TÉCNICO LISBOA

Concept and Evaluation of a Technology-independent Data Acquisition Architecture for *Industrie 4.0*

Pedro João Costa Pinto Prata

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. Birgit Vogel-Heuser
Prof. Susana Margarida da Silva Vieira

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Susana Margarida da Silva Vieira
Members of the Committee: Prof. João Rogério Caldas Pinto
Prof. José Barata Oliveira

June 2019

Acknowledgments

I want to thank to Mr. Emanuel Trunzer, for introducing me to the topic and for teaching me a plethora of new concepts. Also, I am grateful for all the detailed explanations, but mostly for all the rhetorical questions that made me think. It was a huge pleasure to develop the research work at TUM with such support and trust.

Second, I want to thank my supervisor, Prof. Susana Vieira, for all the guidance and availability during my time in Munich.

I would also like to thank Prof. Birgit Vogel-Heuser and the Institute of Automation and Information Systems, for allowing an unknown student to develop his Master Thesis there. Thank you for all the help I received, it made me feel very welcomed in Germany.

Fourth, I would like to acknowledge my friends, that were always there for me throughout this long work. In particular, I would like to thank to Catarina Oliveira, Miguel Chastre, Filipe Amaral, Joana Fonseca, João Pinto, André Pereira and Clara Aparício, for crossing half a continent just to make sure I was working a lot in Munich.

Finally, I want to thank my family. Their everyday support will always be crucial, and I am proud to have you in my life. A final word of appreciation goes to my Mother, since without her none of this would have been possible. I am forever grateful.

Abstract

The fourth industrial revolution, *Industrie 4.0*, motivates research to adapt concepts from Internet of Things and Services and Cyber-Physical Systems to automation. Focus is placed on integrating all data sources from sensors and actuators to management software, to achieve interoperability of heterogeneous systems.

To assure data integration, one should decide on a data acquisition architecture that guarantees data access from all systems, including legacy devices, allowing companies to gradually evolve to *Industrie 4.0* deployments. A middleware-based architecture is a valid answer to the presented challenges.

There is a substantial number of communication technologies implemented in middleware systems, with different specifications and advantages. A technology-independent implementation ensures that migration between communication protocols is eased and makes the architecture more robust and adequate for numerous use-cases.

Two hypotheses are proven in this thesis: a middleware-based data acquisition architecture can be conceptualised and implemented independently of the communication technology; and such implementation ensures a reduction in complexity and effort, when compared with a peer-to-peer legacy approach.

With this thesis, five main contributions are presented. First, a new communication technology, one that abstracts other protocols to ensure technology independence, is created. Following, the concept of a technology-independent data acquisition architecture was created, with its successful implementation being the third contribution. A complexity and effort comparison was done, proving that the developed architecture as lower concept complexity and requires less deployment and migration effort than a legacy approach. The last contribution is a comparison of relevant communication technologies in the field of automation.

Keywords:

Industrie 4.0, Automated Production Systems, Industrial Internet of Things, Middleware Technologies, Data Acquisition Architecture, Communication Technologies

Resumo

A quarta revolução industrial, *Industrie 4.0*, assenta na adaptação, à área da automação, de conceitos das áreas de Internet das Coisas e Serviços e de Sistemas Ciber-Físicos. A integração de fontes de dados, desde sensores e actuadores, até programas de gestão, revela-se prioritária para obter interoperabilidade.

Para optimizar a integração de informação e permitir que as empresas evoluam gradualmente para a *Industrie 4.0*, uma arquitectura de aquisição de dados deve garantir acesso a todos os sistemas, incluindo antigos, o que pode alcançar-se baseando-a em *middleware*.

Existem diversas tecnologias de comunicação implementadas em sistemas de *middleware* com diferentes características. Uma implementação independente da tecnologia facilita a migração entre essas tecnologias, tornando a arquitectura mais robusta e flexível.

No estudo efectuado comprovaram-se duas hipóteses: que pode ser conceptualizada e implementada uma arquitectura de aquisição de dados independente da tecnologia de comunicação e que tal implementação permite uma efectiva redução em complexidade e esforço, comparada com uma abordagem clássica ponto-a-ponto.

Concluído o estudo, extraem-se cinco contribuições. Em primeiro lugar, a criação de uma nova tecnologia de comunicação que permite abstrair diferentes protocolos e garante a independência da tecnologia. A conceptualização de uma nova arquitectura de aquisição de dados independente da tecnologia e a sua implementação, com sucesso, representam as duas contribuições seguintes. A quarta é a demonstração de que a arquitectura é conceptualmente menos complexa e requer menor esforço de implementação e migração, comparando-a com uma implementação clássica. A última contribuição consiste numa comparação entre tecnologias de comunicação relevantes para a automação.

Palavras-chave:

Indústria 4.0, Sistemas de Produção Automatizados, Internet das Coisas Industrial, Tecnologias de *Middleware*, Arquitectura de Aquisição de Dados, Tecnologias de Comunicação

Contents

List of Tables

List of Figures

List of Abbreviations

AMQP Advanced Message Queuing Protocol. 37, 38, 44, 56, 64

aPS Automated Production System. 1, 15, 16, 24, 25, 55

CC Cyclomatic Complexity. 52, 53, 59

CoAP Constrained Application Protocol. 39, 40

CPPS Cyber-Physical Production System. 8, 22, 25

CPS Cyber-Physical System. 1, 2, 7, 8, 23, 24

CT Communication Technology. 1, 4, 6, 11, 13, 16, 17, 23, 24, 27, 29–40, 43, 44, 46, 47, 56, 57, 59–61, 63–66

