Alma Mater Studiorum - Università di Bologna

Dottorato di Ricerca in
COMPUTER SCIENCE AND ENGINEERING

Ciclo XXXV

Settore concorsuale:
09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore scientifico disciplinare:
ING-INF/05 SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Edge and Big Data technologies for Industry 4.0 to create an integrated pre-sale and after-sale environment

Presentata da: Lorenzo Patera

Coordinatore Dottorato:
Prof.ssa Ilaria Bartolini

Supervisore:
Prof. Antonio Corradi

Esame finale anno 2023

Abstract

The fourth industrial revolution, also known as Industry 4.0, has rapidly gained traction in businesses across Europe and the world, becoming a central theme in small, medium, and large enterprises alike. This new paradigm shifts the focus from locally-based and barely automated firms to a globally interconnected industrial sector, stimulating economic growth and productivity, and supporting the upskilling and reskilling of employees. However, despite the maturity and scalability of information and cloud technologies, the support systems already present in the machine field are often outdated and lack the necessary security, access control, and advanced communication capabilities.

This dissertation proposes architectures and technologies designed to bridge the gap between Operational and Information Technology, in a manner that is non-disruptive, efficient, and scalable. The proposal presents cloud-enabled data-gathering architectures that make use of the newest IT and networking technologies to achieve the desired quality of service and non-functional properties. By harnessing industrial and business data, processes can be optimized even before product sale, while the integrated environment enhances data exchange for post-sale support.

The architectures have been tested and have shown encouraging performance results, providing a promising solution for companies looking to embrace Industry 4.0, enhance their operational capabilities, and prepare themselves for the upcoming fifth human-centric revolution.

Contents

1 Introduction

The implementation of Industry 4.0 in smart factories is currently a focus for both large corporations and government bodies, driven by the potential for significant changes in the manufacturing industry. This includes the development of new services, the creation of new business occasions, and the generation of new employment opportunities. The integration of mature Information Technologies such as Internet of Things (IoT), Cloud Computing, and Big Data can enable the digitization of industries. This transformation process requires companies to develop unified data infrastructures that integrate information generated at the operational level, such as data from machines and production chains, with end-user, third-party, and business data.

A key aspect of Industry 4.0 is the integration of third-party systems and services into the manufacturing processes. Such integration can be achieved by incorporating external sensors, devices, or platforms into the factory's current infrastructure. This enables the collection and analysis of data from multiple sources, which can be utilized to optimize production, minimize downtime, and enhance overall efficiency. Additionally, it allows for the integration of external services such as data analytics, machine learning, and predictive maintenance. This can lead to cost reduction, efficiency improvement, and increased adaptability to changes in demand.

Industry 4.0 is not limited to the production and sale of goods but rather encompasses the entire lifecycle of a product. This includes offering advanced services to end-users, such as remote monitoring (post-sale services), as well as providing highly customizable products at the same cost as mass-produced items. This approach allows companies to better understand and meet the specific needs of their customers, resulting in increased customer satisfaction and loyalty. Additionally, it permits greater flexibility in production and supply chain management, enabling companies to respond quickly to changes in demand.

Moreover, integration and cooperation will make a large amount of operational data available at the enterprise level, providing companies with better insights into live pro-

duction performance and the ability to make accurate short-term and long-term business decisions. Also, the benefits of in-premise interoperation between Operational Technology (OT) and Information Technology (IT) improve machine utilization, better-performing production lines, enhanced safety for workers, and the ability to implement new and unexplored business models.

There has been a recent increase in interest and hype surrounding the convergence of OT and IT in the industrial, manufacturing, and academic sectors. According to research firm Gartner [44], by 2020, half of OT service providers will collaborate with IT service providers to expand the range of services provided by Industrial Internet of Things (IIoT) devices used in industrial production sites. While the research community is actively working on the OT/IT convergence topic, currently, there are limited practical and cost-effective solutions available for small and medium-sized enterprises to achieve this convergence goal.

This work presents a set of architectures, developed through close collaboration with companies in the Emilia-Romagna region of Italy, that take advantage of Edge and Big Data technologies while also adhering to the privacy, safety, and security requirements of manufacturing firms. The proposed architectures aim to address the main challenges of OT/IT integration, including the creation of a homogeneous data gathering and processing architecture, the exploitation of software-defined networks and in-network processing potentialities, the implementation of serverless scalable processing at the edge, and the ability to gather data from machinery using low-latency communication protocols.

The remainder of this work is organized as follows: in chapter 2, we provide an overview of the key features and characteristics of Industry 4.0. In chapter 3, we delve into how cloud technologies can be leveraged to support the shift towards Industry 4.0. In chapter 4, we outline the goals and objectives of our research. Chapter 5 presents a review of relevant literature and existing integration architectures. The core of our work is exposed in chapter 6, where we present four architectures for data gathering and management in Industry 4.0, along with experiments and validation tests. The outcomes of our research on OT/IT convergence in the industrial and manufacturing sector are discussed in chapter 7. Finally, we summarize our findings and discuss future research directions in chapter 8.

2 Industry 4.0

We are in the midst of the fourth industrial revolution, which is changing the established paradigms of manufacturing, logistics, and pre and after-sale assistance. This chapter provides an overview of the key features of the four industrial revolutions, highlighting their technological advancements and the impact they had on all industries. It also examines the technological pillars defined by the European Union for Industry 4.0 and the European Servitization Program. Additionally, it covers the market share for Industry 4.0 and a brief look at Industry 5.0 (I5.0), the next industrial revolution, and its characteristics.

2.1 The four industrial revolutions

The industrial revolutions are processes of economic evolution and industrialization of societies from agricultural-artisan-commercial were transformed into industrial systems [41]. Figure 2.1 represents a timeline with the four main revolutions until nowadays. The remainder of this section provides more information about each one of them.

The first industrial revolution is traditionally attributed to the radical innovation in the textile-metallurgical sector with the introduction of the *flying shuttle* and the *steam engine*

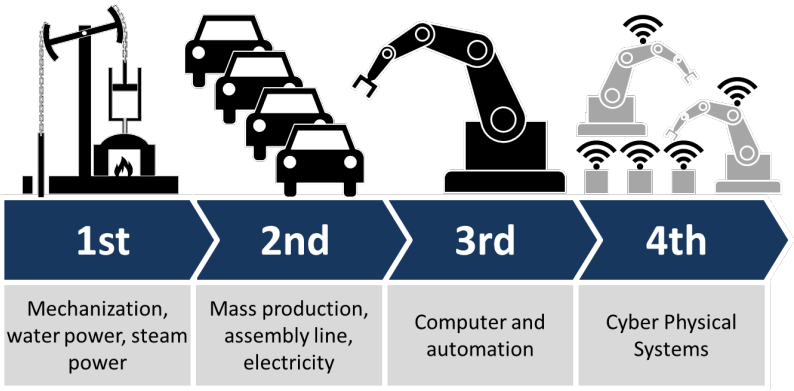


Figure 2.1: The four industrial revolutions [90].

in the second half of the 18th century [26]. As happens with many historical processes, there is no precise date on which it started. Industrialization was pervasive and it changed the socio-economic order of the whole of Europe impacting demographic, agricultural, commercial, transport, cotton, and iron industries. The radical changes in society, driven by the higher quality of products and mass production, transformed Europe from a poor, underdeveloped, and sparsely populated land in the early Middle Ages into the richest and most developed area in the world during the 19th century.

After about 120 years, from 1870 to 1914, a new phase of scientific discovery pushed the world in what is defined as the second industrial revolution [63]. Differently from the first phase of industrialization, which focused on mechanical and steam technologies advancements, this revolution was characterized by the introduction of machine tool industries, sewage systems, electricity, and telegraphs, and by the widespread adoption of railroad networks. The latter improved the mobility of goods and people across countries, enabling import, export, and commerce, pushing the world into globalization. The electrification permits the building of smaller and more efficient engines capable of moving conveyor belts and speeding up the employees' work. The premises can now be electrically lit, and it is possible to work more hours per day without using gas lighting, dangerous and pollutant. The advancements in research impact all sectors, spreading to chemical, petroleum, automotive, fertilizers, and business management.

The third industrial revolution (also known as the Digital Revolution) started in 1945 with the end of the Second World War (WW2) [67]. The war shifted the global scientific leadership from Western Europe to the United States. Pushed by the after WW2 Research and Development (R&D) investments, the United States was able to innovate in biomedical science and Information and Communication Technologies (ICT), becoming the leader of the digital revolution. In March 1971 Intel invented the world's first microprocessor [16] and the third industrial revolution starts spreading across all sectors. All the processes manually controlled by humans, now are monitored and managed by computers, System on a Chip (SoC), and automatic systems. Thanks to advanced computation and control, industries are locally more efficient and can produce high-quality goods in less time.

Although many industrial routines and machinery are monitored and controlled by SoC, processors, and computers, in the third industrial era the Machine to Machine (M2M) communication and Machine-to-Cloud communication are poor and often not existent.

Employees are forced to go to company premises to manually (re-)configure the machines, the goods are produced in series, and customizations have a high cost on a few pieces.

Only with the Fourth industrial revolution, in 2011, the focus shifted from a local to a global and interconnected perspective [54]. German industry was a pioneer, embracing Industry 4.0 (I4.0) in order to increase efficiency, flexibility, and innovation in their operations. The government has also supported the development of I4.0 through funding and initiatives such as the High-Tech Strategy and the Industry 4.0 Platform. The machines equipped with several sensors, advanced Human-Machine Interface (HMI), and emerging ICT technologies now work in a coordinated manner. Enterprises with several plants can control, supervise, and manage the machines via the internet as a logical and unique production unit. The application of Artificial Intelligence (AI) technology to industrial data suggests the best scheduling of work across devices, increasing the overall performance. The data gathering is pervasive and permits authorized stakeholders to access data and offer advanced services and pre- or post-sale support on final goods. With servitization the focus shifts from selling products to selling services, as a way to increase revenue and competitiveness.

However, although it may seem simple, the transition from Industry 3.0 to I4.0 is rich in challenges and issues that must be fixed to bring society to a new era. Germany, as a pioneer in industrial transition, proposed a plan comprised of four main milestones [106]:

1. **Building an always-connected network**
   The network is a core functionality in I4.0. Machine data must be gathered and collected locally or sent to the cloud capable of powerful and faster computation. With 5th Generation (5G) communication technologies and its network slicing, communications can be granted in Quality of Service (QoS) and site-to-site interconnection characteristics. The negotiation of networks' speed and latency permits to avoid degradation of the connections and to continue the communication across devices, smart products, and connected machinery.

2. **Researching on major themes**
   Smart Factory and Intelligent Production themes are crucial for advanced integration between machinery. Smart Factory focuses on AI-powered networks for machine data delivery, adaptive processes, and robotic load handling. Intelligent Production focuses on Advanced HMI, Three Dimensional (3D) object modeling, hyper-customization of products, and other technologies capable of making goods

production more flexible, customizations cheaper, and reducing the cost-per-unit of product. Investments in Research and Development on those themes are enablers in the industrial revolution.

3. **Implement pervasive integration**
   The fourth industrial revolution requires close cooperation between the so-called OT and IT. OTs are systems that are used to manage and control physical processes within an organization. Examples of operational technologies include Industrial Control System (ICS), Supervisory Control And Data Acquisition (SCADA) systems, and Manufacturing Execution System (MES). These systems are often used to monitor and control equipment, machinery, and other physical assets in real-time, and may also be used to collect data and provide insights into the performance of the organization's operations. ITs are systems used to process, store, and transmit digital information. These systems are used by organizations to manage and process data, communicate with customers and partners, and support a wide range of business activities. The integration of operational technologies and information technologies can provide organizations with a range of benefits, including increased efficiency, improved decision-making, and better control over their operations. By integrating OT and IT systems, organizations can gain a more comprehensive view of their operations, which can help them to identify problems and opportunities for improvement. Moreover, horizontal, vertical, and end-to-end integrations are crucial for better interoperability and cooperation within companies.

4. **Realize technological goals**
   As also in [15], there are eight main measurable objectives that can be used to evaluate the progress and effectiveness of Industry 4.0 technologies:

   - *Interoperability*: The ability of different systems and devices to communicate and exchange data with each other.
   - *Information transparency*: The availability of real-time data and information about all aspects of the production process.
   - *Technical assistance*: The use of advanced technologies such as artificial intelligence and machine learning to assist workers in tasks such as decision-making and problem-solving.

- *Decentralized decision-making*: The ability of systems and devices to make decisions and take actions on their own, without the need for central control.
- *Virtual commissioning*: The use of virtual reality and simulation technologies to test and optimize production processes before they are implemented in the real world.
- *Real-time optimization*: The ability of systems to continuously adjust and optimize production processes based on real-time data and feedback.
- *Modularity*: The use of modular, standardized components that can be easily assembled and reconfigured to meet changing needs.
- *Service orientation*: The ability of systems to provide ongoing support and maintenance to ensure that they are always operating at their best.

Industry 4.0 technologies have the potential to improve the sustainability of industrial processes by reducing waste, energy consumption, and emissions. This makes the integration of Industry 4.0 technologies into production processes a key aspect of the shift towards a circular economy, where resources are reused instead of discarded. As such, companies are increasingly considering not just environmental factors, but also social and economic aspects in their sustainability strategies. This focus on sustainability will be further examined in the next subsection 2.1.1.

2.1.1 Sustainability

According to data from the United States Environmental Protection Agency, industrial emissions are responsible for more than 40% of greenhouse gas emissions [33]. Many experts believe that digitization in manufacturing and the emergence of the fourth industrial revolution offers numerous opportunities to reduce carbon emissions. For example, the use of the IoT and AI in production can increase efficiency and flexibility, reduce waste, and minimize the carbon emissions associated with each product. Additionally, I4.0 may enable the development of new business models that shift from mass production to mass customization or even individualized production, which can optimize the consumer market and help to create a low-carbon future [68]. These changes may also contribute to environmental and social sustainability.

There is a strong connection between the economic and environmental performance of manufacturing systems, including those using Additive Manufacturing (AM). Currently,

AM techniques can be economically competitive with traditional processes for small to medium-batch production of metal parts, but the cost of AM machines and materials is still relatively high. As AM becomes more widely used, it is expected that the cost of these materials and machines will decrease, making AM more cost-effective. Additionally, as larger production volumes become more economically feasible with AM, the overall cost-effectiveness of the technology is expected to improve.

According to industrial reports, Industry 4.0 has also a high impact on the recruitment sector [12]. In this environment, automation technologies like industrial robots, automated vehicles, and intelligent machines are replacing humans in tasks like inventory tracking, quality control, and product distribution. While it is expected that Industry 4.0 will eliminate many low to medium-skilled jobs, it is also expected to create new employment opportunities in fields like informatics, mechatronics, process engineering, and system integration [42]. The social sustainability implications of Industry 4.0 extend beyond the creation of digitization-related employment opportunities. The digitization of the manufacturing industry also contributes to a more sustainable and green economy, creating millions of sustainable manufacturing-related job opportunities.

2.1.2 Fourth revolution and COVID-19 Pandemic

The COVID-19 pandemic has had a significant impact on Industry 4.0, with many organizations being forced to adapt quickly to the challenges posed by the pandemic. One key way in which this has happened is through the shift to remote work, as social distancing measures and business closures have made it necessary for many people to work from home. This has required organizations to implement new technologies and practices to support remote collaboration and communication, as well as secure remote access to data and systems. Thus, the pandemic has led to an increased reliance on digital technologies in many industries. For example, virtual reality, online collaboration tools, and e-commerce platforms have become more widely adopted as a way to continue operations and serve customers remotely.

Figure 2.2 shows a bar graph of I4.0 sectors' growth from 2015 to an estimation of 2021. As can be seen, during the first phases of the pandemic (2019-2020) all sectors of industry 4.0 registered increases and no loss in interest and investments. Estimations for 2021 were even more optimistic, estimating increases of more than 10% [84].

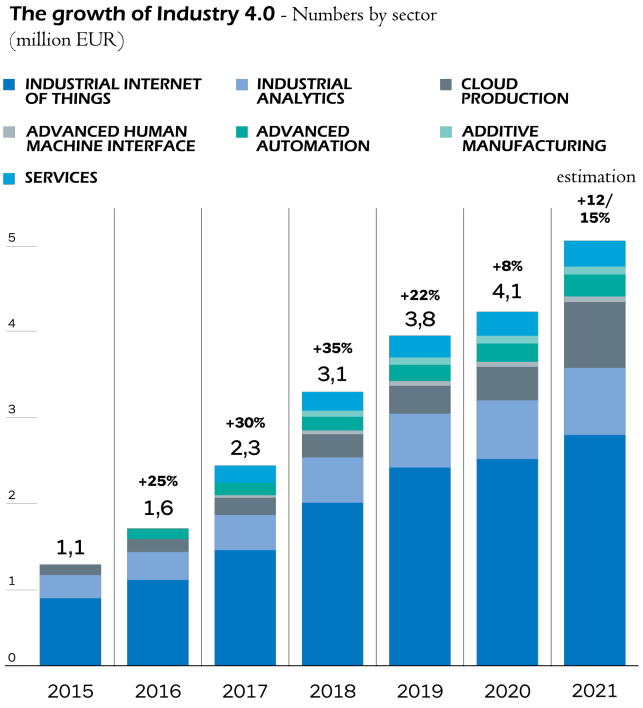


Figure 2.2: Growth of Industry 4.0 sector during COVID-19 pandemic.

Overall, the pandemic has had a significant impact on Industry 4.0, leading to the accelerated adoption of digital technologies and the need for more flexible and resilient business models. In [91], the pandemic is even seen as a booster, that is pushing the world into the fifth industrial revolution.

2.2 Technology pillars

European Union defined a series of technological pillars that leads the transition to a new era of research and advancements. The technological pillars are not only a series of technologies on which all communitarian states are encouraged to invest and to work but also they are proposed as technological sovereignty for an "interconnected, digitalized, resilient, and healthier European society" [34].

For I4.0 these technologies comprehend:

1. **Cybersecurity**
   Digital security for Industry 4.0 involves identifying and protecting against potential cyber threats, such as hacking, data breaches, and cyber espionage, and

implementing measures to ensure the confidentiality, integrity, and availability of digital systems and data. This may include measures such as implementing secure network architecture, using encryption, and regularly updating and patching systems to address vulnerabilities.

2. **Augmented Reality**
   Augmented Reality (AR) is a technology that overlays digital information and graphics on top of the physical world, providing users with an enhanced view of their surroundings. In the context of I4.0, AR can be used to provide workers with real-time information and guidance as they perform tasks on the factory floor, making it easier for them to access important data and instructions without having to refer to separate screens or manual instructions. AR can also be used to visualize complex processes and systems, making it easier for workers to understand and troubleshoot issues that may arise. Some potential applications of AR in Industry 4.0 include training, maintenance, quality control, and logistics.

3. **Big Data**
   Industrial machinery can generate a large amount of data that must be transmitted, elaborated in real-time, and stored. Industrial data can be structured and well-known (e.g., temperature of an oven, pressure of components in a thermal plant) or scattered and sparse (e.g., text, mobile activity, IoT data). Thanks to the usage of big data technologies, information can enhance product and service quality, bringing better knowledge of processes, enabling predictive maintenance, and reducing costs.

4. **Autonomous Robots**
   Autonomous robots perform precise and repetitive tasks in specific contexts, such as cleaning, product mobilization, welding, and others. The development of interconnected and AI-powered robots and their large-scale production will increase efficiency, improve safety, and reduce costs. A completely autonomous robot senses the environment, elaborates the information, works without human intervention, moves, and is safe for humans.

5. **Additive Manufacturing**
   Additive Manufacturing technology permits the building of three-dimensional objects by melting a strand of material, extruding it through an automated arm, and depositing it layer upon layer. Also known as 3D printing, this technique can

be used to create objects with complex shapes or features that would be difficult or impossible to produce using traditional manufacturing methods. There are two main categories of materials used in additive manufacturing: polymers and metals. Polymers tend to be less expensive to purchase and use in printing technologies, but metals offer superior performances. In AM, a digital model of an object is created using Computer-Aided Design (CAD) software.

6. **Simulation**
   The digital simulation of industrial processes permits finding design errors in advance and predicting system performances before production. Training of new employees can be performed through simulation in specialized and high-risk fields, avoiding injuries. The best expression of simulation can be reached through Digital Twin (DT). DT is created to faithfully represent a physical object or process. The object being researched (e.g., an air conditioner) is equipped with a variety of sensors that are connected to the key functioning regions. These sensors generate information about a variety of performance characteristics of the physical device, including energy input, temperature, environmental conditions, and more. The processing system then applies this information to the digital copy, updating its status.

7. **Horizontal and Vertical Integration**
   Integration with stakeholders and with the upper or lower industrial processes is crucial for the development of I4.0. Acquiring or merging with competitors or partners requires horizontal integration to overcome incompatibilities and make systems and machinery interoperable with the new ones. Making new partnerships e.g., with a logistic supplier requires vertical integration for automated shipping of parcels or replenishment of warehouses. Horizontal integration can bring benefits to both companies in the partnership, increasing the economy of scale and respecting the business division principles.

8. **Cloud Computing**
   Cloud Computing makes applications and services accessible everywhere is available an internet connection. Cloud applications can scale up and be distributed and accessed across many devices and IoT gadgets and wearables. The almost infinite computing power offered by cloud computing providers permits analyzing big data

and to extract knowledge from them. Moreover, it acts as an enabler for collaborative manufacturing, allowing for real-time collaboration and data sharing.

9. **Internet of Things**
   IoT permits to send and receive information from/to devices that traditionally were not connected to the internet. The availability of those data allows industries to improve the accuracy of digital twins and to better monitor environmental conditions inside the production plant. Embedded systems and the miniaturization of IoT devices allow the creation of wearable devices equipped with a monitor that can show an augmented reality, enriched in sensor information and instructions. When IoT is applied to the industrial devices, it takes the name of IIoT.

These pillars aim to provide digital security, enhance the physical world with digital information, process and store large amounts of industrial data, automate tasks with robots, create complex objects with 3D printing, simulate industrial processes for better design and training, integrate with partners and stakeholders, access cloud applications and services, and connect devices to the internet for better monitoring and collaboration.

2.3 European servitization program

I4.0 has created an incredible opportunity for the industrial ecosystem. However, the impact is high in terms of upskilling and reskilling of employees, pushed to learn how to use new technologies and smart instrumentations. For Small and Medium-sized Enterprises (SMEs) the transition to advanced industrial environments is quite difficult and requires double effort from Europe and directly from member states with adequate (infra)structures, benefits, and tax reductions.

The primary objective of the Digitising European Industry plan is to establish a platform that collects all the information, protocols, experiences, and large-scale tests conducted by member states, which aims to become the largest database on Industry 4.0, focusing on standardization in the sector. This platform also aims to reach the necessary scale to attract private funding and joint investments from major industrial players, governance elements such as high-level roundtables will be established where representatives of member state initiatives, industry leaders, and social partners, meet with the European Commission and other European stakeholders. In April 2017, the European platform of national initiatives [38] was established, which focuses on collecting information and experiences from

member states on Industry 4.0. As of May 2021, it has been integrated into Futurium [37], which is a platform dedicated to EU policy discussions with European citizens.

The second pillar of the strategy is the establishment of European Digital Innovation Hubs (EDIHs). The main target of these EDIHs are SMEs, start-ups, and mid-caps, companies that do not have sufficient resources to upgrade and implement European guidelines independently. The EDIHs will be located in every member state and connected through an EU-wide network. Member states and regions are encouraged to invest in the creation or strengthening of EDIHs that align with their national or regional digitalization strategy, and they should secure the necessary funding through regional development funds. The European Commission has committed to investing 329.3 million euros per year from 2021 to 2023 in EDIHs [36].

The European strategy includes Public-Private Partnerships (PPPs) as a means of promoting industrial innovation. The goal of this initiative is to increase competitiveness in digital technologies in the EU and establish European leadership. To achieve this, the European Commission has created PPPs and Joint Undertakings in key digital areas such as 5G, big data, High Performance Computing (HPC), cybersecurity, photonics, robotics, and electronic components and systems, which are funded through the EU's Horizon 2020 program.

In order to achieve the objectives of the European digital industry strategy while maintaining user privacy, the European Commission is working on regulating critical aspects of the industry. Two main areas of focus are the free flow of non-personal data and cybersecurity. The free flow of non-personal data aims to enable a competitive data economy within the member states, allowing companies and public administration to store and process industrial data. This regulation covers several key aspects such as the movement of data across borders, switching between different cloud service providers, and providing portability and independence of data flow. The cybersecurity package aims to establish a certification network for enhanced cyber-resilience and refactor the EU's cybersecurity agency, ENISA, to provide support to member states, and contribute to operational cooperation, issuing EU-compliant certificates and crisis management across the EU.

The last and arguably most important pillar of the European digital industry strategy is to prepare citizens for the digital revolution and the impact it will have on their studies and careers. This is accomplished through partnerships between member states, companies, social partners, non-profit organizations, and education providers. These entities collaborate at different levels to address the digital divide among citizens and ensure that everyone

has access to necessary education, regardless of their starting point. The education and training offered range from high-level training for unemployed individuals to training in cutting-edge technologies for IT professionals.

2.4 Market share

It is challenging to provide a single market share for Industry 4.0 as it comprehends a very large number of technologies, each with its own market size and projected growth. However, it is possible to break it down by specific sub-areas to get an estimate of the market share for each technology. Here are a few examples of different technology areas that are part of I4.0, and their estimated market size:

- **Internet of Things**: IoT technology is expected to see significant growth in the coming years. According to research firm MarketsandMarkets, the IoT market is expected to grow from USD 384.5 billion in 2021 to USD 566.4 billion by 2027, growing at a Compound Annual Growth Rate (CAGR) of 6.7% [58].

- **Big Data Analytics**: The big data analytics market is also projected to grow at a significant rate. According to MarketsandMarkets, the big data analytics market is expected to grow at a CAGR of 11.0% from 2021 to 2026 [56].

- **Cloud Computing**: The market for cloud computing is also projected to reach a significant size. According to research firm Mordor Intelligence, the cloud-enabling technology market size is projected to reach USD 46.17 billion by 2026, growing at a CAGR of 8.26% during the forecast period of 2021 to 2026 [65].

- **Artificial Intelligence**: The AI market is also expanding quickly, AI is expected to drive innovation and efficiency across industries, and it is expected to have a market size of USD 407.0 billion by 2027, at a CAGR of 36.2% during the forecast period 2022 to 2027 [55].

- **Industrial Robotics**: Robotics industry is also expanding in various domains such as healthcare, manufacturing, and agriculture, the market size of industrial robotics is expected to grow from USD 15.7 billion in 2022 to 30.8 billion by 2027, at a CAGR of 14.3% [57].

- **Cybersecurity**: As digitalization increases, so does the risk of cyber-attacks and data breaches. This is creating a growing demand for cybersecurity solutions. According to Mordor Intelligence, the global cybersecurity market size is expected to reach USD 2317.02 billion by 2027, growing at a CAGR of 13.37% during the forecast period 2022 to 2027 [66].

The above information provides an overview of some of the major technology areas within I4.0 and their estimated market size. It's important to note that as technology continues to advance, these forecasts are subject to change and are expected to evolve as new applications are developed. However, all the technology areas mentioned are showing growth and have the potential to bring significant benefits to the businesses that adopt them.

2.5 Toward industry 5.0

Although the goals of the fourth industrial revolution have not yet been achieved, the fifth industrial revolution is already present in the literature [40, 59, 64, 96]. I5.0 includes and empowers the Industry 4.0 paradigm by focusing goal of making manufacturability sustainable from economic, ecologic, and societal perspectives. This integration is expected to lead to even greater levels of efficiency, flexibility, and intelligence in manufacturing and other industries. Integrating sustainability and environmental considerations into this model could involve the use of these technologies to support the development of more environmentally-friendly production processes, as well as the use of data analytics and other tools to track and optimize the environmental performance of organizations. In I5.0 machinery and processes will reach new levels of resilience, by incorporating self-healing and self-diagnosing capabilities into industrial systems, so that they can detect and repair problems on their own without human intervention.

Human-centric green I5.0 will place emphasis on creating a sustainable and inclusive environment that promotes quality of life and well-being for all people, while also addressing global challenges such as climate change and resource depletion. It will also focus on creating safer, healthier, and more efficient work environments, integrating them with smart grids.

This new era of the industry is expected to be driven by the integration of cutting-edge technologies such as Artificial Intelligence, IoT, Cloud Computing, and Robotics, as well as new technologies in the field of Energy, such as renewable energy and energy storage, as well as advanced materials, biotechnology and advanced manufacturing.

The human-centric green I5.0 is expected to promote the use of sustainable resources, circular economy, I4.0 systems, and technologies to reduce waste and emissions and create a more sustainable future. Green technologies like IoT, big data, and advanced manufacturing are expected to play a key role in I5.0, which is designed to protect the environment and improve the quality of life for all people.

3 Cloud Technologies

This chapter explores the concept of cloud continuum as it relates to Industry 4.0. The cloud continuum refers to the integration of cloud computing technologies within the manufacturing industry to enable greater efficiency, flexibility, and scalability. The chapter will delve into the different types of cloud services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Function as a Service (FaaS) and how they are used to support Industry 4.0. Additionally, the chapter will cover other key technologies within the cloud, including middleware software, message-oriented middleware and containerization.

3.1 Cloud continuum

Distributed systems and the widespread fast internet access pushed the application from running on a local computer or in-house workstations to servers sparse across the world and accessible "as a Service (aaS)". Data storage, applications, computing resources, and even functions are managed by cloud providers, relieving the burden of management of hardware and software shifts from companies' Developers and System Engineers. Costs of hardware and software are modulated on an as-needed basis. The users can select the best cloud plan for their businesses or pay for single computations. Scalability is granted by the providers, which, based on the user's needs, can add or remove virtual and physical resources from the pool transparently.

The key features of cloud computing are:

- **Remote access**: Cloud computing allows users to access resources from anywhere, on any device, as long as they have an internet connection.
- **On-demand resources**: Cloud computing allows users to only pay for the resources they use, which can be scaled up or down as needed, without the need to invest in additional hardware or software.

- **Reliability**: Cloud providers typically have multiple data centers and employ redundancy and backup systems to ensure that resources are available even if one or more data centers fail.
- **Managed services**: The service provider is responsible for monitoring, maintaining, and troubleshooting these systems, and for ensuring that they are running smoothly and efficiently. This can include tasks such as patch management, software updates, security, and backups. Additionally, Managed Services providers typically offer a Service Level Agreement (SLA) to ensure the quality and availability of their services.
- **Pay-per-use**: Customers are charged for the specific services they use, such as storage, computing power, or data transfer. The charges are typically based on usage metrics, such as the amount of data stored or the number of CPU hours used, granting benefits for both customer and service providers, as it allows them to optimize their usage of resources and manage costs more efficiently.

