ALL
Partner	CRIT
Contributors	IDE, AUTO, SDU
Short description of deliverable	Survey on innovative technologies for an intelligent services and data orchestration platform; Intelligent orchestration platform architecture
Submission date	31/05/23
Type	Report
Audience	Public
Key words	Technologies survey; Orchestration platform; Data platform; Industry 4.0; Smart Manufacturing

ONE4ALL Key Facts

Acronym	ONE4ALL
Project title	Agile and modular cyber-physical technologies supported by data-driven digital tools to reinforce manufacturing resilience
GA n°	101091877
Starting date	01/01/2023
Duration-months	48
Call (part) identifier	CLIMATE NEUTRAL, CIRCULAR AND DIGITISED PRODUCTION 2022 (HORIZON-CL4-2022-TWIN-TRANSITION-01)
Type of Action	HORIZON-RIA HORIZON Research and Innovation Actions
Topic identifier	HORIZON-CL4-2021-TWIN-TRANSITION-01-03
Consortium	10 organisations, all EU Member States
Model GA type	HORIZON Action Grant Budget-Based

ONE4ALL Consortium Partners

N.	Partner	Acronym	Country
1	IDENER RESEARCH & DEVELOPMENT (Coordinator)	IDE	ES
2	INNOPHARMA TECHNOLOGY	INO	IE
3	CRIT	CRIT	IT
4	EXELISIS	EXE	GR
5	UNIVERSITY OF SOUTHERN DENMARK	SDU	DEN
6	AUTOMATIONWARE	AUTO	IT
7	MADAMAOLIVA	MOL	IT
8	HOLOSS	HOLO	PT
9	DORTMUND UNIVERSITY	TUDO	DE
10	ORIFARM	ORI	CZ

Disclaimer

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Executive Summary

This deliverable presents a technological scouting of an open-source Intelligent Orchestration Platform (IOP) conducted with the objective of identifying current trends, opportunities, and challenges in implementing Industry 4.0 reference architectures, data architectures, specifically Kappa and Lambda, and open-source frontend development frameworks.

The investigation on Industry 4.0 reference architectures formed the initial focus of this analysis. Industry 4.0 signifies the next phase in the digitization of the manufacturing sector, with smart factories at its heart. The crucial component of the Industry 4.0 reference architecture is the implementation of IOP, where the interoperability of systems and data transparency are emphasized. Successful implementations of reference architectures were found across a wide range of industry sectors. These examples provide clear evidence of the efficacy and adaptability of such reference architectures, revealing their capacity to enhance operational efficiency, data utilization, and decision-making process through real-time data analysis.

Next, the scouting delved into the exploration of data architectures, focusing on Kappa and Lambda. Lambda architecture is designed to handle massive quantities of data by using batch processing and stream-processing methods. It delivers robust and fault-tolerant capabilities but comes with a higher level of complexity. On the other hand, Kappa architecture is a simpler alternative, proposes to unify the batch and stream processing by considering all data as a stream. Case studies highlighted in the scouting demonstrate successful applications of both architectures, emphasizing the choice between them hinges on the nature and requirements of the specific use case.

The exploration of open-source frontend development frameworks represented the third dimension of this technological scouting. Open-source frameworks are fundamental for developers as they expedite the development process, enhance collaboration, reduce costs, and improve the maintainability and scalability of applications. A range of robust frameworks were presented at high level, including Angular, React, and Vue.js. These have demonstrated significant versatility, strong community support, and comprehensive documentation.

Implementation examples were found from academic publications and were systematically collated and reported as part of this scouting. This data presents tangible evidence of the efficacy of the explored architectures and frameworks and provides an insight into how such technologies can be implemented. These reports serve also as examples on how potential challenges could be managed, by looking at previous implementations and logics.

Each of the aforementioned elements, with their advantages and disadvantages, can serve as tools for the implementation of the IOP to be used within this project. However, their successful implementation demands a comprehensive understanding of the organizations' unique needs and capabilities.

Table of contents

List of acronyms

RCPM	Reconfigurable Cyberphysical Production Module
DT	Digital Twin
IOP	Intelligent Orchestration Platform
SDK	Software Development Kit
PLC	Programmable Logic Controller
RAMI 4.0	Reference Architectural Model Industry 4.0
IIRA	Industrial Internet Reference Architecture
LASFA	LAsim Smart FActory
SCADA	Supervisory Control and Data Acquisition
MES	Manufacturing Execution System
ERP	Enterprise Resource Planning
ROS	Robot Operating System
PLC	Programmable Logic Controller
IoT	Internet of things
OPC	Open Platform Communications
DOM	Document Object Model
CPPS	Cyber-Physical Production Systems
PLM	Product Lifecycle Management
CRM	Customer Relationship Management
MDM	Master Data Management
SCM	Supply Chain Management
APIs	Application Programming Interface
BPM	Business Process Management
OSS	Open Source Software

List of figures

List of tables

1. Intelligent Orchestration Platform Introduction

1.1. Overall concept of the IOP

The overall purpose of the Intelligent Orchestration Platform (IOP) in the ONE4ALL project is to deliver a secure and open-sourced platform that interconnects all digital and physical modules within the manufacturing supply chain (Figure 1). The IOP aims to gather real-time information from these modules and present it through user-friendly interfaces, improving the understanding among end-users (managers and operators) about the production line and how to organise and control it. The platform covers various aspects such as sustainability, product quality, resource consumption, production activities, orders, and business analysis. It also incorporates a suggested decision-making strategy at different levels and throughout the supply chain, with both local and remote access. Additionally, the IOP integrates end-users' feedback into the loop, ensuring their active participation and involvement.

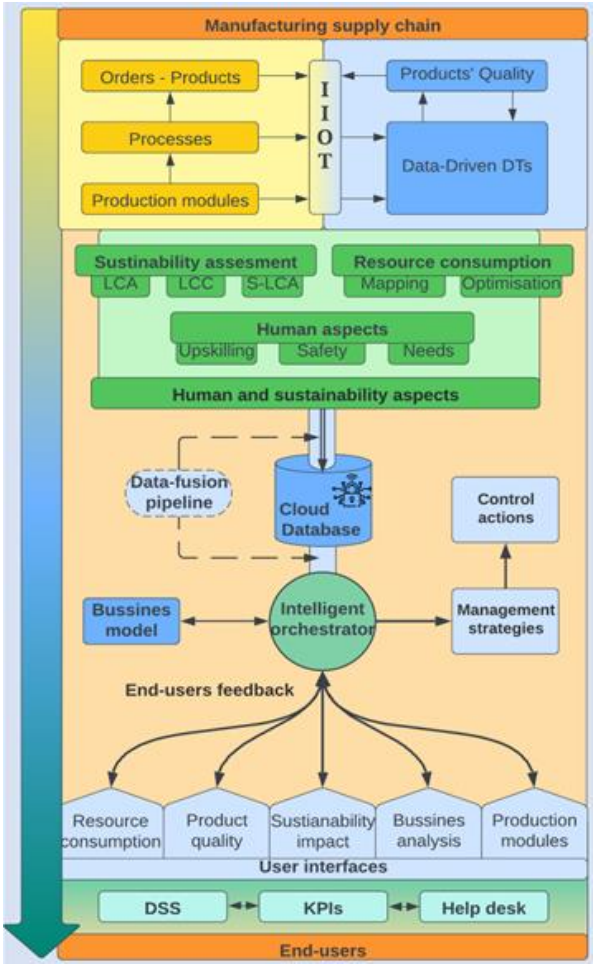


Figure 1: IOP positioning-interconnections overview

1.1.1. Positioning within ONE4ALL

The IOP serves as a central hub within the ONE4ALL project, responsible for connecting and orchestrating modules throughout the manufacturing supply chain. By seamlessly integrating with diverse services and devices, it facilitates smooth communication and data flow. The IOP serves as a gateway for data intake, collecting and processing information from physical and digital devices within the supply chain. It leverages this data to enable the creation and operation of data-driven digital twins, providing valuable insights and empowering end-users in their decision-making process.

Additionally, the IOP acts as a central hub, extending its orchestration capabilities to RCPM. It receives information from collaborative robots, which constitute the physical part of the RCPM, and sends relevant data back to them, ensuring a bidirectional flow of information and enabling efficient collaboration between the IOP and RCPM for optimised operations.

1.2. Purpose and objectives

The purpose of the Intelligent Orchestration Platform (IOP) in the ONE4ALL project is to enhance the supply chain by improving end-user understanding, promoting effective decision-making, and facilitating the integration of smart technologies. The objectives of the IOP include improving end-user understanding through real-time information presentation, assessing end-users in decision-making through a self-learning system, providing support and resources, and facilitating overall comprehension of the IOP through a Software Development Kit (SDK). Additionally, the IOP aims to develop a data fusion pipeline for efficient collection, processing, and formatting of heterogeneous data. With a microservices-based architecture, the IOP enables flexibility, scalability, fault isolation, and efficient resource utilisation.

2. Technological scouting

In this section, a technological scouting analysis is presented that centres around the exploration of industry 4.0 reference architectures and data architectures. In section 4.2.2 a general introduction on commonly used tools for the development and design of the IOP frontend is given. The focal point of our analysis was the requirement for the technologies to be open source. Our objective was to identify prior solutions that not only provide the benefits of openness and collaboration but also possess scientific documentation to support their architecture design and implementation, with a focus on robotic arms, collaborative robots, data architectures and integration of the Robot Operating System (ROS).

To ensure the robustness and reliability of the identified technologies, we sought scientific and technical documentation that provided insights into the aforementioned topics. This documentation allows for a comprehensive understanding of the underlying principles, design choices, and key components of each technology. By assessing the scientific documentation, we gain valuable insights into the scalability, maintainability, and adaptability of the technologies, enabling informed decision-making regarding their suitability for the project's requirements.

Through a systematic approach, we have curated a selection of articles that describe open source technologies that meet the aforementioned criteria. The subsequent sections of this document will provide detailed profiles of each technology.

2.1. Reference Architecture in Industry 4.0

The advent of Industry 4.0 has brought about a significant change in manufacturing operations, necessitating the seamless interaction of end-to-end industrial systems. While many manufacturing companies are still grappling with the challenges posed by Industry 4.0, reference architectures have increasingly become adopted in different areas as guides for engineers on how to structure and operate their systems. Companies make varied experiences with these architectures depending on the specific use cases. However, there is currently no complete understanding of existing representative architectures. Therefore, there is the need to review and analyse existing reference architectures for Industry 4.0, assessing their suitability for facilitating Industry 4.0 procedures and solutions.

In [1], the authors reviewed six different Industry 4.0 reference architectures, discussing their coverage, description, how to use them and different technologies and tools that can support their use. Of the six identified reference architectures, only five are taken into account in this deliverable, since IBM Industry 4.0 architecture is not open source.
To set a standard for the coverage and comparison of Industry 4.0 reference architectures, all were mapped against the five levels of the industrial automation pyramid as a reference guide:

1. **Field Level:** consisting of physical entities, including machines, devices, actuators, and sensors in the field or on the production floor.
2. **Control Level:** represented by a PLC (Programmable Logic Controller) and/or a PID (Proportional–Integral–Derivative controller) controlling one single device at the field level)
3. **System/process Level:** referred as SCADA (Supervisory Control and Data Acquisition), which controls several devices at the field level in a combined way through the output from multiple PLCs.
4. **Operation Level:** consists of systems that monitor the entire manufacturing process from the raw materials to the finished product, also referred to as MES (Manufacturing Execution System).

5. **Enterprise Level:** consists of systems for integrated management of companies, also referred to as ERP (Enterprise Resource Planning).

2.1.1. IIRA

IIRA (Industrial Internet Reference Architecture - Figure 2) is a domain-independent, industry-driven architecture that includes manufacturing, healthcare, energy, smart city, and others, and has been developed by the US-led IIC (**Industrial Internet Consortium**[1]), which has invested in the worldwide adoption of IoT in the industrial context. It is one of the first architectures specifically targeting Industry 4.0 appeared in 2015, about five years after the concepts of Industry 4.0, Industrial Internet, and smart factories had started to become popular. Emerging from accumulated experience in IoT systems, IIRA focuses on the functionalities required in the industrial domain, including business, operation, prognostics, optimisation, information analytics, and monitoring/control of devices. One peculiarity of this architecture is that it details technical issues, mostly reusing the experience from IoT system architectures.