DDS Data Distribution Service. 37, 40, 57

DLL Dynamic Link Library. 43, 47, 51, 59

DTLS Datagram Transport Layer Security. 39

ERP Enterprise Resource Planning. 3, 8, 16, 22, 32, 38

ESB Enterprise Service Bus. 10, 11, 21–26

HMI Human-Machine Interface. xv, 21, 31, 33, 34, 48–50, 55, 56, 59, 64

HTTP Hyper Text Transfer Protocol. 38, 39

iESB intelligent Enterprise Service Bus. 21, 22, 26

IETF Internet Engineering Task Force. 39

IIRA Industrial Internet Reference Architecture. 12

IoS Internet of Services. 7

IoT Internet of Things. 2, 5, 7, 12, 15

IoT RA Internet of Things Reference Architecture. 12

JSON JavaScript Object Notation. 46, 48

LoC Lines of Code. 52, 53, 59

M2M Machine-to-Machine. 23, 36, 39

MAD Median Absolute Deviation. 34

MES Manufacturing Execution System. 3, 8, 16, 32–35, 49, 50, 55, 60, 64

MI Maintainability Index. 52, 53, 59

MOM Message-Oriented Middleware. 1, 3–5, 10–13, 16, 21, 23, 24, 27, 31, 32, 35, 56–58, 60, 63–65

MPS Modular Production System. xv, 32, 34, 48–50, 56

MQTT Message Queuing Telemetry Transport. 38

OEE Overall Equipment Effectiveness. 2, 8, 64

OMG Object Management Group. 37

OPC UA OPC Unified Architecture. 33, 38–40, 48

P2P Peer-to-Peer. 1, 3, 5, 6, 10, 11, 15, 18, 21–27, 35, 43, 50, 55, 57–61, 63–66

PLC Programmable Logic Controller. 3, 7–9, 16, 33, 35, 48

PS Publish-Subscribe. 13, 33, 36–40, 44

QoS Quality of Services. 13, 36–40, 44, 57

RAMI 4.0 Reference Architecture Model for *Industrie 4.0*. 12, 23

REST WS Representational State Transfer for Web Services. 38, 39

RR Request-Response. 13, 33, 36, 38, 39, 48, 51

SCADA Supervisory Control and Data Acquisition. 3, 8, 16

SSL Secure Socket Layers. 37–39

TCP Transmission Control Protocol. 33, 37–40, 48, 51, 58, 60, 61, 65

TIAP Technology-independent Abstraction Protocol. 29, 31, 32, 34–36, 43, 44, 46–48, 51, 56–61, 65, 66

TLS Transport Layer Security. 37–39

UDP User Datagram Protocol. 37–40

UML Unified Modelling Language. 29

VM Virtual Machine. 50

WMFP Weighted Micro Function Points. 52, 53, 59

WPF Windows Presentation Foundation. 49

Chapter 1

Introduction

Advancements in different technological areas, such as Internet of Things and Services and Cyber-Physical Systems (CPS), are on the origin of the next industrial revolution: *Industrie 4.0* [1]. With the vision of improving production while ensuring consumer customized goods, this revolution relies on the achievement of important goals.

One of the important objectives in *Industrie 4.0*, and the one in focus in the present work, is data integration. This can be achieved by the use of communication architectures, defined by Trunzer et al. as ”the description of the overall system layout based on principles and rules in order to describe its construction, enhancement and usage” [2]. Much of these architectures are based in a Message-Oriented Middleware (MOM), a system that works as a middle layer between the different components of the distributed system, abstracting communications while making interoperability possible [3].

A MOM for Automated Production Systems (aPSs) must respect some requirements, discussed in different scientific publications. Focus is given on one: middleware implementation independent of the communication technology (CT) [4–6]. A technology-independent implementation allows companies to take the best advantages of different communication protocols in distinct deployments, while ensuring that improvements are not delayed due to dependence on a specific technology.

This thesis aims to fulfill the aforementioned requirement by proposing a Technology-independent Architecture for Data Acquisition. This architecture is based on a MOM and abstracts the CT specific operation behind a common interface. A comparison with a legacy peer-to-peer (P2P) approach is also carried out, in order to identify the improvements that *Industrie 4.0* based architectures can bring to the engineering process.

The central ideas of this thesis were developed jointly with Mr. Emanuel Trunzer.

In the next sections a more detailed contextualization of the mentioned topics is provided.

1.1 *Industrie 4.0* – Historical Background

In the last 250 years, the world has witnessed the rising of industry over the classical manufacturing of goods, dominantly by hand. These revolutions were marked by the increasing importance and complex-

ity of some technologies, resulting in great improvements in production capacity.

The first industrial revolution took place around the end of the 18th century, and it is marked by the introduction of steam power to drive mechanical engines. This was a period that assisted not only to the mentioned increase in productivity, but it was also a turning point in human history, traduced by a rise in income and population.

In the end of the 19th century, with the emergence of electrical power, the second industrial revolution began. This revolution was marked by the introduction of conveyor belts and division of work, creating mass production assembly lines. One well known example is the case of Henry Ford's assembly line, where production improvements of 10,000% were achieved in less than 5 years [7].

The third industrial revolution began in the 1960's, and was triggered by the use of electronics and information technologies implemented in automation. Technologies like Ethernet, Wireless Networks or Web Technologies facilitate information exchange, allowing for increasing complexity in automation systems [8]. The architectures used are mostly hierarchical, as is the case of ISA-95 [9], also called the Automation Pyramid, present in Figure 1.1. Most enterprises are still on this stage, even though transition to *Industrie 4.0* is already taking place.