Cloud computing, while providing many advantages, also has some potential drawbacks. Starting from security risks, since the servers are remote and accessible via the internet 24/7, there is the possibility of attacks and data breaches. Cloud providers try to put in place all the measures for controlling unauthorized access, but breaches can still occur. An attack on a data center or the network can cause inaccessibility of the cloud and in industrial scenarios even the stop of some production lines. Moreover, since the cloud is accessible only via the internet, a reliable and fast internet connection is required to access it and that can be challenging in rural and isolated areas. Also, control over data can be risky, since cloud providers tend to consider clients' data as part of their business and often deny the possibility of downloading from customers. The control of data can also be crucial in some industrial systems since can contain reserved information and industrial secrecies. The limited customization offered by cloud providers can make it difficult to tailor the service to unique requirements. Costs of per-customer customization can be high and non-convenient. Lastly, maybe the major downside of a cloud approach is the so-called vendor lock-in which refers to the difficulty of moving data, applications, or other resources from one cloud provider to another. A cloud provider can offer many custom Application Programming Interfaces (APIs) and services that other competitors have not, so a switch to another cloud provider can require major changes in the software that, in some cases, could even be impossible.

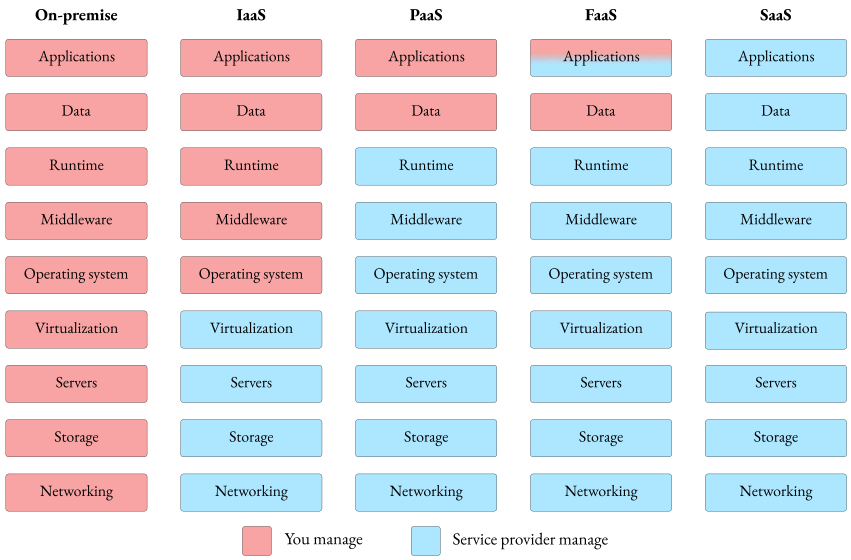


Figure 3.1: Cloud service models.

Cloud computing is generally categorized into four main service models: IaaS, PaaS, FaaS, and SaaS represented in Figure 3.1 that highlights what is managed by the developer or the final user (pink) and what is managed by the cloud provider (light blue). Amazon Web Services, Microsoft Azure, IBM, and Google Cloud Platform are among the top providers of cloud services, offering flexible and scalable solutions for various cloud needs [30].

Before discussing service models, it is important to understand the concept of on-premise systems. These systems are installed and run on a company's own data center, with the company taking responsibility for all related tasks. This can include finding a suitable space to house the servers, negotiating internet connections with providers, and managing the installation of cables, hardware, routers, servers, uninterruptible power supply, and backup systems. Additionally, virtualization services are often necessary to manage the underlying hardware and an operating system is needed to support business-oriented application development. At this level of abstraction, developers can then make informed decisions about the appropriate application middleware, runtime, data storage, and user-facing applications for their specific needs. The following subsections will describe the key characteristics of each Cloud Continuum service model, along with examples of popular open-source implementations.

3.1.1 Infrastructure-aaS

In the IaaS model, the service provider manages all levels of infrastructure, including physical, networking, storage, servers, and virtualization. IaaS providers offer customers the ability to rent, on a pay-as-you-go basis, the infrastructure required to run their applications and services. The key advantage of this model is its high flexibility, as customers can easily scale up and down the infrastructure they need, including Virtual Machines (VMs) and virtualized network elements (such as switches and routers). These VMs, also known as instances, can be selected from a pool and customized to meet the customer's specific needs, with varying levels of CPU, memory, and storage. Operative Systems (OSs) can also be installed and configured via the provider's admin panel, or selected from a catalog of pre-configured options. Custom or specialized OSs can be installed via admin panels and virtualized Secure Shell (SSH) accesses.

Storage services are one of the key components of IaaS. Typically, IaaS providers offer block and object storage. Block storage is suitable for applications that need quick read-and-write operations and immediate access to data. It allows for high performance with databases and transactional data storage and offers control over the location of the storage, providing proximity to application servers. However, block storage has limited scalability options due to its traditional management of data and metadata, making it considered an "old-fashioned" storage option. Object storage is considered the future building block for highly-scalable storage systems. They permit the storage of a large amount of data across multiple regions, scale infinitely to a petabyte or even more, and are often equipped with analytics capabilities. The latter permit organizing and retrieving data easily, providing redundancy and increasing data availability, replicating information across multiple data centers. In conclusion, object storage is typically less expensive than block storage, as it does not require specialized hardware or software. This makes it a more cost-effective solution for big data applications.

Network virtualization lays the further foundation for IaaS. Network virtualization refers to the use of virtualization technology to create virtual networks that emulate the functionality of physical networks. The IaaS admin panel permits to create, configure, manage and destroy customized virtual networks within the underlying shared infrastructure. Several techniques permit to achieve network segregation and virtualization, the most used are [17]:

- **Virtual Local Area Networks (VLANs)**: A VLAN is a network build upon a Local Area Network (LAN) on witch the hosts communicate by using tags (called VLAN IDs). The advantages of this technology comprehend flexibility in terms of network administration and management since all the needed configurations are set via software. Moreover, several VLANs can be logically connected and hosts sparse on different local networks can communicate as if they were local.

- **Virtual Private Networks (VPNs)**: A VPN allows for the connection of geographically dispersed entities or regional sites into a secure, single network. The purpose of a VPN is to establish a private network using public or mixed private-public infrastructures and utilizing Internet Protocol address (IP) routing. VPNs can be used on various operating systems and typically have secure access through the use of login credentials, passwords, certificates, or even biometric features.

- **Software-Defined Networking (SDN)**: In SDN control and data planes are split into two logically separated entities that can be pure software or a mix of software and ad-hoc hardware. This separation allows improved control over data paths, as well as more flexibility and programmability of the network. The control plane can make specialized decisions over data flows that can be prioritized based on policies, destination, source, contained data, or channel congestion. In IaaS can be used to dynamically manage and provision network resources.

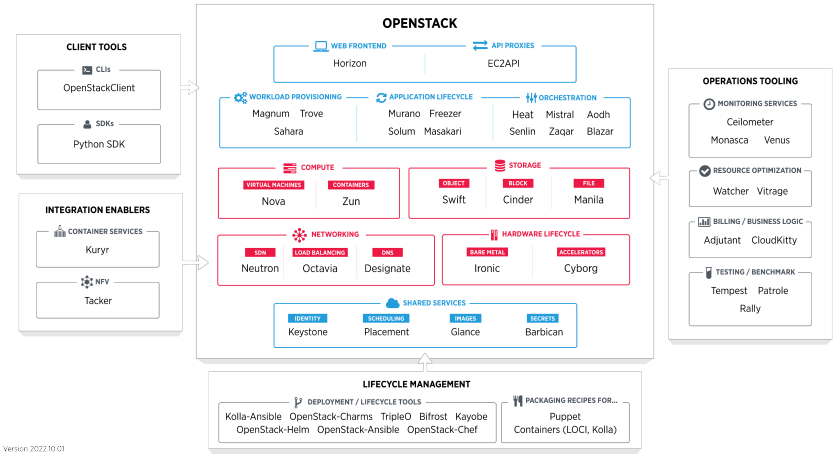


Figure 3.2: OpenStack components map [81].

OpenStack [80] is an open-source cloud computing platform that provides IaaS and it is composed of multiple individual components, or "services", that can be deployed and configured independently based on the user's needs. The latest version of OpenStack "OpenStack Zed" released in October 2022, includes 29 different core services and many additional services for configuration, management, and integration and it is represented in Figure 3.2. However, a minimal installation of OpenStack can be made with only 6 main services (Keystone, Glance, Placement, Nova, and Neutron) plus a couple of suggested commodity and dashboard services (Horizon and Cinder).

Here is a summary of the key components and their functions in a minimal OpenStack installation:

- *Keystone*: Keystone is the authorization and authentication service. It provides a centralized, uniform, and coherent way to securely connect to all other services through its identity services. It also offers a service catalog that allows users to discover all the services available in the OpenStack cluster. Keystone attaches meta information to each service, indicating if it is an admin, internal, or public service, and the port it is available on.

- *Glance*: Glance is the image service for OpenStack. It allows for the discovery, registration, and retrieval of virtual machine images through its Representational state transfer (REST) APIs, which permit querying the service and retrieving an image. With Glance, images are cached in order to decrease latency and increase the availability of images throughout the cluster.

- *Placement*: The Placement service allows for the scheduling and planning of computing resources like virtual machines and containers. It offers a centralized REST APIs for users to query and request resources. Additionally, it maintains an inventory of available resources in the OpenStack cluster and enables the setting of quotas for users in terms of CPU, memory, and storage.

- *Nova*: It is the compute service in OpenStack that enables the creation and management of virtual machines and other compute resources. It interacts with other services such as Keystone and Glance for service discovery and image retrieval. Moreover, it includes a scheduler that is responsible for identifying the appropriate compute node with sufficient resources to host new instances. The APIs permit other components to interact with Nova and to manage VMs.

- *Neutron*: Neutron is the component responsible for the networking of compute instances. It allows users to create and manage virtual networks, including routers, switches, and other networking components. It offers a variety of network options, ranging from basic virtual bridges to advanced virtual networks (e.g., VLAN, VPN, SDN, and others).

- *Horizon*: It is the web-based dashboard service that enables the management of all OpenStack components. It can be accessed over public or private networks and features automatic updates through its interaction with Keystone for service discovery, making new components and capabilities available as they are installed.

- *Cinder*: Last of suggested minimum installation components, Cinder is the block storage service for OpenStack. It provides a way to create and manage block storage devices, such as hard drives, and manage their attachments to compute instances.

OpenStack is an example of a widely used and component-based IaaS platform. Other cloud infrastructure providers offer similar services to OpenStack, some of the most popular include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Alibaba Cloud, VMware vSphere, OpenNebula, and CloudStack.

3.1.2 Platform-aaS

PaaS allows developers to run, develop, execute and monitor applications without having to worry about the underlying infrastructure. The service provider, along with the IaaS-covered layers, abstracts the architecture even more, installing and hiding from the final user the OS, the middleware, and the runtime. This allows developers to focus on writing code and building features, rather than managing servers and other infrastructure components. It also allows for faster and more efficient deployment of applications, as well as automatic updates and upgrades. Key features of PaaS comprehend the total abstraction from the underlying architecture, making the developed application more portable and platform-independent. Scalability functions and replication mechanisms permit to scale-up and down of the application without requiring manual intervention from developers. Develop production times decrease, thanks to tight integrations between offered services. The same multi-tenant infrastructure can be used by many developers in an isolated and safe way, providing each one a dedicated view. Finally, a PaaS infrastructure is kept updated

without the intervention of application developers who can benefit from security patches and other fixes.

As with IaaS, PaaS also has some drawbacks. In this service model, flexibility or customization are limited, since service providers cover only widespread use cases which can guarantee many paying users with contained costs. Given the platform abstraction from the underlying layers, troubleshooting and precise control of the execution environment is difficult or even impossible, causing inability in optimizing application performances. As with all other cloud continuum technologies, PaaS also has the potential for vendor lock-in, dependence on an internet connection, and costs that must be taken into account during the evaluation phase.

Cloud Foundry [18] is a widely used open-source PaaS platform that allows developers to build, scale, and deploy multi-language applications, including Java, Ruby, Python, Go, and .NET, and distribute them on microservice architectures or container-based deployments. It has a large user base and an active community, and commercial options are available for professional users.

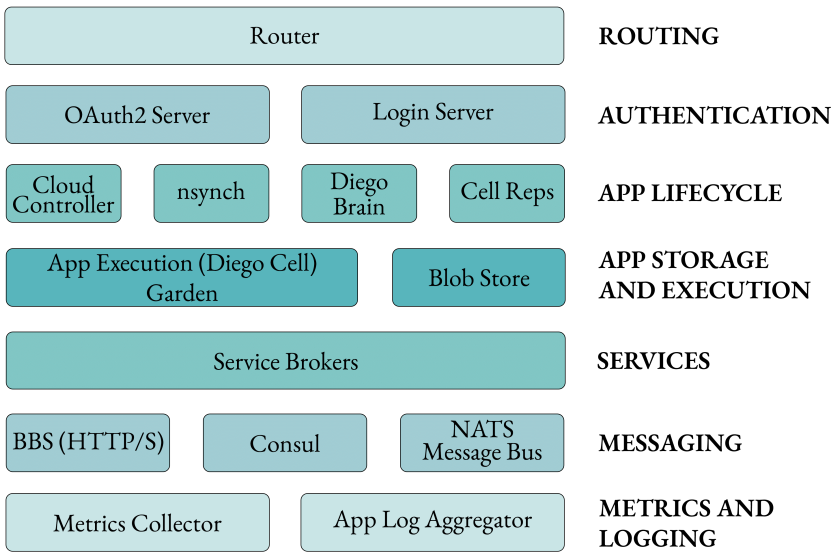


Figure 3.3: Cloud Foundry architecture schematic.

Figure 3.3 represents an overview of Cloud Foundry architectural components. It is composed of 7 different layers:

- *Routing*: This layer is responsible for directing incoming traffic to the appropriate component. The router updates its routing tables by regularly querying the Diego Bulletin Board System (BBS) to determine the current location of a component.

- *Authentication*: The authentication layer allows users to log in and applications to authenticate through a token-based system provided by the OAuth2 server. This ensures that only authorized users and applications have access to the platform and its resources. This layer is crucial for maintaining the security and integrity of the platform by validating the identity of the users and applications that are accessing it.

- *App Lifecycle*: The app lifecycle layer is responsible for managing the registration and updates of applications, as well as controlling the replication of components. This layer ensures that the applications are running in an updated and consistent state by monitoring their health and making necessary adjustments. This includes tasks such as starting and stopping applications, scaling them up or down, and performing maintenance or updates.

- *App Storage and Execution*: The app storage and execution layer are responsible for storing and managing the files, code packages, buildpacks, droplets, and other elements necessary for the functioning of an application. It is the foundation for the execution of the applications on the platform, and it allows for the storage and retrieval of large binary files, application code, and other components.

- *Services*: The services layer is responsible for integrating with third-party databases and SaaS providers. The Service Broker is the main component of this layer and it enables the platform to connect to external services and resources. It allows developers to consume external services and resources, such as databases, messaging systems, and other types of services. The Service Broker also manages the provisioning and binding of services to applications, which means that it is responsible for creating new instances of services and making them available to applications.

- *Messaging*: Cloud Foundry utilizes messaging to facilitate communication between different components within the platform. The internal HTTPs layer allows communication with the Diego Cell to start, stop, and scale applications. The BBS provides a message queue implementation to establish communication between

Diego components. Additionally, the NATS Message Bus is used to communicate with the router, thereby propagating the latest routing table.

- *Metrics and logging*: The Metrics and Logging layer of Cloud Foundry infrastructure allows developers to have real-time metrics and a centralized location for gathering all application logs, which is essential for debugging and troubleshooting. This layer provides valuable insights into the performance and behavior of the deployed applications, helping developers to optimize their code, detect and fix errors, and identify potential bottlenecks. Additionally, the metrics and logging layer allows monitoring of the underlying infrastructure of the platform, enabling the detection of potential issues and maintenance.

Overall, PaaS is a valuable tool for developers looking to build and deploy web and mobile applications quickly and easily. Cloud Foundry is a popular open-source PaaS platform that provides a comprehensive set of tools and services for building, deploying, and scaling applications. It abstracts away the underlying infrastructure, allowing developers to focus on writing code and building features. However, as with any PaaS solution, it's important to weigh the benefits against the potential drawbacks, such as vendor lock-in and costs.

3.1.3 Software-aaS

With SaaS, all management and development of the application are handled by a third-party provider, and the application is made available to customers over the internet. This service level is the last step in the cloud continuum, where customers pay on a pay-per-use or subscription basis to directly use the final application. The main advantage of this technology is that the end-user does not have to pay for infrastructure, and can access the software from any location, browser, and OS. Data is fully managed by the service provider, and the user has access to an integrated view of the application through a reserved area. Examples of SaaS include Customer Relationship Management (CRM) software like Salesforce, human resources management software like ADP, and office productivity software like Microsoft Office 365, G Suite, and Overleaf.

SaaS is often offered for free, as service providers can collect personal information and preferences that they can sell to third-party companies or monetize through targeted advertising. Some software is offered in freemium versions, where the user can try out

the platform before paying for advanced features or continue using it after a trial period. However, since the provider has complete control over the application, data, and usage information of customers, there can be privacy concerns. Other disadvantages are discussed in the previous sections and overlap with other cloud continuum technologies: vendor lock-in, limited customization, limited control, and limited data ownership.

3.1.4 Function-aaS

One of the most recent advancements in Cloud Continuum is represented by FaaS. The term FaaS was first introduced by Amazon Web Services (AWS) in 2014 with the release of their serverless computing platform, AWS Lambda. FaaS, in addition to the features provided by FaaS, offers a high level of support for application development by managing the entire lifecycle of the functions developed. FaaS eliminates the need for developers to manage servers, scaling, or availability, allowing them to focus solely on writing and deploying business functions. That can lead to cost savings as developers only pay for the resources they use on a fine-grained scale, and increased agility as they can quickly and easily deploy new functions.

FaaS is often paired with managed services such as storage, databases, and messaging queues to achieve a Serverless architecture. Since functions are activated by the framework only when there is a request, the usage of resources on the FaaS server is minimal when there are no requests, providing cost savings for the users.

The underlying mechanisms of FaaS involve several components that work together to create and respond to requests. When a function is first invoked, resources such as memory and CPU may need to be allocated and the function's code and dependencies loaded. This process, known as a cold start, can cause a delay in the execution of the function. Alternatively, they can be managed through a pool of resources that are kept in memory and ready to respond to requests after a few more instantiations (warm start) or they can be completely pre-allocated and ready to go (hot start). These different start types can impact the performance of the functions and it's important to understand how they will affect the application and the resource usage in the node.

There are multiple open-source distributed FaaS platforms available such as OpenFaaS [79], KNative [52], OpenWhisk [101], Nuclio [70], OpenLambda [100], and IronFunctions [49]. For the purpose of this paragraph and to better illustrate the concepts

discussed, Apache OpenWhisk is selected as an example, owing to its clean architecture and ease of understanding.

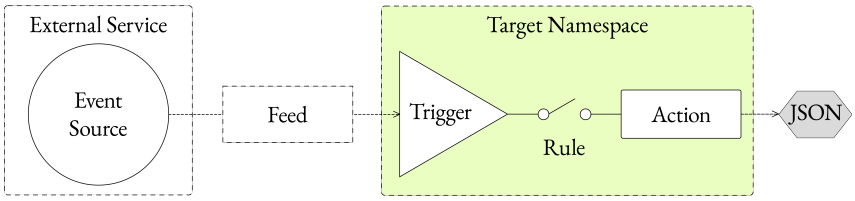


Figure 3.4: OpenWhisk programming model schematic.

The OpenWhisk programming model is based on the schematic in Figure 3.4. Events can originate from various sources such as databases, message queues, chatbots, timers, etc., and trigger the execution of a specific action. An action is a stateless piece of code that runs on the platform and can be written in multiple programming languages, giving developers the flexibility to use their preferred one. Actions can also be combined to form a sequence of actions that execute one after the other. Triggers are implemented through the use of channels and can be connected to a specific action or sequence through rules. Once a rule is set, any event that enters the trigger will initiate the execution of the corresponding action. The final output is in JavaScript Object Notation (JSON) format.

OpenWhisk utilizes several internal technologies such as Docker containers [27] and Apache Kafka broker [98] to abstract the complexity of underlying systems and enables seamless scalability across a cluster of machines.

Like other technologies, also FaaS has some disadvantages. One of the main disadvantages is the cold start problem, which can significantly impact system performance. Additionally, the nature of the technology limits flexibility, as functions are stateless, and maintaining state across multiple function invocations may not perform well, as accessing databases and other sources can be expensive in terms of latency and bandwidth. As with other cloud continuum technologies, FaaS also has the potential for vendor lock-in and security concerns, as functions must be exposed over the internet to be accessed by users.

3.2 Middleware Software

Cloud technologies intertwine with a more general concept of Middleware Software. Middleware acts as a bridge between the operating system or physical hardware, and the applications, allowing them to interact and work together even if they were not designed

to do so. Middleware software provides additional functionalities such as caching and multiple communication capabilities. Using middleware also speeds up the development process and reduces the time it takes for a product to be brought to market.

Middleware grant both functional and non-functional properties to the applications. From a functional perspective, they enable communication between different applications and systems, allowing them to share data and work together. This can include support for various communication protocols, such as Hypertext Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP), and REST, security, integration, many-to-many communication models, and data management. Functional properties are mandatory and can be achieved also with a different kind of software.

On the other hand, the non-functional properties, that grant a high QoS, are challenging to be achieved without the usage of middleware. Principal non-functional properties of middleware comprehend:

- **Performance**: Middleware should be designed to provide good performance in terms of response time, throughput, latency, and scalability under any kind of workload.
- **Reliability**: Middleware engineers should design and develop the middleware to be reliable and have high availability, in order to ensure that it provides consistent and dependable services.
- **Scalability**: Middleware should be designed to be scalable so that it can handle increasing loads and support more users.
- **Security**: Middleware should be engineered to ensure a high level of security so that it can protect data and restrict access to only authorized users.
- **Manageability**: Middleware should be designed to be easy to manage and maintain so that it can be updated and modified as needed.
- **Interoperability**: Middleware should be designed to be interoperable with other systems and applications so that it can work with a wide range of technologies.
- **Extensibility**: Middleware should be designed to be extensible so that new features and capabilities can be added as needed.

- **Flexibility**: Middleware should be designed to be flexible and adaptable so that it can be used in different environments and situations.

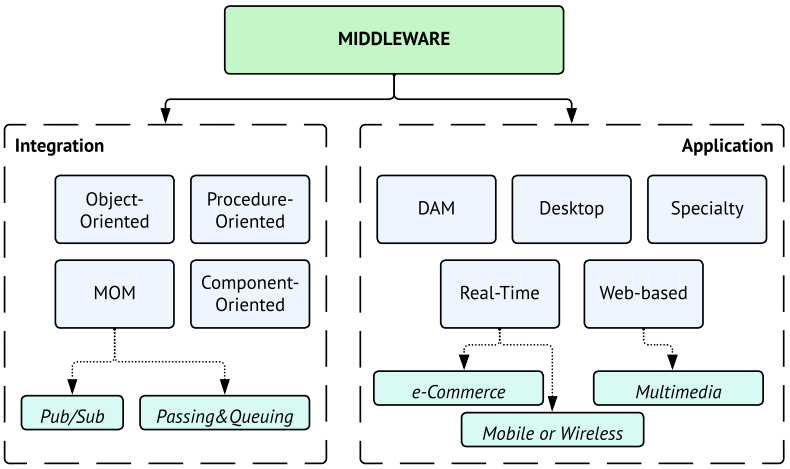


Figure 3.5: Categories of Application Middleware.

As outlined in the taxonomy established by [9], and reported in Figure 3.5, there are a variety of different types of middleware that can be broadly categorized into two main groups: integration and application middleware. Integration middleware enables communication and data sharing between different systems. It provides mechanisms to help different systems interact and work together in a coordinated and uniform manner. The main examples of this category are Object, Procedure, Message, and Component -oriented.

Object-oriented technology utilizes the creation of an object to facilitate the sharing of information and behavior between entities connected to an Object Request Broker (ORB). ORBs convert application language and format data, taking into account the endianness, by utilizing the Stub and Skeleton approach, which converts local data into ORB objects. The middleware then calls one or more services and returns the response to the caller, always passing through local converters. The interaction can be synchronous, asynchronous, or deferred asynchronous. A noticeable example of ORB implementation is the Common Object Request Broker Architecture (CORBA) [103].

Procedure-oriented middleware establishes a specific communication channel between the client and server, which is used to transmit data through marshaling and unmarshalling processes. Unlike object-oriented middleware, the communication is exclusively

synchronous and demands that both nodes are active and prepared to answer requests. In the event of exceptions, errors, or connectivity issues, these are propagated to the application layer through local Stubs and Skeletons. Examples of Procedure-oriented middleware are Remote Procedure Calls (RPCs).

Component-oriented middleware, also known as Reflective middleware, is a flexible architecture that allows for the creation of distributed systems through the use of software components. These components can be reused and combined to build complex systems. Reflective middleware has the ability to automatically assess the state of the system, identify the appropriate services and components to handle specific requests, and dynamically connect the service request to the appropriate service provider. This feature enables the system to adapt to changing needs, optimize resource utilization, and incorporate new functionality. An example of this type of middleware is the Enterprise JavaBeans (EJB) standard.

A Message-Oriented Middleware (MOM) is a type of software application that acts as an intermediary for message communication between senders and receivers. MOMs can be classified into two categories: Passing & Queuing and Publish-Subscribe. Passing & Queuing acts as a router for messages in the system without modifying or interpreting the message or its content. The client of Passing & Queuing sends the message, which is read by a server and then delivered to the correct receiver in a First-in-first-out (FIFO) or other type of order. The characteristics and usage of Publish-Subscribe (Pub/Sub) middleware will be thoroughly examined in the next section 3.2.1, as it is a key focus of this dissertation and will be extensively utilized in the research projects outlined in 6.

Application middleware is a widely-used technology that enables the resolution of specific types of interaction schemes between different software entities. Some examples of this broad category include Data Access Middleware (DAM), Web-based Middleware, Real-Time Middleware, Desktop Middleware, and Specialty Middleware. DAM is used to interact with databases and external resources, providing features such as protection, access control, and Atomicity, Consistency, Isolation, and Durability (ACID) properties. Web-based middleware is designed to make it easy for developers to build websites, e-commerce platforms, or other web-exposed services. Examples of web-based middleware include web application frameworks such as Ruby on Rails, AngularJS, ReactJS, ExpressJS, and Spring Boot. Desktop middleware allows for communication and interaction between local and remote computers, examples include SSH, terminal emulation, and printing services. Real-time middleware specializes in delivering communications with specific latency and

QoS characteristics. They are often used in low-level M2M communications, automotive systems, and IoT systems. Specialty middleware is developed to address specific, unique problems and does not fit into any of the aforementioned categories.

Additionally, an emerging class of middleware uses container technology to assist developers in connecting to resources across multiple clouds.

3.2.1 Message-Oriented Middleware

Remote procedure calls and remote object invocations help to conceal communication in distributed systems, thus enhancing access transparency. However, this mechanism is not always appropriate. In particular, when it cannot be assumed that the receiver is active at the time a request is made, alternative communication services are required. Similarly, the inherent synchronous nature of RPCs, which causes a client to be blocked until its request is processed, may need to be replaced by a different approach such as messaging.

For these purposes, MOM systems, also called message-queuing systems, provide mechanisms to support asynchronous, many-to-many reliable communications [25]. The communication paradigm goes from sender to receiver through a third-party message exchange system that is responsible for collecting, potentially storing, and delivering the message to the final destination. Sender and receiver can be connected to the message-queuing system at different times, and messages can be stored with different Quality of Service (QoS) and retention policies.

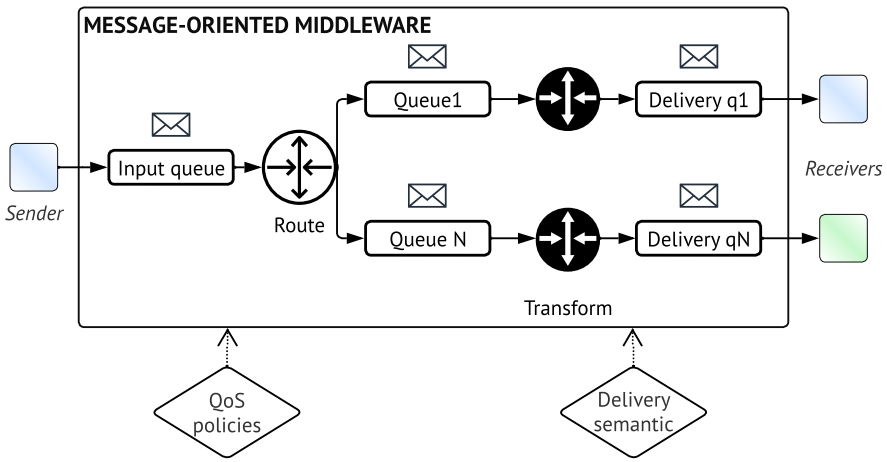


Figure 3.6: Message Oriented Middleware logical architecture overview.

Figure 3.6 illustrates the flow of messages from a sender to receivers. A set of routers and queues ensure that messages are delivered to the correct receiver who has expressed their desire to receive messages (Subscribe) from that channel (Topic). The sender leaves their message in an input queue (Publish) that is managed by the MOM.

There are several semantic delivery policies that can be applied to the messages. The two most commonly used are *at-least-once* and *at-most-once*. *At-least-once* policy guarantees that the message will be delivered to the receiver at least once, even if the delivery process is interrupted. This means that if the message does not reach the receiver, the MOM system will attempt to resend the message. On the other hand, *at-most-once* policy ensures that the message will not be delivered more than once, even if the delivery process fails. This means that if the message does not reach the receiver, the MOM system will not attempt to resend the message. Another semantic policy that can be applied to messages in a MOM system is *exactly-once* delivery. This policy guarantees that the message will be delivered to the receiver exactly once, and no more or less. This means that if the message does not reach the receiver, the MOM system will attempt to resend the message. The latter is particularly useful in cases where data consistency is crucial.

QoS policies are applied to messages to ensure their proper handling and delivery. These policies can include transactional messaging, which ensures ACID properties; durability policies, which control the time a message is retained before it is discarded; presentation and data-processing properties, which dictate how messages are presented and potentially pre-processed before delivery; and resource limits properties, which manage the resources the service can consume [73].

Some examples of MOM include:

- **Apache Kafka** [98]: an open-source, distributed streaming platform that can handle high volumes of data and supports real-time processing of data streams.
- **RabbitMQ** [104]: an open-source message broker software that implements the Advanced Message Queuing Protocol (AMQP) standard.
- **Amazon Simple Queue Service (SQS)** [2]: a fully managed message queuing service provided by AWS that enables you to decouple and scale microservices, distributed systems, and serverless applications.
- **Microsoft Azure Service Bus** [61]: a messaging service provided by Microsoft Azure that enables you to build messaging solutions for applications and services.

- **IBM MQ** [48]: a family of messaging products developed by IBM that provides messaging capabilities for enterprise applications and services.

These are just a few examples, there are many more MOMs available in the market, each with its own set of features and capabilities.

3.3 Containerization

Containers are a way to package software in a consistent and portable way, allowing it to run on any platform, including desktops, traditional IT environments, and the cloud. They use OS virtualization to isolate and control the processes and resources that the software has access. Containers are lightweight, fast, and portable, as they don't require a separate guest OS for each instance, instead they utilize the host OS's features and resources.