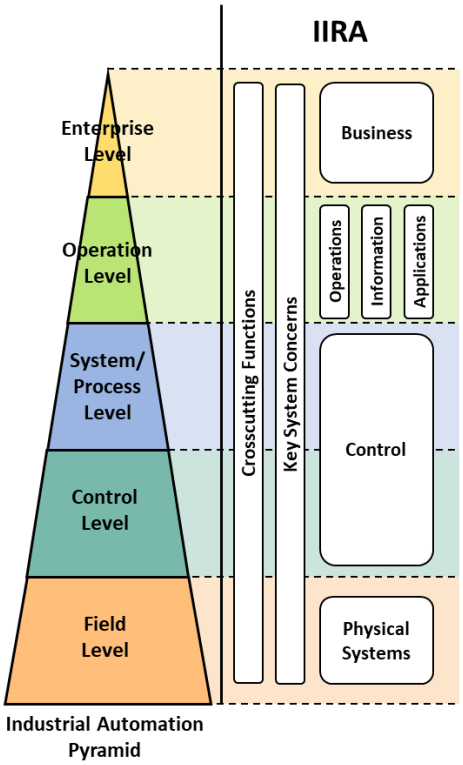


Figure 2: Industrial Internet Reference Architecture graphical overview

The IIRA architecture comprises 8 different modules, namely:

- **Business:** addresses all functional logic to support business processes and activities of business functions.
- **Operation:** addresses provisioning, management, monitoring, and optimisation of industrial control/automation systems.
- **Information:** addresses the management and processing of data from other modules; transforming, persisting, and modeling or analysing data.
- **Application:** responsible for the application logic to realise specific business functionalities.
- **Control:** addresses the industrial control/automation systems receiving data from sensors, applying rules and logic, and exercising control over physical systems.
- **Physical systems:** refers to machinery and devices on the shop floor.
- **Crosscutting Functions:** addresses all functions that enable major system functions with specific characteristics of Industry 4.0 systems, like industrial analytics, intelligence/resilience control, distributed data management, and full connectivity.
- **Key System Concerns:** addresses quality attributes, such as safety, security, resilience, reliability, privacy, scalability, and others, to achieve trustworthiness.

IIRA is presented as a set of architectural viewpoints where the functional view specifically addresses the functional components, their structure and interrelations, as well as interactions with external elements of the environment.

[1] IIRA -> https://www.iiconsortium.org/IIRA/

2.1.2. RAMI 4.0

RAMI 4.0 (Reference Architectural Model Industry 4.0 - Figure 3) is a domain-specific, government-driven architecture by a consortium led by the Association of German Engineers (VDI) and German Electrical and Electronic Manufacturers' Association (ZVEI) has designed RAMI. Like IIRA, it was one of the first architectures specifically designed to address Industry 4.0 requirements. To assure alignment with real industry needs, consortia of companies and research institutions were established to design, evolve, experiment, and adopt it.

One peculiarity of this architecture is to put together different dimensions of the Industry 4.0 space and to describe all crucial components of Industry 4.0. It could be represented as a three-dimensional structure (Value stream, Hierarchy levels, and Layers), RAMI 4.0 is still based on international standards: IEC 62890 (for lifecycles of products and production) as well as IEC 62264 and IEC 61512.

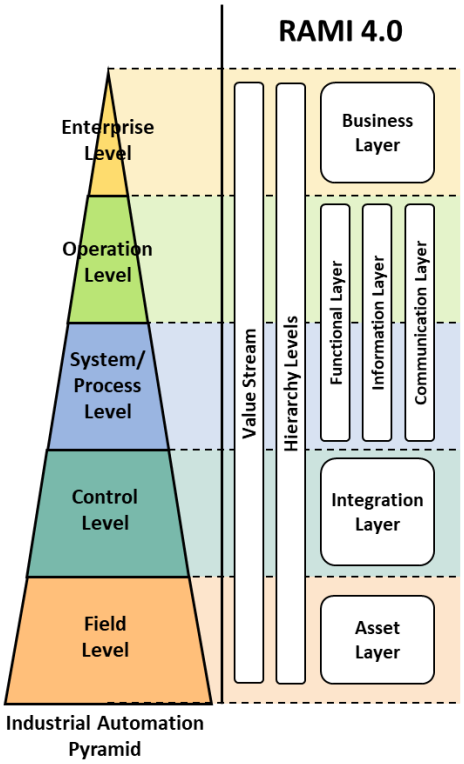


Figure 3: Reference Architectural Model Industry 4.0 graphical overview

The RAMI 4.0 architecture comprises 8 different modules, namely:

- **Business Layer:** addresses business strategy, business environment, and business goals.
- **Functional Layer:** addresses all logical and technical functions of all assets, also enabling remote access and horizontal integration.
- **Information Layer:** addresses the organisation of real-time/non-real-time data to provide a holistic view and information on physical objects, products and materials manufactured.
- **Communication Layer:** addresses the communication between information and integration layers through a uniform data format combined with a means to make the data available.
- **Integration Layer:** addresses the link between the physical and digital world by transforming and connecting the physical objects into the digital world.
- **Asset Layer:** addresses physical objects, such as production equipment, product parts, documents and people.
- **Value Stream:** addresses the management of lifecycles of products and production based on IEC 62890.
- **Hierarchy Levels:** based on IEC 62264 and IEC 61512, plus three layers (Field Device, Product, and Connected World) to enable flexible systems/physical parts and their direct integration into the factory and with external partners.
- ZigBee and Hadoop platform

2.1.3. SITAM

SITAM (Stuttgart IT-Architecture for Manufacturing - Figure 4) is an academic, multi-layered architecture designed within several research projects led by the University of Stuttgart, Germany. Developed for the first time in 2016, it has the purpose of facilitation, which essentially means promoting the sharing and/or reuse of knowledge related to architect Industry 4.0 systems.

This architecture focuses on integration/interoperability in smart factories by presenting three middlewares (for Integration, Mobile, and Analytics). The Integration middleware contains five buses to connect several systems to the entire product lifecycle, from product engineering to usage/support, as well as IT systems, physical devices, and other data sources.

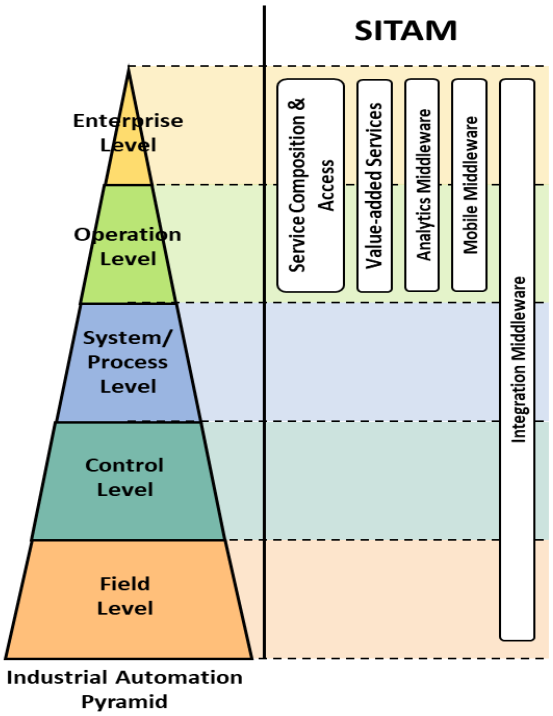


Figure 4: Stuttgart IT-Architecture for Manufacturing graphical overview

The SITAM architecture comprises 5 different modules, namely:

- Service Composition & Access: addresses the composition and access to value-added services by app composer and app marketplace.
- Value-added Services: addresses the services that create new value for the business by packaging and combining functionalities of other middlewares.
- Analytics Middleware: addresses manufacturing-specific analytics components for data-driven factories.
- Mobile Middleware: addresses the provision/acquisition of mobile information to develop and integrate manufacturing-specific mobile apps.
- Integration Middleware: addresses all services and corresponding data exchange to provide mediation and orchestration functionalities.

Regarding enabling tools, the authors report a lack of information about specific technologies. This lack is unfortunately confirmed, since no specific use case was found that specifically addressed the usage of the SITAM architecture.

2.1.4. IVRA

IVRA (Industrial Value Chain Reference Architecture - Figure 5) is a conceptual architecture maintained by the Industrial Value Chain Initiative (2018), Japan. IVRA focuses on an overall view of how a smart factory could detailed into its components, and how its modules could be combined to achieve the general functions of manufacturing.

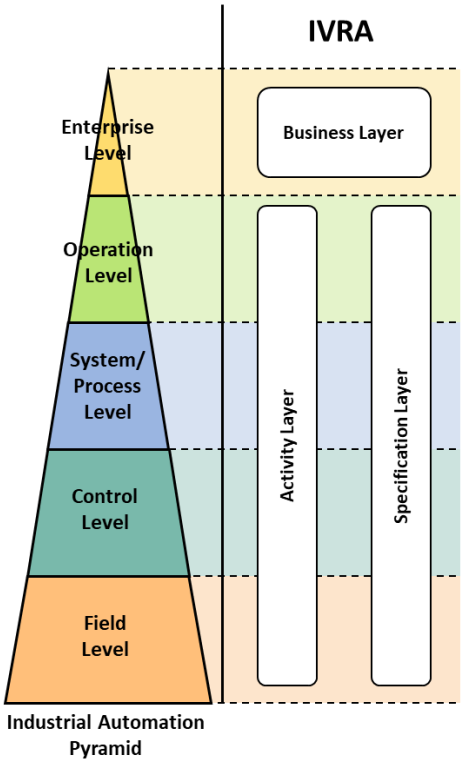


Figure 5: Industrial Value Chain Reference Architecture graphical overview

The IVRA architecture comprises 3 different modules, namely:

- **Business Layer:** addresses the management of business strategies and products/services.
- **Activity Layer:** addresses all concrete activities performed by people, machines, and software.
- **Specification Layer:** addresses the engineering to transmit, process, and reuse knowledge and know-how.

The **Business Layer** itself can be represented as a cube composed of three views:

- asset valuable for manufacturing units: plant, product, process, and personal
- management: regarding quality, cost, delivery, and environment
- activity: the activities Plan, Do, Check, and Action

2.1.5. LASFA

LASFA (LAsim Smart Factory - Figure 6) is a two-dimensional architecture proposed by the University of Ljubljana, Slovenia in 2019. LASFA is based on RAMI 4.0 layers and presents four building blocks (Business Process Management; MES + Digital Twins; Digital Twins for processes, logistics, and products; and Control process communication Production processes), which contain well known systems (e.g., ERP, MES, and PLM), digital twins, and the information/ data flow among them.