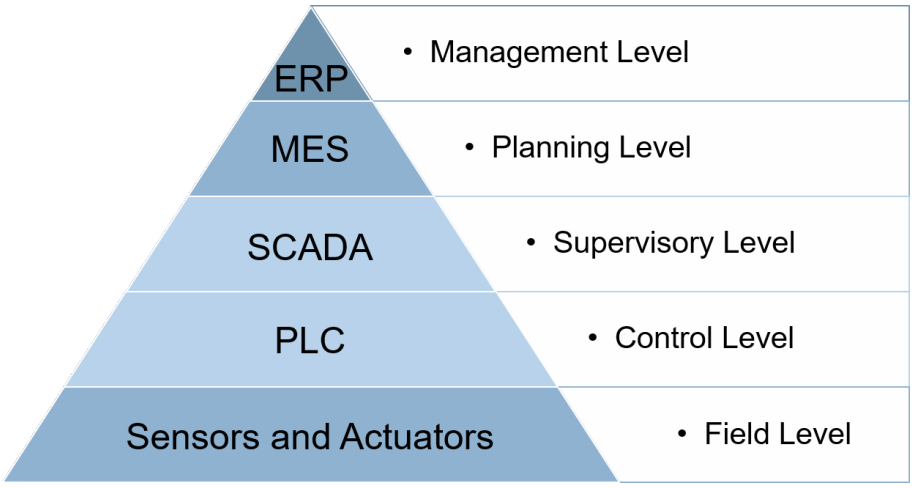


Figure 1.1: The five levels of the Automation Pyramid.

The term *Industrie 4.0* was first used in 2011 in Germany, and has been of great interest for the industrial and scientific fields ever since. There is no single definition, but it can be explained as: "the technical integration of CPSs into manufacturing and logistics and the use of the Internet of Things and Services in industrial processes" [1]. The vision is to create smart factories, capable of profitably deliver products in accordance to the personalization manufacturing paradigm, where consumers customize products according to their needs [7].

The main goal is to improve Overall Equipment Effectiveness (OEE) by optimizing the manufacturing process with algorithms and data [10]. Large amounts of data are analysed with emergent tools based on, for instance, machine learning, to extract important knowledge and achieve the OEE's improvement.

An important aspect in the path to *Industrie 4.0* is the capacity to deal with aforementioned quantities of data, made possible with the advancements in the field of Big Data. In the context of *Industrie 4.0*, and respecting the Internet of Things (IoT) principle of making all data available to all the participants in the

system, data integration is a goal to achieve. This a task that is not compatible with the classical ISA-95 hierarchical structure, where communication is made solely between adjacent levels [11]. Accordingly, focus must be given in ensuring data integration between all levels of the automation pyramid.

1.2 Data Integration in Automation

Data Integration in automation can be described by three aspects [12], further discussed in Chapter 2:

- horizontal integration, in the same level of automation;
- vertical integration, between the different levels;
- and temporal integration, along the system's life cycle.

The traditional automation pyramid (Figure 1.1) is not coherent with the data integration viewpoint for *Industrie 4.0*. In ISA-95, data is shared hierarchically, making the Manufacturing Execution System (MES) never aware of data present in the Programmable Logic Controllers (PLCs) or the upper level of the hierarchy, Enterprise Resource Planning (ERP), unaware of data in the Supervisory Control and Data Acquisition (SCADA) system. In other words, vertical integration is limited only to the adjacent levels. While this was common practice in the beginning of the third industrial revolution, soon this hierarchical structure began to appear inadequate, as suggested in [13], where a heterarchical architecture, very similar to a MOM-based architecture, was suggested. More recently, this problem was also emphasize in [11], as a result of an increasing necessity for data transparency in all levels of the automation pyramid, due to the appearance of smart devices and agents capable of making decisions in lowest levels of the hierarchy, for instance. It is interesting to note that, in both cases, this deficiency was identified before the term *Industrie 4.0* was first used, which emphasises the need for a new communication paradigm.

As mentioned before, in *Industrie 4.0* the integration viewpoint is fairly simple: everything must be connected and data must be available. In a greenfield deployment, where every system is added considering the requirements of *Industrie 4.0*, the implementation of a communication architecture can be made in a P2P way, without increasing complexity. In a brownfield environment, where legacy devices are present, the specific technologies of each device must be considered, in order to achieve the integration goal.

Considering this second type of deployments, that accounts for the big percentage of enterprises who use technologies that still have a long lifetime, is one of the gaps in the literature identified by Cimini et al. [14]. This gap, if not addressed, can slow down the progression of already existing enterprises. One solution for this problem is the use of a MOM.

1.3 Message-Oriented Middleware: Solution for *Industrie 4.0*

MOM is a software component that serves as a middle layer between the communications of a distributed system, ensuring interoperability between heterogeneous devices. It is characterized by Sauter as a ”key

enabler for integration" [12]. When a MOM is used in a system, messages are sent directly to this layer and redirected to the expected receiver. This software relies on a communication protocol, or CT, that should be used by all the participants of the system.

It makes sense to consider a MOM as a solution for legacy automation systems that want to achieve *Industrie 4.0*. Legacy devices present should implement common functions uniformly, by using data adapters, to ensure homogeneous interaction through the middleware. The use of these adapters ensure a reduction in application-specific software, increasing transparency, flexibility and extensibility while ensuring easier maintainability [5].

By using a MOM, when a new legacy device is added to the system, only one data adapter needs to be deployed: adapting the legacy CT to the middleware's CT.