Containerization makes it possible to package the developed application and all its dependencies in a single bundle, avoiding environment misconfigurations and missing dependencies. Containers run on top of a container engine (also called container runtime) software that manages the entire lifecycle of hosted containers and isolates them from each other and from the underlying guest OS. The container engine is responsible for configuring all the resources to meet the needs of the application, creating volumes and binding them with the underlying filesystem, creating bridges, VLANs, overlay networks, opening ports on the OS, and connecting them to the correct container that is exposing the service.

Although the concept of containers has existed for decades, the modern era of containers began in 2013 with the introduction of the Docker [27] container engine, which made it more accessible for developers.

Docker is an open-source container engine that enables fast and lightweight containerization capabilities. It is organized as a client-server application, with the core daemon (*dockerd*) acting as a long-running server and a Command Line Interface (CLI) (*docker*) acting as a client. From the CLI, it is possible to pull, run, stop, list, and manage images, templates of applications, and dependencies, similar to a snapshot in a VM environment.

The increasing growth of container-managed software and the wide-spreading of Docker and other container engines pushed the development of Container Orchestrators, software dedicated to maintaining the consistency and the desired QoS in a pool of

managed container engines. Container orchestration tools like Kubernetes [53], Docker Swarm [28], and Mesos [99] come into play, providing a way to manage and automate the deployment, scaling, and management of containerized applications at scale.

Overall, containers are an essential aspect of modern software development and deployment, and their use is likely to continue to grow in popularity. Practical usages in I4.0 of containers and container orchestrators will be explored in Chapter 6.

4 Objectives

Both I4.0 and I5.0 aim to establish value chains spanning heterogeneous industrial domains, enhancing reuse, increasing production flexibility, and exhibiting resiliency in times of disruption.

Sensing technology, big data, and AI have proven viable for automating, managing, and optimizing a wide range of non-industrial processes. Recently, this practice is expanding in the industrial domain [46]. The current manufacturing landscape comprises heterogeneous machines and production facilities capable of autonomous message exchange, generating data at an ever-increasing speed, and all data could provide useful information and could be used proactively for optimized control and business-related purposes [97]. This capability could bring fundamental improvements to the industrial processes in manufacturing, engineering, supply chain, and life cycle management [105].

However, a big obstacle in achieving this goal, especially for SMEs, is the obsolete and rigid separation between technologies that characterize departments involved in product manufacturing (working machines and production lines) and departments committed to managerial tasks [93]. The nature of SMEs frequently suggests that, in contrast to larger businesses, they run with a small staff or a single owner. As many other companies, also SMEs have to decide between creating custom in-house software applications or selecting third-party software solutions. Many times, SMEs are forced to choose the latter due to limited resources. Even though ready-made solutions may be efficient, utilizing them frequently comes with limitations, like the inability to customize or integrate them into the company model and workflow. Third-party solutions might also need tweaks and adjustments, which could raise prices and present new, unexpected difficulties, and fragmentation in protocols and tools.

Moreover, industrial automation has taken a conservative approach, opting for rigid separation between the Operation and Information Technology domains. However, it is becoming obvious that I4.0 and even more I5.0 will have a very significant impact only

with a full convergence of OT/IT that will push for the deep joint exploitation of most recent computing and communication technologies.

Zooming in on the OT layer, a wide range of protocols co-exist which are incompatible with one another, leading to fragmentation that makes it difficult to provide a coherent and consolidated view of the assets and processes.

This work proposes solutions to a set of common OT/IT convergence issues, exploiting the edge computing paradigm to enable fast, customizable, and reliable data sharing on the OT/IT boundary. Our infrastructures rely on a layered middleware approach, where each layer aims to best face and comply with the different requirements of the OT and IT layer, respectively, and is also capable to provide a better synergy of the two layers.

Moreover, in this work, we propose a framework coupled with a protocol suite for QoS-enabled semantic routing in industrial networks. Such a framework fits the characteristics of modern manufacturing environments and permits overcoming the necessity of flexible computation and data aggregation in industrial domains. In particular, the presented solution relies on overlay networking to provide a semantic routing substrate that operates both at the application and network layers. The application layer consists of a MOM and several Application Gateways (AGWs), whereas the network layer combines SDN and In-Network Processing (INP). The overarching ambition is to enable i) loosely coupled, asynchronous communications, ii) fine-grained traffic management, and iii) in-network traffic optimization.

To provide context for these works within the realm of OT/IT integration, this section will outline key objectives that serve as the foundation for all the proposed solutions.

The blurring of the OT/IT strict boundary would open the door to the next-generation I5.0 industrial applications, allowing for fine-grained monitoring and control of individual assets and processes via Digital Twin technology [13, 39]. In the IIoT context, in particular, in production manufacturing plants, cloud/edge computing is considered a relevant opportunity that can significantly contribute to blurring the current separation of OT&IT domains through the design of edge nodes where compute/storage/networking functionalities could converge.

As shown in Figure 4.1, several hierarchical layers of edge nodes with different capabilities can be deployed, distributing the resources along to support the execution of industrial applications and their data storage, thus giving rise to a more fluid model identified by the aforementioned Cloud-to-Thing Continuum. The Automation Pyramid is solely responsible for controlling and monitoring production processes in the machine locale.

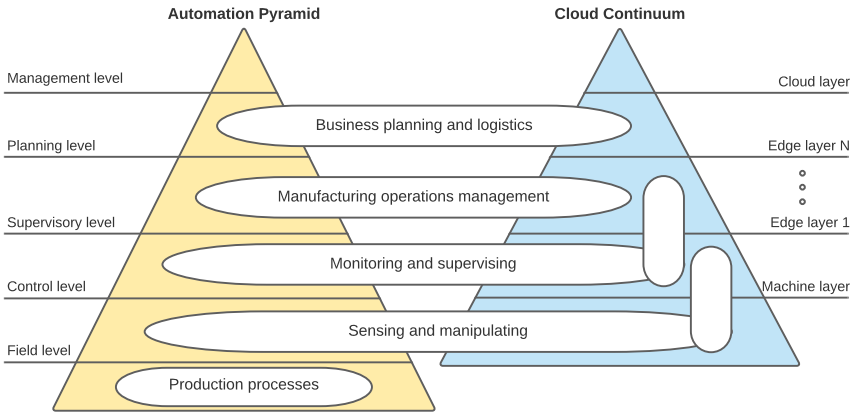


Figure 4.1: Automation pyramid remapped on Cloud continuum representation.

As we move from control to supervisory levels, we aim to gather knowledge and enable remote monitoring, supervision, and predictive maintenance by distributing computation across multiple edge layers close to the production floor. The planning layer can take advantage of data collected from various industrial operations and, in collaboration with management, can be dispersed across edge nodes and cloud layers.

It is clear, however, that just introducing support for the execution of industrial applications at the edge nodes is not sufficient. Seamless integration at all levels of the infrastructure (cloud and edge) is needed to ensure the application QoS specifications.

In this context, edge computing and cloud-continuum play a crucial role in enabling the design and implementation of novel distributed control functions with parts that are hosted on the edge nodes located in the production plant premises and close to the controlled sensors/actuators, primarily to increase reliability and decrease latency [83].

These edge-enhanced cloud architectures provide several benefits compared to a pure data center-based approach: application latency is reduced because of vicinity to end-points; inter-domain traffic is diminished because, for example, Multi-access Edge Computing (MEC) nodes stay in the telco operator network; sensitive information/processing (e.g., of monitoring data related to the manufacturing process that can reveal competitive advantages) can be maintained at industrial edge gateways in the premises of end-points, while global status visibility can be employed, e.g., when needed for global machine learning optimization, by interacting with pure data center-based cloud resources [1].

The following chapter provides an overview of the key technologies currently utilized in the industrial sector to gather, monitor, control, and export data from machinery.

5 Related Works

In this section, we provide a view of the main existent platforms and standards concerning I4.0 machine integration and data management. Without any pretense of being exhaustive, the goal of the section is to facilitate the identification of the critical aspects and stress the differences with respect to the solutions that we will propose in the next Chapter 6.

We present three emerging standards. The first one is the Reference Architectural Model Industrie 4.0 (RAMI 4.0) [86], a three-dimensional spaced model defining how to approach I4.0 issues in a structured manner. Thus, we explore SCADA [95], a technology widely is being used in industrial and in manufacturing plants for over 30 years. Then, we investigate the Open Platform Communications (OPC) protocols [76], which focuses on secure and real-time M2M interoperable communications. Finally, we expose two examples of low-level protocols used in I4.0: Modbus and Profibus.

5.1 Reference Architecture Model Industrie 4.0

In Figure 5.1, RAMI 4.0 is depicted. The model ensures that all the entities involved in the platforms can communicate in a uniform and standardized within a service-oriented architecture. It makes wide use of the *divide and conquer* principle, splitting the complexity into several packages, including data privacy and IT security. The model develops in three distinct yet complimentary dimensions:

- **Hierarchy Levels (IEC62264/IEC61512)**: This axis models the environment surrounding the industry. It spans from the product to the perspective of a connected world, opening the system to other external enterprises, devices, and smart things in general. It is compliant with the 62264 and 61512 IEC standards. The former is an enterprise standard for system integration having its roots in the ANSI/ISA-95 [50] international standard: it helps to define boundaries between the enterprise systems and the control systems. The latter defines reference models for batch control (as it

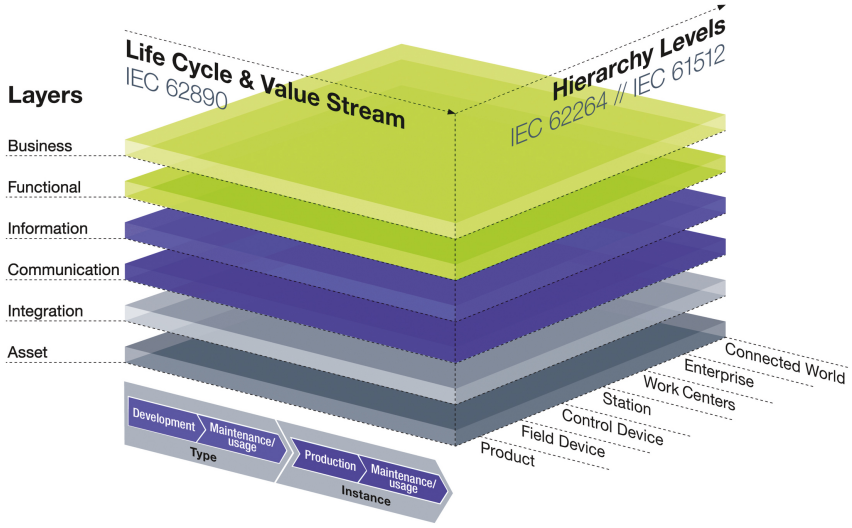


Figure 5.1: Reference Architectural Model Industrie 4.0.

is used in the process industries) and the terminology explaining the relationships between these models and the terms.

- **Life Cycle & Value Stream (IEC62890)**: This axis identifies two main phases. The first one (*Type*) defines the entry of design, development, and test orders carried out up to the first sample and the production of the following prototypes. At this stage, therefore, the type of product, machine, etc. is represented. Only at the end of all tests and the corresponding validations, the type is certified and released for series production. The second phase (*Instance*) identifies the products manufactured according to the general type described in the previous phase. Each product represents an instance of a specific type and has a unique serial number. Then the requests are transferred and delivered to customers. On the customer side, the products are initially just types. They become instances when they are installed in a specific aggregate system.

- **Layers**: This axis models the classical partition of Cyber-Physical Systems (CPS). At the bottom, the physical things in the real world are represented (sensors, actuators, etc.). The physical object interfaces directly with its digital representation (*Integration layer*). The digital representation is then shared with the surrounding entities through the *Communication layer*. From the data, it is now important to extract information, depending on the nature of the asset. Then the systems can be

integrated into a unique *Functional layer*, providing input for the top *Business layer* which enables the development of new business processes and better organization of the plants.

5.2 Supervisory Control and Data Acquisition

SCADA [95] systems are centralized entities devoted to controlling sensors, actuators, and assets, and to triggering corrective actions over them. SCADA systems are often equipped with advanced Graphical User Interface (GUI) that helps operators monitor the plant. The control actions over the plant can either be done by the system or manually driven by an operator. In SCADA systems the controlled plant can be highly sparse in space and the assets need to guarantee a connection with the SCADA servers in order to be monitored and controlled. The assets are often called *field devices* since they act directly on the plant with operations such as opening and closing valves and breakers. The manners in which SCADA applications interact and control the assets are several. Usually, every SCADA-compliant component has to expose an access protocol, such as MODBUS [22] or a carrier such as the MQTT [72] or the AMQP [71] protocols, through which the asset state can be monitored. Moreover, SCADA usually hides the complexity of the underlying protocols, giving the illusion of a unique interface of interactions.

5.3 Open Platform Communications

Ethernet technology has seen steady growth in adoption in the industrial automation sector, leading to overcoming the well-known Fieldbus family of the technology, becoming a *de facto* standard in the OT domain. A multitude of Ethernet variants has been developed and deployed over the years: PROFINET, EtherCAT, Modbus-TCP [105]. However, these technologies are incompatible with one another, leading to fragmentation at the OT layer, making it difficult to provide a coherent and consolidated view of assets and processes.

OPC operates in the context of device inter-communication in a client-server manner. Its main purpose is a unique and standardized way to easily and securely exchange data between different industrial platforms from multiple vendors. The OPC products are more than 35.000 and the specifications help to overcome interoperability issues and to eliminate the need for post-production standardization efforts [77]. Also, there are several OPC specifications. The Classic one [76] derives from the Microsoft Distributed

Component Object Model (COM/DCOM) [60]. The whole set of protocols is broken down into three main categories, according to the type of data that can be accessed: Data Access (DA), Alarms and Events (AE), and Historical Data Access (HDA).

With the advent of the Service-oriented Architecture (SOA) approach [3], in 2008 OPC specifications too evolve into a more complex and powerful architecture: the OPC Unified Architecture (OPC-UA) [75], proposed as a platform-independent standard facilitating interoperability among vendors. Similarly to RAMI 4.0, OPC-UA adopts a multi-layered approach that targets the I4.0 emerging problems.

At first, the standard adhered to a Client/Server paradigm: an OPC UA server provides access to data and functionality structured in an object-oriented information model, while clients interact with the information model via a set of standardized services. Communication takes place in this setting, by following the classical request-response model. This interaction does not suit our application needs as: (i) it introduces strong coupling between different system parts, and (ii) this communication model impedes is not suited to meet the performance required by a hard real-time system.

For this reason, Part 14 of the OPC UA specification defines an extension of OPC UA based on the Publish/Subscribe (Pub/Sub) communication paradigm. In this communication model, an application can play the role of either publisher or subscriber (even both sometimes), where the former is the source of data, while the latter consumes that data.

The communication between publishers and subscribers is message-based: the publisher sends the messages to a message-oriented middleware, without taking into account the possible number of subscribers. Likewise, subscribers show interest in one or more types of data without having any specific information about the publishers. The Pub/Sub model is best suited for applications where location independence and scalability are required.

The MOM is a well-known infrastructure used to send and receive messages in distributed systems, pursued by OPC UA, suited to many use cases in the industrial domain. More specifically, OPC UA Pub/Sub supports two different MOM architectures:

1. **Broker-based**: the core component of the MOM infrastructure is a message broker. Subscribers and publishers use standard messaging protocols like AMQP or MQTT to communicate with the broker [69], with messages being published to specific queues (e.g., topics, nodes) exposed by the broker. The broker is tasked with translating messages from the messaging protocol of the publishers to the messaging protocol of the subscribers.

2. **Broker-less**: in this form the MOM is the network infrastructure, capable of routing datagram-based messages, and subscribers and publishers use a datagram-oriented protocol like User Datagram Protocol (UDP). The broker-less model is intuitively the one embodying the best performance, and therefore best suited to fulfill our system requirements. Addressing the needs of this deployment model, the specification defines a custom UDP-based protocol, called UADP [77] which relies on a multicast scheme for communication among parties.

Focusing on the implementation of the brokerless, a subscriber entity registers to a multicast group represented by an IP address in a special range. Data sent to this address are forwarded to all members of the group. This delegates a large part of the publisher's complexity to the existing IP network infrastructure (router, switches, and so on).

While OPC UA is the most significant IoT protocol proposed to address the fragmentation and communication needs at the OT layer, it does not fully address the needs of the IT layer. In this context, we require solutions and frameworks capable of handling high-throughput data transfer in a reliable and secure manner, while these features are not the primary concerns of OPC UA.

5.4 Modbus data communication protocol

Modbus [22] is an open-source and royalty-free serial communication protocol that was first developed by Modicon in the late 1970s. It allows industrial devices to communicate in a master-slave configuration, enabling the reading and modification of machine registers. The process of reading and writing registers is straightforward, with the master able to connect to multiple slaves and request values or modifications to registers at specific addresses.

In Modbus RTU, data is transmitted through Universal Asynchronous Receiver-Transmitter (UART) technology in a series of bytes at baud rates ranging from 1200 to 115200 bits per second. In a Modbus RTU network, there is only one master and each slave is identified by a unique 8-bit address, responding only if it recognizes its address in the request. Modbus requests consist of various components such as device address, function code, register number, register count, data, and a checksum. The function code can be of read or write type and includes identifiers for the type of register being accessed (e.g. read coil, write multiple holding registers, write multiple coils, etc.).

Modbus TCP allows for the transmission of requests and responses using data packets encapsulated in the Transmission Control Protocol (TCP) protocol. This enables connectivity to machinery via widely used Ethernet networks, eliminating the need for outdated serial cables.

While both Modbus RTU and Modbus TCP allow for easy access to machine registers, they do not provide any access control mechanisms or cryptographic protections for transmitted data. This makes the machinery vulnerable to security risks and unauthorized access.

Given the lack of built-in security features in the Modbus protocol, it is typically used in physically or technologically isolated networks in the OT domain, with access restricted to only devices that have been granted permission. However, this approach can limit the widespread adoption of I4.0, as many machines still rely on the Modbus protocol and companies may not be willing to replace them with newer, more secure equipment.

5.5 Profibus data communication protocol

Profibus is a widely used fieldbus protocol that has gained significant popularity over the past 20 years, with an estimated 66 million devices installed by 2021. It is commonly used to connect Programmable logic controllers (PLCs), sensors, actuators, and other industrial equipment. Profibus is robust and can handle a large amount of data exchanges, including high-speed and real-time control messages. It also offers a more flexible and modular solution compared to other protocols like Modbus, allowing for proactive management of industrial machinery without service disruption.

Profibus is divided into two main categories: Profibus-DP (Decentralized Peripherals) and Profibus-PA (Process Automation).

Profibus-DP is a communication technology that allows for the unification of different transmission technologies, such as wired, optical, and wireless, and presents a consistent view of devices and underlying protocols to the upper layers of the system. It is the core of the Profibus solution and enables the use of the same communication protocol across different devices and machinery that can all be connected to a single cable. Data rates from 9.6 Kbps to 12 Mbps are supported.

Profibus-PA is used for communication between devices in critical process control systems, it allows for the use of the same two cables for both power and communication, making it suitable for use in hazardous environments. It also has explosion protection

capabilities and special hardware for specific applications. However, the protocol and data management remain the same but the bandwidth is limited to 31.25 kbps.

Lastly, PROFIBUS Security Suite is a comprehensive set of security features that are designed to protect PROFIBUS networks from cyber threats and unauthorized access. The suite aims to ensure the availability, robustness, integrity, authenticity, authorization, and confidentiality of data. It can be used in three main security classes, each of which focuses on a specific subset of properties. Class 1 covers only robustness, Class 2 covers integrity and authenticity, and Class 3 includes all mechanisms to ensure all the properties.

6 Architectures for I4.0 data gathering and management

In this chapter, we present a comprehensive overview of four architectures that we have developed for data gathering and management in Industry 4.0 environments. We will cover different aspects of data gathering and management: cloud-enabled smart data collection, QoS-enabled semantic routing, low-latency m2m communication support, and serverless processing at the edge. Additionally, we will present infrastructures and practical examples that demonstrate how these architectures can be functional in real-world Industry 4.0 scenarios. This will include experiments that test the effectiveness and functionality of the proposed solutions. The architectures will be evaluated based on collected metrics generated with the usage of simulated physical assets and when meaningful, fault tests.

6.1 Reference Scenario

The present work proposes advancements to a research line undertaken in collaboration with many manufacturing companies based in the "Packaging Valley" district located in Emilia Romagna, Italy [23]. Grounding on requirements elicited from the district manufacturing companies, we aim to develop tools to concretely support manufacturing companies in the transition to Industry 4.0.

During the first steps of collaboration, we drafted a guideline for the implementation of the convergence of the factory OT and the IT layers as a way to enable the Industry 4.0 transition[5]. In the following scientific contributions, we discussed the main technological requirements on the basis of a smart factory scenario and a first draft architectural view of a data gathering and integration platform (SIRDAM4.0) [10, 11, 24]. The platform aims to address some of the challenges raised by IIRA and OPC-UA. At the current stage, the platform provisions data gathering and structuring at the OT layer, and transmission of such data to upper layers. In particular, it implements a communication pattern between

the company departments that is based on asynchronous messages exchange carried out by a MOM that adopts the publish-subscribe model. The MOM enforces IT/OT integration at the data level that acts as a data conveyor from bottom to top layers, but at the moment inhibits the flow of control commands from top layers downwards, so to avoid by design the possible threats due to exposing the OT to the direct control of an external entity.

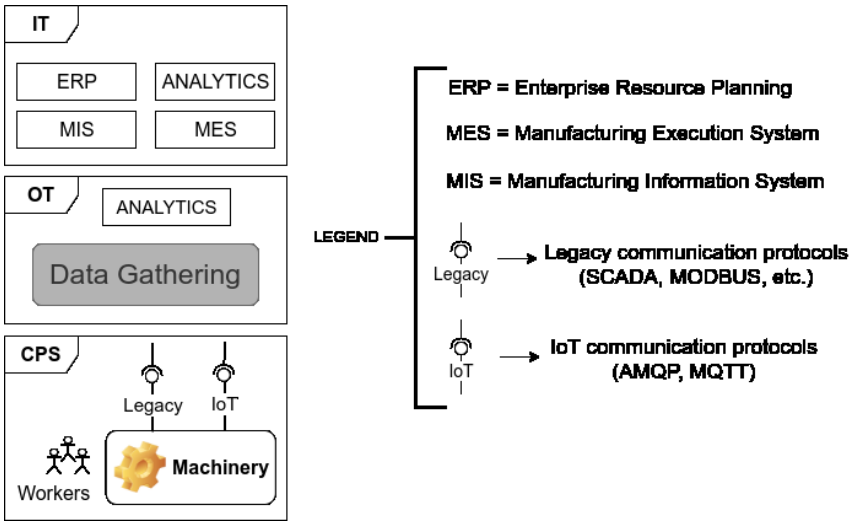


Figure 6.1: Technological layers in a manufacturing industry.

In Figure 6.1, we have depicted a typical representation of the technological layers of a manufacturing factory. At the bottom layer, machines and workers strictly interact for production purposes. Production data generated by machines are used by workers to correctly operate the production line. Here, relevant data streams are characterized by great speed and variety, since a high volume of data is generated by the many working machines. Furthermore, a strict data access mechanism is required to prevent malicious intruders from stealing confidential information or injecting data that could eventually bring the operational layer to an unsafe state.

Data Gathering in the OT layer represents the process of real-time collecting data produced by machines at the CPS layer. The OT layer consumes data gathered for operational purposes transferring them to upper layers for business purposes. Data flows generated by this function are usually filtered before reaching the IT layer (not all production data are useful at upper layers). Once here, they feed a number of IT tools to help business experts to control production and coordinate other managerial tasks. The *Data Gathering* system shall act as a *separation layer* for the underlying layer so as not to expose machine-

production data but, at the same time, to shield the shop floor from any attempt to make direct access to machines.

Let us now consider a typical industrial scenario involving many actors in the manufacturing sector, and focus our attention on the data gathering and management issues that this scenario will raise. In the prospected scenario, two competing manufacturing companies *Company A* and *Company B* run production lines in Imola and Milan respectively. Both company production lines are operated by machinery manufactured by a Machine Manufacturer Company (MMC). Let us also assume that Company A needs to increase production in order to catch a new business opportunity, so as to establish a new production line in Bologna. Due to this expansion, both *Company A* premises will deploy a LOCAL IT layer equipped with some data filtering and aggregation tools, while only Imola premise will also deploy a CENTRAL IT layer responsible for aggregating data coming from the two LOCAL IT layers before feeding them to business-level ERP software. Finally, we can assume that other companies deploying MMC machinery may join the scenario (e.g., Company N).

To defend and grow its market share, MMC aims at continuously improving its product so as to take advantage of gaining information on production data of machines running at customer premises: such data, if timely analyzed and processed, will help MMC to spot machine misbehavior and detect potential causes (to cite a few: misconfiguration of machine parameters, machine design defects, assembly defects). Unfortunately, disclosing customers' production data to MMC poses a huge security problem. Customers, for obvious reasons, refrain from disclosing anyone (not even the machine manufacturer) their data. Yet, with the opportunity of receiving by MMC a timed and more effective technical support, customers may be willing to disclose agreed portions of their production data. MMC by accessing these data could promptly detect and diagnose run-time anomalies, suggest more effective machine parameters' configuration/setting, advise machine part replacement, etc.

This research aims to enable advanced data gathering scenarios, capable of transferring data from the machine level to the cloud, respecting secrecy, confidentiality, and QoS requirements. Technologies such as MOM, SDN, Containerization, Time-Sensitive Networking (TSN), INP, and FaaS must cooperate to achieve the desired results with the right degree of elasticity, both for OT and IT requirements.

To enforce the described scenario and meet the need of all stakeholders in terms of availability and accessibility to relevant data, we considered the following requirements:

R1. *Timely access to data*. At the shop floor level, data must be made timely available and accessible. The purpose is twofold: complying with the near-real-time constraints of the targeted production process and promptly undertaking countermeasures in case of potential hazards.

R2. *Handling of heavy workloads*. Depending on the market demand, to deal with requests for increasing production, new machines might have to be added to production lines. The data gathering system must be able to absorb spikes in data generation and guarantee a timely data delivery also in case of heavy workloads.

R3. *Controlled access to data*. Access to data needs to be regulated. Data must be carefully partitioned and made available to the intended recipient (be it the shop floor, the company business department, or the machine manufacturer) only for specific use.

R4. *Tolerance to faults*. In order to maximize the company profit, close to 100% machines operational continuity has to be granted. Faults occurring at any level, and in particular, at the OT layer, have to be solved at a time that is compatible with the criticality of the data that could potentially get compromised by a shutdown.

The definition of frameworks for gathering data at the OT layer and making them safely accessible to stakeholders is an important step that poses the basis for OT/IT convergence in industrial settings.

In the following Subsection 6.2, we will present the design and implementation of a platform that collects OT data and ensures real-time access to data consumers, addressing proposed requirements. In Subsection 6.3, we will delve deeper into how SDN and semantic routing can improve performance and reduce the workload on the nodes of the architecture. Additionally, in Subsection 6.4 we will a case study, which focuses on low-latency M2M communication support. Finally, in Subsection 6.5, we will examine how serverless processing can make a difference by saving resources on the nodes involved in the data analysis and transfer.

6.2 Cloud-enabled Smart Data Collection

This section presents an architectural model for collecting and analyzing data from manufacturing machines. The experience in the field allows us to identify and address the challenges associated with handling large amounts of data from complex machines. The proposed platform aims to be an approach that is independent of the underlying physical level, enabling SMEs with legacy machines to embrace the fourth industrial revolution. The architecture is designed to meet the specific needs of a single enterprise while also being adaptable to the needs of many companies undergoing I4.0 transformation. The resulting platform is able to process data in near real-time while maintaining efficiency and security, avoiding the need to transmit all data over external networks for scalability and data protection. The design is modular and independent of most technologies used, a key feature for I4.0 given the wide diversity of technologies used such as network protocols, storage and search platforms, and providers. Additionally, the architecture includes several layers to ensure security and reliability throughout the system.

6.2.1 SIRDAM4.0 Architecture

We report the first architectural model, also presented in [10, 11] that addresses the IT/OT convergence issues. Furthermore, we discuss some relevant details of the software prototype of the data gathering platform we implemented and employed during the tests.

Experience gained in past collaborations with important manufacturing enterprises and corroborated by an in-depth study of the main revolutionary industrial specifications, helped us draft some guidelines [5] that have driven the design and implementation of Support Infrastructure for Reliable Data Acquisition and Management in Industry 4.0 (SIRDAM4.0).

The platform supplies the following services:

- gathering of data produced by manufacturing machines;
- long-term storage, fast processing, and user-friendly presentation of such data at the OT layer;
- secure mirroring of operational data flows for use by IT departments;
- secure and selective provision of operational data to third party stakeholders.

SIRDAM4.0 design principles inspire mainstream I4.0 standardization activities such as RAMI 4.0, IIRA, and OPC UA. A feature SIRDAM4.0 borrowed from all specifications is secure and selective access to information guaranteed to both the stakeholders acting inside a modern I4.0 manufacturing company and those operating outside (e.g., the machine vendor).

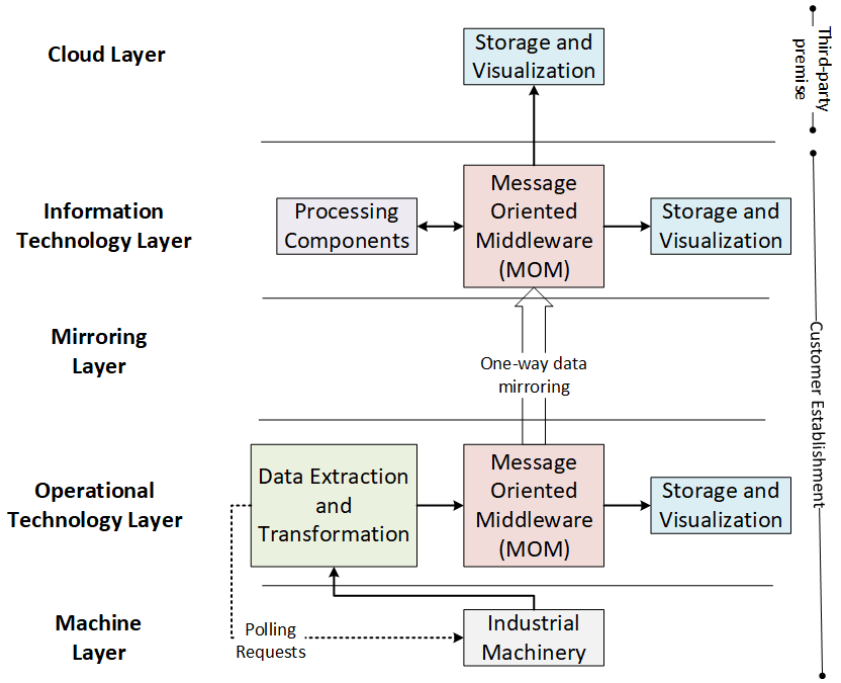


Figure 6.2: SIRDAM4.0 overall architecture schema.