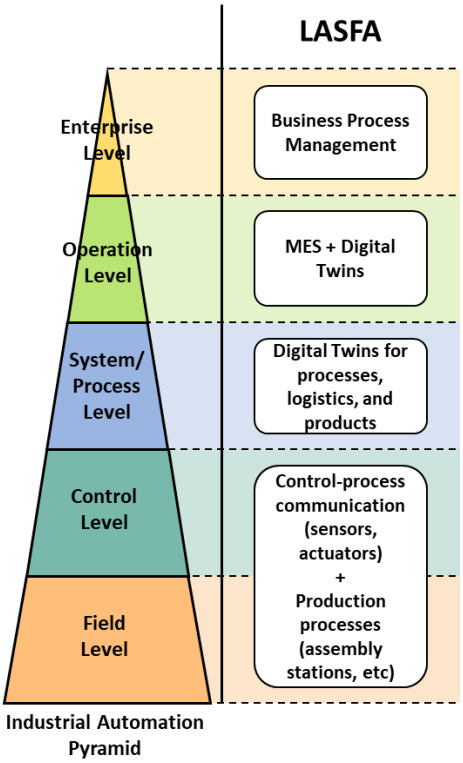


Figure 6: LAsim Smart FActory graphical overview

The LASFA architecture comprises 4 different modules, namely:

- Business Process Management: provides an integrated and continuously updated view of core business processes for business planning and strategy.
- MES + Digital Twins: provides the real-time tracking of the transformation of raw materials to finished goods, control of multiple elements of the production processes, and support for the manufacturing decision makers.
- Digital Twins for processes, logistics and products: provides the digital twins of factory processes, including local processes, logistics, production line, production cell, warehouse, and workplace.
- Control-process communication + Production processes: gives an overview of machinery, sensors, actuators, and other devices on the shop floor

2.1.6. Architectures comparison at general level

Reference architectures have already made important contributions, mainly regarding the overall structure of Industry 4.0 systems. The complex and interconnected nature of Industry 4.0 systems can be observed trough this comparison (Figure 7), which sometimes transcend organisational and

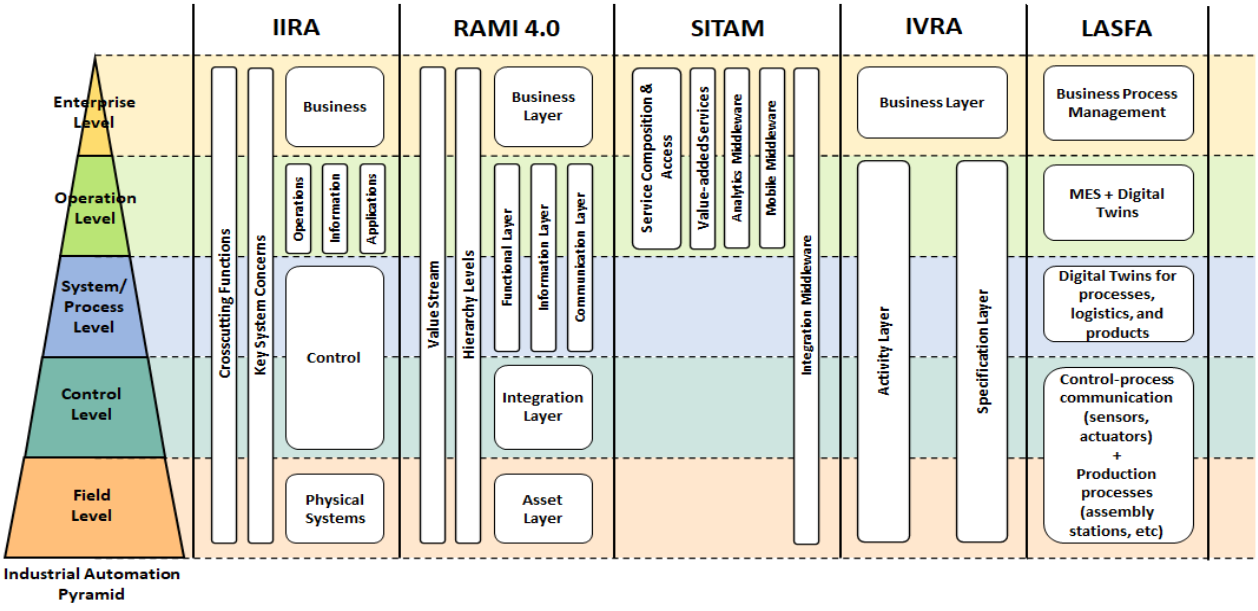


Figure 7: Comparison of reference architectures in industry 4.0

technical boundaries, thus leading to reference architectures that are broader, covering everything from the business level to technical issues.

Looking at the overall scenario, there is a large trend towards the reuse of successful IoT experiences, practices, and technologies, which has been explored as an efficient means to connect physical objects (e.g., machines and devices on shop floors) to the Internet, taking then advantages of its benefits and addressing its drawbacks such as the breaches of privacy in IoT.

All architectures cover the different levels of the pyramid from the business to the shop floor levels. Moreover, modules of the architectures sometimes cover more than one level of the pyramid, reinforcing the nature of Industry 4.0 systems in which there is no clear distinction among these levels. IIRA, RAMI 4.0, and SITAM present modules crosscutting and covering all levels, while other architectures have modules distributed in more than one.

While IIRA and RAMI 4.0 are among the most popular and frequently mentioned architectures, all analysed architectures share the issue of a clear and punctual standardisation method. Moreover, IIRA, SITAM, and LASFA also have the purpose of facilitation, which essentially means promoting the sharing and/or reuse of knowledge related to how to architect Industry 4.0 systems. Architectures involving consortia (IIRA, RAMI 4.0, IVRA) focus on multiple organisations, to be established and adopted; while academic architectures (SITAM and LASFA) have usually focused on a single or a couple of organisations.

It can be noticed that only IIRA, RAMI 4.0, and LASFA address digital twins as reference architectures.

RAMI 4.0 and IVRA present high-level views to communicate what smart factories should/could encompass. While RAMI 4.0 is complemented with a textual description of its view, IVRA presents other overall views detailing each layer, together with a textual description. These two architectures do not explicitly describe how the layers (and their individual elements) could be interconnected. Hence, both architectures require considerable decisions and refinements to be made when used in a given industrial project.

3. Tools and Practical applications survey

3.1. General approach for Industry 4.0 transition

P. K. Illa and N. Padhi identify and describe general key areas that should be encompassed in an Industry 4.0 architecture in their research [2]. The authors do not address any specific aforementioned architecture but try to give a general insight for the configuration of an Industry 4.0 architecture.

The identified areas are reflected in Figure 8. The main blocks are: Manufacturing Applications, Enterprise Applications, IoT Platform, Data Visualization and Control. For each of them, a detailed description of the building blocks is given. The description is reported in this deliverable, since it could give an important insight for the development of the IOP structure, independently from the adopted architecture.

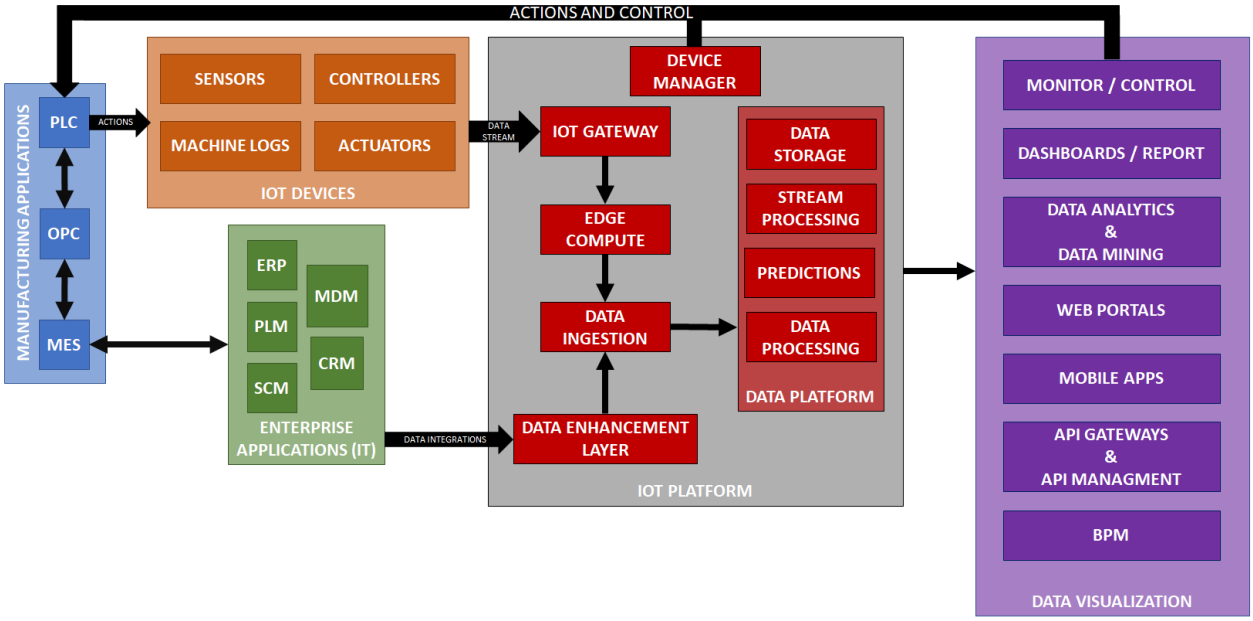


Figure 8: Industry 4.0 usual organisation architecture

- Manufacturing Applications
  - **Manufacturing Execution System (MES):** is an OLTP (Online Transaction Processing) system for the shop floor. It records all the transactions at the shop floor such as material movement, consumption, rework, scrap etc. It can be a packaged application or even a home-grown application. Packaged MES are off-the-shelf products available in the market and are then customised to meet the needs of a plant. Sometimes certain factories have very specific needs, so they develop their own MES internally. Either approach has its own advantages and disadvantages, and the factory needs to evaluate what is best for it.
  - **Programmed Logic Control (PLC):** controls the coordination among the equipment, process steps and operators to produce finished goods. Usually, more than one PLC is present in the plant, which is managed by a Master PLC.

- **Open Platform Communications (OPC):** is the Middle Layer which facilitates communication between MES and PLC. MES logs all the transactions performed on the shop floor and PLC guides equipment to perform process steps. MES must continuously communicate with the PLC to record the transactions in real-time. Middle layer such as OPC or SECS/GEM facilitate this communication.
- **Supervisory Control and Data Acquisition (SCADA):** is a supervisory system which controls equipment, processes and devices. It interacts closely with equipment, PLCs and MES to perform "supervisory" role on the shop floor.

- Enterprise Applications
  - **Enterprise Resource Planning (ERP):** is a Packaged OLTP (Online Transaction Processing) system. It provides a consolidated platform to operate multiple business processes such as Manufacturing, Source-to-Pay, Order-to-Cash, Planning, Accounting, Costing, Consolidation, Intercompany transfer etc.
  - **Product Lifecycle Management (PLM):** is a centralised repository for SKUSs (Stock Keeping Units).
  - **Customer Relationship Management (CRM):** facilitates customer ordering process. It provides a platform to manage quotes and sales orders.
  - **Master Data Management (MDM):** provides the up-to-date information of critical data such as customer, location, product etc. across various system in the organisation.
  - **Supply Chain Management (SCM):** normally is a packaged application with a focus on Supply Chain Planning, Forecast Management and Production Planning.

- Data Visualisation and Control
  - **Operational Dashboard:** offers real-time visibility of all the applicable sensors, devices, and machinery.
  - **Control & Governance:** monitors and controls the machines or sensors from Command center based on the real-time monitoring or anomaly detection.
  - **Data Analytics:** analyses collected data to identify patterns or trends. Predictive and preventive measures are applied through data analytics and machine learning algorithms.
  - **Portals & Mobile enablement:** provides the analytics and metrics as graphical user interface for consumers applications or mobile devices.
  - **API Gateway:** exposes the functionalities through APIs (Application Programming Interface) to enterprise applications for demand forecasting, inventory management, traceability and even to participate in BPM (Business Process Management) orchestrations.

- Since the IoT Platform is the core that transforms a factory to a smart factory, and it consists of several layers and components, the authors give a more detailed description of each component, graphically represented in Figure 9.