A middleware-based architecture can also be seen has a temporary solution that takes into account the current state of automation systems, the automation pyramid, and provides a way of integrating data, as illustrated in Figure 1.2. With this solution, companies can fulfil data integration while ensuring gradual migration to a *Industrie 4.0* deployment, without the additional cost of replacing all the legacy devices, allowing them to keep their productivity competitive and leave the legacy ISA-95 structure functioning in parallel.

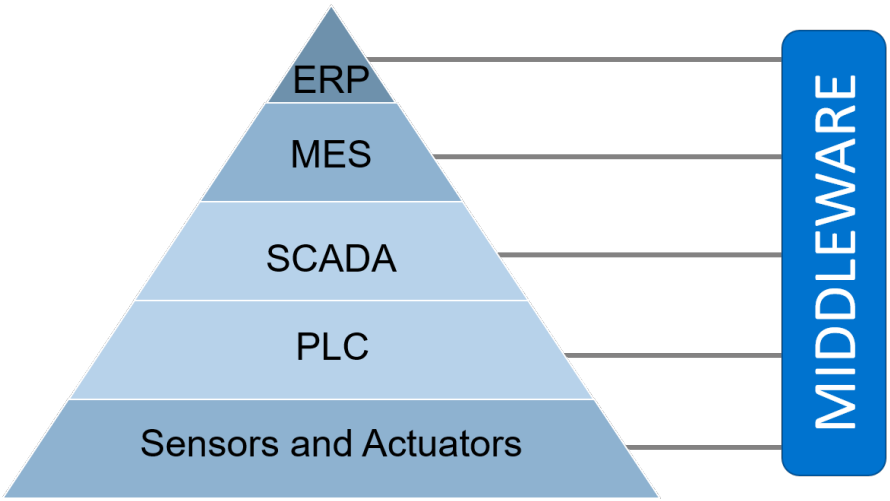


Figure 1.2: Legacy communications integrated with a middleware.

In greenfield deployments, the use of a MOM-based architecture can also be recommended, since it accounts for a reduction in system complexity. Distributed automation systems can consist of a large number of clients, so it can be advantageous to have all systems connecting only to one component.

This kind of MOM-based architecture can already be found in communication architectures present in the literature, as is the case of the UDaTA [2] or the PERFoRM project [15].

As mentioned before, the MOM works by standardizing the communication protocol used in a distributed system to the specific CT it uses, making all systems (including the middleware) dependent on the protocol. This dependence on technology slows down innovations and the change induced by *Industrie 4.0*, since an architecture that abstracts specific technologies can adapt more smoothly to changes and innovation [16].

1.4 Objectives and Contributions

The present work aims at developing a Data Acquisition Architecture based on a Message-Oriented Middleware (MOM), filling a gap in the scientific research related to the implementation of *Industrie 4.0* deployments in environments with legacy devices. Two main hypothesis must be realised:

- Data acquisition architecture can be independent of the technology used in the middleware's communication;
- Data acquisition architecture, when compared with a legacy peer-to-peer (P2P) approach, will present best results in terms of complexity of the concept, effort in the solution's deployment and effort in migrating between different communication technologies.

Considering the first hypothesis, technology independence (or platform independence) is defined by Almeida et al. [17] as "a quality of a model that relates to the extent to which the model abstracts from the characteristics of particular technology platforms." It is important to understand the relevance of creating an architecture independent of the communication's protocol used. By going through an analysis in the literature, one can find requirements for MOM stated in publications from two different fields that come together in *Industrie 4.0*, Industry and Internet of Things (IoT), and one from *Industrie 4.0*.

Regarding IoT, Razzaque et al. made a survey where a set of requirements for middleware were outlined [4]. The second Architectural Requirement discussed is related to interoperability, where it is stated that the "[middleware] should work with heterogeneous devices/technologies/applications, without additional effort from the application or service developer."

Regarding Industry, guidelines from the Association of German Engineers have put some emphasis on MOM for Industrial Automation [5]. The first requirement to be stated is the abstraction of communication, where it can be read that one of the key tasks of a middleware is "the provision of middleware-specific services (...) irrespective of the realisation offered by the technology."

Regarding *Industrie 4.0*, Theorin et al. [6] developed a data integration architecture, and state that "[service-oriented architecture] applications should be self-describing, discoverable, and platform- and language-independent."

Besides being stated as a requirement in the literature, there are clear advantages in a technology-independent middleware-based architecture. Different protocols have different features, representing a trade-off between advantages and disadvantages. A company might be interested in this architecture if they have distinct deployments where it is beneficial to change the communication technology in use.

Another scenario is when a company wants to change the communication protocol of its systems. During the migration process, being able to easily switch between technologies in order to keep the current deployment running and being able to test the new one is very valuable, since effort in migration might represent a considerable cost.

Considering the second hypothesis, it is clear that obtaining positive results in a comparison with a legacy approach proves the relevance of the solution.

Complexity of implementation is evaluated and discussed, proving that this solution should be taken into account by enterprises seeking to implement *Industrie 4.0*.

The effort comparison defines which architecture requires more work for the developer (in terms of time). The effort (time) to migrate the architecture to a different CT is also compared. This proves that enterprises might benefit from this architecture, since a reduction in working hours translates in a reduction in costs.

The main contributions of this work are the development of a technological independent data acquisition architecture concept and its implementation with different technologies in a prototypical use-case applied to *Industrie 4.0*.

Another contribution of this work is the definition of a new communication protocol. This was developed for the architecture and it abstracts different technologies to ensure technological independence for the concept.