SIRDAM4.0 architecture depicted in Figure 6.2 reflects the physical separation of OT and IT layers enforced in most production sites and addresses their *integration at data level*.

Despite SIRDAM4.0 does not exhibit strict compliance for what concerns the implementation of entities and classes, its architecture largely adheres to OPC UA specifications. Indeed, SIRDAM4.0 follows the interoperability principle reiterated in part 1 and in part 14 of the OPC UA specification, which advises the use of Pub/Sub communication pattern. Along with that goal, we placed a message-oriented middleware (MOM) between machines and the tools charged with data elaboration and storage tasks. We also decoupled the data type of the machine registers from the specific communication protocol by "flattening" the data type according to a unifying scheme, following the prescription reported in part 14 of the OPC UA specification.

Stemming from the fact that most SMEs have little or no economic resources to undertake the I4.0 transformation, SIRDAM4.0 also supports the gathering of data produced by existing legacy assets and its integration with modern MOMs and IoT protocols. Specifically, protocols from the SCADA family, still widely used in manufacturing realities [45, 62, 89], are fully supported by the platform.

The data streams generated at **Machine layer** are characterized by high speed, large varieties, and big volumes, due to the number of different machines operating on the shop floor. As often reiterated in both IIRA and RAMI 4.0 standards, companies need proper solutions to manage data in a secure and reliable way, avoiding damages to surrounding people and to the machines themselves during remote operations. We take the OPC UA advice (part 2) to restrict access to this layer in order to achieve the right level of security and safety. As a mandatory practice of I4.0 specifications, the **OT layer** needs to provision very low data latency, good bandwidth, enhanced security mechanisms, and resilience. In this layer, we placed a component to collect data from sources (*Data Extraction and Transformation*) and a *MOM* capable of delivering such data to consumers in a Pub-Sub fashion, and of guaranteeing a data latency compatible with near-real-time constraints. Keeping latency low allows the software of this layer (*Storage and Visualization*) to align with the update frequencies of the machines and carry out fast data processing.

On top of the OT layer, the **Mirror layer** offers support to implement a fine-grained control of the convergence. In line with the vision of the Chinese IMSA standard that recommends flexibility of management policy in relation to the requirements of the considered industrial application domain, different data storing policies (what, when, and how data have to be exchanged between OT and IT layers) can be enforced at this layer. The Mirror layer may also serve as a backup of OT-generated data, thus ensuring the whole platform a good degree of robustness with respect to potential faults of the Machine layer.

Technical and maintenance departments, as well as management and logistics, are consumers of information: the formers consume raw information, while the latter are interested in aggregated information. At **IT Layer**, multiple stakeholders need to consume different portions of the available data set that is fed with data coming from underlying layers. Then, we decided to replicate here the OT layer component scheme, which provides for a MOM distributing data according to Pub-Sub, and a set of tools (to be used by data consumers) devoted to the processing, storage, and visualization of data. The actual distinction between OT and IT at the design level lies in the relaxation of the requirements

at the IT level, where no work machines operate and the risk of jeopardizing workers' safety is much lower.

Finally, SIRDAM4.0 opens to the involvement of third-party stakeholders (TPS), i.e., potential partners, sharing common goals with the company, that could generate value from production data. Being TPS outside of the manufacturing establishment, in the architectural view we collocated them in the **Cloud Layer**. This tier collects selected data coming from production sites and runs analytics over it. As a data consumer, the *Storage and Visualization component* is allowed to subscribe only to specific topics published by the IT layer MOM. This mechanism aims at avoiding any leakage of private and confidential company data that does not serve TPS purposes.

To conclude the discussion on architectural aspects, we would like to remark some of the platform features that are also strongly accounted for in the I4.0 vision of RAMI 4.0 standard:

- Isolation of the Machine layer;
- High availability of production data and real-time data processing at OT layer;
- Secure and selective access to production data by process stakeholders, be they company IT departments or external partners.

6.2.2 Platform Implementation

This section provides some implementation details of the software prototype of the SIRDAM4.0 platform. Our prototype makes use of state-of-art and open software tools with the goal not to build an enterprise commercial product, but of implementing a proof-of-concept for all our claims and proving that the first step towards I4.0 transition can also be taken by SMEs while keeping the transition cost low.

After surveying a list of candidate software tools that might fit our needs, we decided to use Apache Kafka Broker to implement the MOM component of our architecture and some event streaming tools offered by the Confluent suite [32]. A Kafka Broker instance can handle a data ingestion rate as high as 420K messages/second [47] while guaranteeing an almost constant performance in terms of end-to-end message delay. Should the user need to handle a higher throughput, multiple Kafka Broker instances can be clustered and run as a more powerful broker. Clustered brokers also implement a data replication

scheme that provides the system with high system resiliency against sudden and unexpected software faults.

Machine and OT layer

Figure 6.3 depicts a schematic view of machinery populating the *Machine layer* and the software components implementing the functionality of the OT layer discussed in the previous section. Since the fourth industrial revolution encompasses the interconnection of the machines, achieved during the third industrial revolution, the transition requires incorporating all legacy patterns and communication protocols. We made the choice of supporting SCADA protocols, which in most cases are hard-coded in the firmware of manufacturing machines. Although we tested MODBUS TCP, other protocols such as Profibus, CANOpen, and DeviceNet must be supported as well. Of course, modern IIoT protocols like Message Queue Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP) should be supported too, as explicitly advised in part 14 of OPC UA.

OT layer interfaces with the underlying work machines. In the following, we provide a description of the software tools implementing the functionality offered by each architecture component that populates this layer.

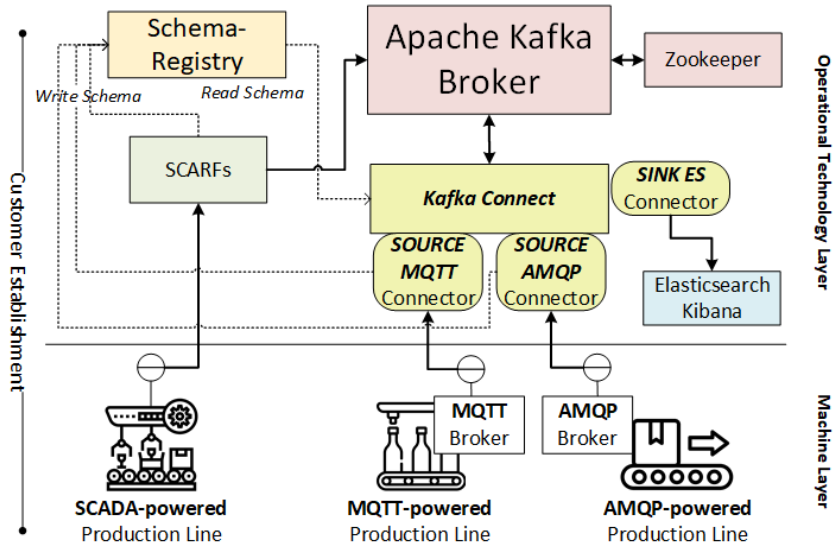


Figure 6.3: Operation Technology Layer schematic.

Data Extraction and Transformation

Software components devoted to provisioning the service of data extraction and transformation are listed below:

- SchemaRegistry. OPC UA specification, in parts 3 and 5, addresses the standardization of components (objects and servers) and registries inside shop floors via "AddressSpace" and "Information Model". In compliance with the OPC UA specification, SchemaRegistry implements the repository of the schema describing the format of production data, useful for carrying out operations on production data. As suggested in Figure 6.3, schema can be uploaded to or retrieved from SchemaRegistry through simple read and write operations.

- SCADA Reader and Forwarder (SCARF). As mentioned before, many machines are powered with SCADA capabilities, i.e., can interoperate via a protocol of the SCADA family. The SCARF component, implemented on top of the pymodbus tool v2.1.0, interfaces with SCADA-powered machines and carries out the following tasks: data retrieval, data validation, data serialization, and data forwarding to Kafka Broker. Most SCADA protocols do not provide spontaneous sending of their production data; SCARF can poll machine registry at predefined and configurable time intervals. As a general principle, a SCARF instance is instructed to read from a machine registry. In a production chain, usually populated by many work machines producing a non-negligible load of big data, we carefully adopted and tailored a lightweight format for data compression, namely AVRO, a data serialization framework arranging information in a compact binary format. SchemaRegistry stores the AVRO schemes for serialization and deserialization purposes. Eventually, SCARF instances send serialized data to the Kafka broker.

- IoT connectors. Different than the SCADA-powered machines, which require a polling mechanism to implement data gathering, IoT-powered machines interface to message brokers that implement the Pub-Sub mechanism. To gather data produced by such machines, a potential consumer just subscribes to machine topics and gets data while they are published by machines. We decided to support the interaction and communication with machines powered by MQTT and AMQP messaging protocols. Specifically, in the Machine layer, MQTT and AMQP messages are managed by *Eclipse Mosquito* [31] and *RabbitMQ* [104] message brokers respectively.

In the OT layer, *MQTT connector* and *AMQP connector* of the Confluent suite act as subscribers of messages published by production machines. The Kafka Connect components are open-source Kafka plugins containing converters and connectors to interface the Kafka broker with external platforms, both source and destination of data.

Message Oriented Middleware

The *Apache Kafka Broker* implements the MOM component of the architecture. It is a typical message broker that supports the Pub-Sub mechanism for distributing messages among participants. In this layer, data producers (i.e., the publishers) are SCARFs, MQTT, and AMQP Brokers via the respective connectors, while Elasticsearch is the only data consumer (subscriber). We would like to stress that Kafka Broker represents the software component that physically "shields" the OT layer from the overlying layers, but at the same time, it is where the first step of IT/OT convergence is taken. It represents a gate through which production data can flow upwards to reach stakeholders (both internal and TPS). To enforce security, no message originated by the IT layer is allowed to transit to the OT layer: all stakeholders of overlying layers can just act as subscribers of OT layer topics.

Finally, Kafka Broker calls upon Zookeeper as a coordinating central point for retrieving services such as naming and distributed synchronization.

Storage and Visualization

We selected *Elasticsearch* as a long-standing storage tool and *Kibana* for presenting data to the customers. We chose the document-oriented Elasticsearch storage tool for its speed, scalability, and search options features. Kibana guarantees a high customization level, which allowed us to define a dashboard for each plant (and a view for every machine inside it), for its capabilities of defining users, roles, access to data, and for the nice rendering of interactive graphs, tables, and pie charts. The *Sink ES connector* is a subscriber of Kafka Broker topics that consumes the data, deserializes them via a schema retrieved from Schema Registry, applies any required transformation, and delivers them to Elasticsearch for storage.

Mirror, IT, and Cloud Layer

The company can deploy resources for mirroring wherever there is hardware availability. Therefore, the *Mirroring layer* could potentially collapse inside either IT or OT. With the conceptual division proposed in our architecture, we want to remark that the logic of mirroring is customizable and under the control of the company. This feature allows fine-grained control of the company, in defining policies for configuration of the system and in data protection. The Kafka MirrorMaker is a stand-alone component that copies a subset of topics from OT to IT layer (see Figure 6.4). In practice, MirrorMaker places a consumer at the source broker (the OT layer) and a producer at the destination broker (the IT layer). The company can also choose a specific distribution and replication level for the MirrorMaker component. By mirroring the OT Kafka broker, we make the whole system gain availability, avoiding a single point of failure, and enforce the separation principle between OT and IT layers.

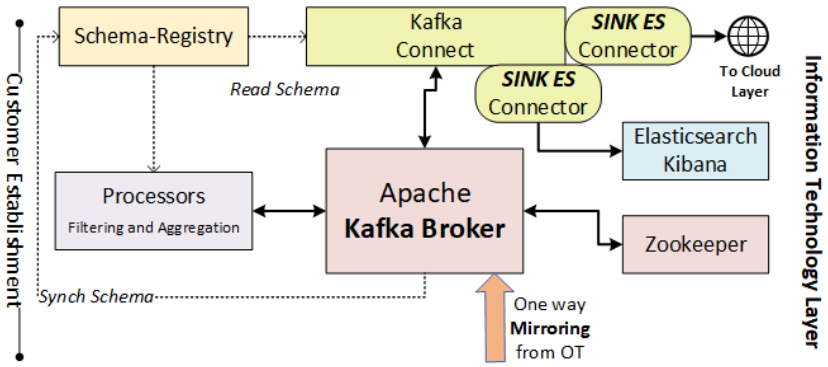


Figure 6.4: Information Technology Layer schematic.

The *IT layer* is populated with the same software modules as the OT layer, with the exception of SCARFs, given that the IT layer does not have a connection with physical machines. As mentioned in previous sections, data in the IT layer are a full copy or a subset of data in the OT, depending on the MirrorMaker configuration policy. Due to the absence of humans in contact with production machines, the time constraints are more relaxed with respect to the OT layer: the topics update frequency is lower and it is possible to deploy specific data analytic logic and advanced software modules. As shown in Figure 6.4, we added data processing modules using Kafka Processors, components able to aggregate and transform data before sending information to the Kafka Broker. The company can add custom business logic (such as lambda functions) for data manipulation.

We have placed two connectors consuming messages from the Kafka Broker: one brings data to Elasticsearch storage, the other forwards information to the Cloud layer.

The main objective of the *Cloud layer* is the aggregation and presentation of data coming from production sites. In order to reach this layer, data forwarded by the IT layer will have to use a secure communication channel, as they have to traverse the public Internet. State of art solutions guaranteeing integrity and confidentiality (SSL/TLS) will be used to enforce data security. In this layer, TPS will use software that fits their business needs. Recalling the example scenario described in Section 6.1, machine vendors can deploy diagnostic, predictive maintenance, and other after-sales software.

6.2.3 Experiments

In this section, we report the result of a thorough assessment of the platform with respect to the performance indicators previously discussed in Section 6.1: functionality, timely access, scalability, controlled data access, and resilience. We made use of virtualization techniques to implement the platform software prototype: specifically, virtual machines (VMs) realized the physical separation between all layers (OT, IT, View). To achieve system scalability and resilience, as well as flexibility when adding new platform features, we adopted the *microservices* programming paradigm. The microservice-based approach allowed us to develop a horizontally scalable and robust system to easily adapt to the dynamics of the input workload and to tolerate potential run-time faults.

For the message latency tests, we address a typical manufacturing scenario with near-real-time constraints for what concerns the availability at the IT layer of information collected by the work machines, which is the most common use case in medium and small manufacturing realities. In such contexts, as shown by experiments, the message delay shall never overcome 100ms, which is compatible with the classical timing of near-real-time systems [19, 92].

We consider the case of a typical SME owning two production sites (plants) located in two different cities, say Imola and Bologna, that belong to the same productive district. The SME intends to implement scalable and robust data gathering on both premises. We will prove that this goal is easily achieved by means of our platform.

We arranged a VPN service to connect the two sites and isolate the machines from each other inside each site. We used the OpenStack infrastructure manager to deploy VM instances. We containerized all the components discussed in Section 6.2.2 using the

Docker tool and called upon Kubernetes and Rancher to orchestrate and control the services. Docker containers run inside VM instances in their turn.

Bologna Plant	*Imola Plant*	*Remote Cloud*
	Central IT layer VM - Zookeeper - Schema-registry - Kafka Broker - Processors - Kafka Connect - Elasticsearch - Kibana	**View Layer VM** - Elasticsearch - Kibana
Local IT layer VM - Zookeeper - Schema-registry - Kafka Broker - Processors - Kafka Connect	**Local IT layer VM** - Zookeeper - Schema-registry - Kafka Broker - Processors - Kafka Connect	
Local OT layer VM - MODBUS Server - SCARF - Zookeeper - Schema-registry - Kafka Broker - Kafka Connect - Elasticsearch - Kibana - MirrorMaker	**Local OT layer VM** - MODBUS Server - SCARF - Zookeeper - Schema-registry - Kafka Broker - Kafka Connect - Elasticsearch - Kibana - MirrorMaker	

Table 6.1: Testbed deployment: Locations, VMs, and Kubernetes Pods

Table 6.1 depicts all VM instances, microservices running within each VM, and the physical location of VMs.

Each production site implements OT, Mirroring, and Local IT layers. We collapsed OT and Mirroring layers in one VM for the sake of simplicity and because usually, in a real deployment, these entities are on the same local network. In the case of Imola, the Central IT Layer is also deployed. Central IT gathers and aggregates data coming from Imola and Bologna Local IT layers, emulating a real-world deployment where the headquarters also

act as data collection points. As mentioned in Section 6.1, this layer includes IT software serving all company departments, such as those for managing staff or project cycles.

Almost all VMs are equipped with Ubuntu 18.04, 16 GB of RAM, 100 GB of HD, 8 logical cores, and a connection of up to 1GBps. Only the VMs emulating the IT layer are provided with the same operating system, HD size, and connection rate, but are assigned 8 GB of RAM and 6 logical CPUs.

For the simulation of the physical asset, we used ad-hoc MODBUS servers to set the values of the machine registers. The snippet of code in Listing 6.1 shows the mapping, used by the SCARF, to extract data from the machines, before sending their serialization to the Message Broker.

```
{
   "LOCK_VALV_1" : {
      "register_type" : "single_register",
      "register_index" : 4,
      "unit" : "seconds"
},
   "INJECTOR_LVL_1" : {
      "register_type" : "single_register",
      "register_index" : 17,
      "unit" : "mm"
},
   ...
}
```

Listing 6.1: JSON mapping SCARF example.

We want to show an example of our mapping between the physical register from MODBUS to a higher-level value, such as a float. The serialization follows the AVRO key-value schema presented in the next Listing 6.2.

Finally, we remark that for each test discussed below, we reported statistical values obtained from multiple reiterations of the experiment.

```
{
   "name" : "INJECTOR_LVL_1",
   "type" : {
      "type" : "record",
      "name" : "lvl_injector_1_oil",
      "fields" : [
         {
            "name" : "value",
            "type" : "float"
         },
         {
            "name" : "unit",
            "type" : "string"
         },
         {
            "name" : "description",
            "type" : "string"
         }
      ]
   }
}, ...
```

Listing 6.2: AVRO key-value schema example.

Delay assessment tests

In our initial test suite, we implemented SCARFs to add a timestamp to each message read from MODBUS, and Elasticsearch performs the same function for the documents coming from the connector. Thus, each message includes a *creation timestamp* from the first element of the chain and a *storage-inserted timestamp* from the last one. In addition to the vast amount of useful data for queries, tracking multiple timestamps enables us to model the delay across the entire infrastructure.

We conducted experiments using 1, 5, 10, and 12 simulated physical assets, each one associated with 7 SCARFs (one for each simulated functional unit). Based on our experience, the number of simulated machines is representative of what is typically found in a medium-sized company's assembly line (10-12 machines that update 6 pools of registers every 10 seconds and one pool every 5 seconds). As a result, each simulated machine

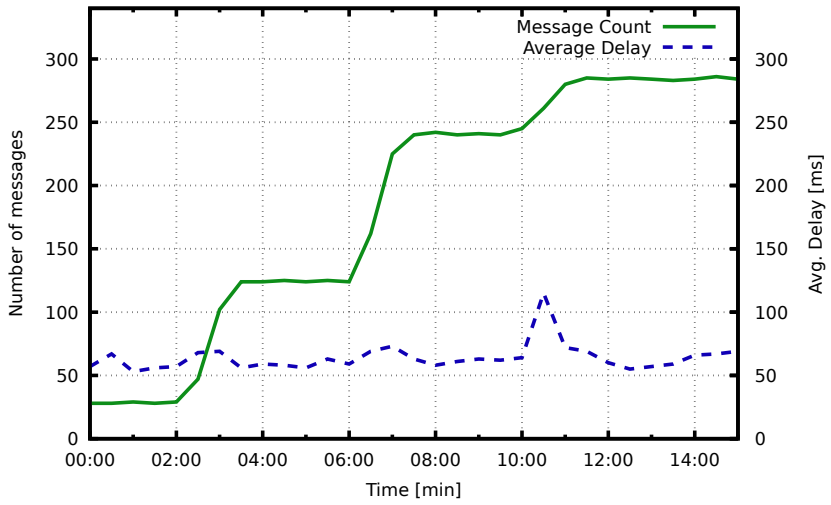


Figure 6.5: Delay test using 1-5-10-12 simulated physical assets.

generates 24 messages every 30 seconds. Figure 6.5 illustrates the delays in the View layer with respect to the gathering of messages in the OT Layer.

The graph is intended to demonstrate how the number of messages increases as the number of machines attached to the system increases, while the delay remains constant, thereby proving the scalability of the system in the face of a growing data volume. The right vertical axis shows the increasing number of messages exchanged, corresponding to an adaptation of the architecture to more resources. Each step of the green line corresponds to a test with an increasing number of simulated machines and the subsequent reading of many messages: starting with 24 messages and one machine, then with 120 messages and 5 machines, then with 240 messages and 10 machines and finally with 288 messages and 12 machines. The left vertical axis and the corresponding red line indicate the packet delay in tracing the most critical and long path inside the platform, from the production machine to their effective storing in the view layer. We take measurements by reading the number of messages from Kibana every 30 seconds. The horizontal axis shows a time period of 15 minutes.

In a second test, we further stress the platform by performing the same increase in the number of messages and containers used to collect them, but this time with 2, 10, 20, and 24 simulated physical assets, so in the last step, the platform was processing 600 messages, as seen in Figure 6.6.

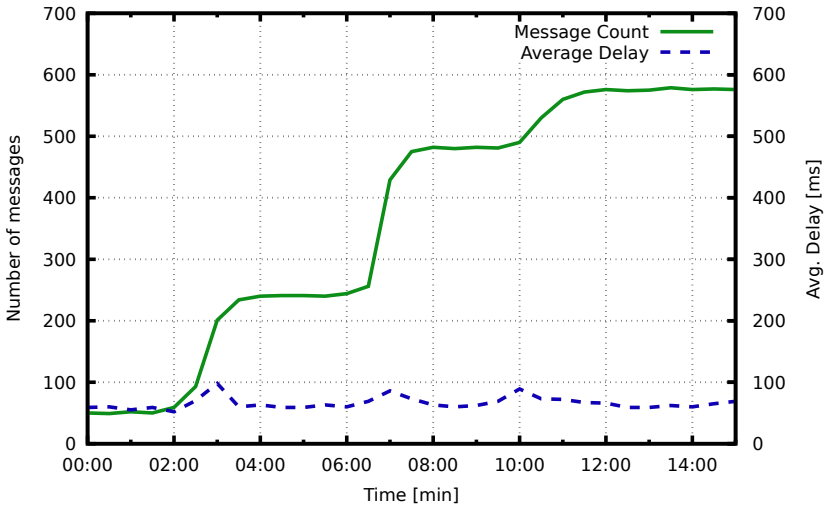


Figure 6.6: Delay test using 2-10-20-24 simulated physical assets.

The results from these tests demonstrate the feasibility of the platform for scenarios requiring near real-time requirements. The average delay is 70 milliseconds to make the data available from the machines to the last View layer. We can see small delay peaks during an increase in message throughput due to the settling of components, but the platform results in general are scalable, showing almost constant delays up to about 1 200 messages per minute processed.

OT layer stress test

The stress test aims at assessing the impact of a sudden increase in message load on the platform performance. In real situations, the number of messages to handle can raise due to an increase in the rate at which machine registries are polled or when new machines are deployed on the shop floor. We will show that, in spite of a substantial increase in the number of messages, the delivery of messages is not affected, thus guaranteeing the message consumer good performances in terms of delivery time, so as to comply with the manufacturing sector requirements. Results show that the platform meets the requirements $R1$ and $R2$ set out in Section 6.1.

The target of the first test is the machine/OT stack. We considered scenarios involving both legacy equipment (SCADA-powered machines) and modern ones (IIoT-powered machines). This specific test addresses just the OT layer of one stack, so we arbitrarily decided to carry it out on the Bologna plant. For tests on SCADA-powered machines,

we used ad-hoc MODBUS servers emulating the machine registers as data generators, so as to have the possibility to arbitrarily introduce load spikes and sudden increases. It is worth mentioning that the SCARF component deployed at the OT layer supports several communication protocols of the MODBUS family (TCP, UDP, Serial ASCII, Serial RTU, Serial Binary). For the purpose of the experiment, the TCP one is used since machines are equipped with an ethernet interface. This choice will not affect the generality of the experiment outcome because the SCARF component behaves as an adapter or a separation layer between the underlying protocols and the data format our platform deals with. In case of changing the communication protocol used by pymodbus, we just need to change the MODBUS client communication type in the SCARF component. Furthermore, in our tests all the virtualized components are running on machines connected via ethernet TCP/IP protocols, so changing the communication protocol of pymodbus actually does not affect the "real" underlying communication substrate.

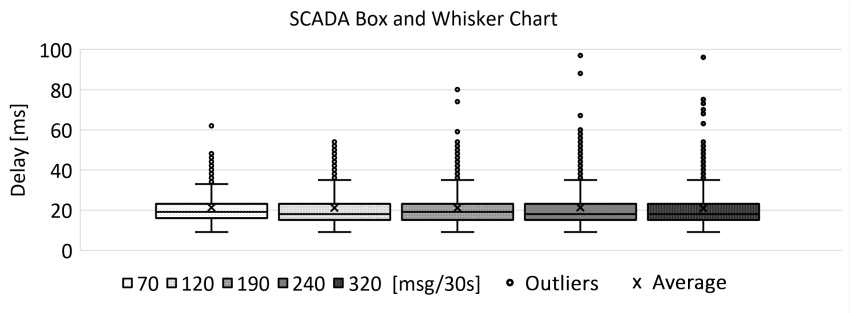


Figure 6.7: SCADA OT delay test.

We define *message delay* as the time lapse between the time when a data sample is read from a machine register and the time when the same data is stored to the Kafka broker deployed at the OT layer, i.e., when it becomes available to consumers. We were able to track the delay trend by adding metadata to this sample, a creation timestamp, and a storage timestamp respectively. We then investigated the capability of the platform to handle the message loads produced by different numbers of machines on the shop floor. We assume that every emulated machine is equipped with 7 functional units, each exhibiting exactly one register. Each machine is configured to produce 24 messages every 30 seconds. We observed the performance of the platform handling messages produced by 3, 5, 8, 10, and 13 machines, which corresponds to loads of 70, 120, 190, 240, and 320 msg/30secs respectively. Each experiment assessing the performance of a given load lasted 30 minutes and was repeated 10 times. Figure 6.7 reports the statistics of each experiment results

in the form of Box and whisker plots. By looking at the plots, it appears very clear that, despite the increase in load, the average message delay is stably set around 20ms. Also, data is very much condensed as evidenced by the short distance between the first and the third quartile, and by the narrow extension of the whiskers. We can conclude that the system is fairly capable of absorbing load fluctuations by providing a constant performance that fits the real-time requirements of OT environments.

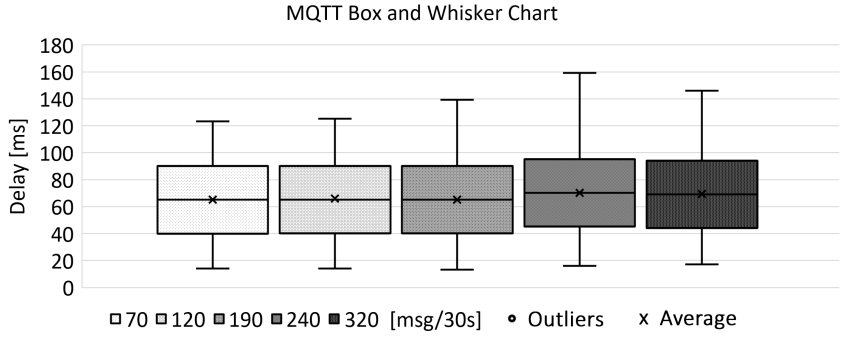


Figure 6.8: MQTT delay test.

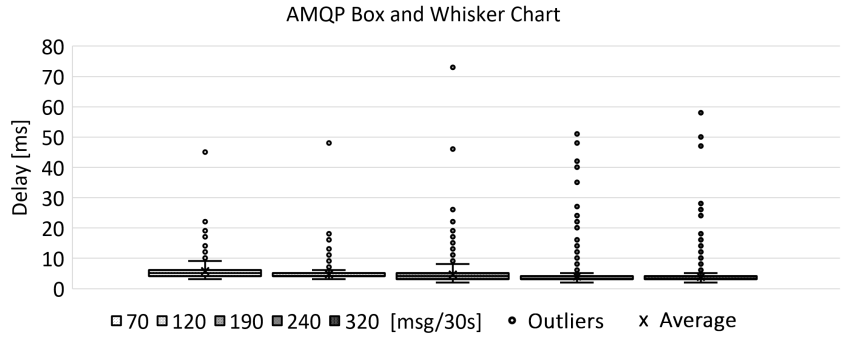


Figure 6.9: AMQP delay test.

The same experiment on data delay was conducted on IoT-powered machines equipped with MQTT and AMQP brokers respectively. We remind that, in this case, we used ad-hoc Kafka connectors to extract data from MQTT/AMQP brokers and send it to the Kafka broker, which will eventually publish it. We will focus on the time span from when a data sample is produced by the machine sensor to when it is available to the Kafka broker consumers. We tested the system tolerance with respect to the same increase of message loads employed in the SCADA experiment. As shown in Figure 6.8, in the case of

Mosquitto the message delay is around 70 ms, while for RabbitMQ, depicted in Figure 6.9, it is about 4ms. In both cases, the increase in load did not impact the performance, thus confirming the good scalability of the proposed solution in realistic industrial settings.

IT layer stress test

The IT-level scalability test aims at assessing the performance of the Central IT Kafka broker when it is loaded with messages coming from multiple Local IT plants. To measure that, we set up a test bed reproducing the scenario of a company running seven production plants, located in different places geographically distant from each other, that send data to the Central IT layer. The data transfer dynamics vary from plant to plant and are not predictable: to such uncertainty, many indicators contribute the switch off/on of machines determined by the production schedule, the different rates at which machine registries produce data, and the variability of the network bandwidth available during the data transfer. Each plant is emulated via a software message producer with the message rate randomly changed over time throughout the test. This configuration produced an overall message load on the Central IT layer that is variable in time: the objective of the test was to assess the performance of the Central IT broker in response to such variations of the input load.

Each Producer produces messages within a time frame of 35 minutes, split into 5 intervals of 7 minutes each, with increasing message throughput. Each interval is characterized by an average message rate plus (or minus) a random delay with a 60% bound of the specific message rate of the interval. The producers wake up at a different time, thus emulating with effectiveness a real scenario in which some production machines are operating, while others are off. Moreover, the highest peak (2 240 msg/30s) is reached when all producers are active and generate messages at a rate of 320 msg/30s.

In Figure 6.10, we report both the overall message load trend and the observed average and standard deviation of the message delay (discrete points curves). Each point represents the average (standard deviation, respectively) delay of messages arriving in a 30s time window. From time 0 to T1 the producers gradually increase the overall system load, sending concurrently up to 2 240 msg/30s. At T1, the load suddenly decreases to 1 235 msg/30s due to the disconnection of 3 plants. From T1 to T2 the message rate continues to grow, reaching a new local maximum of 1 780 msg/30s. At T2, two producers gradually stop production.