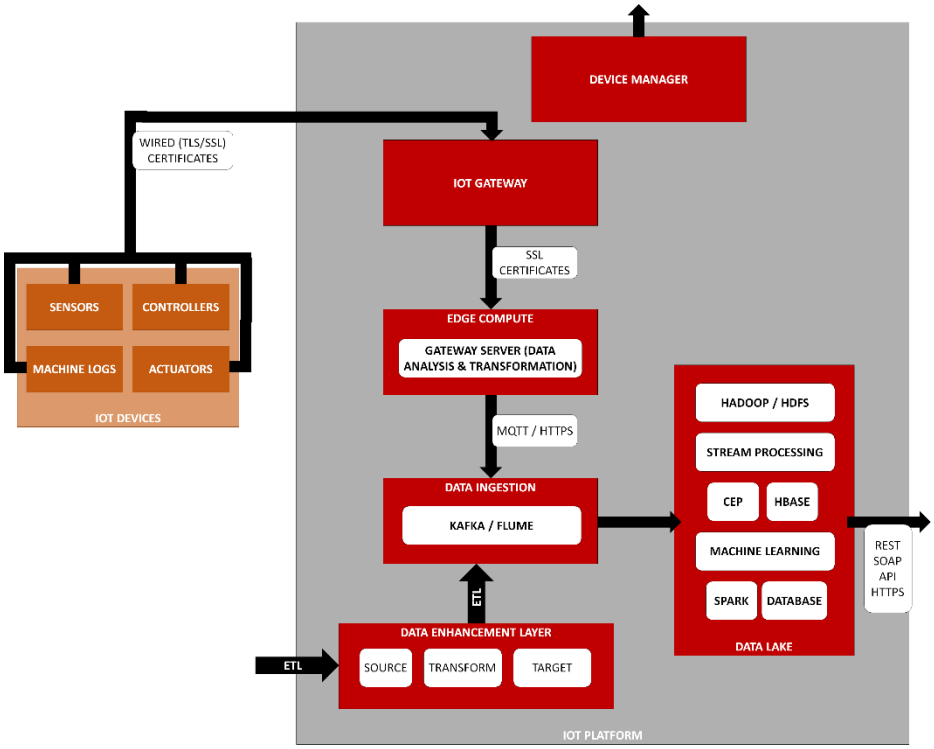


Figure 9: IOT platform architecture

- **IoT Gateway:** provides connectivity to sensors and equipment. It mainly deals with communications, protocols, security, infrastructure and network topology. The devices are IP-enabled and uniquely identified in the network. Sensors in the factory are generally wired using ethernet cables (TCP/IP) or connected through wireless (RFID/ZigBee/Bluetooth). As the cost of sensors and wireless protocols are rapidly decreasing, the sensors have become so economical and ubiquitous, they can even be scaled with wireless mesh networks.
- **Edge Compute:** consists of gateway servers or router services that performs real-time computing necessary to make quick local decisions on data streams for low-latency controls. The decision services are integrated with Device Manager to transmit control parameters to PLC or OPC for controlling and optimising system operations. Edge computing happens closer to the devices and decisions are made local to the equipment for low-latency operations, without waiting for decisions from the subsequent layers of Data Lake.
- **Data Ingestion:** data is streamed from multiple source applications into Data Ingestion Layer, where the data is processed and further transformed. The data from multiple sources with different formats (i.e., timeseries, event streams, log streams, structured, semi-structured, unstructured) are transformed into enterprise's canonical or standard formats. Data serialisation format (i.e., protobuf, avro, thrift) is selected depending on speed and consistency. Data quality and harmonisation should also be taken into consideration depending on how well the data is properly maintained in the source applications. Depending on the volume of data, reusable data pipelines can be established using Apache Kafka cluster or Flume to receive massive amounts of data.
- **Data Lake:** data is stored in the distributed HDFS cluster, RDBMS (i.e., Oracle, MySQL, MS SQL) and NoSQL (i.e., Cassandra, Mongo) database depending on the type of data and usage. Apache Spark is used for real-time analysis and it is several times faster than MapReduce. Data is stored as relational format using Spark SQL for data processing of structured data.

Semi-structured or Unstructured data is processed and analysed with Spark scripts that are developed using scala, python or java. Spark in turn offers out-of-box Machine Learning (ML) libraries to train and test the data sets, establish reusable pipelines and then apply prediction or clustering algorithms.

- **Data Integrations:** enterprise applications and Manufacturing applications are usually connected through Middleware and ETL tools. The data from these different enterprise systems are extracted through Middleware or ETL tool to the Data Lake or Data Ingestion layer. The data is usually processed through a series of stages known as Data Staging area, where the data is enhanced, transformed and enriched to a standardised and shareable form. In Data Staging area, the information is usually joined and combined with multiple IT applications to generate canonical models before feeding to Data Lake.

The authors also describe enabling technologies, together with the advantages and disadvantages of establishing an IoT platform using: OSS (open-source software), commercial distributors and using PAAS (Platform-as-a-service). The second and the third options are both based on outsourcing to software provider companies. Since the purpose of this deliverable is to explore open-source solutions, only the first option is reported below.

Open-source software: in this approach, OSS is used to establish all the necessary components in the organization's data center. OSS Apache Hadoop framework, along with its plethora of modules, is used to establish the IoT platform. Hadoop Distributed File System (HDFS), HBase and NoSQL databases (Cassandra or MongoDB) are the usual choice for data storage. For data computations, Apache Spark and MapReduce are used. Apache Flume and Sqoop are used for data movement and connectivity from different sources such as machine logs, enterprise databases and IT applications. Apache Kafka and Storm are used for real-time data streaming from sensors. Apache Mahout, MLLib and Spark ML are used for applying machine learning algorithms. In Table 1 are presented the advantages and disadvantages of this approach.

Table 1: OSS Advantages and Disadvantages

Advantages	Disadvantages
Solution specific to the organization's requirements can be achieved.	Requires strong expertise in wide variety of technologies and skills
Offers high level of flexibility in terms of achieving project goals, change and customisations.	Involves high level of complexity and requires huge effort as the solution is entirely home-grown.
Complete ownership of data, product stack and processes	Hindered by the challenges of being an early adopter and prepare to discover issues as the products mature.
Completely driven by organisation's vision, expertise and execution	
Benefit from early adoption of technology trends, raise as technology leaders and capture market opportunities before rest of the industry and its competitors catches-up	

3.2. RAMI 4.0 for robotic arm control

In recent studies, such as the one published by T. Lins et al. [3], there is presented a customisation of RAMI 4.0 for robotic arms applications. The authors focused on the adoption of various technologies in order upgrade industrial equipment (robotic arm) to Cyber-Physical Production Systems (CPPS). The issue of standardisation is addressed with the support of a custom platform that implements features to work independently of the type of equipment. According to the authors, the peculiarity of the RAMI 4.0 architecture for this specific application is its Administration Shell. The Administration Shell is the interface that connects the physical object to the industry 4.0 and is responsible for storing all the data and information about the asset. It also servs as a standardised interface for network communication that can integrate passive objects.

Each physical object has its Administration Shell, and the connection occurs through the communication offered by Industry 4.0, with the Administration Shell forming the digital part and the physical object forming the real part.

In order to integrate the CPPS in an Industry 4.0 environment the following functional components, divided in infrastructure, communication and application, are taken into account:

- Infrastructure
    - **Embedded Board Component (EBoard-C)**: must support various connections, buses and interfaces, such as General-Purpose Input/Output (GPIO), Pulse Width Modulation (PWM), serial ports and Universal Serial Bus (USB), as well as support for IoT devices. The EBoard-C also has the function of identifying the IoT devices and scanning the signals collected by the IoT devices.
    - **IoT Devices Component (IoTDev-C)**: works in conjunction with the EBoard-C component, where the IoT devices (such as cameras and other sensors) are installed, providing also the packages that must be activated to detect the IoT devices.
- Communication
    - **Network Component (NET-C)**: responsible for connecting industrial equipment to the Industry 4.0 network. The purpose of this component is to select the right interface and driver to establish the connection with the Industry 4.0 network.
    - **OPC Component (OPC-C)**: this component is divided into two sub components: the **Server-M** module and the **Equipment-M** module. The Server-M module is responsible for connecting the CPPS to OPC servers. The Equipment-M connects CPPS to the industrial equipment. During the routine, the Server-M module scans the network in search of the OPC-UA servers. After detecting the service, the Equipment-M module scans for existing industrial equipment. The servers and collected equipment are saved in a database so that the industry can use the information. In the Equipment-M module, besides to collect the information, it is possible to carry out operations of writing, changing equipment variables, such as enabling and disabling functionalities.
    - **SDN Component (SDN-C):** the SDN-C component enables two modules in the CPPS. The first module is the **Controller-M**, which allows CPPS to manage the network resources, as well as to mount the data plan and send to the commuters. The second module, the **Commuter-M** allows the CPPS to receive the data from an SDN controller to decide to route CPPS packages.

- Application
    - **Database Component (DB-C)**: is the component responsible for connecting CPPS to a database, supporting several databases. This component has two modules: **ServerDB-M**, responsible for making a database server available, to storing information for quick and short-term responses, while the module **ConnectDB-M** is responsible for the connection with external databases. Usually present in a cloud, the module performs a network scan and connects to the database to store information that requires long-term processing or storage.
    - **Remote Access Component (RA-C)**: responsible for releasing remote access to the CPPS, making it possible for the user to access the CPPS externally.
    - **Web Service Component (WEB-C)**: responsible for making a Web server available.
    - **Monitoring Component (MON-C):** responsible for collecting CPPS information, and with the help of the WEB-C and DB-C components, make available some information CPPS to users, such as status, connectivity, components used, etc.
    - **Cloud Component (CLOUD-C)**: responsible for making support packages available to cloud computing. This component has 2 modules: the **Node-M** module, which is responsible for providing the necessary packages for CPPS to become a cloud node, and the **Migration-M** module responsible for assisting and managing migrations with legacy software.

These functional components are integrated into the seven layers of the RAMI 4.0 architecture as follows and represented in Figure 10.

- **Asset Layer:** contains the components of the infrastructure with the EBoard-C with functions of the board and the IoTDev-C activating the IoT devices.
- **Integration Layer:** contains the component OPC-C, responsible for the digitisation of the information of industrial equipment, as well as EBoard-C, with the function that digitises the information IoT devices.
- **Communication Layer:** contains the NET-C component, which uses an interface installed in the board to connect to the communication network of the Industry 4.0, and the SDN-C component to actively connect the CPPS on an SDN network.

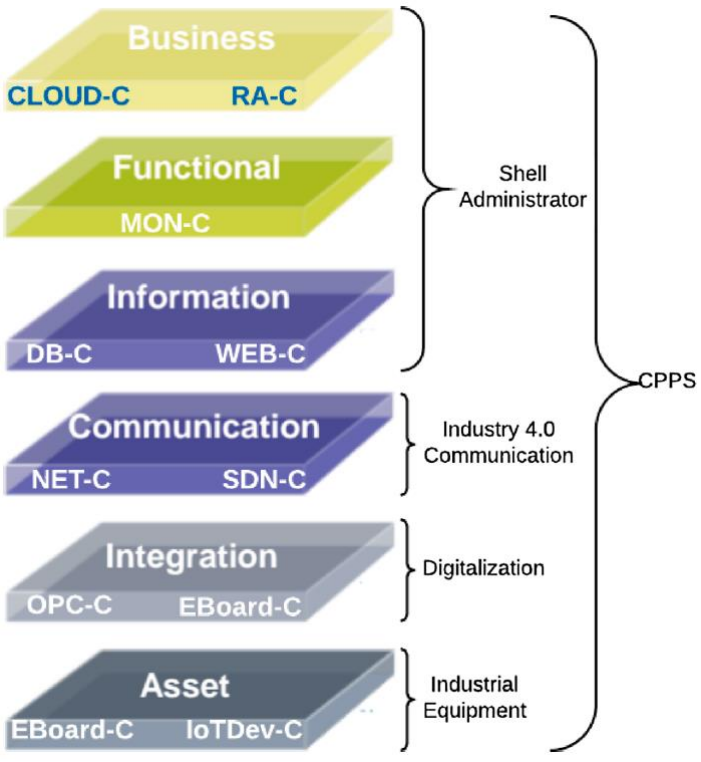


Figure 10: Functional components integration within RAMI 4.0 Architecure layers

- **Information Layer:** contains the DB-C and WEB-C components, with the storage, search and availability of the CPPS information.
- **Functional Layer:** contains the MON-C components, which monitor the operation of the CPPS, as well as the resources and the communication.
- **Business Layer:** contains the Cloud-C component that through cloud computing provides means for composing, organising and managing services of Industry 4.0, while the RAC component offers remote access for users to manage CPPS resources.

With the presented integration strategy, the authors have developed a custom platform for the integration of robotic arms in an Industry 4.0 scenario (Figure 11).

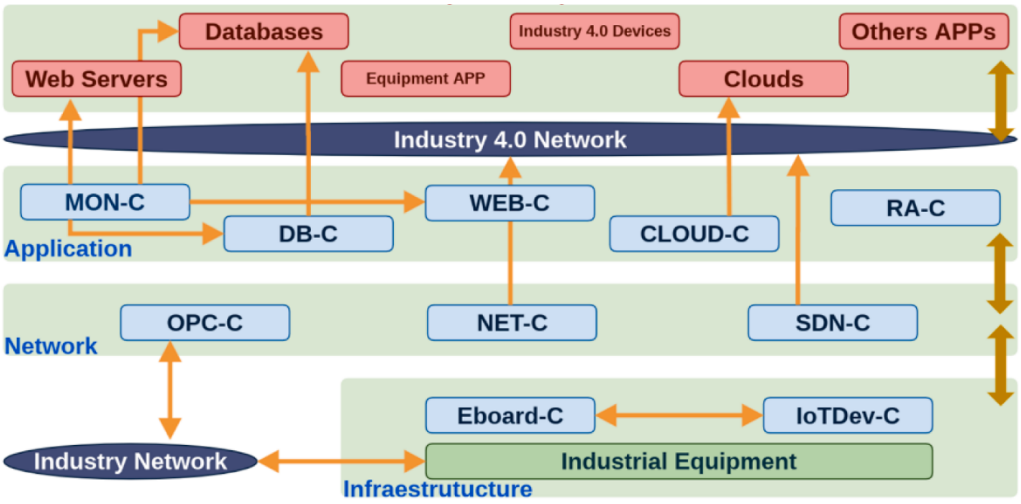


Figure 11: Practical application of RAMI 4.0 in a platform

The platform consists of three interconnected structural layers: Infrastructure, Network/Communication and Application.