This work also contributes with a comparison of the concept and implementation with a legacy P2P architecture.

A fifth contribution is a theoretical comparison of communication protocols. This comparison not only focus on the most relevant aspects for the field of automation, but also on the most important technologies used in automated production, filling a gap in the literature for comparisons of the kind.

Further, the output of this work was submitted as a conference paper.

1.5 Thesis Outline

The remainder of the thesis starts with a theoretical introduction of important concepts, some of them already discussed, in Chapter 2. It continues, in Chapter 3, where the requirements for a technology-independent data acquisition architecture are derived, based on the two hypothesis stated in Section 1.4. The statement of the requirements is followed, in Chapter 4, by an evaluation of relevant solutions in literature. These solutions are set side by side with the requirements present in this work, to identify the main research gaps that need to be filled. In Chapter 5, the concept of the architecture is developed, along with a comparison of relevant CTs and the definition of the metrics that will be used to evaluate the solution. Chapter 6 describes the implementation, both of the architecture developed and of the classical legacy approach, followed by the evaluation, comparison and discussion of both approaches, in Chapter 7. The work is concluded in Chapter 8, where the most important achievements are stated and future work is outlined.

Chapter 2

Theoretical Background

The purpose of this chapter is to give context to the concepts used throughout the rest of the work. It is not intended to get into the specifics of every topic mentioned. Instead, it serves as an introduction to the relevant details.

2.1 *Industrie 4.0* Drivers

As presented in Chapter 1, *Industrie 4.0* is based on two recent technological fields: Internet of Things and Services and Cyber-Physical Systems (CPS).

To start, a side-note should be made. Throughout this work, the term used to define the fourth industrial revolution is *Industrie 4.0*, a German expression that translates to *Industry 4.0*. This name is used since it was in Germany that this concept was introduced, and it is used this way in publications written in English [1].

2.1.1 Internet of Things and Services

Internet of Things and Services, can be understood as the consideration and combination of two paradigms: Internet of Things (IoT) and Internet of Services (IoS).

The paradigm of IoT is based on a network of objects, or *things*, that interoperate to achieve a common goal, envisioning ”anytime, anywhere, anymedia, anything” communications [18]. These *things* can go from simple home appliances to complex industrial PLCs, and must be capable of communicating with each other. New communication protocols are required to achieve the envisioned network of highly heterogeneous *things* [19].

IoS is a paradigm where everything in the internet is presented as a service [20]. It is not as widely used as IoT, nevertheless, it represents an important addition to the functionality of the *things*, since a service must also incorporate important knowledge about the business area and possible applications. Thus, as presented in [1], effort must be done in connecting this two paradigms, creating the Internet of Things and Services.

2.1.2 Cyber-Physical Systems

As pointed out by Gunes et al. [21], a CPS is defined from different perspectives by the research community, so there is no single definition. The most essential is the integration of a physical component with a computational representation (cyber). The cyber part of the CPS should be capable of monitoring and controlling its physical counter-part.

Nowadays, and in the context of *Industrie 4.0*, focus is given on creating Cyber-Physical Production Systems (CPPSs), a production system that integrates at least one CPS. For instance, Lee et al. [22] propose an architecture that serves as a guideline for implementing CPSs in the manufacturing industry, allowing the improvement of product quality and reliability.

In conclusion, *Industrie 4.0* is based on creating a computational representation of all systems involved in automation. The referred systems communicate interchangeably and interoperate as services that seek to achieve a common objective and improve the OEE. This creates smart factories that can make autonomous decisions and, at the same time, give constant inputs to human workers, allowing monitoring and controlling.

2.2 The Automation Pyramid

The Automation Pyramid, represented in Figure 1.1, was presented in Chapter 1 as the classical architecture used after the third industrial revolution. An overview of some important aspects of this architectural style is given in this section.

This concept is defined in the standard IEC 62264 [9], commonly known as ISA-95, to describe automated interfaces between the manufacturing control and enterprise functions. It focuses on reducing the risk, cost and errors, that usually occur in the interfaces' implementation.

Five levels of automation are defined. The lower one, considered the *Level 0*, is the *Field Level*. It is where the sensors, actuators and other field-devices are present, with different input and output signals used for the physical work in the production floor. Following, there is the *Control Level*, where the PLCs are present, to control the devices from the lower level, taking information from the sensors and making decisions on the outputs to send. Next, *Level 2* is the *Supervisory Level*, with SCADA systems that receive data from different PLCs and decide on the processes being used by those systems to control the industrial process. Finally, the model ends with the *Planning Level* and the *Management Level*, defined by software systems like MES and ERP respectively.

As stated in the Chapter 1, data in the Automation Pyramid is shared hierarchically, only between adjacent levels. This benefits suppliers from automation devices, that can limit its development to focus on meeting requirements for sending and receiving data only from nearby levels [11].

The shape of a pyramid is due mostly to the size and frequency of data exchange between levels. For instance, the field level sends data in small packages but in real time, while the MES sends large batches but with frequencies that can go from day-to-day to monthly. Thus, in total, both the total data

size and the frequency of exchange are reduced by going up in the hierarchy's levels.

2.3 Enterprise Integration

Data integration in an enterprise, as explained in Chapter 1, is described by three aspects [12]: horizontal integration, vertical integration and temporal integration. This section's purpose is to explain the different aspects of data integration in the automation hierarchy.

Horizontal integration, refers to the exchange of data and interoperability of devices and software systems used in different stages of the manufacturing process. Each level of the automation pyramid is horizontally integrated, so communication happens between PLCs, for instance. Information exchange between different companies is also possible in horizontal integration.