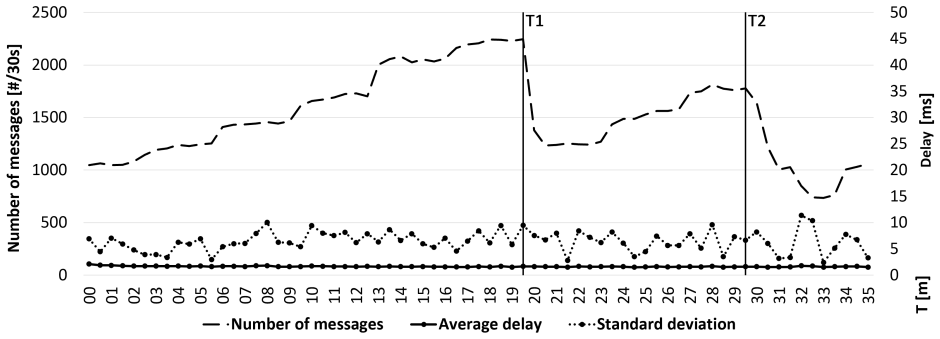


Figure 6.10: Central IT Kafka broker stress test.

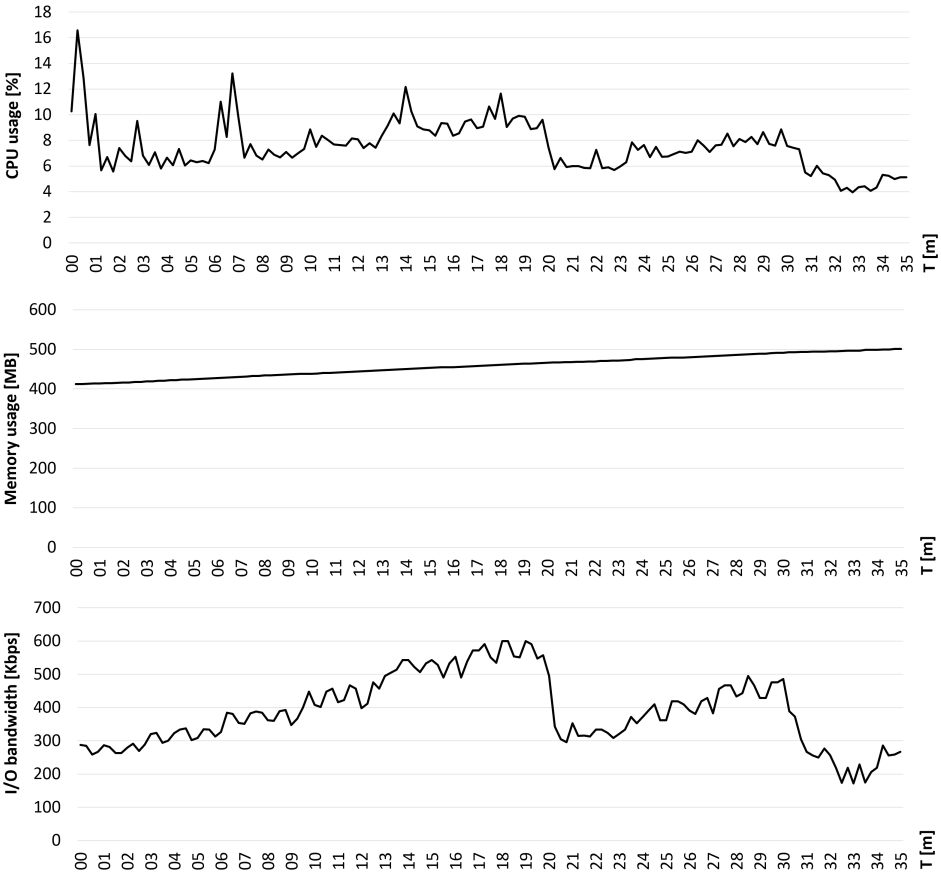


Figure 6.11: Central IT Kafka broker resources usage.

The reader will note that the average delay curve is almost steady (hitting a value of around 1.65ms) independently of the message load fluctuation, while the standard de-

viation keeps below 10ms. Figure 6.11 shows the resource consumption of the Docker container that runs the Central IT Kafka broker during the test. We ran the experiment 20 times with different random seeds and changed the start instant of the plants. All experiments showed quite comparable performances in terms of message delay and resource consumption.

Controlled data access test

In Section 6.1, we thoroughly discussed the advantages of deploying a data gathering and sharing system in a typical industrial manufacturing setting. We then implemented a message broker-based support to gather data from the OT layer and move it to the IT department where it is eventually consumed. Currently, the platform allows data to flow only from OT to IT layers. Should malicious actors gain access to the message brokers, they could steal precious data or convey tampered information to the management department. To accomplish a full IT/OT integration, in the future we will allow data to flow from the IT downwards to the OT, thus exposing the shop floor to further security and safety issues. In fact, in that case, malicious intruders could exploit the system to inject control data that will eventually cause damage to machines or put the workers' safety at serious risk. In order to face these and future security issues, we have provided the platform with support to prevent malicious or unauthorized access to production data. In the following, we disclose the details of the experiment run to test the implemented access control mechanisms.

First of all, in our testbed, we leverage a VPN to secure communication among VMs (layers) within a plant, as well as inter-plant communication (in our case, communication between the Local IT of Bologna and Central IT of Imola). SSL/TLS is used to guarantee data integrity and confidentiality of communication with TPS premises. In order to reinforce security at the OT layer, we exploited the Kafka Access Control List (ACL) feature to manage access (both in reading and writing mode) to the topics of the OT layer Kafka broker. ACLs can be defined for each topic with a very fine-grained policy.

According to Table 6.1, the Imola Central IT layer consumes data coming from the underlying OT layer and data coming from the Bologna OT layer. We assume that an intruder managed to gain access to the environment and steal the identity of a producer (e.g. *Cooler*) that is allowed to post messages only to the "bologna_capper-1-cooling" topic. When the tampered publisher Cooler tries to push data to the "bologna_capper-1-data-cycle" topic, which it is not granted to write, the producer is notified about the denial of

authorization, while the Kafka authenticator log reports what has happened: timestamp of the failed attempt, the host from which the attempt originated, and the involved topic.

What has been shown is a very simple rule, but through the ACL mechanism, the platform can enforce more complex access control rules at any layer (OT, Local IT, Central IT). That allows for preventing the execution of malicious or accidental read/writes operations on topics. In fact, this tool guarantees that all stakeholders, both internal and external to the company, are granted access just to information of which they are the intended recipient. Let us conclude by noting that this test demonstrates that the platform meets the requirement $R3$ mentioned in Section 6.1.

Fault-tolerance test

The objective of our last experiment is to test the platform's capability to react to potential faults. In the implemented data gathering system, we aim to guarantee continuous support to the data ingestion and migration towards the upper layer. Unexpected and sudden interruptions of the mentioned support may cause data blackouts that can severely impact the efficiency and efficacy of processes that need to consume operational data. When a fault occurs, a plan needs to be promptly enforced to recover as quickly as possible and restore the previously provided quality of the service.

Once again, we focus on the OT layer as it represents the data entry point of the platform. Particularly, we intend to preserve the service continuity of the message broker, since faults at this layer may in turn compromise the service continuity of upper-layer brokers. We remind that Kafka brokers are implemented as containerized services running inside VMs. Faults can be of many different types (a crash of the container/VM, a hardware failure of the hosting PC). Whatever can go wrong at runtime is a fault the system will have to deal with.

To face faults, we exploit Kafka replication by deploying a redundant number of Kafka brokers in the OT layer. Each topic replication factor is set to 2, meaning that a topic is configured to have 2 replicas (one is the *Master*, the other one is the *Slave*) residing in two of the available brokers respectively. Slave replicas will function as a backup of their respective Masters. Kafka takes care of distributing topic replicas among the brokers, managing the faults of brokers, and keeping topics in sync among all replicas.

For the test purpose, we deployed three brokers (*B0*, *B1* and *B2*) and created three topics (*bologna_capper_data_cycle*, *bologna_capper_plasticizer_data*, and *bologna_capper_-absolute_totalizers* respectively). Then, we configured producers to send messages on the

Active Brokers	Replica Brokers	Replicas In Synch
TOPIC bologna_capper_data_cycle		
T1 - [B0,B1,B2]	[B0,B1*]	[B0,B1]
T2 - [B0,B1]	[B0,B1*]	[B0,B1]
T3 - [B0,B1,B2]	[B0,B1*]	[B0,B1]
TOPIC bologna_capper_plasticizer_data		
T1 - [B0,B1,B2]	[B1,B2*]	[B1,B2]
T2 - [B0,B1]	[B1*,B2]	[B1]
T3 - [B0,B1,B2]	[B1,B2*]	[B1,B2]
TOPIC bologna_capper_absolute_totalizers		
T1 - [B0,B1,B2]	[B1*,B2]	[B1,B2]
T2 - [B0,B1]	[B1*,B2]	[B1]
T3 - [B0,B1,B2]	[B1*,B2]	[B1,B2]

Table 6.2: Resilience test: brokers per topic. Brokers denoted with an asterisk hold a Master replica.

three topics at an overall rate of 192/sec. The experiment consists of getting the whole system up to work at time $T1$, tearing down *B2* at time $T2$ (we simulate the broker fault by killing the broker instance), and getting *B2* back to work at time $T3$. The blackout of B2 lasts for about 5 minutes.

Table 6.2 shows the dynamics of the system in the course of the experiment. In the "Replica Brokers" column we reported the brokers holding the replica of the considered topic (the one denoted with an asterisk is the broker holding the Master replica). The reader may note that topic *bologna_capper_data_cycle* is not affected by B2 blackout because none of its replicas are held by B2. Topic *bologna_capper_plasticizer_data* has a Master replica in B2 and a Slave replica in B1. At time T2, being the Master replica unreachable due to B2 fault, prosumers are redirected to the B1 Slave replica. At time T3, when B2 will be again up and working, the B2 Master replica is synchronized with the Slave, and prosumers are redirected back to it. For what concerns *bologna_capper_absolute_totalizers*, no action is taken at time T2 since B2 is holding a Slave replica. Prosumers will keep using the Master replica held by B1. When B2 recovers, the Slave replica is synchronized with the Master. Of course, with the two replicas configuration, service continuity of a topic is guaranteed as long as at least one replica is available. For highly unstable or overloaded systems, it is advisable to increase the redundancy of the number of brokers and/or topic replicas.

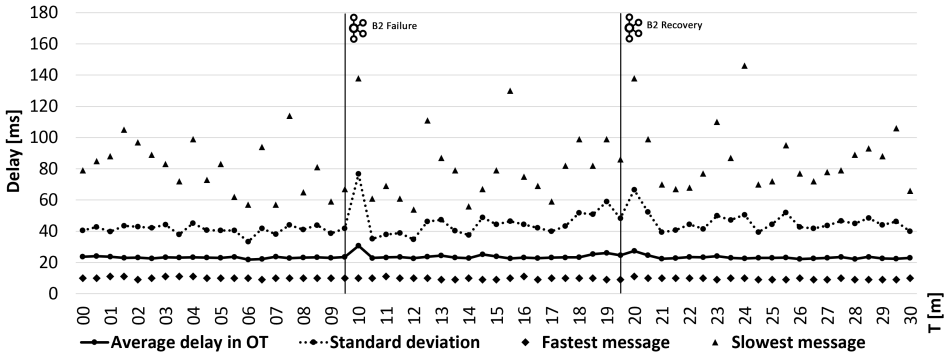


Figure 6.12: Resilience test: average and standard deviation of message delay.

In Figure 6.12 we depicted the average and standard deviation of message delay recorded during the experiment. For each observation, maximum and minimum delay values are also reported. The reader may notice that no message is lost both at time T2 and time T3: small glitches of both the average and the standard deviation curves at the two instants of time prove the robustness of the system, which then is proved to meet the requirement *R4* set out in Section 6.1.

We also monitored the trends of CPU usage, memory occupancy, and I/O throughput of the docker containers running the three brokers reported in Figure 6.13. B2 recovery at T2 causes a glitch in CPU usage and peaks in I/O throughput trends. B2 CPU usage irregular transient (small consecutive peaks) is due to the progressive reactivation of the Docker container's modules. I/O throughput peaks observed at T2, where B2 is the highest, are due to the synchronization of topics among the brokers. As far as B1 and B3 are concerned, except for the I/O throughput, almost no significant resource usage change was observed throughout the experiment. Test results show that thanks to proper management of the underlying message brokering, SIRDAM4.0 is able to absorb sudden and long-lasting faults, guaranteeing the reliability and service continuity of the system with no decrease in the performance level.

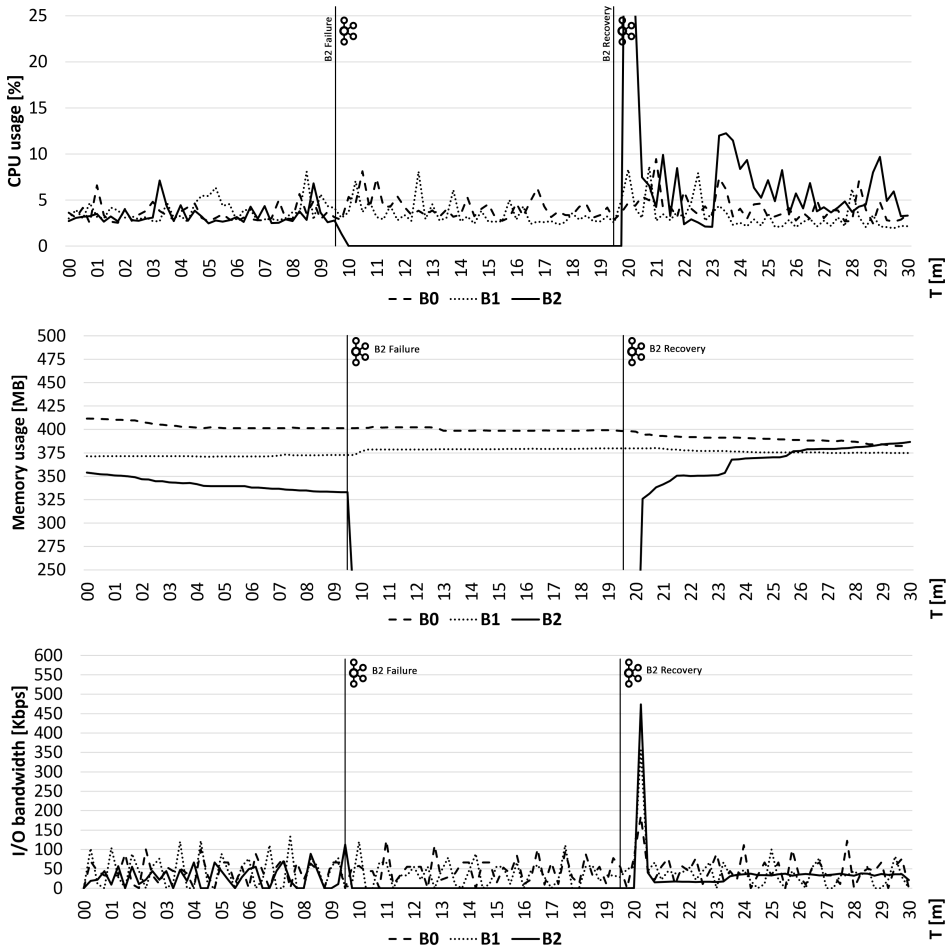


Figure 6.13: Resource usage of the Docker containers running the three brokers.

6.3 QoS-Enabled Semantic Routing

Future industrial networks will be characterized by an unprecedented degree of heterogeneity and complexity. Traditional solutions, mainly based on the direct interconnection of machines one to each other and machines towards the control room, cannot provide the required degree of flexibility. Therefore, there is the need to adopt novel solutions able not only to manage the deluge of data generated by the underlying industrial machinery layer, but also to allow the dynamic and flexible reconfiguration of the topology driven by QoS requirements. By considering the momentum of MOM as an enabler of the I4.0 vision, we believe it will become a pillar of future industrial ecosystems.

However, let us note that even if the adoption of MOM enables critical features to facilitate message dispatching by making it independent on actual machine location, it does not provide features allowing to control how packets flow through middle network devices along the data routing path. In fact, once a message is sent from a broker to a receiver (or vice versa), the path the message traverses and the QoS network elements provide cannot be managed, since out of the visibility and control scope of the MOM. While this approach allows to abstract from low-level networking details, the ability to dictate the behavior of underlying network devices is essential in industrial networks that typically require satisfy of stringent QoS requirements.

To address the above open issues, we claim the importance of adding an SDN control plane so to ease the management and dynamic reconfiguration of network elements that act as the (distributed) communication substrate between the machines and the MOM. In addition, the SDN controller could provide network-wide abstractions to define and enforce fine-grained network policies. Accordingly, we propose to integrate MOM and SDN paradigms in a joint solution that allows the QoS-enabled dispatching of MOM-based messages by also limiting the management burden on network operators and machine technicians. At the top level, the MOM infrastructure is in charge of identifying the set and (abstract) location of destination nodes a message should be dispatched to. At the bottom level, application gateways, deployed close to machines, act as intermediaries among machines (and their proprietary industrial protocols) and the MOM (based on widely adopted standard protocols) by allowing to specify the required QoS in a semantically enriched manner. In the middle level, the SDN controller exploits its centralized point of view to (re)configure the communication substrate, and the network elements therein,

in relation to the current state of the network as well as QoS requirements identified by application gateways.

Based on the collaboration among the MOM infrastructure and the SDN controller, network elements can be properly configured to i) select the best route towards the destination and forward messages accordingly, ii) manage competing traffic flows in a coordinated manner, e.g., to ensure prompt dispatching of mission-critical messages even if at the expense of less critical messages, and iii) enforce in-network processing to reduce the network utilization, e.g., by merging consecutive packets in only one. To this purpose, we enhance packets by adding custom tags to specify how network devices should manage them in terms of QoS and processing. Based on such tags, network devices exploit dynamically deployed traffic rules and ad-hoc software modules to enable proper rerouting, per-traffic flow prioritization, and in-network processing with the goal of significantly improving networking performance and reducing the overall network overhead.

For example, by considering two traffic flows between the MOM broker and a machine, proper routing table management allows to forward traffic flows tagged as "mission-critical" via a large-bandwidth low-latency path (if available). In addition, a QoS software module can selectively delay packets of traffic flows tagged as "not-urgent", where the magnitude of the imposed delay can also depend on the current level of network saturation. Finally, an in-network processing module can exploit the knowledge about the carried data model to manage packet content. In particular, let us note that the knowledge of packet semantic allows the adoption of a wide range of highly expressive traffic flow management mechanisms. For instance, it is possible to forward packets only if they satisfy a given rule, e.g., if they carry temperature values greater than a threshold, or to apply functions to send pre-processed values, e.g., sending only one packet with the average temperature resulting from a series of received temperature values. From a functional point of view, the in-network processing layer sits atop the data forwarding layer. As in the case of SDN deployment, we do not argue that all the network devices should provide the in-network processing capability. Instead, we promote a pragmatic approach where legacy and novel solutions cooperate effectively. Since the SDN controller holds a network-wide view, it knows which network devices offer in-network processing functions and which not. Therefore, traffic can be optimally handled by maximizing the in-network processing (e.g., routing of packets carrying values that can be averaged towards network devices providing that aggregation function) while ensuring QoS requirements.

To support the dynamic and semantic-driven management of routing, QoS, and in-network processing, we propose to enrich packet semantic with the following five tags: source ID, destination ID, QoS, in-network processing, and data model:

- source ID and destination ID are logical names that decouple the "who" and "where." In fact, the overload of semantic affecting traditional IP addresses intrinsically prevents seamless mobility. Moreover, since messages are actually sent to the broker by application gateways, it is not possible to exploit the sender IP address to discriminate among different machines;
- the QoS tag allows to specify if a traffic flow should be considered either as mission-critical or non-critical, possibly defining multiple levels. Of course, this tag can be enriched with other priority levels in case it is required to support finer-grained management of traffic flows, such as gold/silver/bronze traffic types [6];
- the in-network processing tag provides information about the possibility to process the packet directly on network devices. When in-network processing is allowed, such a tag also specifies the function to apply and the target field(s);
- the data model tag identifies the syntax of the payload. Both the SDN controller and the in-network processing modules deployed on network devices take advantage of that tag. The former exploits the information provided by the data model tag to decide where to route the packets (perhaps not all the network devices can process all the data models), whereas the latter takes advantage of it to understand how to process the packet.

6.3.1 Architectural draft

The second proposed architecture [7, 8], mostly working at the application layer, adopts the typical SDN approach by identifying two main areas: *Control Plane* and *Data Plane*. In the *Control Plane* area we deploy: the MOM controller, interacting with the MOM broker; the SDN controller, controlling network elements; and the Gateway controller, managing the many application gateways deployed in the environment. In the *Data Plane* area takes place the implementation of: the MOM, the Gateway components, and, in the middle, network elements equipped with in-network processing modules running atop them and managed by the SDN controller.

Each component has different duties and responsibilities:

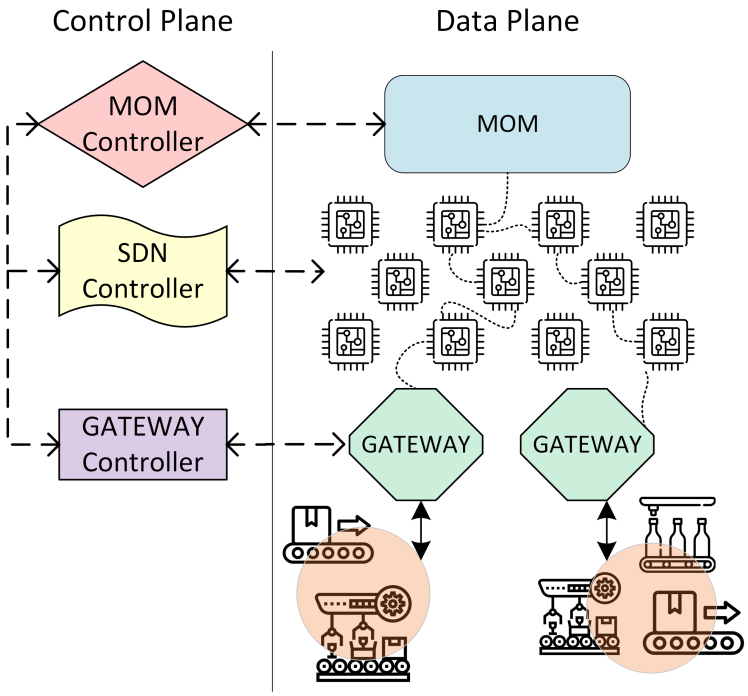


Figure 6.14: Functional/layered view of the SDN-MOM distributed architecture.

- The **MOM Controller** is demanded to sniffing and re-routing the traffic flowing into the MOM topics. It subscribes to several MOM topics to receive messages and to analyze the traffic. It consumes part of the header and can return back onto the MOM the message, performing decisions based on the header and on the information received from the SDN Controller and the Gateway Controller. The messages can be forwarded to a specific topic, duplicated among different topics, or consumed and pulled out from the flow. At the same time, the MOM Controller issues information that will be used from the SDN controller to correctly configure the SDN devices for achieving the desired level of QoS on the specific flow.

- The **Message-Oriented Middleware**-MOM is one of the core pieces of our infrastructure. It is the logical single point of communication between several firm sectors. It contains topics written by the Gateways and can be read by multiple other Gateways, based on the plant communication requirements. The MOM is responsible for guaranteeing differentiated QoS policies with different semantics. Typically, the at-most-once semantic can be used for best-effort machinery traffic. Otherwise, at-least-once semantic can be used for monitoring mission-critical assets

and for control traffic. Moreover, some messages can be sent with high priority, guarantying differentiated traffic management and avoiding congestion.

- The **SDN Controller** centralizes network intelligence in a separate component, disassociating the packet forwarding process from the routing processes. The SDN Control Plane consists of one or more controllers that are considered the brain of the network, where all intelligence is embedded. The SDN Controller configures the network resources. In our infrastructure, the SDN Controller has full knowledge of the network and the paths, guarantying a fine-grained ruling of the traffic coming from the Gateways. Differentiated policies can be applied based on the content of the messages, following the MOM Controller rules. The traffic can be duplicated, aggregated, blocked, and re-routed on different data paths. The SDN Controller role is dual: on the one hand, it has to gather and make accessible to the MOM Controller information about machinery position, router congestion, and gateway policies. On the other hand, it has to translate the MOM Controller policies to actual configurations on network entities.

- The **Gateway Controller** emits control messages directed to the Gateway of the infrastructure. It works in strict coordination with the MOM Controller and SDN Controller, to avoid congestions and to maintain topic abstractions coherent with the real machine distribution. Its management duties comprehend: checking of the state of all the gateways, which must be configured coherently with the machine on which are acting; synchronization with SDN and MOM controllers, that can send re-configuration messages to avoid congestion. Practically, it can manage the header that is applied from the Gateways to each packet, modify the priority of the messages, and define levels of QoS applied directly to the data-extraction phase.

- The **Gateway** duties comprehend the data gathering, the data transformation to an internal MOM-specific representation, the header addition, and the interconnection between the industrial machinery world and the MOM topic-centric world. In industrial scenarios, it is common to have machines that use different languages and protocols for data exporting and representation (e.g. Modbus, Profibus, OPC UA, OMG DDS, EtherCAT [29]). For this purpose, the Gateways can be specialized with ad-hoc libraries and push or pull strategies based on the specific machinery from which to gather information. Moreover, the QoS can be managed directly at

this level, avoiding high useless throughput when the plant is working in a normal condition.

Figure 6.14 depicts a schematic of the entire infrastructure. Dashed paths between controller entities in the control plane, and between control and data planes represent the management/configuration data exchanges that are logically separate from data flows. Continuous line paths represent data flows between machines and our architecture. Finally, dotted paths represent network data paths dynamically (re)configured according to the SDN Controller policies.

6.3.2 Experiments

To demonstrate the feasibility and efficiency of the proposed solution, the section outlines primary performance results achieved based on our proof-of-concept prototype. On the one hand, we want to demonstrate how the adoption of our gateways allows to enrich packets with meaningful tags, sent by a plethora of gateways to a (logically) centralized MOM broker. On the other hand, we focus on the computational overhead imposed on intermediate nodes performing in-network processing, with the primary goal of demonstrating its scalability in terms of packet rate on nodes with relatively limited hardware capabilities.

The testbed comprises 4 Amazon EC2 VMs based on Ubuntu Server 20.04 LTS and equipped with 1 vCPU and 1 GB of RAM. The first VM acts as industrial machinery simulator, simulating an arbitrary number of machines transmitting messages based on pre-defined templates. The second VM acts as Gateway, collecting packets from the machines, enriching packets with proposed tags, and routing them to the MOM broker. The third VM hosted between the Gateway and the MOM broker runs the in-network processing module. Lastly, the fourth VM acts as a MOM broker.

Integrated SDN-MOM Semantic Routing

The SDN Controller and in-network processing modules act as core functional services in our architecture. As the reader may know, the main SDN implementations (e.g., Openflow [74]) work by forwarding to the SDN controller the packets of the flow not having a configured compatible rule for the delivery. The SDN Controller, thanks to its complete knowledge of the monitored locality, can make different routing decisions based on information about packet source or destination addresses, current network topology,

and traffic, and also taking into account information arriving from external sources, in our case the MOM and Gateway Controllers.

Our implementation configures the networking rules based on application-specific header information to perform semantic routing on the network devices and on top of the MOM. In order to have in each moment the correct header associated with the payload, we decided to enrich the messages directly at the Gateway level of the infrastructure, so that only complete messages traverse the platform. The gateways are configured to set for each machine the specified header, containing information about QoS, but also additional information about machine `geo_position`, and `innet_processing`. In the following Listing 6.3, there is an example of the header added to each message ingested in the data platform.

```
"machine_id":"m00000001",
"machine_serial":"123456789",
"source_id":"m00000001",
"destination_id":"rabbitmq",
"geo_position":{
        "continent_name":"EU",
        "city_name":"Bologna",
        "country_iso_code":"IT",
        "region_name":"Emilia-Romagna",
        "location":{
                "lat":"44.493690",
                "lon":"11.343080"
        }
},
"innet_processing":{
        "func":"sum",
        "field":"TEMP_1.value"
},
"payload_description":"sensor_temp",
"QoS":5,
"message_rate":100
```

Listing 6.3: JSON header example

Going further from pure application-level additional information, `message_rate` and the `QoS` tags need in-depth analysis. The `QoS` tag contains an integer from 1 to 10 that commands the desired quality in the entire platform. The SDN Controller exploits that tag to guarantee a correct path for reaching the quality expected and, where appropriate, for reducing the data quantity using the `innet_processing` additional details. The `message_rate` tag is expressed in milliseconds and represents the time interval between a data gathering and another.

Since the data volume traversing the platform can be managed by both Gateways and in-networking modules, we have conducted some tests to demonstrate how those techniques can be used in a conjunct profitable manner. In our first test, we exploit the Gateway capabilities of modulating the traffic entering the platform based on the `message_rate` tag. For this experiment, we defined three `message_rate` levels: bronze (110 ms), silver (60 ms), and gold (30 ms). During the execution, we change the levels by communicating the `machine_id` and the new `message_rate` values to the Gateway Controller that will re-configure the correct Gateway. On the MOM VM, we run the AMQP broker implementation (the open-source RabbitMQ [104]).

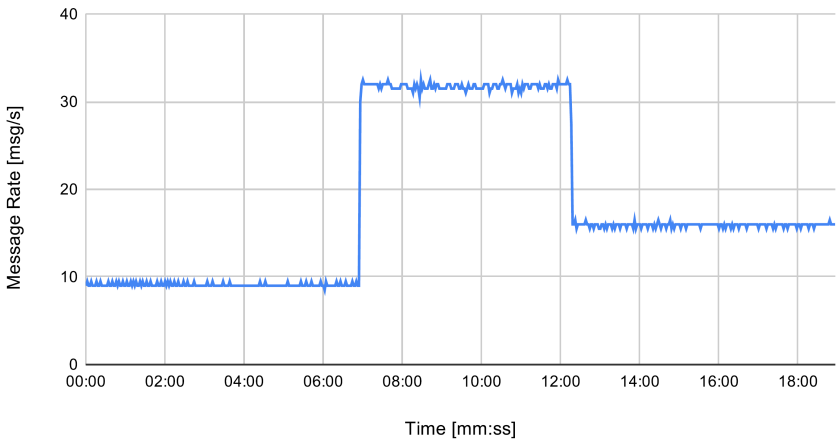


Figure 6.15: Bronze, gold and silver levels managed on Gateway.

Figure 6.15 shows the three levels of data gathering modified dynamically during one execution of about 20 minutes. At time 6:56 the `message_rate` changes from bronze to gold to supply the higher detail of the processes running on machinery. At time 12:16 the `message_rate` is set to silver quality.