- The **Infrastructure Layer** provides all support and interconnection of the components with the industrial equipment.
- The **Communication Layer** makes access to the industrial networks freeing the access of these networks to the other functional components.
- The **Application Layer** is responsible for the interaction of the CPPS functional components with the components present in the Industry 4.0.

Each functional component has direct access to the information, and functionalities of the other components.

In the **Infrastructure Layer**, through the EBoard-C component, the platform has support for several embedded boards, as well as packages to support the IoT devices. In order to achieve interoperability of the platform with the various embedded boards in the market, the implementation of the CPPS Retrofitting components used primarily the Python language. In order to attend to the IoT devices on the platform, the IoTDev-C component supports various packages and drivers, such as the GPIO and PWM packages.

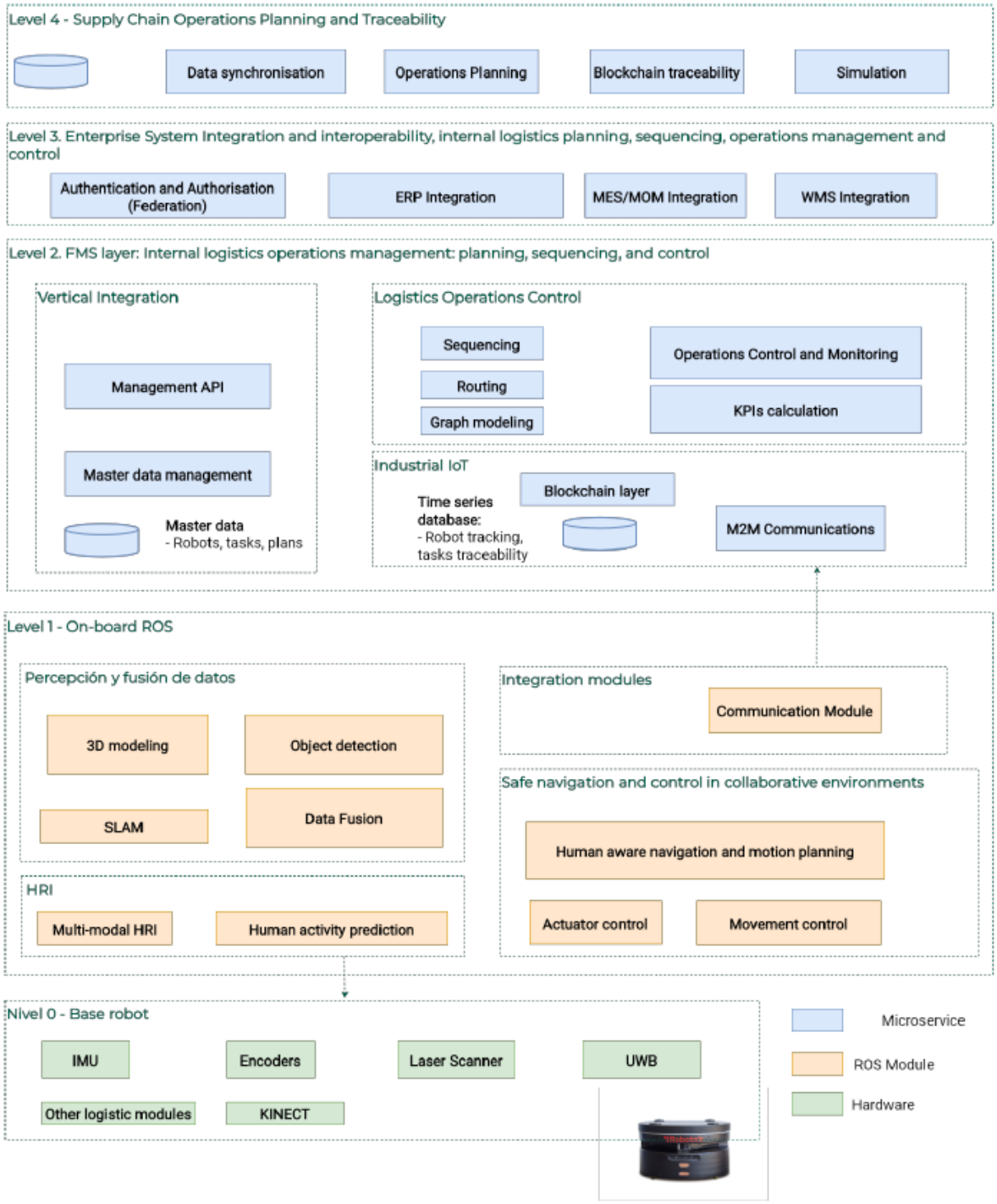
In the **Communication Layer** the NET-C component of interfaces supports for connection with the industry 4.0. The interface may be native to the board embedded or used through a dongle. According to the network used in the industry, the configuration is made. Therefore, the platform detects the communication of the Industry 4.0 following the configuration made and then, connects CPPS in

Industry 4.0. The OPC-C component identifies industrial equipment via the OPCUA protocol, which can be done automatically or manually. First, the OPC servers are detected. Then, for each OPC server the variables of each equipment of the industry are collected. With the detection of IoT devices and industrial equipment, the platform integrates the communication, being able to manage all the devices and make decisions based on the information collected by the platform. The SDN resource provided by the SDN-C component adds CPPS to the commuter and network controller functionality, making CPPS able to make decisions regarding packet forwarding and network resource management. On the platform OpenVswitch is available, which is an open code implementation of an SDN commuter and the SDN controllers. On the platform, it is possible to enable one or both features at the same time.

In the **Application Layer**, the platform provides access to several databases through the DB-C component, which contains some client software, among which we can mention MySQL, PostgreSQL, SQLite, and Open database connectivity (ODBC), MongoDB, NoSQL MySQL. In order to guarantee the CPPS to have an independent Web service, the platform offers the option of enabling an Apache server to manage the information generated by the CPPS itself through the WEB-C component. Monitoring of CPPS and CPPS-connected devices is performed by the MON-C component, which collects the information, saves it to the database and makes it available through the web service. The CLOUD-C component allows integration with Ubuntu Cloud, Microsoft Azure, Opennebula and Eucalyptus. In addition to typical applications in Industry 4.0, the cloud can receive native applications of the equipment through migration. The RA-C component provides remote access with the Virtual Network Computing (VNC), Secure Shell (SSH) and Remote Desktop Protocol (RDP), in which all available protocols can be enabled or only one of the user's preference.

3.3. IIRA for Autonomous Mobile Robots

In [4] the authors propose an architecture, based on the IIRA architecture, for the implementation of robotic platforms with the purpose of internal logistics. The proposed architecture is specifically designed for the management of Autonomous Mobile Robots (AMR), which represent a collaborative approach to internal logistics automation. The reference architecture is based on Robotic Operating System (ROS), and aims to meet the requirements of small and medium-sized companies.



The definition of the reference architecture is based on industry standards and best practices for open robotics, and it follows the recommendations of the ISA/IEC 62443 standard on security in industrial control systems. Specifically, the different levels are described as follows:

- **Level 0 - Industrial Level:** this level groups together the physical components of industrial systems, mainly actuators and sensors (cameras, laser scanners, encoders to detect products). The selected base hardware to implement the reference architecture comprises ROS compatible logistic robots.
- **Level 1 - Control Level:** this level contains the system-level control elements of the components at Industrial Level. In a robotic platform, this level groups the Human-Robot-Interaction modules, the modules to model the environment, including humans, and the modules to control the robot:
    - human navigation and motion planning
    - actuator control
    - movement control

Modules for data fusion to enable the integration of different indoor positioning systems are also located at this level. From a technical point of view, the modules at this level are implemented as ROS modules, interconnected in a level-1 ROS network, deployed in an edge platform or on-board computer with GPU acceleration to achieve the required performance. This is particularly important for modules that rely on neural networks, like Yolov4 object detection and human activity prediction. The performance of these modules has been successfully tested using a NVIDIA Jetson module. Communication with higher layers is implemented through ROS bridge modules (communication modules) that act as secure conduits to exchange information with components of the Operation Level, using:

    - Message Queue Telemetry Transport Protocol (MQTT)
    - OPC Unified Architecture (OPC UA) protocol

- **Level 2 - Operation Level:** this level groups together the operation and supervision systems, such as operator terminals/consoles and monitoring applications. In a robotic platform, this level groups the main functional blocks of the fleet management system, including functions to control and orchestrate the robotic fleet, calculate Key Performance Indicators (KPIs) for logistic operations monitoring, as well as functions to enable communication between Control Level and higher levels according to security specifications for Industrial Control Systems. From a technical point of view, these modules are microservices deployed in a microservice orchestration platform like Kubernetes. The machine-to-machine communication module provides an endpoint (e.g. MQTT broker) used to send control commands to and receive status feedback from the robotic fleet. The information is stored in a time series database microservice to enable robot tracking and tasks traceability. To facilitate vertical integration, this level implements microservices to manage master data and expose management functions to the Enterprise Level services or external systems through a management Application Programming Interface (API).
- **Level 3 - Enterprise Level:** this level contains the equipment and systems to provide support to the company's business processes, such as ERP. This level of the reference architecture groups functional blocks to implement Role Based Access Control to the FMS (Flexible Manufacturing System) functions, including federation with external authentication and authorisation services, as well as functional blocks to integrate with other enterprise systems,

like the ERP, MES/MOM, or WMS integration. From a technical point of view, these functions are mapped to microservices. Communication with Operation Level functional blocks is achieved through the management API, so that Operation Level and Enterprise Level services are decoupled to achieve inter-level isolation.

- **Level 4 - Supply Chain Level:** this level is introduced to extend the solution to the supply chain level, enabling collaboration among supply chain collaborators. This level groups advanced services to enable trustable supply chain traceability using blockchain technology, data services to enable the integration and synchronisation of data distributed across different platforms, supply chain operations planning services, and simulation services to simulate internal logistic processes.

From a security perspective, to adequately protect the components at the Industrial Level, the authors suggest the adoption of a "Defense In Depth" strategy. This defense strategy is based on the establishment of different security controls to protect critical systems at lower levels. Communications must always be made from the lower levels to the upper levels (communications are not allowed to be initiated in the opposite direction), and all communications between levels must use secure, properly protected conduits (security paths between two levels).

3.4. Specific Data Architectures and practical applications

3.4.1. Kappa Architecture for the applications of online ML on data streams

The authors of [5] proposed an architectural concept for the applications of online Machine Learning on data streams based on the Kappa Architecture in conjunction with the use of container-based microservices. The overall structure of the architecture and all the modules are presented through a diagram block in Figure 12.