Vertical integration is similar to horizontal, but it considers systems from different hierarchical levels. When an enterprise has the control level exchanging data directly with the management level, for instance, it is considered to be vertically integrated between those levels. Ideally, all levels can communicate and cooperate with each other.

In temporal, or end-to-end [1], integration, the entire value chain of a system or product is taken into account, from product development to production and services.

An enterprise should work to achieve integration in the three levels previously mentioned. When the systems are integrated vertically and horizontally, temporal integration is also facilitated and the deployment becomes similar to the desired *Industrie 4.0* scenario, where interoperability and flexibility of all systems involved is a requirement.

2.4 Data Integration Levels

The previous section described the three aspects for enterprise integration, according to the automation hierarchy. This constitutes one of the possible dimensions that can be analysed when defining integration [23]. Another relevant dimension is the four integration levels: Hardware, Platform, Syntactical and Semantic Level.

According to Izza [23], integration in the hardware level is related to compatible hardware and networks, while the platform level encompasses the operating system or the database platform, for instance. In the syntactical level, the way the data is represented should be coherent.

Two systems integrated in a syntactical level must share data in the same format or being able to read data sent by the other device. For example, if one machine A sends a *string* to machine B, and B is able to receive it, they are considered syntactically integrated.

In the semantic level, the intended meaning of the data is commonly understood by the systems present. Considering the previous example, A and B are considered integrated at the semantic level if B is able to extract knowledge from the *string* sent by A. One way of achieving this is with a common information model.

Each of these levels is built on the previous one. For example, A and B can never be integrated at a syntactical level if they are not integrated at the hardware level. When dealing with communications, one must guarantee hardware integration before escalating to the subsequent levels.

With the theoretical concepts presented in this section and in Section 2.3, it is possible to give a clearer definition of integration. For example, in the same enterprise, it is possible to have data vertically integrated in terms of platform and semantically integrated in a horizontal viewpoint. As shown in [24], the concept of integration can take several forms, thus the importance of clarifying how it is used throughout this work.

2.5 Message-Oriented Middleware in Industry

The use of a MOM in production systems, with its principal requirements, is going to be briefly discussed in this section. Focus is given on the Enterprise Service Bus (ESB), an architecture proposed by Chappell [25] that can be seen as a classical middleware implementation widely used in the industrial field.

MOM is defined by Izza [23] as ”one of the key technologies for building asynchronous large-scale enterprise systems”. Publishers and consumers are decoupled, since the middleware is the software responsible for the communication, allowing clients to stay autonomous of each other. Three different types of MOM are defined, according to the way messages are handled: message passing, where messages are sent in a unidirectional way; message queuing, that provides reliability by adding message persistence; and publish/subscribe messaging, that allows ”one to many”, ”many to one” and ”many to many” messaging.

The broker is the software entity that takes care of the communication in a MOM-based architecture. In most cases, the terms broker and middleware can be used interchangeably.

Requirements for middleware systems can be found in a variety of different sources. Two examples are analysed.

Razzaque et al. [4] groups middleware requirements in two sets: services that middleware should provide and architectural requirements. The services can be functional, such as resource discovery, data management or event management, or non-functional, such as scalability, real-time, reliability or security. In terms of architectural requirements, the MOM should provide programming abstraction and interoperability, while being context-aware and autonomous, for instance.

Guidelines from the Association of German Engineers [5] define requirements such as abstraction of communication, information security, interoperability and flexibility. It can be seen that there are similarities between the requirements from both publications.

As presented in Chapter 1, both sources mention that a MOM should be technology-independent.

The ESB [25], a solution widely used in industry for integration, is an architecture that uses a communication bus to decouple systems while managing the communications between them. This bus can be considered a MOM and was created to avoid P2P architectures and reduce integration complexity in an enterprise.

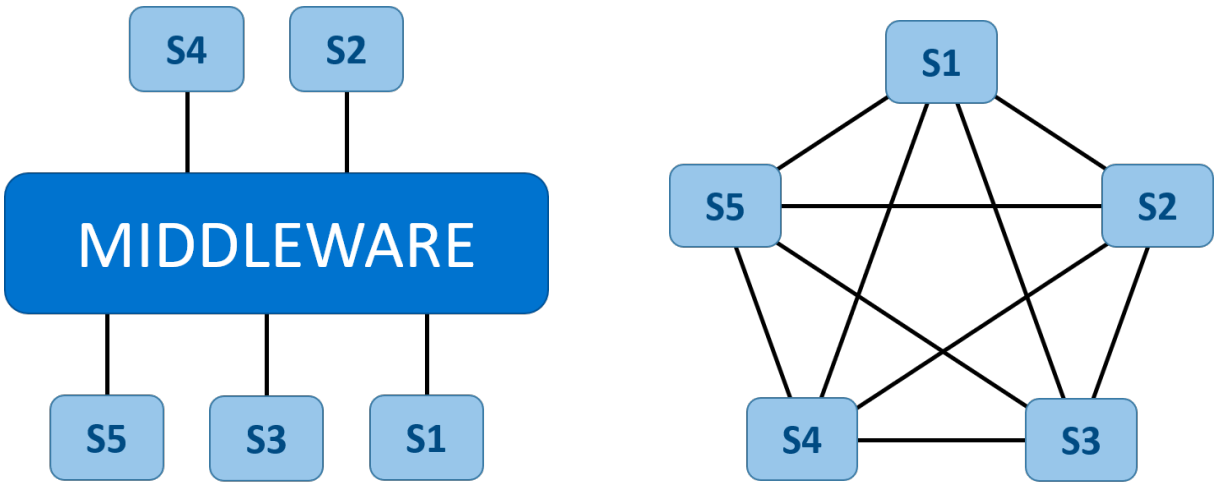
This system is used for enabling a service-oriented architecture, an architecture where participants provide their functionality as services, by providing the connectivity layer between those services. The use of an ESB is traduced in an increased flexibility, since it can discover, use and modify the meta-data, conditions and constraints used in connections between service requester and provider [26].