In-network Processing

This section presents the experiments conducted to investigate the overhead introduced while performing in-network packet processing tasks to aggregate packets being part of the same flow. For the sake of simplicity, the Gateway directly sends all generated traffic to the VM running the in-network processing module. We implemented an in-network processing module that sorts the incoming traffic in queues and performs an average function on fields representing temperature values when a queue grows to 10 elements. Such a module takes advantage of the tags we introduced to enrich the packet semantic and identifying packets of different flows. In particular, a flow is identified by the logical names of sender and receiver (source ID and destination ID, respectively) and the aggregation function. Using that information, the in-network processing module inspects packet headers and sorts packets in different queues according to their tags.

As soon as a queue reaches a certain threshold, i.e., 10 packets, the in-network processing module applies the aggregation function on the target payload fields and sends out a single packet containing the computed result, i.e., the average value. The primary outcome is a sharp decrease of packets sent to the following node, since the in-network processing module lowers the overall network usage by in-situ processing of packets in relation to the in-network processing configuration, 90% in the case above.

However, such a network traffic reduction entails the following (at least) two important consequences. First, since packets are inspected, queued, and processed, they reach their destinations with an additional delay, due not only to packet queuing, but also to packet processing. Therefore, packets with strict QoS requirements may not be eligible to flow through in-network processing modules (the SDN controller must dictate alternative paths for them accordingly). Second, the in-network processing task itself may require more computational resources than those available when overwhelmed by high network traffic volumes. Through the experiments described in the following, we seek to evaluate how the incoming packet rate, namely, number of packets per (s)econd, affects the performance of in-network processing. In other words, we aim to outline the relationship between the incoming packet rate, the CPU load, and the outgoing packet rate.

The experiments vary along two dimensions. The first dimension is the percentage of CPU exploitable by the in-network processing module. Since the in-network processing module can be deployed on nodes already performing other activities, we assume that it cannot take advantage of all the available processing capabilities, since reserved for other

functions. To this end, we limit the CPU usage of the in-network processing module to 25% and 50% of the overall processing availability using `cpulimit` [82]. The second dimension is the packet rate sent by the application gateway to the in-network processing module. We generate increasing packet rates ranging from 2K to 16K in step of 2K packets/s.

To make the experiments reproducible in an automated manner, we developed a test framework based on Ansible [88], a configuration management tool, and several shell scripts. The choice of Ansible over other configuration management tools available on the market is mainly due to its intrinsic simplicity. In fact, Ansible does not require anything else except an SSH connection between the control node (where Ansible is installed) and managed nodes. We use `tcpdump` to capture the network traffic and a Python package called `psrecord` [4] to monitor the CPU activity of the in-network processing module.

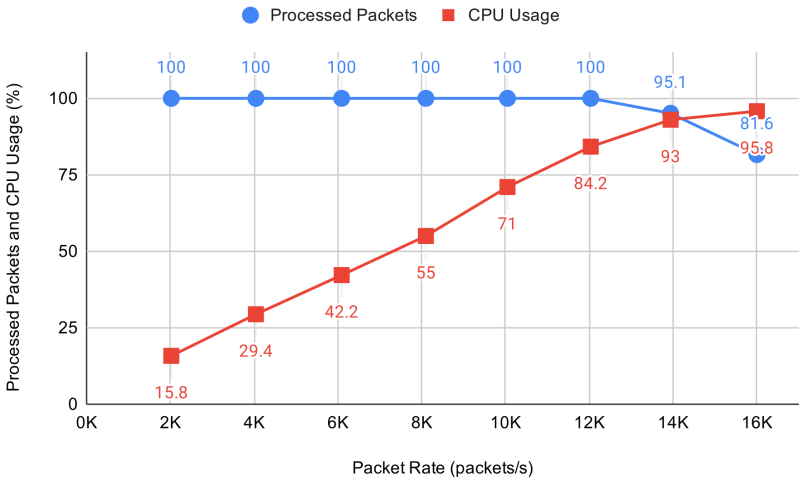


Figure 6.16: CPU usage limited to 50% of the total.

As Figure 6.16 shows, the in-network processing module can successfully handle the totality of the incoming packets with 50% of the CPU and incoming packet rates up to 12K packets/s. The CPU usage grows up to nearly 100% of the total when the incoming packet rate is 16K packets/s. Interestingly, in that circumstance, the in-network processing module can handle roughly 80% percent of the incoming packets (by losing the remaining 20%). In the light of those results, the SDN controller may decide to redirect up to 12K packets/s towards the in-network processing module to make possible the 100% of processed packets while not overloading the CPU. As the CPU maximum availability

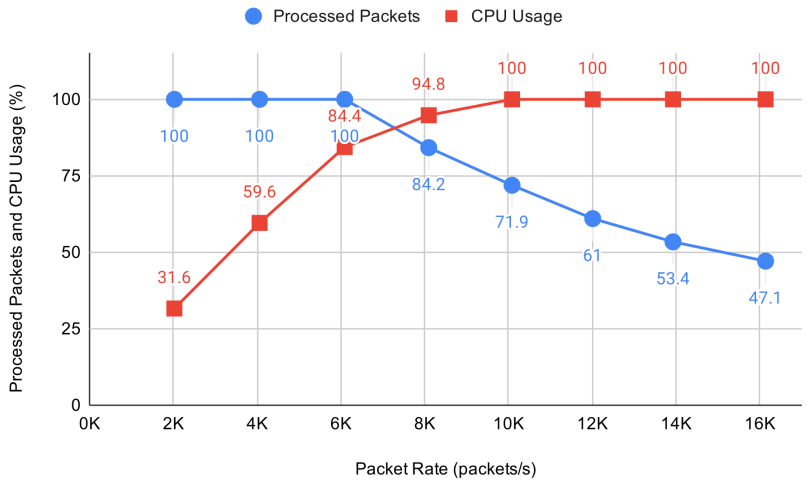


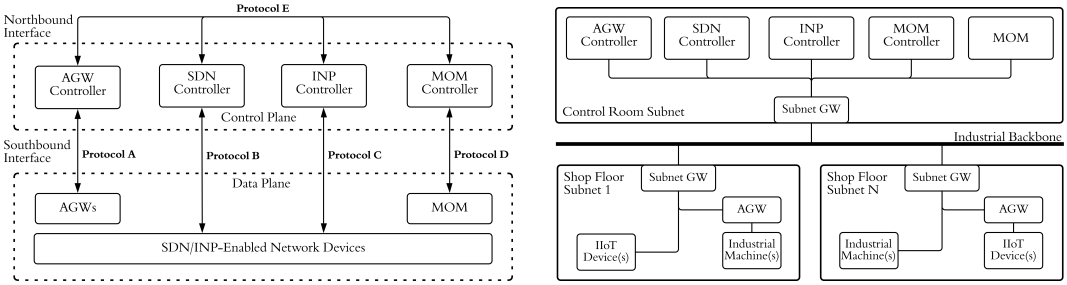
Figure 6.17: CPU usage limited to 25% of the total.

decreases to 25% of the total, the in-network processing module cannot handle more than around 6K packets/s. However, it is interesting to note how the performance dramatically decreases as the incoming packet rate grows. As Figure 6.17 shows, with an incoming packet rate equals to 16K packets/s, the in-network processing module cannot even handle 50% of packets. On the one hand, in-network processing can lower the number of packets traversing the network and reduce network utilization consequently. On the other hand, without detailed knowledge on the limits of the in-network processing modules deployed on the field, they can quickly become bottlenecks for the whole network.

6.3.3 Architecture generalization and protocols

As mentioned above, the solution we propose operates at both the application and network layers. The architecture design follows the SDN approach, clearly distinguishing the control plane from the data plane and identifying the interfaces (i.e., protocols) between such planes. In this regard, Figure 6.18a provides a layered architecture overview. The architecture can be seen as a generalization of the one presented in Figure 6.14 and adds an INP (In-Network Processing) Controller so that all controlling units are represented.

The control plane comprises AGW, MOM, SDN, and INP controllers to monitor and configure AGWs, MOM, and SDN/INP-enabled network devices, respectively. The rationale behind the architectural components is the following. AGWs sit close to shop floor components that cannot support our protocols. For example, some legacy industrial



(a) Functional/layered view. (b) Network topology.

Figure 6.18: Architecture overview.

machines may not support IP-based communications and, at the same time, cannot even be updated due to restrictive manufacturer policies. Then, the MOM decouples senders and receivers, sorts messages in topics of interest, and provides delivery semantics. Although the MOM enables critical features in message dispatching, it does not control how packets traverse the network. This is where SDN and INP come in. Specifically, SDN-enabled network devices put into action fine-grained traffic management (flow steering and prioritization), whereas INP-enabled network devices perform in-network traffic optimization (data filtering and aggregation).

Figure 6.18b places control and data plane components within the industrial domain, as described above. Specifically, the controllers and the MOM are in the control room subnet (plant/enterprise level). SDN/INP-enabled network devices are both subnet gateways and industrial backbone elements. Lastly, AGWs are in the shop floor subnets, close to the shop floor entities not compliant with the architecture interfaces. Note that each shop floor subnet may be provided with one or more AGWs, and a single AGW may serve one or more shop floor entities.

The protocols used by the links and the interaction flows between the architectural components are the following. Starting from the lowest layer, in the data plane we find machine data packets enriched by the respective AGWs with the Data Header, adopted to identify the data flow and the generating machine in a unique way. Protocols A to D implement the southbound interfaces of our architecture and command AGWs, SDN/INP-enabled network devices, and MOM. Lastly, Protocol E commands the infrastructure and configures the control plane, synchronizing all components on a unique shared state. For the sake of clearness, in the final subsection, we report the sequence diagram in case of new topic subscription.

Data Header

The Data Header is attached to every packet traversing the system by the AGWs of our architecture. Each component that receives a new unforeseen Data Header in the packet forwards it to its controller and waits for routing/processing/flow rules to be set.

Field	Type
flowId	int16
machineId	int16
machineSerial	String

Table 6.3: Structure of Data Header

Table 6.3 shows an implementation of the Data Header, which keeps track of three fields:

- **flowID**: 16-bit user-defined integer that identifies the flow in a unique way in the system.
- **machineId**: 16-bit machine identifier. It can be logically set based on user necessities.
- **machineSerial**: String displaying the machine serial number. It can be dynamically read from machine registers or manually set on the AGWs.

Protocol A

Protocol A is adopted between the AGW controller and AGWs. Its purpose is to pass information and configurations about which machine to connect for gathering data and with which frequency. Moreover, it contains the header field, to be put as header of every message gathered by the specific AGW.

Table 6.4 details Protocol A:

- **header**: the Data Header, applied to each packet outgoing from the AGW.
- **crud**: 2 bit flags for identifying Create, Read, Update and Delete of a new or existent configuration.
- **ttl**:time to live [ms] of the configuration. After that time the AGW stops sending out new messages. If set to 0 the configuration is permanent.

Field	Type
header	Data Header
crud	2bit
ttl	uint32
ipFrom	ipAddr
ipTo	ipAddr
destTopic	String
semanticDelivery	3bit
machineProtocol	String
machineUrl	String
pollingInterval	int8
geoPosition	geoURI [94]
applicationType	String

Table 6.4: Protocol A: AGW controller to AGWs

- **ipFrom**: IP address of the AGW interface on which send out messages through the platform.
- **ipTo**: IP address of the destination MOM.
- **destTopic**: destination topic of the messages.
- **semanticDelivery**: 3 bits identifying the semantic of the flow in the MOM. Typically, it can be at-most-once, at-least-once, and exactly-once, but others can be defined based on the specific implementation of the MOM.
- **machineProtocol**: protocol for extracting data from the machine. Examples can be MODBUS [22], Profibus [87], EtherCAT [35], and OPC-UA [75].
- **machineUrl**: Url address of the machine supervised by the AGW.
- **pollingInterval**: interval in [ms] for polling data extraction.
- **geoPosition**: position in space of the machine supervised by the AGW expressed in compliance to RFC 5870 [94].
- **applicationType**: application type header attached to the message body by the AGW. Examples can be 'application/json' or 'application/xml'.

Protocol B

Open Networking Foundation - "OpenFlow Switch Specification" [78] is adopted as Protocol B to enable the interaction between SDN controller and the SDN-enabled network devices. The SDN approach decouples data from control plane access, making the introduction of new network functionalities simple and well structured on SDN-compliant hardware. A custom SDN controller uses the data shared with the Protocol E to control switches and to forward the data to the correct INP-enabled network device for processing and, lastly, to the MOM.

Protocol C

We have selected the P4 [102] language to define data structures of this level. Unlike a generic language like C or Python, P4 is a domain-specific language with a set of constructs optimized for network data forwarding.

In our architecture, the P4 language is used to administrate the INP-enabled network devices and to create processing pipelines that digest the messages in the AGW-MOM path.

Protocol D

Protocol D is in charge of configuring and exchanging information between MOM controller and MOM. Protocol D is further split into two sub-protocols (i.e. D' and D") as follows.

Protocol D' configures the MOM to forward on a list of topics the data with the specified header and to exchange the semantic delivery QoS settings. The D' message is triggered every time that the MOM receives a new unforeseen flow in input and requires the MOM controller flow information.

Protocol D" is used to communicate the topic subscribers to the control plane to correctly set the network paths with the specified priority. The D" message is sent by the MOM to the controller every time there is a new subscription to the managed topics. It is necessary for guaranteeing the QoS also between the MOM and the subscribers and triggers an update on the SDN controller via the Protocol E.

Table 6.5 reports Protocol D':

- **header**: the Data Header, applied to each configuration for identifying the traffic on which to apply the rule.

Field	Type
header	Data Header
crud	2bit
ttl	uint32
semanticDelivery	3bit
forwardOnTopics	List<String>

Table 6.5: Protocol D': MOM controller to MOM

- **crud**: 2 bit flags for identifying Create, Read, Update and Delete of a new or existent configuration.
- **ttl**: time to live [ms] of the configuration. After that time the MOM stops forwarding/prioritizing the flow. If set to 0 the configuration is permanent.
- **semanticDelivery**: 3 bits identifying the semantic of the flow in the MOM. Typically, it can be at-most-once, at-least-once, and exactly-once, but others can be defined based on the specific implementation of the MOM.
- **forwardOnTopics**: the topics on which to forward the specified flow.

Field	Type
header	Data Header
crud	2bit
ipFromMom	ipAddr
topic	String
priority	3bit
subscribers	List<ipAddr>

Table 6.6: Protocol D": MOM to MOM controller

Table 6.6 reports Protocol D":

- **header**: the Data Header, applied to each configuration for identifying the traffic on which to apply the rule.
- **crud**: 2 bit flags for identifying Create, Read, Update and Delete of a new or existent configuration.
- **ipFromMom**: IP address of the MOM interface on which send out messages through the platform.

- **topic**: starting topic of the messages.
- **priority**: SDN priority of the flow outcoming the MOM.
- **subscribers**: a list of IP addresses subscribed to the specified topic. They are necessary for updating the SDN rules.

Protocol E

The northbound interface is managed in a uniform way between all entities by sharing information via Protocol E. Protocol E is configures the system and shares the state of the incoming data transmissions between all the controllers. It contains all the fields that we defined in the previous subsections together with the crud flags marking and differentiating new configurations from updates.

Table 6.7 describes Protocol E:

- **header**: the Data Header, applied to each configuration for identifying the traffic on which to apply the rule.
- **crud**: 2 bit flags for identifying Create, Read, Update and Delete of a new or existent configuration.
- **ttl**: time to live [ms] of the configuration.
- **semanticDelivery**: 3 bits identifying the semantic of the flow in the MOM.
- **priority**: SDN priority of the flow outcoming the MOM.
- **ipFrom**: IP address of the AGW interface on which send out messages through the platform.
- **ipFromMom**: IP address of the MOM interface on which send out messages through the platform.
- **ipTo**: IP address of the destination MOM.
- **destTopic**: destination topic of the messages.
- **forwardOnTopics**: the topics on which to forward the specified flow.
- **subscribers**: a list of IP addresses subscribed to the specified topic.

- **inp**: a list of field coupled with the specific INP function to apply on.
- **applicationType**: application type header attached to the message body by the AGW.
- **geoPosition**: position in space of the machine supervised by the AGW expressed in compliance to RFC 5870 [94].
- **machineProtocol**: protocol used by AGWs for extracting data from the machine.
- **machineUrl**: Url address of the machine supervised by the AGW.
- **pollingInterval**: interval in [ms] for polling data extraction.

Field	Type
header	Data Header
crud	2bit
ttl	uint32
semanticDelivery	3bit
priority	3bit
ipFrom	ipAddr
ipFromMom	ipAddr
ipTo	ipAddr
destTopic	String
forwardOnTopics	List<String>
subscribers	List<ipAddr>
inp	List<(field: String, func: String)>
applicationType	String
geoPosition	geoURI [94]
machineProtocol	String
machineUrl	String
pollingInterval	int8

Table 6.7: Protocol E: Controller to controller

New Topics Subscription Sequence Diagram

Figure 6.19 presents the sequence diagram in case of new topic subscription. By delving into finer details, the subscription of a new data consumer to the MOM triggers an inter-

action with Protocol D" (Table 6.6). Such interaction is handled by the MOM controller, which triggers a proper reconfiguration of the underlying SDN-enabled network devices.

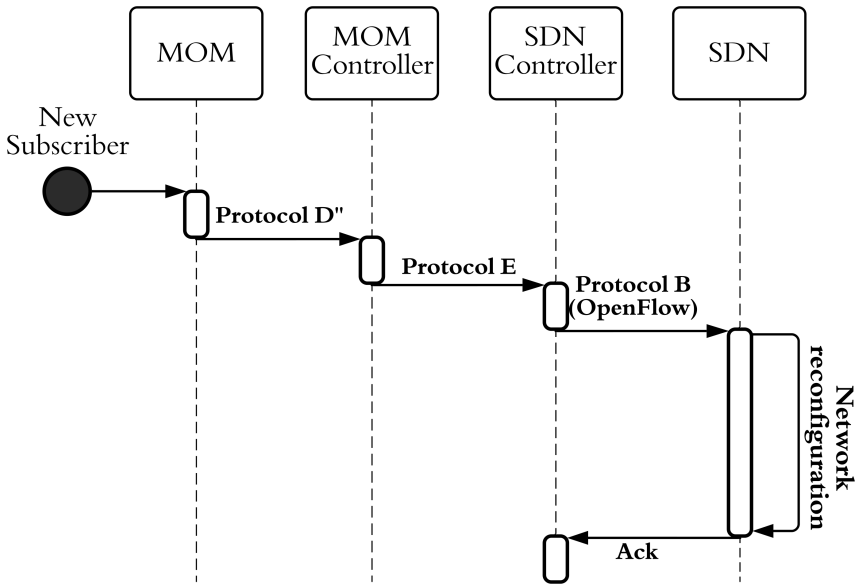


Figure 6.19: New topics subscription sequence diagram.

Figure 6.19 details the sequence diagram showing the interactions triggered by the addition of a new subscriber to a MOM topic. The MOM communicates the updated list of subscriber IPs to the MOM controller via Protocol D". MOM controller unleashes a synchronization phase (Protocol E) within the northbound interface to the SDN controller.

The SDN controller has complete knowledge of the network components and thus chooses the best path to achieve the desired priority based on provided priority bits. Once the best path is calculated, the SDN controller configures the network equipment with Protocol B (OpenFlow) to route the packets accordingly from the MOM to the new subscriber, guaranteeing the correct QoS to the final data consumer.

6.4 Low Latency m2m Communication Support

In our previous work in section 6.2 we proposed SIRDAM4.0, a multi-layer architecture to monitor legacy industrial equipment during their operations inside customer plants. The last proposed architecture provides near-real-time data gathering powered by two Apache Kafka installations, in OT and IT, respectively. In the proposal, several SCARF, dedicated software components acting as both adapter and gateway interface directly with Modbus-TCP machinery and export data in the OT Kafka instance. From here on, the data is then forwarded to the specific Kafka topic at the IT layer.

Collected results were encouraging: the components of the platform maintain a constant delay at the increase of the number of messages in transit in the system, thus confirming the scalability of the architecture. However, the latency assessed in the upper layer is about 70 ms, completely inadequate for M2M communication and incompatible with OT needs.

This last work [85] improves our prior proposal in several different directions. First, it splits the functions of the SCARF components into two distinct ones: adapters and gateways. This decoupling allows for a pluggable machine layer that can be enriched and support additional languages on the shop floor. That also allows the Gateway to be deployed on an edge computing layer, so decoupling OT- and IT-related functionalities. At the same time, we intend to define a support capable of enabling and expressing the necessary different convergence requirements of IT and OT layers. In addition, we state that the proposed solution is going to be assessed in a real testbed scenario, by using real machine data.

6.4.1 Architecture

Herein, we describe the approach taken to effectively blur the OT/IT boundary, so enabling fast, reliable, and secure operational OT data exchange towards the IT layer. As anticipated, the approach relies on a Gateway component that resides on the OT/IT boundary and a two-layered middleware solution aimed at fulfilling both the functional and the non-functional requirements of each layer.

System Components and Integration

Our proposal relies on the OPC UA Pub/Sub for M2M communication at the OT layer and uses Apache Kafka, a high-throughput, low-latency Message-oriented Middleware (MoM), for data gathering from multiple OT sites towards the IT domain. Though in principle, the OPC UA standard allows to reach and convey data above the OT layer to the upper layers SCADA, IT, Cloud, it is mainly a low-level interoperability protocol allowing fast transmission of data. At the IT layer, it is the Kafka MoM that permits the handling of large volumes of data in a secure and reliable manner, while at the same time, presents an extensible framework with a rich ecosystem of tools for IT.

Figure 6.20 shows a schematic representation of the main components of the architecture. The dotted line denotes the separation of the OT, IT, and Machine areas. In the Machine layer, the assets use several low-level and heterogeneous protocols some of which adhere to standard specifications with open-source implementation (e.g., OPC UA over TSN), meanwhile the majority do not generally interoperate, are closed source, and, most times, proprietary. In our proposal, the area on top of the assets, namely the OT, acts as a homogenization layer, by abstracting away from the upper layers the technical details of the specific protocols. The OT layer has a pluggable architecture, so to allow specific adapter components to be added dynamically into the infrastructure.

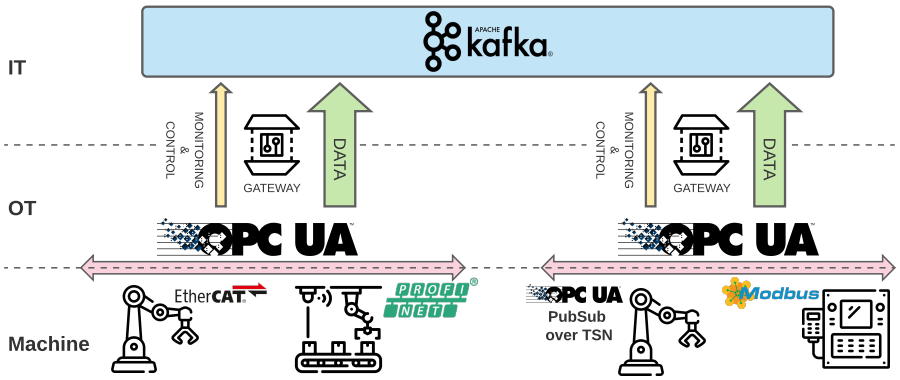


Figure 6.20: Architecture overview diagram.

After configuration, the adapter initiates the data collection according to the machine-specific language, exposing the machine information model via the common OPC UA standard. As anticipated, the rationale of that choice is to have a representation of the information common among the machine and the upper layers. We point out that our

architecture can support different adapter deployment strategies, depending also on the computational resources available on the specific industrial asset: if the machine has enough resources, the adapter can be directly deployed on it; otherwise, the adapter can be deployed elsewhere and is connected to the machine via the network fabric.

On top of the OPC UA protocol, we use the OPC UA Pub/Sub specification for message exchange inside the single shop floor. In the figure, the shop floor is depicted as an arrow above the machinery. Herein, heterogeneous traffic needs to co-exist and can vary from safety-critical control traffic to best-effort ones. In practice, data are gathered via a Gateway component which listens on OPC UA Pub/Sub endpoints and sends data to the Kafka MOM. Gateways are customized via configuration files that specify machine addresses and registers that must be manipulated and re-exposed on Kafka topics. In next subsection we report an example of the configuration file.

One goal of the Gateway is to differentiate between heterogeneous flows, namely raw sensor data and data deriving from monitoring processes on the shop floor. The former represents the information exposed by the industrial machine, containing data regarding its internal state. Additionally, the monitoring flow comprises the data and metrics related to networks, industrial processes, etc., supported by Kafka, by providing mechanisms and engineering options.

More specifically, on the producer side, we need to prioritize monitoring and control data traffic. To achieve that, we use different topics and different partitioning levels per data type, where the monitoring and control topics are configured to have a single partition and a higher degree of replication of that partition. This engineering option guarantees a total ordering of sent messages and an enhanced fault tolerance. Concerning the raw sensor data, the topics are configured to have multiple partitions and a lower degree of replication, guaranteeing higher input/throughput rates and lower memory usage.

At the consumer side, we use Apache Kafka differentiated semantics: At-Most-Once, At-Least-Once, and Exactly-Once for the commit management setting [21]; for monitoring and control data we exploit an Exactly-Once semantic; while for the data traffic, we use an At-Least-Once semantic, for faster reading.

Concluding, Apache Kafka supports Access Control Lists [20] via the so-called ACL Authorizers. This feature can be used in industrial settings since data confidentiality is of paramount importance.

Access control allows us to apply fine-grained access policies on the topics, by defining groups of authorized readers and writers and improving the security of the entire

infrastructure. On the other side, considering the stringent latency requirements at the OT layer, we assume it is not directly exposed to the external world, hence no particular security mechanisms are in place and the software running in this domain is certified and guaranteed not to pose any threat. These aspects deserve further investigation, and we are currently looking into the adoption of lightweight security mechanisms into our solution.

Bootstrapping the System

To bootstrap the system, one needs to provide some essential configuration parameters, binding the components together and initiating a structured information exchange towards the IT layer. The steps involved are as follows:

- **Configuration**: the Gateway component is issued a structured configuration file, containing the addresses of OPC UA enabled assets, such as IP addresses and multicast network groups on which to register the industrial asset internal state. Other configuration parameters contain information regarding the Kafka endpoints and topics on which to publish messages, QoS level mappings, and their publication frequency. For the sake of clarity, an example configuration file is reported in Listing 6.4.
- **Discovery**: the Gateway queries the OPC UA server(s) to verify the representation of the data. In this phase, the Gateway also checks if the OPC UA reported registers are coherent to what has been reported in the configuration file.
- **Operations**: once the discovery phase completes with success, the Gateway subscribes to the multicast network groups, starting the flow of messages, which upon reception in a specific protocol dialect, are un/marshaled to a (configurable) JSON representation. Depending on the data type, the messages can be sent on different channels. For instance, the level can be set on high-quality and ordered for "control" flows, guaranteeing fast and reliable delivery, while sensor messages, can be sent with a non-ordered semantic, depending on customer-specific policies.

From this point onwards, we consider that the data are available and can be fetched from the Kafka topics, and that data can be read by multiple consumers, depending on specific access policies.

It is noteworthy to point out that the decoupling of the OT and IT layer, through the use of a lightweight configurable Gateway, enables us to implement advanced control

features addressing reliability and scalability in scenarios of high ingress traffic. To this aim, we are currently investigating the design and implementation of a lightweight control and management plane, allowing for the run time deployment of customized coordination and synchronization schemes among the Gateway components.

```
{
  "machines":[
    {
      "name":"MACHINE_1",
      "ip_address":"192.168.0.3",
      "transport_profile":"http://opcfoundation.org/UA-Profile/Transport/
          pubsub-udp-uadp",
      "network_address_url":"224.0.0.18:4840"
    }
  ],
  "kafka":{
    "cluster_ip_addresses":[
      "192.168.1.2"
    ],
    "topic":"myTopic"
  },
  "publishers":[
    {
      "data_group_name":"datagroup-1",
      "writer_group_id":"1",
      "registers":[
        "PRESSURE_1",
        "OVEN_TEMPERATURE_1"
      ],
      "interval":"100",
      "QoS":"data"
    }
  ]
}
```

Listing 6.4: Example of JSON configuration file used by the Gateway.

6.4.2 Experiments

The goal of the experiment is to validate our architecture so as to show its capacity to work while suiting different constraints in an effective way. We intend to show the capacity to support QoS specifications of low-latency flows at the OT layer, while at the same time assessing the capability of providing high-throughput and quality data to the IT layer. To this end, we have developed a testbed depicted in Figure 6.20.

To fully assess the functional capabilities of our proposal, we have deployed a real testbed of five nodes, hosting different functionalities related to the OT and IT layer and where nodes are connected via a dedicated network consisting of a 1 GB switch. While this network setting may not be as rich as a real deployment scenario, it suffices the purpose of this work, aimed at testing and assessing the functional components of our architecture in an operational environment. For completeness, Table 6.8 reports the characteristics of the deployed nodes.

Name	Component	Operating System	CPU	RAM	Network
Node 1	Machine Simulator 1	Ubuntu 20.04.3 LTS	Intel Core i5-2400 CPU @ 3.10GHz	8 GB	1 Gpbs
Node 2	Machine Simulator 2				
Node 3	Gateway				
Node 4	Kafka Consumer				
Node 5	Apache Kafka	Ubuntu 20.04.3 LTS	Intel Core i5-3470 CPU @ 3.20GHz	16 GB	1 Gpbs

Table 6.8: Testbed deployment: components, OS, and hardware characteristics.

Two nodes of the infrastructure are dedicated to traffic simulation. For this part, we rely on some software packages emulating realistic industrial machine traffic, build and developed from scratch starting from actual industrial machinery specifications, as seen in 6.2.

More specifically, the first simulator (Node 1) simulates an industrial asset, by exposing its internal operational state via the Modbus/TCP protocol. A Modbus adapter at the machine layer can read and extract the information in a protocol-agnostic format, successively exposing and structuring the machine information by using the OPC UA data

model. Finally, the data is transmitted by using the OPC UA Pub/Sub protocol. Let us note that the adapter acts both as a subscriber to and publisher of data depending on the configuration and purpose. Then, the information is available to be received by all other entities present in the network (machines and gateways). A consumer, receiving the data emitted by the first simulator, is deployed at Node 2, where the software implemented represents an OPC UA Subscriber. That subscribes to the first simulator and begins receiving the messages published by the first machine. This behavior simulates a typical sensors-to-controller scenario.

Next, Node 3 hosts the Gateway component where it subscribes to the messages sent by the simulator present in Node 1. These are the same messages received by the simulator in Node 2. Node 4 is a Docker-based Kafka deployment that receives messages produced by the Gateway that acts as a producer. Finally, Node 5 hosts a Kafka consumer, consisting of a custom program that receives messages from specific Kafka topics. The consumer allows us to estimate the transit time that takes a message from Node 1 to a consumer in the IT department of the factory or directly to the Cloud.

In order to accurately measure time, nodes are synchronized by using the Precision Time Protocol (PTP) to extract fine-grained metrics in the time domain. Toward that goal, the node hosting the Gateway is configured as the controller, providing a reference clock for all other entities that participate in the PTP domain, whereas the others act as responders. For additional details on the implementation front, we refer the reader to the public repository containing the source code of the project [43].