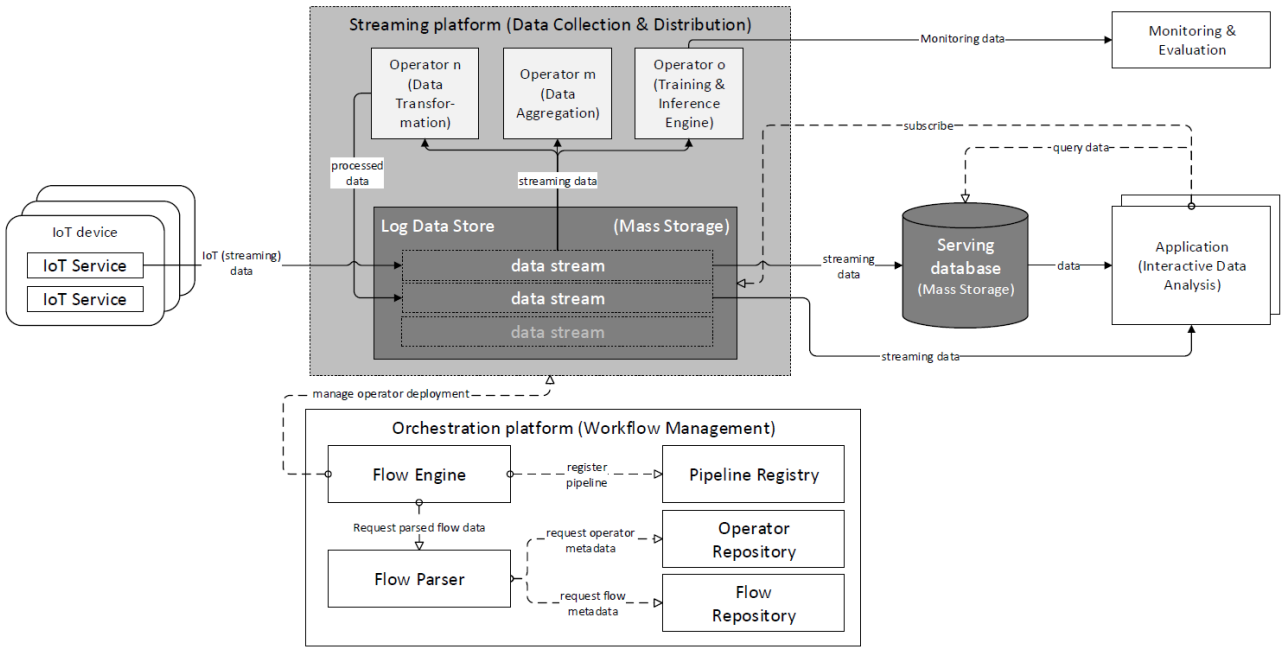


Figure 12: Kappa Architecture in practice

The suggested architecture specifically addresses the management of a high number of data streams in an IoT environment, and comprises following components:

- **Streaming Platform:** is the central component of the architecture, which handles data collection as well as distribution and is composed of different sub components.
    - **Log Data Store:** ingests the data streams using IoT middleware solutions and saves them. The messages in these data streams as well as the ones, which have already been processed in the streaming platform, are saved in sequential order using topics and partitions to organise the data and to enable parallel access.
    - **Serving Database:** if needed, data may be pushed to this component, thus allowing ad-hoc queries on the data. Together, the Log Data Store and the Serving database are used for mass storage of data.
    - **Analytics Operators:** are small, single-purpose microservices, encapsulated using container technology (Docker4) that perform data processing in the streaming platform. Analytics operators may be composed into analytics pipelines, linking their inputs and outputs, allowing for complex data transformation as well as aggregation and application of advanced statistical and ML-based methods.

- **Orchestration Platform:** acts as a workflow management. It implements the capabilities to develop, manage and deploy analytics pipelines based on predefined analytics operators. This component it's also composed of different sub components:
    - **Flow Engine:** is the main subcomponent of the orchestration platform. It offers different adapters to interface with container-orchestration and management systems, such as Kubernetes or Rancher, and deploys analytics operators based on analytics flows. These are flow-based graphs, which describe the data flow between the inputs and outputs of analytics operators.
    - **Operator Repository and Flow Repository:** all analytics flows are saved in the flow repository, whereas analytics operators metadata are saved in the operator repository.
    - **Flow Parser:** is called from the Flow Engine after a user requests the instantiation of an analytics flow. It delivers the the corresponding deployment data, including analytics operators to be deployed as well as their configuration in terms of input data mapping. The deployment data is composed by the Flow Parser using analytics flow data from the Flow Repository and analytics operator metadata from the Operator Repository.
    - **Pipeline Registry:** in this component, new analytics pipelines (composed by Analytics Operator containers and registers) are started by the Flow Engine. A single Analytics Operator always subscribes to at least one topic with IoT data, which is saved in the Log Data Store of the Streaming Platform and merges data streams if needed. After it has processed a message of a data stream, an Analytics Operator writes the resulting message back to the Log Data Store into a separate topic. Consequently, it is possible for external applications to subscribe to these topics and receive streams of processed data at all stages of an analytics pipeline.

The entire architecture is prototyped entirely relying on open-source software, for instance:

- The components of the Streaming Platform are built using Apache Kafka and its software ecosystem.
- Analytics Operators are written in Java. To ensure their integration with the orchestration components, a Java library based on Kafka Streams was developed.
- The Serving Database was built using InfluxDB.
- The components of the Orchestration Platform are developed using Go and Python and expose their functionalities via REST-based CRUD endpoints.
- The metadata is saved on document-oriented database systems, e.g. MongoDB.
- The fronted application was built using Angular. Relying on flow-based notation, this graphical tool can be used to design analytics flows and wire analytics operators without the need to write programming code.
- Since the implementation of ML algorithms in a large number of analytics pipelines poses a problem in terms of the availability of processing resources, online ML algorithms where chosen. Their main advantage is, that the processing time does not increase with growing data sets. Combining this type of algorithm with the kappa architecture approach allows for highly scalable data processing while still keeping its flexibility in terms of data re-processing and changing analytics pipelines at consumer level.
- In order to use online ML, a new Analytics Operator, which provides online ML capabilities, was implemented. This Analytics Operator is a combination of the Training and Inference Engine. Additionally, the current model state is saved in the Analytics Operator, negating the need for an external model storage. Monitoring & Evaluation of the utilised ML algorithm is achieved by writing the corresponding data into the processed messages. Therefore, no external services are needed.

3.4.2. Lambda Architecture for real-time visualisation of sensor data in Smart Manufacturing

In [6] the authors propose an approach, based on the Lambda Architecture, for real-time visualisation of sensor data in smart manufacturing. In Figure 13 the Lambda Architecture for this practical example is presented graphically through diagram blocks.

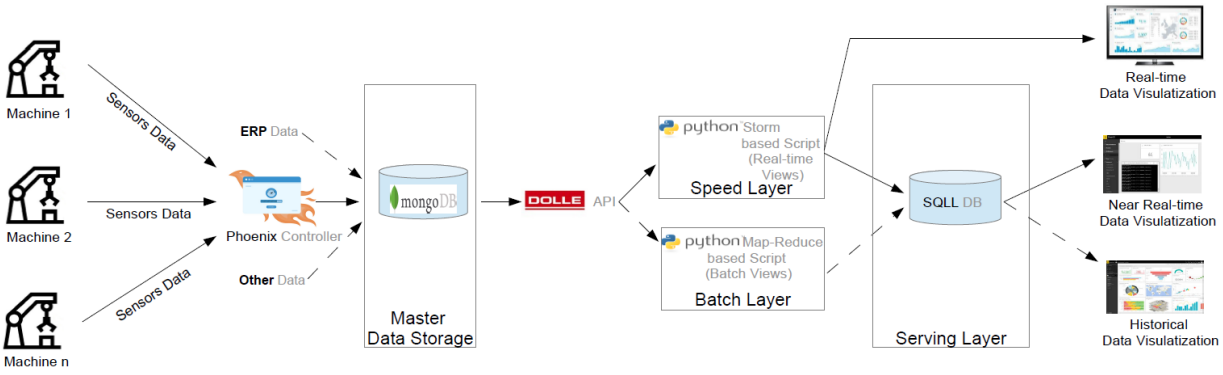


Figure 13: Lambda architecture in practice - real-time visualisation of sensors

The proposed lambda architecture consists of following main modules:

- **Sensor network:** consists of sensor nodes deployed throughout the machines to measure the status of the machines such as, machine on/off, pace in, pace out and fault/error.
- **ERP data integration:** contains details about products, job executions and work calendar.
- **Integration of other external data:** may include indoor temperature, humidity, etc.
- **Master data storage layer for persistence:** The data transmitted by the sensors is streamed into the master data storage (MongoDB) by **Phoenix controller**. The Phoenix controller captures data (ranging from 1 to 10 times per second) from the sensor network and submits the data to MongoDB. The controller submits the data of a sensor in MongoDB, only if there is a change in the state of the sensor. If there is no change in the state of the sensor the value will not be stored. Further, external data such as ERP data, is also submitted to the master data storage. The sensor data (JSON format) contains three attributes. The first three rows in the sensor data snapshot read as follows:
    - Timestamp: represents the date and time of the sensor data acquisition.
    - Value: corresponds to the state of the binary sensor (either TRUE or FALSE)
    - Port: indicates the sensor number.

Similarly, the ERP data snapshot reads as follows:

- Planned start and stop: represent the planned job execution date and time.
- Actual start and stop: denote the actual job execution data and time.
- Item-id: identifies the product that is being produced.
- Machine-id: represents the machine
- Name: represents the type of product being produced.

A custom-built web application programming interface (Dolle API) allows for access to the sensor and external data from the master data storage by means of an HTTP GET request to provide the data to the speed layer for processing.

- **Speed layer:** runs on the Apache Storm platform and provides real-time views.
- **Batch layer:** runs on the Hadoop MapReduce platform and delivers precomputed batch results.
- **Serving layer:** consists of a SQL Server and stores the output from the speed and batch layers.

The real-time views are directly streamed to the dashboard for real-time visualisation using Dash (Python framework for building web applications). The real-time views are also delivered to the serving layer with some latency for near real-time visualisation.

Le logic and steps behind the production monitoring are the following:

1. Read the sensor data from the master data storage
2. Feed the data to the Speed Layer
3. Once in the Speed Layer, following operations are performed on the data:
    - Filtering according to the Values (TRUE or FALSE)

- Aggregation of the port values in key-value pairs
- Display on the dash board the aggregated values
- Storage of the aggregated values in the Serving Layer (SQL Database) for near real-time visualisation
- Reset the key-pairs to zero

The implemented architecture adopts also Machine Learning models to analyse data and find production anomalies in real-time. To do so, the incoming real-time sensor data is compared to standard patterns and an alert is raised when some abnormal event is predicted to happen. The proposed algorithm uses the Batch Layer to train the model to detect abnormal patterns. The MapReduce job runs at predefined intervals and updates the coefficients of the pattern detection models in the Serving Layer. The speed layer reads the incoming streaming data from the MongoDB and detects abnormal patterns by applying a pattern detection algorithm, which dynamically uses the latest calculated model coefficients obtained from the Serving Layer. The pattern detection algorithm choose by the authors is an Auto-Regressive (AR) model with varying coefficients.

4. Intelligent Orchestration Platform

The Intelligent Orchestration Platform (IOP) aligns with the principles and concepts of the Industrial Internet Reference Architecture (IIRA) and RAMI 4.0. It serves as a foundational infrastructure layer for integrating and managing diverse technologies and systems in industrial environments. The IIRA emphasizes interoperability, security, and connectivity, which the IOP supports by facilitating the integration of different technologies and ensuring efficient data exchange and collaboration.

Similarly, RAMI 4.0 focuses on digitizing and networking industrial processes, and the IOP leverages its concepts to create a scalable and flexible infrastructure. By incorporating Cyber-Physical Systems (CPS), IoT devices, and other components, the IOP optimizes the management and control of technologies within industrial environments. This allows for the seamless integration of diverse systems and enhances overall efficiency.

The IOP's data and ML architecture follow the principles of the IIRA and RAMI 4.0. The data architecture enables efficient collection, storage, and processing of industrial data, ensuring interoperability and easy integration from various sources. The ML architecture within the IOP adopts a flexible approach, accommodating both the Kappa and Lambda architectures. This allows for real-time data processing, comprehensive insights, and supports decision-making processes in industrial systems.

4.1. IOP Data & ML architecture

The IOP utilises a data and machine learning architecture to process and analyse data, enabling intelligent decision-making. Integrating data ingestion, storage, processing, and machine learning capabilities empowers organisations to leverage their data effectively for improved performance and decision-making.

In the constantly evolving world of big data processing, two prevalent paradigms have emerged: the Kappa and Lambda architectures. Both architectures are designed to process large volumes of data and provide actionable insights in real-time or near-real-time, although they approach this goal in distinct ways.