In Chapter 4, state of the art architectures for industry are analysed, and different types of MOM are presented.

2.6 Peer-to-peer Network for *Industrie 4.0*

A Peer-to-peer (P2P), or point-to-point, network is a system where connection between clients is made directly, without intermediaries. It is common in small networks, for instance in home environments, but it is also considered a classical approach for industrial communications.

In Figure 2.1, a comparison between two scenarios, one with a MOM and the other with a P2P, is presented. The number of communication lines in the MOM scenario (Figure 2.1(a)) corresponds to n, being n the number of clients present in the system. For a fully connected P2P scenario (Figure 2.1(b)), the number of communication lines is $n(n-1)/2$. Thus, with respect to communication lines, the order of the system is n in the MOM architecture and n^2 in the P2P. This relation is represented in Figure 2.2.



(a) Middleware Architecture

(b) Peer-to-peer Architecture

Figure 2.1: Comparison between the number of communication lines in a fully connected architecture with five clients: from System 1 (S1) to System 5 (S5).

The number of communication lines is seen as a way to measure complexity, since a system with a high number of connections is harder to maintain. Plus, a P2P scenario presents a challenge in terms of scalability, since adding a new client means creating $n-1$ new connections, while in the MOM-based system of systems it is only necessary to create one connection.

Considering that, in *Industrie 4.0*, all systems should be connected, it becomes impractical to use a P2P architecture [27]. In a case where every system uses a different CT, a substantial amount of effort would be needed to create a P2P scenario with more than three or four clients, since the quantity

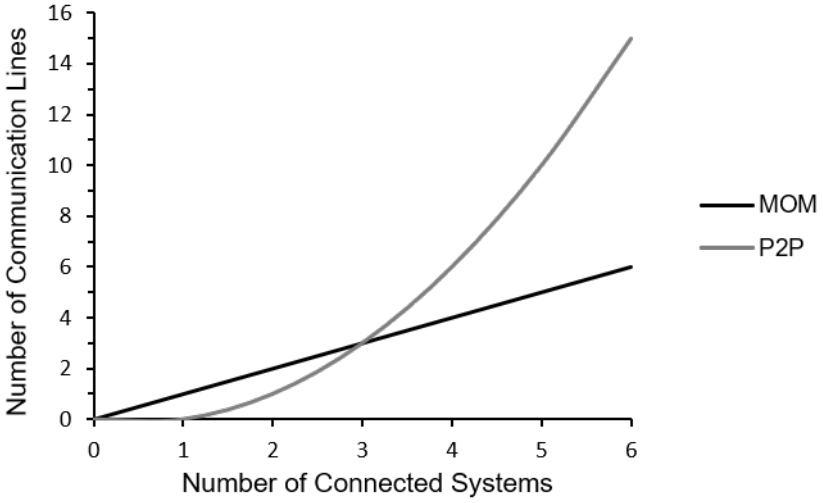


Figure 2.2: Graphical comparison of a peer-to-peer and a middleware architectures.

of connections would quickly scale. Thus, MOM-based architectures present a clear advantage for *Industrie 4.0* applications.

2.7 Reference Architectures

In the present section, three reference architectures for *Industrie 4.0* are going to be briefly presented. These are architectures that encompass important principles accounted in the fourth industrial revolution, and that are presented in a high level of abstraction, providing guidance for the development of specific architectures. Full-range implementations are non-existent.

The Reference Architecture Model for *Industrie 4.0* (RAMI 4.0) [28] is focused on manufacturing and proposes a three dimensional model to represent the *Industrie 4.0* space. In the vertical axis, layers are used to represented different perspectives, such as business, communication and integration. The left-horizontal axis represent the product life cycle with its value streams. The last axis contains the location of functionalities and responsibilities, from the product to the connected world, passing through the levels present in the automation pyramid. This model defines components by combining life cycle and value stream (horizontal axes) with a hierarchical approach (layers of the vertical axis).

The Industrial Internet Reference Architecture (IIRA) [29] is an open-architecture for *Industrie 4.0*, focused on interoperability, mapping applicable technologies and driving technology development. The IIRA describes four viewpoints, defined by analysing different use-cases, identifying relevant stakeholders and considering *Industrie 4.0* concerns. The four viewpoints are: business, usage, functional and implementation.

The Internet of Things Reference Architecture (IoT RA) [30] describes a top-down approach that uses industry best practices. To start, the most important IoT characteristics were collected and abstracted into a conceptual model, that was used to derive a system based reference model. This final model is

then broken down in five architecture views: functional, system, user, information and communication view.

A deeper analysis of each reference architecture is not considered important, serving this section as a high level introduction. Nonetheless, it is possible to identify similarities between them, such as the division of the *Industrie 4.0* space in different layers/viewpoints/views.

2.8 Relevant Communication Principles

Since this work is focused on communication architectures in the industrial environment, some important communication principles should be introduced.