Performance Assessment

To assess the proposal, we measure the message latency from the OT-to-IT layer, under varying traffic regimes.

Figure 6.21 shows the latency in the OT level (Node 1 → Node 2), whereas Figure 6.22 measures the end-to-end latency, that is from the OT layer to the Kafka consumer in the IT layer (Node 5). In both cases, the latency is computed as the time period between the receiving and sending time at the application layer, by sending messages with a different rate, from 400 to 1500 messages per second.

Figure 6.21 shows that the latency between the two simulated machines remains stable while increasing in the number of messages/second up to 1500 per second. Most importantly, we always observe a sub-millisecond latency, which is the required latency expected at the OT layer, in particular for the communication between different machines or PLCs.

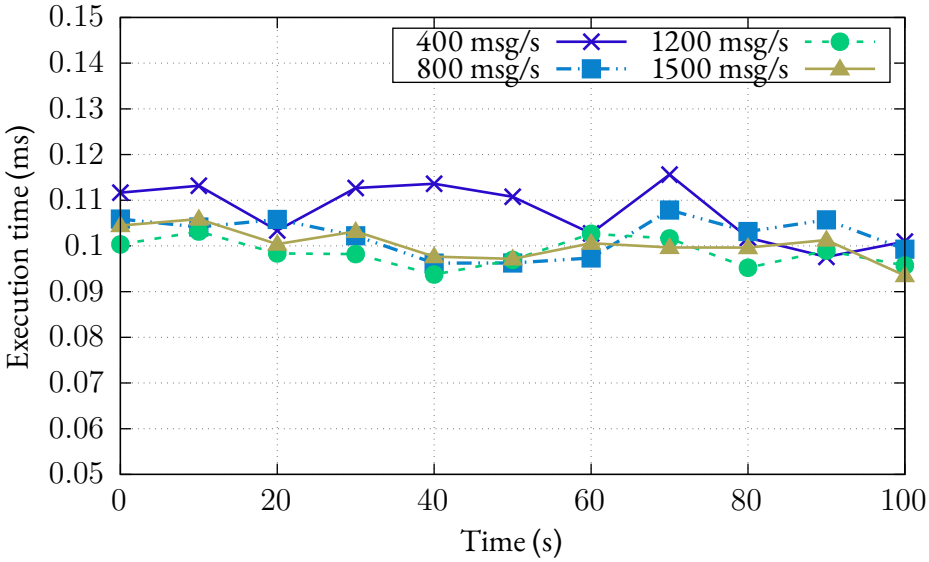


Figure 6.21: Machine-to-machine communication latency under varying message load of the OT layer.

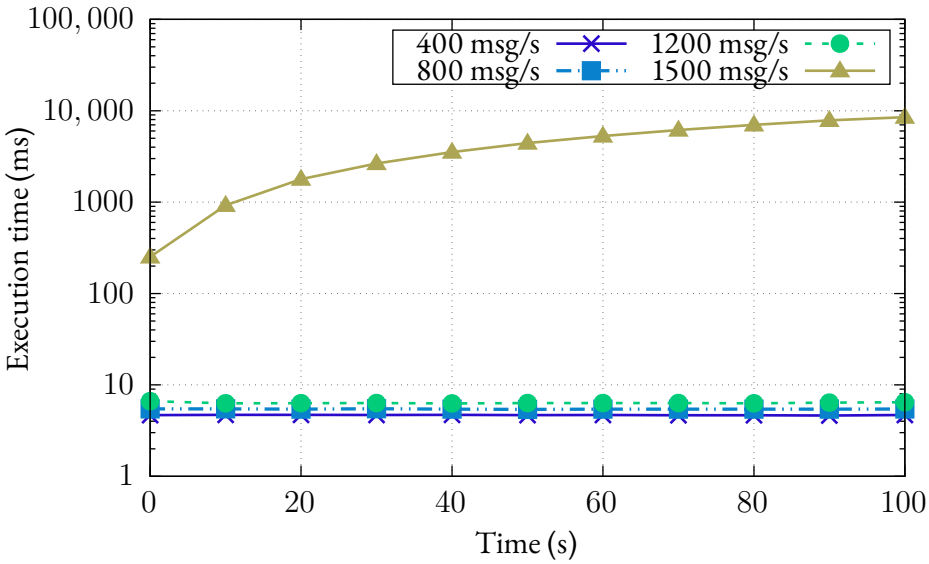


Figure 6.22: Machine-to-consumer communication latency under varying message load of the IT layer.

Figure 6.22 shows the end-to-end latency measured at the IT level for the same message rate above, by exhibiting a latency that is an order of magnitude higher than the one sensed in the OT level. That increase is expected when considering both the number of software

components the message must traverse, and the latency introduced by the Kafka MOM features. The latter has been configured to manage the forwarding of the messages to the consumer by imposing a total ordering and ensuring exactly one semantics (single topic/single partition) which is particularly important when conveying safety-critical information from the OT. The effects of the above-mentioned semantics are clearly visible in the 1500 message/second configuration, causing up to an exponential growth in latency. In fact, in this setting, the rate mismatch of servicing input data, marshaling of messages to IT-layer compliant format, and their emission to the respective output queue, creates an increasing backlog of messages over time. This trend gives evidence that not all data and information exchanged in the OT could be sent to IT whenever low latency and no data loss are requirements. To solve that mismatch, the OT layer can be equipped with selective pre-processing capability, specifically using filtering and aggregation, to better coordinate the different layers and to alleviate the burden at the OT/IT bridging point.

That line is also confirmed by data shown in Figure 6.23, showing the Gateway CPU usage, evidencing an increase in CPU usage trend, augmenting together with the increase of message arrival rate, while still having plenty of resources that could be devoted to other computational tasks.

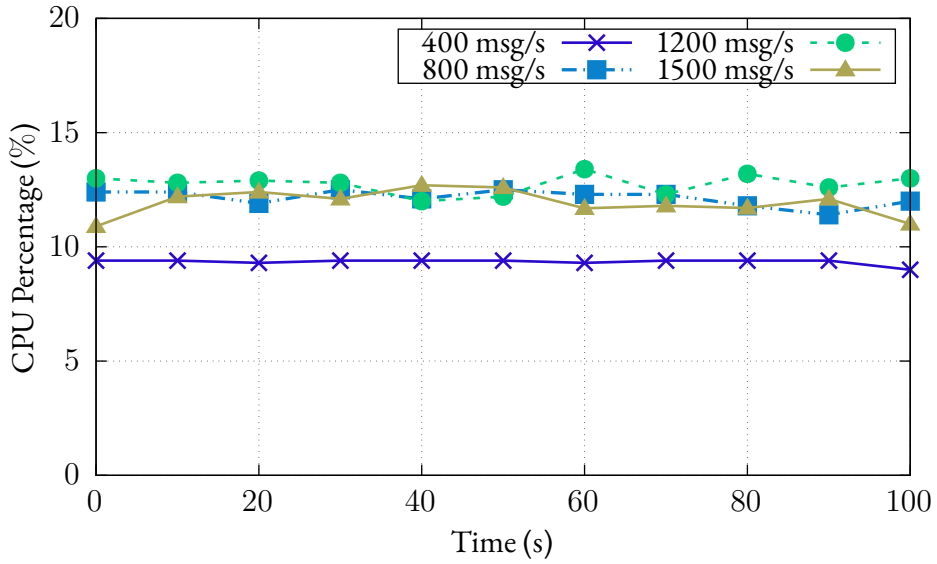


Figure 6.23: Gateway CPU usage under varying message loads.

In conclusion, this work presented a set of architectures that aim to tackle the main integration issues in the IT/OT convergence in the industrial and manufacturing sector.

By leveraging the potentialities of Edge and Big Data technologies, we proposed architectures that respect the privacy, safety, and security constraints imposed by manufacturing companies. The experimental results confirm the effectiveness of our approach, showing sub-millisecond latencies at the OT layer and greater flexibility at the IT layer. Additionally, it's worth mentioning that container orchestration is an important aspect to consider when managing a large number of containers sparse in several production environments, and it is something that could be further explored in the next work that hides container management issues exploiting the capabilities of Serverless processing.

6.5 Serverless Processing at the Edge

Motivated by a data sharing scenario, extending SIRDAM4.0 conceptual architecture, we present the design of an OT/IT convergence solution equipped with edge processing capabilities [14]. The edge computing capability could alleviate the risks, providing an e.g., cloud-free, integration of the plant floor data (OT) with the enterprise resource planning system(s) (IT). This resulting integration layer is fed with data through an event-driven architecture, acting as a data conveyor from bottom to top layers, inhibiting direct exposure of the OT layer by design e.g., malicious access to machines. To cope with potential traffic spikes and adhering to an event-centric architecture, the edge processing layer embraces a serverless approach exploiting the FaaS paradigm, capable of elastic data processing in the upstream path. In principle, these processing capabilities could be extended and deployed beyond the OT/IT boundary allowing for a fine grained and contextual transformation of data.

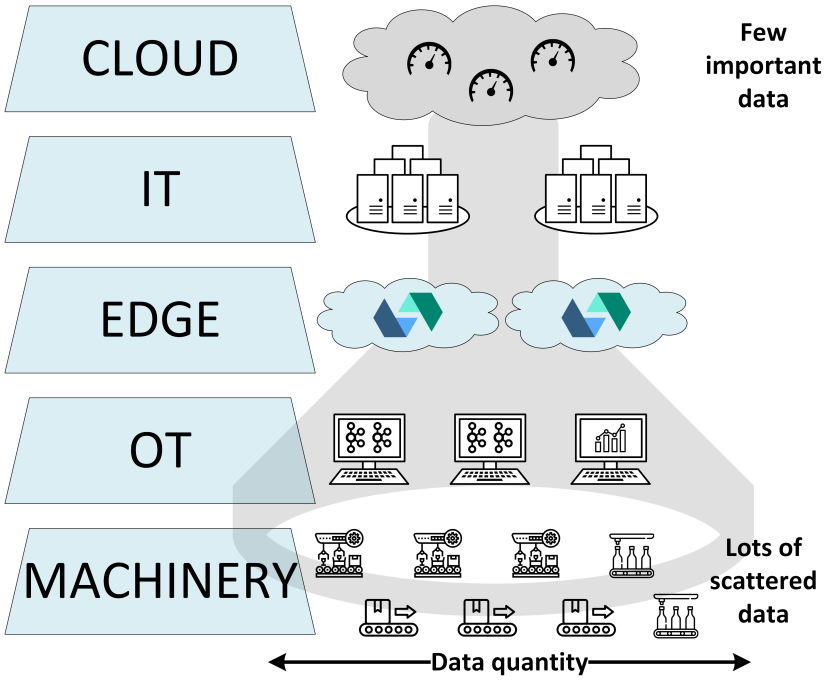


Figure 6.24: Single site OT/IT integration approach.

In Figure 6.24 is depicted the integration approach that we used for this work. Starting from the bottom layer, we find the machines and workers interacting for production purposes. Of paramount importance is the correct configuration of the machines e.g.,

duty cycling regimes, aimed at maximizing the production rate while keeping the plant floor as safe as possible. At this layer, relevant data streams are heterogeneous in terms of type and size, potentially amounting to huge data quantities being generated. Furthermore, a strict data access mechanism is required to prevent data theft and intrusions that could lead to the alteration of the application logic.

Moving upwards, we find the OT layer where the real time data collection generated by CPS actors occurs. Here the data are consumed and gathered for operational considerations, in turn, synthesized and fed to the IT layer for decision making purposes. Concerning the edge processing capabilities, conceptually this functionality resides in the OT/IT boundary but in general, can be distributed virtually anywhere in an upward path, viewed as a vertical stream processing fabric acting on specific data flows. It is important to point out that the data gathering system(s) at the OT layer shall act as a DMZ for the Machine layer so as to shield the plant floor from any attempt to make direct access to machines.

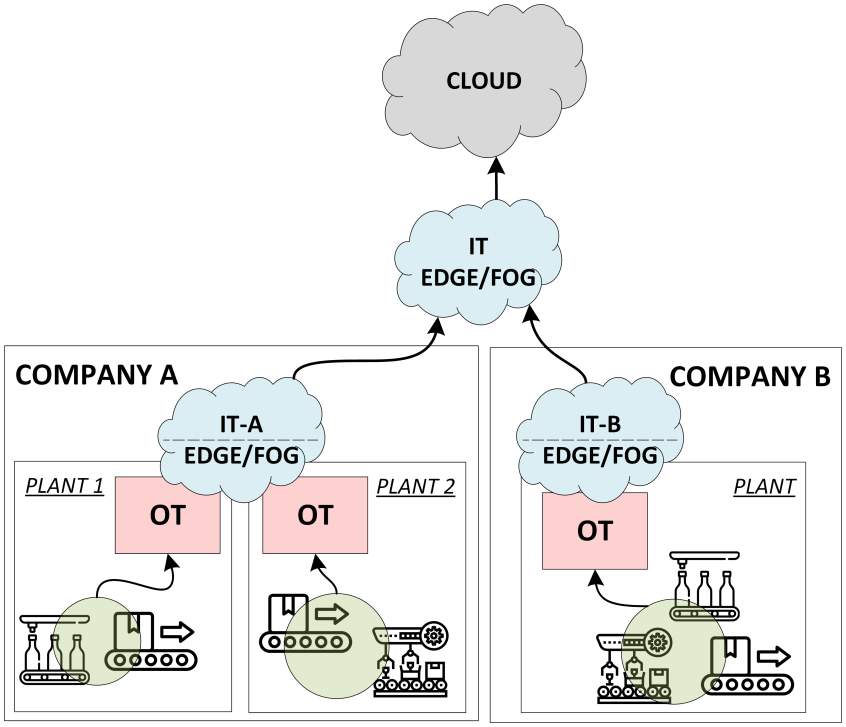


Figure 6.25: A distributed, hierarchical integration approach exploiting an edge/fog compute fabric.

Zooming out from the layered perspective, we can envision a generalized, geographically distributed industrial scenario accounting for different stakeholders as depicted in Figure 6.25. To explain the rationale of the scenario, let us assume that Company A decides to further extend its production line footprint in a different geographical location. In this case, both production lines are run by the same company each with a local deployment of the layered architectural components described above. To gain intelligence, Company A could deploy a central IT office tapping into data sources originating from the different plants. This higher level integration could be achieved through the deployment of an edge/fog node physically collocated with either plant or residing elsewhere. This exercise could be opened to additional competing companies giving rise to an extended ecosystem. The reason for doing so is to allow other market players e.g., machine manufacturing companies, to gain access to machine and/or assembly line operational data generated on their customers (companies) premises. This horizontal integration could pave the road to productive cooperation between different stakeholders, contributing to further better the products and harmonize production lines. However, one needs to acknowledge potential security and privacy risks associated with data disclosure phenomena, valid arguments that refrain companies from disclosing sensitive production data. Enabling the above scenario while meeting the needs of all stakeholders in terms of data confidentiality and quality of service, we propose a technological solution discussed in the next section.

The contribution of this work is three-fold: (i) we present a practical edge computing approach for the OT/IT convergence problem, generalized to contemplate for different stakeholders in the industrial landscape (ii) present the implementation details of a serverless processing solution exploiting the FaaS paradigm (iii) validate the proposal under realistic settings. The remainder of this section is structured as follows: in Architecture Subsection we discuss the implementation details, presenting various system components and how they are glued together, while Section Experiments presents an assessment of the proposal.

6.5.1 Architecture

Relevant technological details concerning the available machine protocols and OT analytics are discussed in Section 6.1. Overall, the functionalities offered by the platform can be summarized as follows:

- Gathering of data produced by manufacturing machines with support for a variety of protocols and dialects.
- Long-term storage, fast processing and user-friendly presentation of such data at OT layer.
- A serverless edge processing layer servicing operational data flows for use to IT department(s).
- Offline/online processing and selective provisioning of machine data in the upstream path.

The third architecture, depicted in Figure 6.26, reflects the physical separation of OT and IT layers enforced in most production sites and addresses their integration at the data level. Our design follows the interoperability principle reiterated in part 1 and in part 14 of the OPC UA specification, which advises the use of a publish-subscribe communication pattern, allowing for flexible management policies and very low communication latency at the layer boundaries.

The data streams generated at the Machine layer are characterized by high speed, large varieties, and big volumes, due to the number of different machines operating in the plant floor.

Without loss of generality, currently our proposal embodies a best-effort processing capability e.g., of diagnostic data produced by machines, while it also strives to achieve horizontal scalability so that new machines connected to the platform can be properly managed. As often reiterated in RAMI 4.0 standards, companies need proper solutions to manage data in a secure and reliable way, avoiding damages to surrounding people and to the machines themselves during remote operations. We take the OPC UA advice (part 2) to restrict access to this layer in order to achieve a proper level of security and safety achieved by configuring a set of policies enforced by a *Message Oriented Middleware* (MOM) solution.

As a mandatory practice of I4.0 specifications, the OT layer needs to provision very low data latency, good bandwidth, enhanced security mechanisms, and resilience. In this layer, we find a component aimed at collecting data in (near) real-time (*Data Extraction and Transformation*) and the MOM capable of delivering data to consumers in a publish-subscribe fashion, and of guaranteeing low latency and security. Keeping latency low

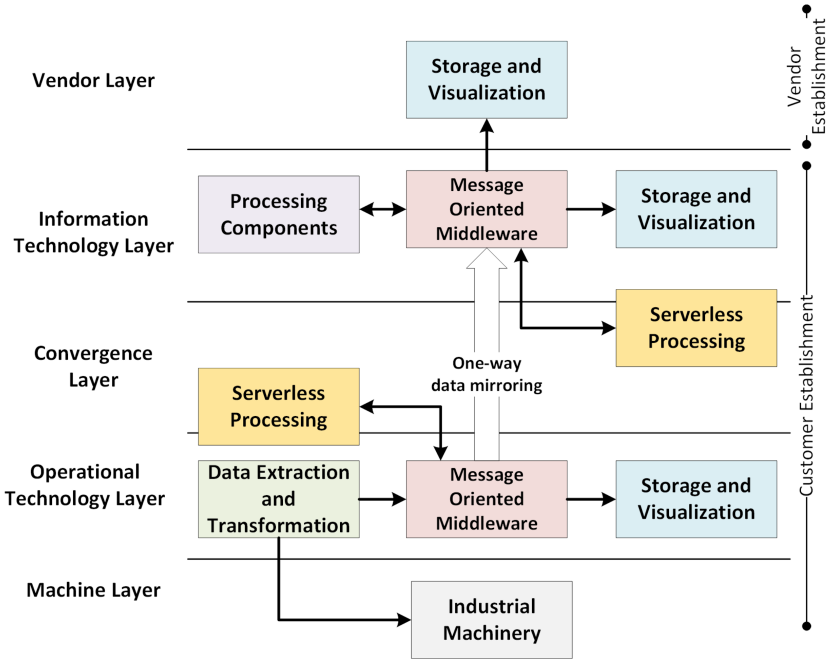


Figure 6.26: Functional components of the platform.

allows the software of this layer (*Storage and Visualization*) to align with the update frequencies of the machines.

On top of the OT layer, the Convergence layer supports the implementation of fine-grained access and control of the simple operations performed on the various data streams. In line with company policies, at this layer one can configure and enforce different data storing and processing functions, that is specify what transformations e.g., windowed min/max, averages, random sampling, etc., can be applied to the data, and when and how data the data can be exchanged between OT and IT layers. The processing capability at this layer can be dynamically turned on and off allowing for a dynamic zoom-in and out of specific data streams originating from the plant floor. At a very basic level, the layer could also serve as a backup of OT generated data, thus ensuring the whole platform a good degree of robustness with respect to potential faults of the Machine layer.

The serverless model adheres to an event-driven approach and allows for a fine-grained scaling and usage of resources with a one-to-one mapping between events (data flows) and processing functions. It consists of a software layer made up of configurable connectors fed with data by external events, triggering the execution of functions either pre-configured or attached dynamically at run-time (later on). Depending on the configuration and type

of processing, the layer can be hosted on commercial of the shelf (COTS) edge nodes or more capable ones exploiting HPC grade resources.

Moving upwards, at the IT layer, multiple stakeholders need to consume different portions of the available data. Data streams might be fetched from persistence support or streamed directly from the lower layer(s). In particular, in the latter scenario, the data are further processed in a serverless fashion applying more advanced operations such as map-reduce ones. Resources at this layer are dedicated to more complex offline/online analytics.

In specific, the layer is equipped with a MOM component, distributing data according to a publish-subscribe model, and a set of tools and APIs (to be used by data consumers) devoted to the processing, storage, and visualization of data. An additional distinction between OT and IT at a design level lies in the relaxation of the requirements at the IT layer, where no machinery operates and risk concerns are not an issue.

Finally, the platform opens to the involvement of third-party stakeholders, i.e., potential ecosystem partners, that could generate value from production data. In the architectural view, this entity can be collocated at a higher level edge/fog node or in the cloud. This tier collects selected data coming from production sites and runs analytics over it. As a data consumer, the *Storage and Visualization component* is allowed to subscribe only to specific topics published by the IT layer MOM. This mechanism aims at avoiding any leakage of private and confidential company data that do not serve the purposes.

6.5.2 Implementation

We highlight the implementation choices made in our proof of concept implementation of the platform. The OT *Data Extraction and Transformation* components have been discussed and analyzed in the previous Section 6.1. In a nutshell, these components are responsible for extracting and transforming data from a machine-specific implementation to a MOM standard representation.

Concerning the data flow from the bottom to upper layers, we rely on a MOM solution implementing a many-to-many, publish-subscribe communication paradigm. This component consists of an Apache Kafka [98] instance granting potential low latency and high throughput. The MOM is sized to collect all the data coming from machinery and, depending on the layer it operates and configuration, keeps a raw backup copy of the data. The MOM solution deployed on the *OT-Convergence* layer boundary and the one deployed

on the *Convergence-IT* layer boundary could be the same instance or separate instances running on different systems. The flow of data is guided by the use of pre-configured topics and specific policies.

As a *Storage and Visualization* solution we adopt the Elasticsearch-Logstash-Kibana (ELK) stack [32] allowing for a fine-grained visualization of the data via dashboards. In the current version, the serverless processing is implemented with an all-in-one node deployment of Apache Openwhisk [101] optimized to operate on a limited resource scenario i.e., COTS edge/fog node, with short living incoming tasks at a high rate. Openwhisk uses container management technology to dynamically scale up/down resources, creating ephemeral environments in which to execute the functions.

The IT MOM component is specialized to scale depending on the traffic generated from the connected OT layer(s). Again, thanks to the advanced clustering capabilities, enhanced performances, and a wide active community, our choice is Kafka. The IT serverless processing component is provisioned as a clustered full-featured installation of Apache Openwhisk configured to operate on HPC hardware, exploited for computational intensive on-line and batch operations on a huge quantity of data e.g., map-reduce like tasks.

The integration between Openwhisk and Kafka is achieved through the use of adapter components configured as back-end services in the Openwhisk platform, able to ingest data from the Kafka middleware, trigger the processing of containerized functions. The output of this processing can be resent on the same or in a different Kafka topic. The quality, the frequency of ingestion, and the delivery of those messages are customizable and adaptable to the different scenarios of IT and OT.

On a last note, the Kafka ACL mechanism can be defined per stream through fine-grained policies, granting a desired security of the system.

6.5.3 Experiments

The main objective of our evaluation is to assess the feasibility of our platform and, in particular, of the serverless processing component at the Convergence layer. In the following, we provide first some details about the experimental testbed, then we show collected results and we conclude with a discussion of the experimentation results.

We have set up a testbed comprised of three virtual nodes equipped, respectively, with 24, 16, 12 GB of memory and 8, 8, 4 virtual CPUs to emulate the OT, Convergence,

and IT layers. The Convergence layer hosts an instance of the OpenWhisk FaaS software module configured in standalone mode. The machine traffic is generated by an emulator obtained through a collaboration with a company situated in the *Packaging Valley* district in the region of Emilia Romagna, Italy. In this analysis, the amount of traffic generated mimics that of several machines and the objective is to evidence some characteristics and dynamics of the layered architecture.

The assessment is structured in two experiments, each aimed to validate the edge processing capabilities under different conditions. In both experiments, the FaaS layer is equipped with a random sampling function that filters out only a fraction (1/10) of the messages to traverse the *Convergence-IT* boundary. This operator allows the IT layer to receive a steady amount of traffic for the desired topic(s).

The first experiment simulates a real use case scenario where a hypothetical IT analyst needs to acquire more information on a particular phenomenon and decides to remotely zoom-in into a specific set of data. In this experiment, the zoom-in operation employs a simple identity operator, merely acting as a forwarder of the data towards the IT layer, but other more complex operators are available. On a technical level, the zoom-in operation comprises many steps among which is the deactivation of the random sampling function from a specific data stream (i.e., Kafka topic) and the activation of the new one. The (de)activation process presents itself in different forms depending on the actual conditions of the containerized FaaS environment. In particular, if a container running a prior function is available and not involved in actual processing, the code of the new function can be injected and activated to process incoming traffic, otherwise, a new container image is prepared and activated from scratch. The latter phenomenon is referred to as *cold start* and is more time consuming when compared to the former one.

In the second experiment, only a random sampling strategy is enabled, but we simulate a variable traffic behavior by turning the machines on and off. In this case, we validate the proposal by showing the difference between the traffic generated at the OT and traffic going through the *Convergence-IT* layer boundary.

Performance Assessment

Starting from the first experiment, Figure 6.27 shows the evolution of the traffic quantity and message delay as measured at the IT layer. Up until time T1 only a fraction of the overall messages traverses the *Convergence-IT* boundary due to the random sampling strategy applied at the processing layer. At time T1, we simulate a zoom-in request to a

particular topic and this corresponds to a reconfiguration of the support, through the injection of the new function, and eventually an associated increase in the number of messages at time T2.

At time T3 the zoom-in operation is interrupted and the traffic load entering the IT layer returns to the default configured system behavior.

Referring to the delivery delay, one can observe a spike at time T1 attributed to the *modus operandi* of the FaaS module and in specific to the *cold start* phenomena. Initially, the FaaS environment needs to scale-out to grant the appropriate amount of resources needed to cope with the incoming traffic, reaching a peak in terms of message delay at time T1. At this point forward, a sufficient number of containers are available and the new zoom-in function is injected, recycling available containers. At time T3, the ingress traffic starts a steady decrease and as a consequence the FaaS environment scales-in the resources needed to process the data, leading to the observed additional peak delay at time T4. It is important to note, that in a production-ready environment, several optimizations addressing the cold start phenomena can be put in place in order to reduce the delay variations [51].

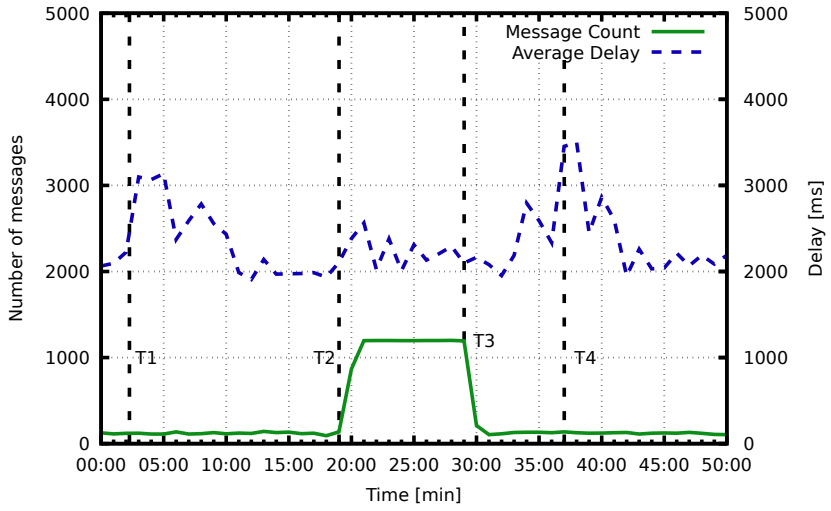


Figure 6.27: Traffic and message delay evolution in time. At time T1 a zoom-in operation is triggered feeding a steady flow of raw process data at the IT layer.

Figure 6.28 shows a complementary view onto the experiment. Initially, only a fraction of the generated data is forwarded to the IT layer until the zoom-in function is enacted. This corresponds to an increase in traffic load entering the IT layer. Similarly, when the

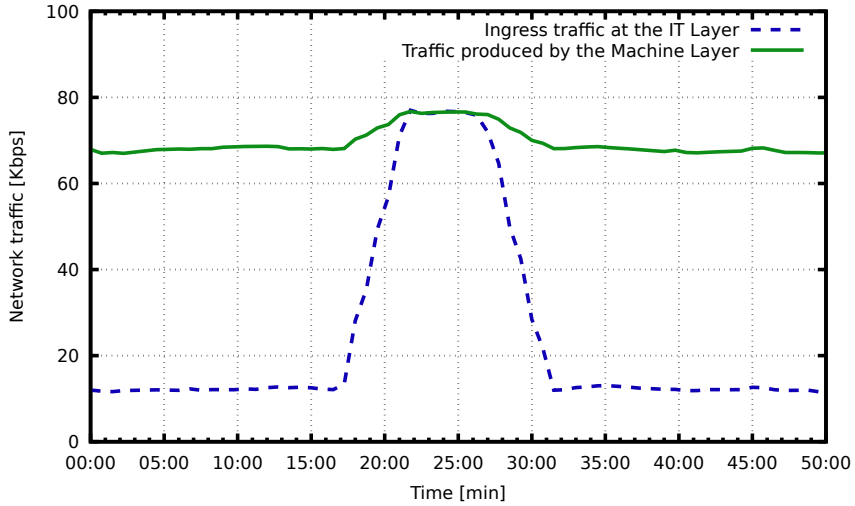


Figure 6.28: Comparison of network traffic at the IT and Machine layer.

default operator is restored, the load at the IT layer gradually decreases until it reaches a steady regime.

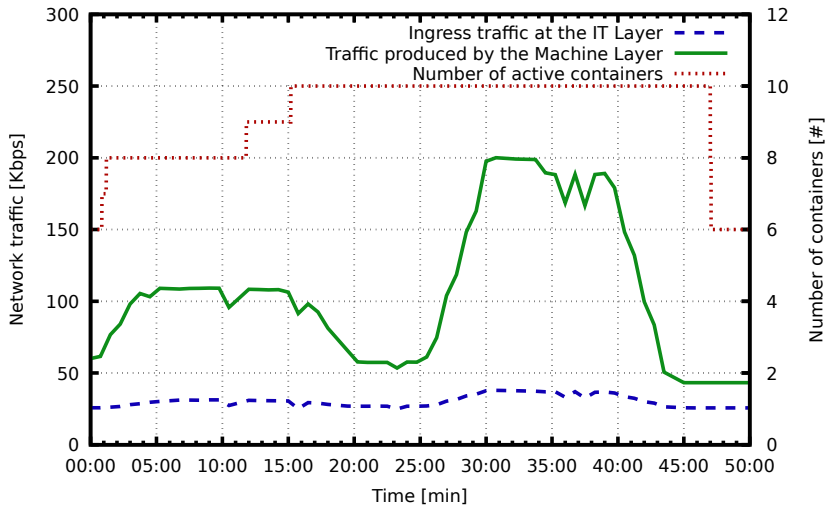


Figure 6.29: Comparison of network traffic at the IT and Machine layer.