4.1.1. Kappa vs Lambda

Kappa Architecture is a software architecture used for processing streaming data. The main premise behind the Kappa Architecture is that it can perform real-time and batch processing, especially for analytics, with a single technology stack. It is based on a streaming architecture in which an incoming data series is first stored in a messaging engine like Apache Kafka. From there, a stream processing engine will read the data, transform it into an analysable format, and then store it in an analytics database for end users to query.

The Kappa Architecture supports (near) real-time analytics when the data is read and transformed immediately after it is inserted into the messaging engine. Enabling recent data quickly available for end-user queries. It also supports historical analytics by reading the stored streaming data from the messaging engine later in a batch manner to create additional analysable outputs for more types of analysis.

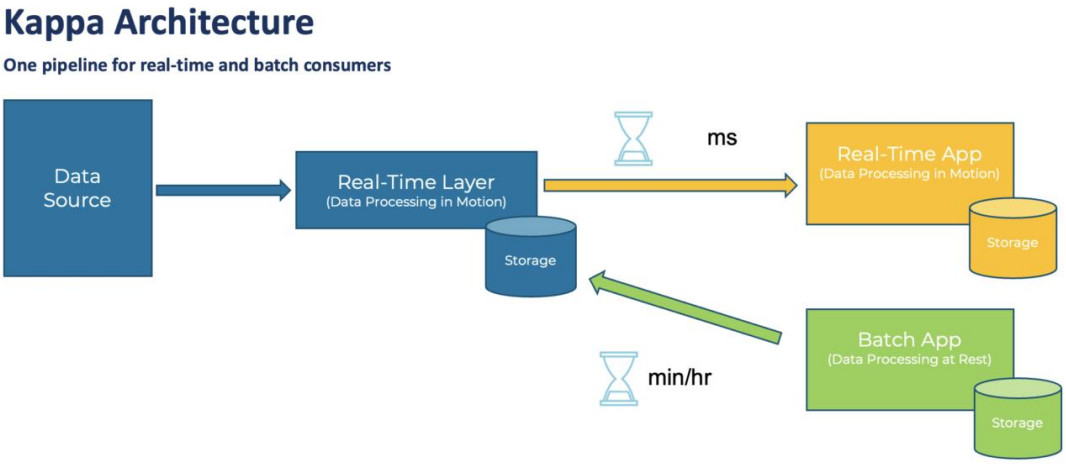


Figure 14: Kappa architecture simplified overview

Lambda Architecture. A data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. Lambda architecture includes batch, speed, and serving layers. This approach enables processing data in real-time but also easy re-processing of batched static datasets.

This approach attempts to balance latency, throughput, and fault tolerance by using batch processing to provide comprehensive and accurate views of batch data while simultaneously using real-time stream processing to provide views of online data.

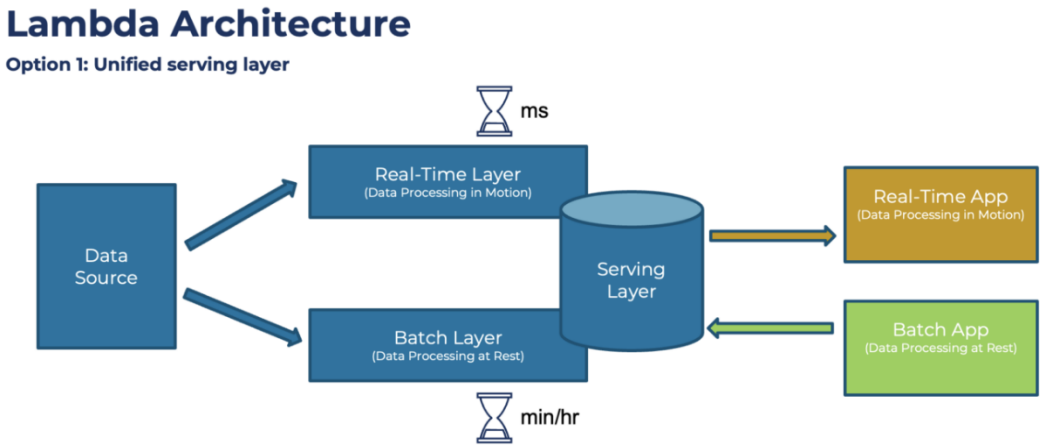


Figure 15: Lambda architecture simplified overview

In conclusion, both Lambda and Kappa architectures are viable for managing vast amounts of data in real-time. However, the specific needs and priorities ultimately determine the choice between them. The Lambda architecture's strength lies in its fault-tolerance and scalability, making it ideal for situations demanding rigorous data handling and historical data analysis. Still, it brings a level of complexity in setup and configuration that might not always be suitable.

On the contrary, with its simplicity and ease of use, the Kappa architecture provides an appealing option for scenarios where real-time processing is a key priority and data loss is not a major concern. Kappa also has the advantage of potentially delivering superior performance, even though it might not offer the same level of fault-tolerance as Lambda.

Given these considerations, it has been decided to initially adopt the Kappa architecture for its simplicity, given our current requirements. Furthermore, while it might not encompass the full range of capabilities the Lambda architecture provides, it still utilises the same core technologies, thus

making future transitions less challenging should we ever need to introduce a Batch layer for more rigorous data handling.

The strengths of both these architectures lie in their shared technology base, meaning that if our needs evolve, it is possible to switch from Kappa to Lambda without substantial reinvestment in new technologies. This approach allows us to start simple with Kappa and remain flexible for future adaptations, assuring that we are well-prepared to handle any shifts in our data processing needs.

4.1.2. IOP backend architecture

The backend architecture of the IOP following a Kappa or Lambda architecture encompasses several key concepts to enable efficient data processing and management:

1. Data Ingestion: The architecture includes a data ingestion layer responsible for collecting and processing data from various sources. The primary goal is to reliably and efficiently collect data and make it available for processing in real-time and batch processing layers.
2. Real-Time Processing: The real-time layer handles the processing and analysis of data as it arrives, typically in a streaming fashion. This layer focuses on processing data in real-time to extract valuable insights and make timely decisions.
3. Batch Processing: Alongside real-time processing, the architecture supports batch processing. This involves processing large volumes of data in scheduled or periodic batches to generate aggregated reports, perform complex computations, or perform offline analytics.
4. Storage: Data storage is a critical component of the architecture. It involves storing both raw and processed data, ensuring it is accessible for future analysis, reporting, and retrieval. The architecture may employ distributed file systems or databases capable of handling large-scale data storage.
5. Serving Layer: The serving layer is responsible for providing data to end-users or other systems in an easily consumable format. It focuses on delivering insights, reports, visualisations, or other relevant information derived from the processed data.
6. Containerisation and Orchestration: Containerisation enables packaging applications and their dependencies into lightweight, portable containers. Orchestration tools provide automated management, scaling, and deployment of containerised applications, simplifying the administration and operation of the system.
7. Continuous Integration and Continuous Deployment (CI/CD): The architecture supports CI/CD practices, enabling automated processes for building, testing, and deploying software applications. It streamlines the development workflow, ensuring efficient collaboration and quick deployment of updates or new features.

The development of the IOP will leverage a comprehensive set of advanced technologies and tools. These include data ingestion frameworks, real-time processing frameworks, storage solutions, containerisation and orchestration platforms, and continuous integration and deployment tools. These technologies and tools will enable the efficient development and operation of the IOP, ensuring scalability, reliability, and streamlined workflows.

Data Ingestion Layer

The data ingestion layer plays a crucial role in both Kappa and Lambda architectures, as it is responsible for collecting and processing incoming data from various sources. The primary goal of this layer is to reliably and efficiently collect data from different sources and make it available for processing in the real-time and batch-processing layers.

- Kafka is often used as the primary data ingestion system. Apache Kafka is a distributed streaming platform that is designed to handle high volumes of data in real-time. It is open-

source and can be used for building real-time data pipelines and streaming applications that are reliable, scalable, and fault-tolerant. Kafka is built on top of a publish-subscribe model and uses a distributed architecture to provide high throughput, low latency, and high availability.

Real-Time Layer

The real-time layer of a Lambda or Kappa architecture is responsible for processing and analysing real-time data as it arrives, typically in a streaming fashion. Here are some example tools commonly used for the real-time layer of these architectures:

- Apache Flink is an open-source stream processing framework that provides powerful APIs and features for building real-time data processing applications. Flink provides advanced windowing and state management capabilities and can be easily integrated with other systems like Kafka and Hadoop.
- Apache Storm is a distributed real-time stream processing system widely used for processing high-volume, high-velocity data streams. Storm provides a simple programming model and flexible scalability options and can be integrated with various data sources, such as Kafka and HDFS.
- Apache Spark Streaming is a real-time processing framework part of the larger Apache Spark ecosystem. Spark Streaming provides a high-level programming model that allows developers to process data streams using Spark's powerful batch processing engine, enabling real-time data processing at scale.

Storage

In a Kappa or Lambda architecture, data must be stored for batch and real-time processing. Here are some examples of storage tools that can be used in these architectures:

- Apache Hadoop is an open-source framework for storing and processing large-scale data. Hadoop provides a distributed file system (HDFS) for storing data, as well as a batch processing system (MapReduce) for analysing data. Hadoop can be easily integrated with other tools, such as Apache Kafka and Apache Spark.
- Apache Druid is a high-performance, distributed OLAP (Online analytical processing) database that is designed for real-time analytics. Druid provides fast query response times and can handle both streaming and batch data, making it ideal for real-time analysis in a Lambda architecture.

Serving Layer

The serving layer of a Kappa or Lambda architecture is responsible for serving data to end-users or other systems in a format that is easy to consume. Here are some examples of tools that can be used in the serving layer of a Kappa or Lambda architecture:

- Apache Superset is an open-source, enterprise-ready business intelligence (BI) web application that is designed to explore and visualise data in real-time. Superset provides a user-friendly interface for creating interactive dashboards and visualisations, and it supports a wide range of data sources and visualisation types.
- Trino is a powerful distributed SQL query engine that allows users to query and analyse large amounts of data stored in various data sources, making it an ideal solution for organisations that need to process and analyse massive datasets. Trino's distributed architecture allows it to process queries in parallel across multiple nodes, and it supports a wide range of data formats and connectors, including Hadoop, Cassandra, and MySQL, among others.

Containerisation and Orchestration

Containerisation and orchestration are critical components of modern software architecture, and Docker and Kubernetes are two popular tools for packaging and deploying software applications consistently, efficiently, and scalable.

- Docker is a containerisation platform that allows developers to package their software applications in containers. Containers are lightweight, standalone executable packages that include all the necessary dependencies and libraries to run an application. Docker provides a consistent and reliable environment for running applications, regardless of the underlying infrastructure.
- Kubernetes is an open-source container orchestration platform that allows developers to automate containerised applications' deployment, scaling, and management. Kubernetes provides a highly resilient and fault-tolerant infrastructure for running applications at scale. In addition, it can automatically manage containerised workloads across multiple nodes, making managing and scaling applications easier.

Continuous Integration and Continuous Deployment (CI/CD)

CI/CD is a set of best practices and tools that enable developers to automate the process of building, testing, and deploying software applications. With CI/CD, developers can continuously integrate code changes into a shared repository, automatically test and validate the code, and automatically deploy the changes to production.

- GitHub Actions is a powerful and flexible tool for automating the software development process, available on the code repository platform GitHub. With GitHub Actions, developers can easily build, test, and deploy their software applications directly from their GitHub repository.

4.2. IOP User interfaces

4.2.1. Overall structure and work plan

Frontend refers to the part of a software system that users directly interact with. It includes modules/interfaces that present information and provide functionalities to users. These interfaces are designed based on the connection with the system's services, as defined in an ontology (M18 in D3.3). The frontend aims to comprehensively gather and present information from multiple modules/interfaces, integrating the Decision Support System (DSS) and the digital maturity and sustainability assessment module.

The development of frontend involves two main steps. Firstly, the design of the modules/interfaces and the identification of the information presented and functionalities of each interface are performed. This step includes defining the connection between the interfaces and the modules/services of the system using the provided ontology. Secondly, the actual development of the modules/interfaces and their online connectivity takes place. This development follows the guidelines and protocols defined in T3.1 and T3.2, ensuring proper implementation and integration. The participation of INO and AUTO is required to align with physical components from WP4 and modules from WP1.