In the second experiment, the machines generating the data are randomly turned on/off to emulate a variable traffic behavior. Figure 6.29 shows the network load at different layers of the platform along with the number of containers active in the FaaS environment (Convergence layer). The traffic at the Machine layer fluctuates over time while this

difference is less felt at the IT layer due to the adopted random sampling strategy. In correspondence, the FaaS environment scales-in accordingly in order to cope with the incoming traffic, anticipating the traffic spikes, gradually scaling out the resources.

7 Key Findings

To address the challenges at both the OT and IT levels, we propose novel and general models for collecting, ingesting, processing, and displaying data from production machines. Through our experience working closely with manufacturing companies, we have gained insights and developed a general platform, as outlined in the following.

First, the platform must have **high performance** to handle the fast and varied streams of I4.0 information. Industrial production sites generate a high volume of big data that express the current operating parameters of the machines on the shop floor. Therefore, advanced tools are needed to build an I4.0 platform that can handle large data volumes. The proposed architectures, detailed in Chapter 6, use cutting-edge technologies such as Apache Kafka, Rancher, Kubernetes, OpenFlow, P4, and OpenWhisk to ensure efficient data ingestion, service orchestration, network flows management, in-network processing, and serverless processing capabilities.

Second, **security and safety** are essential in any shop floor environment. By ensuring high performance at the machine level, we can locally perform actuation actions while maintaining the safety of workers and machines. Separating the machine level from higher layers also improves security and safety, as well as reduces costs and resource consumption. Additionally, information security must protect proprietary data extracted from machines that vendors want to keep private. Strict access mechanisms in this layer prevent malicious intrusions and theft or alteration of application logic that could lead to dangerous situations or information leaks.

Third, the platform should **facilitate the transition to I4.0** by allowing manufacturing companies to keep their old equipment in operation without a complete replacement. The neat layer separation allows for customized data extractors for higher layers without replacing the existing ecosystem.

Fourth, the platform must **divide duties and information** by reporting only a subset of OT data in the upper layers. The OT layer must store all data from machines, creat-

ing a detailed historical database of all machine operating parameters. This division of information reflects the different needs of technical and managerial departments.

Finally, the platform should **replicate information and business logic** at different layers to increase scalability and flexibility, while also reducing costs and resource consumption. This allows for the efficient use of resources and the ability to adapt to changing business needs.

The architectures shown in Sections 6.2, 6.3, 6.4, and 6.5 demonstrate how it is possible to create software architectures capable of responding to the requirements imposed by the I4.0 and upcoming I5.0 revolutions.

We presented the design and implementation of SIRDAM4.0 in 6.2, a platform that supports large-scale data gathering for industrial operations. The platform is designed to be fast, scalable, controlled, and robust in providing access to information. We created a geographically distributed test bed that mimics a scenario of small and medium-sized enterprises owning production plants in different locations. These plants use a combination of modern (IIOT-based) and legacy (SCADA-based) communication protocols. Our tests have shown that the platform can handle the constraints imposed by the convergence of these technologies, including timely access to data, secure and selective access to information, and tolerance to unexpected faults. Additionally, we used open-source tools to build the software prototype of the platform, which not only keeps costs low for adopters but also makes the platform easily extensible and reusable.

The QoS semantic routing platform in 6.3 presented a new solution using SDN, MOM, and INP to manage traffic flow in an Industry 4.0 environment. It uses gateways near industrial equipment to add semantic information to packets sent to the MOM broker, including location, desired QoS, and content. SDN modules on network elements then use these tags to perform detailed management of each traffic flow. The architecture and protocol suite were designed to meet demanding QoS requirements and are suitable for modern manufacturing environments, offering an overlay networking solution that works at both the application and network layers.

We then presented a practical solution for addressing the issue of OT/IT convergence that respects strict M2M latency requirements in 6.4. The solution utilizes a two-layered MOM approach that acts as an interoperability layer for OT and enables the efficient transfer of large amounts of data to IT, exploiting the best features of OPC-UA Pub/Sub and Apache Kafka. The proposal was tested in a real-world environment using actual machine data and was shown to effectively handle increasing data volume.

The last work presented a solution to the problem of integrating OT and IT by using a conceptual edge/fog node as a Convergence layer with storage and processing capabilities in 6.5. A general approach was proposed for extending the data-sharing ecosystem to other industry stakeholders, using a hierarchical edge/node structure with processing capabilities. The platform is based on a MOM component that acts as a conveyor belt for data, which can be processed and refined at different layers of the architecture using an event-centric serverless processing model that can scale dynamically and autonomously. This last proposal was also tested in a real-world environment using real data, and the results showed its feasibility in a manufacturing scenario.

8 Conclusions and Future Works

Industry 4.0 represents a significant shift in the manufacturing industry, with the potential to revolutionize the way companies operate. By leveraging mature Information Technologies such as IoT, Cloud Computing, SDN, and Big Data, companies can digitize their operations and achieve a new level of efficiency, adaptability, and customer satisfaction. The COVID-19 pandemic has accelerated the adoption of Industry 4.0 technologies and practices, as companies aim to increase their resilience and adapt to changing market conditions. The integration of digital technologies has allowed for remote monitoring and control of production processes, reduced the dependence on manual labor, and improved supply chain visibility. The resulting increased adaptation and agility have proved crucial in enabling companies to quickly respond to shifts in demand and supply chain disruptions caused by the pandemic. By exploiting the capabilities of Industry 4.0, companies can not only improve their operations in the short term but also enhance their competitiveness and ability to respond to future challenges in the long term.

Industry 4.0 encompasses the entire lifecycle of a product, from production to sale and beyond, allowing for advanced services and highly customizable products. Thanks to the integration of product usage analytics, and third-party systems and services into the manufacturing process companies can reduce costs, improve quality and foresee changes in goods demand. The development of novel decentralized data gathering, analytics, and processing platforms grants support to advanced pre-sale and after-sale services that increase the overall product value and service revenue.

This dissertation addressed the main challenges of OT/IT integration, by proposing a set of architectures that take advantage of Edge and Big Data technologies while also adhering to the privacy, safety, and security requirements of manufacturing firms. Nowadays, manufacturing plants not only produce a deluge of non-mission-critical traffic but also include mobile equipment. This leads to a challenging domain, where computing and

networking resources within and among control and shop floor subnets are no longer as abundant as in the past and potentially adopt different (even proprietary) protocols.

The proposed platforms aim to create a homogeneous data gathering and processing environment by using software-defined networks, in-network processing elements, serverless processing at the edge, and gathering data from machinery using low-latency communication protocols. The shown technologies can be selected based on the specific industry needs or in conjunction to maximize the configurability and flexibility of the architecture. Thanks to a close collaboration with companies in the Emilia-Romagna region of Italy, we demonstrate that these architectures adhere to the requirements, and can provide significant benefits and competitiveness to small and medium-sized enterprises.

The conclusions of this dissertation suggest that while the research presented represents progress towards the goal of Industry 4.0, there is still much work to be done to fully realize the potential of the fourth industrial revolution. Future studies should aim to boost the security and dependability of Industry 4.0 architectural proposals, while also seeking innovative ways to utilize generated data for business expansion and new prospects. The European Union's NextGenerationEU digital plan targets to raise the digitalization level, introduce faster and more trustworthy 5G connections and enhance healthcare, transportation, and education. The industrial sector must not fall behind and seize this one-of-a-kind chance by integrating these advancements.

In particular, it is crucial to explore the usage of federated approaches, allowing for the efficient dispatching of MOM-based messages between different Industry 4.0 domains. The experimental testbeds should be extended to include wider and more complex topologies, encompassing globally distributed firms and allowing for dynamic deployment and adoption of new industrial machines and control room components, as well as supporting federation among different manufacturing plants.

Although I4.0 may seem like an arrival point, it is one of the starting and integration points towards Industry 5.0. The latter focuses on the integration of human-centered, resilience, self-management, and ethical values with industrial technologies, allowing advanced scenarios and paving the way to an ecosystem of interconnected, sustainable, and green services. Overall, this thesis highlights the potential of Industry 4.0 to transform the manufacturing industry, anticipating the upcoming Industry 5.0, and emphasizing the importance of continued research and development in this field.

Acronyms

3D	Three Dimensional
5G	5th Generation
aaS	as a Service
ACID	Atomicity, Consistency, Isolation, and Durability
AGW	Application Gateway
AI	Artificial Intelligence
AM	Additive Manufacturing
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AR	Augmented Reality
AWS	Amazon Web Services
CAD	Computer-Aided Design
CLI	Command Line Interface
CRM	Customer Relationship Management
DT	Digital Twin
EDIH	European Digital Innovation Hub
FaaS	Function as a Service
FIFO	First-in-first-out
HMI	Human-Machine Interface
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
I4.0	Industry 4.0
I5.0	Industry 5.0
IaaS	Infrastructure as a Service
ICS	Industrial Control System
ICT	Information and Communication Technologies

IIoT	Industrial Internet of Things
INP	In-Network Processing
IoT	Internet of Things
IP	Internet Protocol address
IT	Information Technology
JSON	JavaScript Object Notation
LAN	Local Area Network
M2M	Machine to Machine
MES	Manufacturing Execution System
MMC	Machine Manufacturer Company
MOM	Message-Oriented Middleware
MQTT	Message Queue Telemetry Transport
OPC	Open Platform Communications
ORB	Object Request Broker
OS	Operative System
OT	Operational Technology
PaaS	Platform as a Service
PLC	Programmable logic controller
PPP	Public-Private Partnership
QoS	Quality of Service
R&D	Research and Development
RAMI 4.0	Reference Architectural Model Industrie 4.0
REST	Representational state transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SCADA	Supervisory Control And Data Acquisition
SCARF	SCADA Reader and Forwarder
SDN	Software-Defined Networking
SIRDAM4.0	Support Infrastructure for Reliable Data Acquisition and Management in Industry 4.0
SLA	Service Level Agreement
SME	Small and Medium-sized Enterprise
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol

SoC	System on a Chip
SSH	Secure Shell
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
WW2	Second World War

Bibliography

1. I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck. "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions". *IEEE Communications Surveys Tutorials* 20:3, 2018, pp. 2429–2453. DOI: 10.1109/COMST.2018.2815638.
2. Amazon Web Services Inc. *Fully managed message queuing for microservices*. Last visited in Jan. 2023. URL: https://aws.amazon.com/sqs/.
3. A. Arsanjani. "Service-oriented modeling and architecture". *IBM developer works* 1, 2004, p. 15.
4. astrofrog - GitHub. *Record the CPU and Memory Activity of a Process*. Last visited in Jan. 2023. URL: https://github.com/astrofrog/psrecord.
5. P. Bellavista, F. Bosi, A. Corradi, L. Foschini, S. Monti, L. Patera, L. Poli, D. Scotece, and M. Solimando. "Design Guidelines for Big Data Gathering in Industry 4.0 Environments". In: *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2019, pp. 1–6. DOI: 10.1109/WoWMoM.2019.8793033.
6. P. Bellavista, A. Dolci, and C. Giannelli. "MANET-oriented SDN: Motivations, Challenges, and a Solution Prototype". In: *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2018, pp. 14–22. DOI: 10.1109/WoWMoM.2018.8449805.
7. P. Bellavista, M. Fogli, L. Foschini, C. Giannelli, L. Patera, and C. Stefanelli. "A Framework for QoS-Enabled Semantic Routing in Industrial Networks: Overall Architecture and Primary Protocols". In: *2022 IEEE Future Networks World Forum*. 2022, pp. 1–6.
8. P. Bellavista, M. Fogli, L. Foschini, C. Giannelli, L. Patera, and C. Stefanelli. "QoS-Enabled Semantic Routing for Industry 4.0 based on SDN and MOM Integration". In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. 2021, pp. 1–6. DOI: 10.1109/HPSR52026.2021.9481869.
9. T. A. Bishop and R. K. Karne. "A Survey of Middleware." In: *CATA*. 2003, pp. 254–258.

10. F. Bosi, A. Corradi, G. Di Modica, L. Foschini, R. Montanari, L. Patera, and M. Solimando. "Enabling Smart Manufacturing by Empowering Data Integration with Industrial IoT Support". In: *2020 International Conference on Technology and Entrepreneurship (ICTE)*. 2020, pp. 1–8. DOI: 10.1109/ICTE47868.2020.9215538.

11. F. Bosi, A. Corradi, L. Foschini, S. Monti, L. Patera, L. Poli, and M. Solimando. "Cloud-enabled Smart Data Collection in Shop Floor Environments for Industry 4.0". In: *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*. 2019, pp. 1–8. DOI: 10.1109/WFCS.2019.8757952.

12. D. Brougham and J. Haar. "Smart Technology, Artificial Intelligence, Robotics, and Algorithms (STARA): Employees' perceptions of our future workplace". *Journal of Management & Organization* 24:2, 2018, pp. 239–257. DOI: 10.1017/jmo.2016.55.

13. A. Bujari, A. Calvio, L. Foschini, A. Sabbioni, and A. Corradi. "IPPODAMO: A Digital Twin Support for Smart Cities Facility Management". In: *Proceedings of the Conference on Information Technology for Social Good*. GoodIT '21. Association for Computing Machinery, Roma, Italy, 2021, pp. 49–54. ISBN: 9781450384780.

14. A. Bujari, A. Corradi, L. Foschini, L. Patera, and A. Sabbioni. "Enhancing the Performance of Industry 4.0 Scenarios via Serverless Processing at the Edge". In: *ICC 2021 - IEEE International Conference on Communications*. 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9500286.

15. H. Cañas, J. Mula, M. Díaz-Madroñero, and F. Campuzano-Bolarín. "Implementing Industry 4.0 principles". *Computers & Industrial Engineering* 158, 2021, p. 107379. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2021.107379. URL: https://www.sciencedirect.com/science/article/pii/S0360835221002837.

16. S. Cass. "Chip Hall of Fame: Intel 4004 Microprocessor". *IEEE Spectrum*, 2018.

17. N. M. K. Chowdhury and R. Boutaba. "A survey of network virtualization". *Computer Networks* 54:5, 2010, pp. 862–876. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2009.10.017. URL: https://www.sciencedirect.com/science/article/pii/S1389128609003387.

18. Cloud Foundry. *Open Source Cloud Native Application Delivery*. Last visited in Jan. 2023. URL: https://www.cloudfoundry.org/.

19. Cloudera. *Architectural Patterns for Near Real-Time Data Processing with Apache Hadoop*. Last visited in Jan. 2023. URL: https://blog.cloudera.com/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/.

20. Confluent Inc. *Authorization using ACLs*. Last visited in Jan. 2023. URL: https://docs.confluent.io/platform/current/kafka/authorization.html.

21. Confluent Inc. *Kafka Consumer - Offset Management*. Last visited in Jan. 2023. URL: https://docs.confluent.io/platform/current/clients/consumer.html#offset-management.

22. Control Solutions Minnesota. *Modbus 101 - Introduction to Modbus*. Last visited in Jan. 2023. URL: https://www.csimn.com/CSI_pages/Modbus101.html.

23. Convergence Consulting Srl. *The Packaging Valley*. Last visited in Jan. 2023. URL: https://thepackagingvalley.com/.

24. A. Corradi, G. Di Modica, L. Foschini, L. Patera, and M. Solimando. "SIRDAM4.0: A Support Infrastructure for Reliable Data Acquisition and Management in Industry 4.0". *IEEE Transactions on Emerging Topics in Computing* 10:3, 2022, pp. 1605–1620. DOI: 10.1109/TETC.2021.3111974.

25. E. Curry. "Message-oriented middleware". *Middleware for communications*, 2004, pp. 1–28.

26. P. M. Deane. *The first industrial revolution*. Cambridge University Press, 1979.

27. Docker Inc. *Docker - Develop faster. Run anywhere.* Last visited in Jan. 2023. URL: https://www.docker.com/.

28. Docker Inc. *Swarm mode overview*. Last visited in Jan. 2023. URL: https://docs.docker.com/engine/swarm/.

29. P. Drahoš, E. Kučera, O. Haffner, and I. Klimo. "Trends in industrial communication and OPC UA". In: *2018 Cybernetics & Informatics (K&I)*. IEEE. 2018, pp. 1–5.

30. P. Dutta and P. Dutta. "Comparative study of cloud services offered by Amazon, Microsoft & Google". *International Journal of Trend in Scientific Research and Development* 3:3, 2019, pp. 981–985.

31. Eclipse. *Mosquitto, an open source MQTT broker*. Last visited in Jan. 2023. URL: https://mosquitto.org/.

32. Elasticsearch B.V. *What is the ELK Stack?* Last visited in Jan. 2023. URL: https://www.elastic.co/what-is/elk-stack.

33. EPA - United States Environmental Protection Agency. *Global Greenhouse Gas Emissions Data*. Last visited in Jan. 2023. 2019. URL: https://www.epa.gov/ghgemissions/global-greenhouse-gas-emissions-data.

34. G. Erboz. "How to define industry 4.0: main pillars of industry 4.0". *Managerial trends in the development of enterprises in globalization era* 761, 2017, p. 767.

35. EtherCAT Technology Group. *EtherCAT - the Ethernet Fieldbus*. Last visited in Jan. 2023. URL: https://www.ethercat.org/en/technology.html.

36. European Union. *ANNEX - European Digital Innovation Hubs for 2021 - 2023, Brussels, 10.11.2021 C(2021) 7911 final*.

37. European Union. *Futurium, Your voices, your future*. Last visited in Jan. 2023. URL: https://futurium.ec.europa.eu/en.

38. European Union. *Implementing the Digitising European Industry actions*. Last visited in Jan. 2023. URL: https://ec.europa.eu/digital-single-market/en/digitising-european-industry-digital-day.

39. M. Fera, A. Greco, M. Caterino, S. Gerbino, F. Caputo, R. Macchiaroli, and E. D'Amato. "Towards Digital Twin Implementation for Assessing Production Line Performance and Balancing". *Sensors* 20:1, 2020. ISSN: 1424-8220. DOI: 10.3390/s20010097. URL: https://www.mdpi.com/1424-8220/20/1/97.

40. P. Fraga-Lamas, J. Varela-Barbeito, and T. M. Fernández-Caramés. "Next Generation Auto-Identification and Traceability Technologies for Industry 5.0: A Methodology and Practical Use Case for the Shipbuilding Industry". *IEEE Access* 9, 2021, pp. 140700–140730. DOI: 10.1109/ACCESS.2021.3119775.

41. C. Freeman and F. Louçã. *As time goes by: from the industrial revolutions to the information revolution*. Oxford University Press, 2001.

42. C. B. Frey and M. A. Osborne. "The future of employment: How susceptible are jobs to computerisation?" *Technological Forecasting and Social Change* 114, 2017, pp. 254–280. ISSN: 0040-1625. DOI: https://doi.org/10.1016/j.techfore.2016.08.019. URL: https://www.sciencedirect.com/science/article/pii/S0040162516302244.

43. A. Garbugli and L. Patera. *Source code - PoC IT/OT Convergence Middleware*. Last visited in Jan. 2023. URL: https://github.com/MMw-Unibo/poc-itot-convergence-mw.git.

44. Gartner Inc. *When IT and Operational Technology Converge*. Last visited in Jan. 2023. URL: https://www.gartner.com/smarterwithgartner/when-it-and-operational-technology-converge/.

45. S. Ghosh and S. Sampalli. "A Survey of Security in SCADA Networks: Current Issues and Future Challenges". *IEEE Access* 7, 2019, pp. 135812–135831.

46. M. Haseeb, H. I. Hussain, B. Ślusarczyk, and K. Jermsittiparsert. "Industry 4.0: A Solution towards Technology Challenges of Sustainable Business Performance". *Social Sciences* 8:5, 2019. ISSN: 2076-0760. DOI: 10.3390/socsci8050154. URL: https://www.mdpi.com/2076-0760/8/5/154.

47. G. Hesse, C. Matthies, and M. Uflacker. "How Fast Can We Insert? An Empirical Performance Evaluation of Apache Kafka". In: *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. 2020, pp. 641–648. DOI: 10.1109/ICPADS51040.2020.00089.

48. IBM Inc. *IBM MQ*. Last visited in Jan. 2023. URL: https://www.ibm.com/products/mq.

49. Iron.io. *Open Source Serverless Computing*. Last visited in Jan. 2023. URL: https://open.iron.io/.

50. A. ISA. "ISA-95.00. 03-2005 Enterprise Control System Integration Part 3: Activity Models of Manufacturing Operations Management, ISA-The Instrumentation". *System, and Automation Society*, 2005.

51. E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, et al. "Cloud programming simplified: A berkeley view on serverless computing". *arXiv preprint arXiv:1902.03383*, 2019.

52. Knative Authors. *Serverless Containers in Kubernetes environments.* Last visited in Jan. 2023. URL: https://knative.dev/docs/.

53. Kubernetes. *Production-Grade Container Orchestration*. Last visited in Jan. 2023. URL: https://kubernetes.io/.

54. Y. Liao, F. Deschamps, E. d. F. R. Loures, and L. F. P. Ramos. "Past, present and future of Industry 4.0-a systematic literature review and research agenda proposal". *International journal of production research* 55:12, 2017, pp. 3609–3629.

55. MarketsandMarkets. *Artificial Intelligence Market - Report Code: TC 7894*. Last visited in Jan. 2023. 2022. URL: https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-market-74851580.html.

56. MarketsandMarkets. *Big Data Market - Report Code: TC 1521*. Last visited in Jan. 2023. 2022. URL: https://www.marketsandmarkets.com/Market-Reports/big-data-market-1068.html.

57. MarketsandMarkets. *Industrial Robotics Market - Report Code: SE 2733*. Last visited in Jan. 2023. 2022. URL: https://www.marketsandmarkets.com/PressReleases/industrial-robotics.asp.

58. MarketsandMarkets. *IoT Technology Market - Report Code: SE 2509*. Last visited in Jan. 2023. 2021. URL: https://www.marketsandmarkets.com/Market-Reports/iot-application-technology-market-258239167.html.

59. A. Massaro. "Information Technology Infrastructures Supporting Industry 5.0 Facilities". In: *Electronics in Advanced Research Industries: Industry 4.0 to Industry 5.0 Advances*. 2022, pp. 51–101. DOI: 10.1002/9781119716907.ch2.

60. Microsoft. *[MS-DCOM]: Distributed Component Object Model (DCOM) Remote Protocol*. Last visited in Jan. 2023. URL: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/4a893f3d-bd29-48cd-9f43-d9777a4415b0.

61. Microsoft. *Azure Service Bus*. Last visited in Jan. 2023. URL: https://azure.microsoft.com/en-us/products/service-bus/.

62. L. I. Minchala, S. Ochoa, E. Velecela, D. F. Astudillo, and J. Gonzalez. "An open source SCADA system to implement advanced computer integrated manufacturing". *IEEE Latin America Transactions* 14:12, 2016, pp. 4657–4662.

63. J. Mokyr and R. H. Strotz. "The second industrial revolution, 1870-1914". *Storia dell'economia Mondiale* 21945:1, 1998.

64. D. P. F. Möller, H. Vakilzadian, and R. E. Haas. "From Industry 4.0 towards Industry 5.0". In: *2022 IEEE International Conference on Electro Information Technology (eIT)*. 2022, pp. 61–68. DOI: 10.1109/eIT53891.2022.9813831.

65. Mordor Intelligence. *Cloud enterprise management market - growth, trends, covid-19 impact, and forecasts (2023 - 2028)*. Last visited in Jan. 2023. URL: https://www.mordorintelligence.com/industry-reports/cloud-enterprise-content-management-market.

66. Mordor Intelligence. *Cybersecurity market - growth, trends, covid-19 impact, and forecasts*. Last visited in Jan. 2023. URL: https://www.mordorintelligence.com/industry-reports/cyber-security-market.

67. D. C. Mowery. "Plus ca change: Industrial R&D in the "third industrial revolution"". *Industrial and corporate change* 18:1, 2009, pp. 1–50.

68. J. M. Müller, D. Kiel, and K.-I. Voigt. "What Drives the Implementation of Industry 4.0? The Role of Opportunities and Challenges in the Context of Sustainability". *Sustainability* 10:1, 2018. ISSN: 2071-1050. DOI: 10.3390/su10010247. URL: https://www.mdpi.com/2071-1050/10/1/247.

69. N. Naik. "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP". In: *2017 IEEE international systems engineering symposium (ISSE)*. IEEE. 2017, pp. 1–7.

70. nuclio. *Automate the Data Science Pipeline with Serverless Functions.* Last visited in Jan. 2023. URL: https://nuclio.io/.

71. OASIS. *Advanced Message Queuing Protocol*. Last visited in Jan. 2023. URL: https://www.amqp.org/.

72. OASIS. *MQTT: The Standard for IoT Messaging*. Last visited in Jan. 2023. URL: https://mqtt.org/.

73. OMG - Object Management Group. *OMG Data Distribution Service (DDS)*. Last visited in Jan. 2023. URL: https://www.omg.org/spec/DDS/1.4/PDF.

74. *ONF SDN Evolution*. ONF TR-535. Version 1.0. Open Networking Foundation. 2016. URL: https://opennetworking.org/wp-content/uploads/2013/05/TR-535%5C_ONF%5C_SDN%5C_Evolution.pdf.

75. OPC Foundation. *OPC Unified Architecture*. Last visited in Jan. 2023. URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture.

76. OPC Foundation. *Open Platform Communications Classic*. Last visited in Jan. 2023. URL: https://opcfoundation.org/about/opc-technologies/opc-classic/.

77. OPC Foundation. *The Industrial Interoperability Standard*. Last visited in Jan. 2023. URL: https://opcfoundation.org/.

78. Open Networking Foundation. *OpenFlow Switch Specification*. Last visited in Jan. 2023. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

79. OpenFaaS Ltd. *Serverless Functions, Made Simple.* Last visited in Jan. 2023. URL: https://www.openfaas.com/.

80. OpenStack. *Open Source Cloud Computing Infrastructure*. Last visited in Jan. 2023. URL: https://www.openstack.org/.

81. OpenStack. *OpenStack Components Map - v20221001*. [Image]. 2022. URL: https://www.openstack.org/openstack-map.

82. opsengine - GitHub. *CPU Usage Limiter for Linux*. Last visited in Jan. 2023. URL: https://github.com/opsengine/cpulimit.

83. J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". *IEEE Communications Magazine* 55:5, 2017, pp. 80–87. DOI: 10.1109/MCOM.2017.1600935.

84. L. Orlando. *Il Covid non ferma l'hi tech. Nuovo record per Industria 4.0*. Last visited in Jan. 2023. 2021. URL: https://www.ilsole24ore.com/art/il-covid-non-ferma-l-hi-tech-nuovo-record-industria-40-AEU2E7o.

85. L. Patera, A. Garbugli, A. Bujari, D. Scotece, and A. Corradi. "A Layered Middleware for OT/IT Convergence to Empower Industry 5.0 Applications". *Sensors* 22:1, 2022. ISSN: 1424-8220. DOI: 10.3390/s22010190. URL: https://www.mdpi.com/1424-8220/22/1/190.

86. Plattform Industrie 4.0. *Alignment Report for Reference Architectural Model for Industrie 4.0*. Last visited in Jan. 2023. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-manufacturing.html.

87. Profibus - Profinet. *Profibus*. Last visited in Jan. 2023. URL: https://www.profibus.com/technology/profibus.

88. RedHat. *Ansible*. Last visited in Jan. 2023. URL: https://www.ansible.com/.

89. J. Reeser, T. Jankowski, and G. M. Kemper. "Maintaining HMI and SCADA Systems Through Computer Virtualization". *IEEE Transactions on Industry Applications* 51:3, 2015, pp. 2558–2564.

90. C. Roser. *Industry 4.0 revolutions*. [Image]. 2015. URL: https://commons.wikimedia.org/wiki/File:Industry_4.0.png.

91. Z. Sarfraz, A. Sarfraz, H. M. Iftikar, and R. Akhund. "Is COVID-19 pushing us to the fifth industrial revolution (society 5.0)?" *Pakistan journal of medical sciences* 37:2, 2021, p. 591.

92. S. Saxena and S. Gupta. *Practical real-time data processing and analytics: distributed computing and event proc event processing using Apache Spark, Flink, Storm, and Kafka*. Packt Publishing, 2017.

93. E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund. "Industrial Internet of Things: Challenges, Opportunities, and Directions". *IEEE Transactions on Industrial Informatics* 14:11, 2018, pp. 4724–4734. DOI: 10.1109/TII.2018.2852491.

94. C. Spanring and A. Mayrhofer. *A Uniform Resource Identifier for Geographic Locations ('geo' URI)*. RFC 5870. 2010. DOI: 10.17487/RFC5870. URL: https://www.rfc-editor.org/info/rfc5870.

95. K. Stouffer and J. Falco. *Guide to supervisory control and data acquisition (SCADA) and industrial control systems security*. 2006.

96. M. Sverko, T. G. Grbac, and M. Mikuc. "SCADA Systems With Focus on Continuous Manufacturing and Steel Industry: A Survey on Architectures, Standards, Challenges and Industry 5.0". *IEEE Access* 10, 2022, pp. 109395–109430. DOI: 10.1109/ACCESS.2022.3211288.

97. N. Tapoglou, J. Mehnen, and J. Butans. "Energy Efficient Machining Through Evolutionary Real-Time Optimization of Cutting Conditions on CNC-Milling Controllers". In: *Experiments and Simulations in Advanced Manufacturing*. Springer, 2021, pp. 1–18.

98. The Apache Software Foundation. *Apache Kafka*. Last visited in Jan. 2023. URL: https://kafka.apache.org/.

99. The Apache Software Foundation. *Apache Mesos*. Last visited in Jan. 2023. URL: https://mesos.apache.org/.

100. The Apache Software Foundation. *OpenLambda*. Last visited in Jan. 2023. URL: https://github.com/open-lambda/open-lambda.

101. The Apache Software Foundation. *OpenWhisk - Open Source Serverless Cloud Platform*. Last visited in Jan. 2023. URL: https://openwhisk.apache.org/.

102. The P4 Language Consortium. *P4 Language Specification*. Last visited in Jan. 2023. URL: https://p4.org/p4-spec/docs/P4-16-v1.2.1.html.

103. S. Vinoski. "CORBA: Integrating diverse applications within distributed heterogeneous environments". *IEEE Communications magazine* 35:2, 1997, pp. 46–55.

104. VMware, Inc. *RabbitMQ*. Last visited in Jan. 2023. URL: https://www.rabbitmq.com/.

105. M. Wollschlaeger, T. Sauter, and J. Jasperneite. "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0". *IEEE Industrial Electronics Magazine* 11:1, 2017, pp. 17–27. DOI: 10.1109/MIE.2017.2649104.

106. K. Zhou, T. Liu, and L. Zhou. "Industry 4.0: Towards future industrial opportunities and challenges". In: *2015 12th International conference on fuzzy systems and knowledge discovery (FSKD)*. IEEE. 2015, pp. 2147–2152.