During the deployment phase (M30 to M36) with the collaboration of MOL and ORI, the frontend will be developed further. The tasks involve deploying the Interactive Online Platform (IOP) in the facilities of MOL and ORI. The IOP is expected to provide a comprehensive presentation of information gathered by the interfaces, integrating the DSS and the digital maturity and sustainability assessment module. The frontend will be constantly maintained based on the results of the demonstration activities (WP5) to ensure its functionality and performance.

4.2.2. Technologies for development and deployment

Open-source front-end development and design tools have become increasingly popular due to their flexibility and accessibility. Nowadays, there is a need for graphical user interfaces for a website or an application to be user-friendly in order to impart knowledge from the server side effectively. As a result, front-end frameworks serve as the foundation for the advancement of front-end deployment. These tools offer a wide range of features that help developers create user-friendly interfaces and deploy them remotely to end users. Some commonly used open-source tools for front-end development and design are:

- **React:** is a front-end JavaScript library for developing user interfaces and related components. It employs the MVC architecture with a different table of presentation and data availability. One of its distinguishing features is the Virtual DOM (Document Object Model) and how it handles document access and manipulation. The DOM interacts well with HTML and XML documents, causing them to behave similarly to a tree structure, and each HTML element functions as an object. To create the components, React uses the JSX coding style, with a mix of HTML quotes and tag syntax. It breaks down larger components into smaller ones that can be managed separately and individually. Its core feature is component reusability, which facilitates collaboration and reuse in other parts of the application. One aspect that must be considered is that due to its multiple and constant updates, it is hard to create proper documentation, which affects the learning curve for beginners.

- **Angular:** is a comprehensive JavaScript framework for building complex web applications. It provides a complete set of tools for building UI components, managing data, and handling user interactions. Angular also offers support for server-side rendering, making it suitable for large-scale applications. It is a Typescript-based development platform developed by Google. Angular is a component-based framework for developing a set of tools for developers to use to create, build, test, and modify code and a collection of well-integrated libraries. It also allows scaling single-page applications to enterprise-level applications based on specific needs. It can benefit from a large learning and support community and decreases the amount of code since most important features (such as two-way data binding) are provided by default. Despite the large community, the initial learning curve can be steep due to its complexity and great variability of use.

- **Vue.js:** is a progressive JavaScript framework for building user interfaces. It focuses on simplicity and ease of use, allowing developers to create interactive UIs with a reduced amount of effort. Vue.js also offers built-in support for animations, routing, and state management. Since small in size, it is simple to install and download while it offers simplified binding of existing applications and provides extensive documentation. It also assists developers in understanding peer front-end frameworks such as Angular.js, React, and others. Its MVVM (Model-View-ViewModel) architecture makes it easier to handle HTML blocks. Unlike Angular and React, it's beginner friendly and comprises detailed documentation. One thing that must be considered is that, while fast and easy, it may not be suitable for large projects.

- **Svelte:** is an innovative JavaScript compiler designed to produce high-performance user interfaces. It was created in 2017 and is still in its early stages of development. It is distinct in that it does not employ a virtual DOM, but a specialised JavaScript Virtual Machine designed specifically for creating user interfaces. This allows Svelte to be up to ten times faster than other platforms, such as Angular and React frameworks. Therefore, svelte is a good choice if a small and quick application is needed by a small team of front-end developers, including beginners. However, due to its relatively small community, it might be unsuitable for large projects since this front-end framework is not widely used, and there might be a lack of the necessary help and tooling.

- **jQuery:** is a fast, lightweight, cross-platform JavaScript library for easily manipulating HTML documents and handling events from a client-side perspective. It provides a simple and intuitive API for handling common tasks such as DOM manipulation, event handling, and AJAX requests. jQuery is widely used in web development and can easily integrate with other libraries and frameworks. One of the main benefits of using jQuery is that it can significantly reduce the amount of JavaScript code required to achieve the desired effect. It allows developers to write more concise code, easier to read and understand and often faster to execute than pure JavaScript. Overall, jQuery is a powerful and widely used tool in web development, with a large and active community of developers continually contributing to its development and improvement.

- **Backbone.js:** is a well-known JavaScript library that provides web applications with proper structure by providing models with customised events and major key-value binding. There are libraries of enriched APIs that include functions, declarative event handling, and views. It communicates well with the current API via a RESTful JSON user interface. It is lightweight since it only uses two JS libraries: Underscore.js and jQuery. This language is ideal for creating single-page web applications and keeping multiple aspects in sync. With the help of the Backbone Layout Manager, developers can use predefined perspectives. It is based on the Model-View-Controller (MVC) architecture, where models represent the data and the logic behind it, views define the user interface, and controllers act as the glue between the two. An aspect that is best considered is that it does not support two-way data binding.

- **Semantic-UI:** created in 2014, it is a CSS (Cascading Style Sheets) framework based on organic language syntax. It provides a collection of pre-built UI components and design elements for building responsive and user-friendly websites and web applications. It uses natural language principles to describe classes, making it easier for developers to understand and remember the classes for specific UI elements. It also supports a wide range of plugins and integrations with other libraries and frameworks, such as React and AngularJS. Some of the key features of Semantic-UI include a responsive grid system, a variety of UI components such as buttons, forms, menus, and models, support for theming and customisation, and a collection of utility classes and helpers. It is also designed to be accessible and compatible with modern web technologies and browsers. Unfortunately, also this framework suffers from a small community.

Overall, these open-source tools provide developers with a wide range of options for creating user-friendly and accessible front-end interfaces that can be easily deployed to end-users remotely. However, it is difficult to say which of the presented frameworks is the most suitable since each has strengths and weaknesses, not to mention multiple complexities and frequently released updates bringing new features to the table.

5. Conclusions – Next steps

5.1. Development plan

The development plan for the IOP backend architecture encompasses the design, implementation, and integration of various components to enable efficient data processing and management. With a focus on real-time and batch processing, robust data storage, seamless data delivery, containerization, and CI/CD practices, the plan ensures scalability, reliability, and streamlined workflows. Additionally, the development of an SDK facilitates easy integration and interaction with the IOP, enhancing its usability for developers and fostering a vibrant ecosystem.

> Data Ingestion Layer:

- Select and configure an appropriate data ingestion framework, such as Apache Kafka.
- Set up Kafka clusters and topics to handle the incoming data streams.
- Develop connectors or adaptors to integrate with various data sources.
- Implement data validation, transformation, and enrichment processes as needed.

> Real-Time Processing Layer:

- Choose a real-time processing framework like Apache Flink, Apache Storm, or Apache Spark Streaming based on the specific requirements.
- Set up the chosen framework and integrate it with the data ingestion layer.
- Design and implement real-time data processing workflows and analytics pipelines.
- Define and configure windowing, state management, and fault-tolerance mechanisms.

> Batch Processing Layer:

- Determine the batch processing requirements and select appropriate tools, such as Apache Hadoop and Apache Spark.
- Set up Hadoop clusters and configure the distributed file system (HDFS).
- Develop batch processing jobs using MapReduce, Spark, or other suitable technologies.
- Implement scheduled or triggered batch processing workflows based on the required frequency.

> Storage:

- Determine the storage requirements for raw and processed data.
- Select and set up the storage solutions, such as HDFS for batch processing or Apache Druid for real-time analytics.
- Configure data replication, partitioning, and backup strategies for data durability and availability.
- Integrate the storage solutions with the data ingestion and processing layers.

> Serving Layer:

- Identify the data delivery requirements and select appropriate tools, such as Apache Superset or Trino.
- Set up the serving layer tools and configure data connectors and access controls.
- Design and develop interactive dashboards, visualizations, or APIs to serve the processed data.
- Ensure scalability, performance, and security of the serving layer components.

> Containerisation and Orchestration:

- Choose Docker as the containerisation platform and Kubernetes as the orchestration platform.
- Containerize the backend components and their dependencies using Docker.
- Set up Kubernetes clusters and configure deployment manifests for each component.
- Define scaling policies, health checks, and load balancing configurations.
- Automate the deployment and management of containers using Kubernetes.

> Continuous Integration and Continuous Deployment (CI/CD):

- Configure a CI/CD pipeline using GitHub Actions or other suitable tools.
- Set up automated build processes to package and test the backend components.
- Define test suites and quality assurance processes to ensure code correctness.
- Implement deployment automation to promote validated changes to production.
- Monitor the CI/CD pipeline for errors, failures, and performance issues.

> Testing and Quality Assurance:

- Develop comprehensive test plans covering unit tests, integration tests, and end-to-end tests.
- Implement automated testing frameworks and tools to validate the functionality of the backend architecture.
- Perform load testing and performance testing to ensure scalability and responsiveness.
- Conduct security testing and vulnerability assessments to protect against threats.
- Continuously monitor and optimize the backend architecture based on test results and feedback.

> Documentation and Knowledge Sharing:

- Create detailed documentation for the backend architecture, including design decisions, configurations, and workflows.
- Document the setup and configuration processes for each component.
- Define, design and implement the SDK.
- Develop comprehensive documentation and usage guides for the SDK.
- Develop user guides and troubleshooting guides.
- Facilitate knowledge sharing within the development team through code reviews, technical presentations, training sessions and an online help desk available in the IOP.

5.2. MLOps in the IOP

The IOP adopts MLOps practices to streamline model deployment, monitoring, and management processes. This involves the implementation of automated workflows that cover data preprocessing, model training, evaluation, and deployment stages. By utilising continuous integration and deployment (CI/CD) pipelines, the IOP ensures efficient updates and version control. Robust monitoring and logging systems are employed to track model performance, detect anomalies, and facilitate proactive maintenance and optimisation efforts.

5.3. Cloud

The IOP leverages cloud computing to harness the scalability and flexibility of cloud infrastructure. It utilises on-demand provisioning of computing resources, storage, and networking. Cloud services enable elastic scaling to adapt to varying workloads. Data storage and processing are efficiently

managed through cloud-based databases and object storage, ensuring secure and reliable access from anywhere.

5.4. Open-source approach

The open-source approach of the IOP emphasises collaboration, customisation, and innovation by leveraging open-source software and frameworks. It enables developers to access, modify, and contribute to the platform, ensuring transparency, flexibility, and community involvement. This approach promotes interoperability, avoids vendor lock-in, and facilitates integration with a wide range of technologies and tools.

In line with the open-source approach, the IOP utilises popular open-source technologies such as Apache Kafka for data ingestion and event streaming. This enables seamless collection and processing of large volumes of data in real-time. Additionally, Apache Flink is employed for real-time stream processing, providing scalable and reliable solutions for analysing data as it arrives. The IOP also utilises open-source frameworks like Apache Hadoop and Apache Druid for efficient data storage, processing, and analytics. These frameworks empower the IOP to handle diverse data sources and support advanced analytics capabilities.

References and Resources

[1] E. Y. Nakagawa, P. O. Antonino, F. Schnicke, R. Capilla, T. Kuhn, and P. Liggesmeyer, "Industry 4.0 reference architectures: State of the art and future trends," *Comput Ind Eng*, vol. 156, Jun. 2021, doi: 10.1016/j.cie.2021.107241.

[2] P. K. Illa and N. Padhi, "Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics," *IEEE Access*, vol. 6, pp. 55162–55170, 2018, doi: 10.1109/ACCESS.2018.2872799.

[3] T. Lins and R. A. R. Oliveira, "Cyber-physical production systems retrofitting in context of industry 4.0," *Comput Ind Eng*, vol. 139, Jan. 2020, doi: 10.1016/j.cie.2019.106193.

[4] F. Fraile and R. Poler, "Robotics Platforms for Internal Logistics: A Technical Architecture Proposal," 2022.

[5] T. Zschornig, J. Windolph, R. Wehlitz, and B. Franczyk, "A Cloud-based analytics architecture for the application of online machine learning algorithms on data streams in consumer-centric internet of things domains," in *IoTBDS 2020 - Proceedings of the 5th International Conference on Internet of Things, Big Data and Security*, SciTePress, 2020, pp. 189–196. doi: 10.5220/0009339501890196.

[6] N. Iftikhar, B. P. Lachowicz, A. Madarasz, F. E. Nordbjerg, T. Baattrup-Andersen, and K. Jeppesen, "Real-time visualization of sensor data in smart manufacturing using lambda architecture," in *DATA 2020 - Proceedings of the 9th International Conference on Data Science, Technology and Applications*, SciTePress, 2020, pp. 215–222. doi: 10.5220/0009826302150222.