First, this work will focus on two messaging patterns: Request-Response (RR) and the Publish-Subscribe (PS). This refers to the way producers and consumers of messages communicate. In the RR messaging pattern, the consumer requests the message to the producer, who then responds with the requested message. In PS, the consumer subscribes to receive messages from a certain producer, which then publishes its messages to all the current subscribers. In this last pattern, producers and consumers are decoupled. Each pattern is convenient in some scenarios, for instance, when the consumer wants messages in a frequency significantly lower than the frequency of the producer, then a RR pattern is more advantageous. Another example is the case of a producer broadcasting to more than one consumers. In this last case, it is beneficial to use a PS messaging pattern.

Another important principle is the centralisation or decentralisation of the broker in a MOM-based architecture. In a decentralised architecture, the broker is divided in different locations, and it is capable of making decisions independently of the other parts. If the broker is centralised, the message handling is made in a single location. The exception is the case where the broker is distributed, but it can be considered centralised if different brokers make decisions dependently of each other. In a decentralised middleware the control can be harder, but it has the advantage of avoiding a single point of failure, something that can happen in a centralised architecture.

Quality of Services (QoS) are policies that some CT present in the exchanging of data. These might be related to the way data is delivered, for example *at most once* or *at least once* delivery, or to different parameters related to usage of resources, like reliability and durability.

Chapter 3

Requirements for the Architecture

The present chapter states the requirements to be fulfilled by the technology-independent architecture for data acquisition. Chapter 1 offered a motivation and two important hypothesis that should be restated here:

- 1st Hypothesis – a data acquisition architecture can be independent of the technology used in the middleware's communication;
- 2nd Hypothesis – a technology-independent data acquisition architecture, when compared with a legacy peer-to-peer (P2P) approach in terms of complexity of the concept, effort in deployment and effort in migrating between different communication technologies, will represent a better option.

These hypothesis were already discussed and should now be scrutinised to obtain the needed requirements.

3.1 Application for *Industrie 4.0*

The production of data is increasing exponentially, originating a recent field of research: Big Data. For instance, it is estimated that the volume of digital data doubles every two years and that, between 2012 and 2020, the investment in information technologies will increase by 40% [31]. Big data was described by Wu et al. [32] using the HACE theorem: "Big Data starts with large-volume, heterogeneous, autonomous sources with distributed and decentralized control, and seeks to explore complex and evolving relationships among data."

In *Industrie 4.0*, large amounts of data are generated by the IoT, creating the need for new models and tools to handle this data [33]. The support of these concepts and technologies by aPS is one of the main requisites in *Industrie 4.0*. Therefore, effort must be taken in adapting existing approaches to the industrial field, considering the specific *Industrie 4.0* constraints, most importantly: response time, network latency and reliability [34].

The aim of this thesis is to create an architecture for data acquisition, as stated in the first hypothesis. As will be presented in Chapter 4, there is a large number of studies and concepts of architectures for

Industrie 4.0, and the developed concept should focus on that field. Solutions from different research fields should not be considered, due to the specific industrial constraints.

The first requirement - R1 - can then be stated: the architecture should be applied in the field of *Industrie 4.0*. This means that data acquisition should be implemented in aPSs, integrating different levels of automation, for example considering a PLC or SCADA sending data to a MES or ERP system. This requirement is relevant due to the need of taking into account the particularities present in the Industrial field, that might differ according to the deployment.

While the first requirement could be applied only at a theoretical or simulation level, it is also important to reach the practical implementation level. For example, reference architectures, as the ones discussed in Chapter 2, are very important on a conceptual level, however they reach an abstraction level that makes the implementation unfeasible.

Therefore, a second requirement - R2 - is derived: the architecture should be implemented and validated in a real environment. This ensures that the aforementioned constraints are not only theoretically taken into account, but also practically overcome, proving the applicability of the architecture in a real industrial apparatus. Thus, data must be obtained from industrial plants.

3.2 Interoperability using a Message-Oriented Middleware

Due to long lifespans of industrial systems, legacy devices are present in brown-field industrial deployments. In most cases, the creation of a greenfield implementation would require a lot of investment for replacing all legacy systems, making them unfeasible. Therefore, the goal of *Industrie 4.0* connectivity must be to integrate all systems and components, enabling interoperability [27].

Interoperability between data sources and systems simplifies data integration, handling and sharing. A standard interface is a good way of achieving this goal. Having in mind the life cycle of different services, support for legacy systems is required in an *Industrie 4.0* architecture.

It is now possible to state the third requirement - R3: interoperability of data sources and systems. The different systems present in the aPS should work together, interconnected by a standard interface. This requirement keeps in mind brownfield implementations with legacy devices, with different CT, integrating them via a MOM.

To successfully evaluate this requirement, two sub-requirements must be stated. First, the architecture should have a standard interface, a middleware, integrating all devices' communication. Second, the presence, and integration, of legacy devices should be taken into account. These systems must be retrofitted with a data adapter, allowing them to communicate with the other devices present in the architecture. R3 is considered fulfilled if these sub-requirements are satisfied. A summary is presented in Table 3.1.

Table 3.1: Interoperability of data sources and systems – Sub-Requirements.

R3	Description	Evaluation
a.	Standard Interface	All devices must communicate through a single interface.
b.	Integration of legacy devices	Legacy devices should be integrated, by sending or receiving data from other components.

3.3 Technology-independent Architecture

As discussed in Section 1.4, the development of a concept independent of the communication protocol is relevant in the present context. Technology independence is advantageous since it abstracts CTs' specific logics and allows the architecture implementations in different deployments, taking companies a step further in *Industrie 4.0*. This is a need identified in the literature [4, 5], as previously presented.

Along these lines, the fourth requirement - R4 - is stated: technology-independent concept. This requirement is derived directly from the first hypothesis of this thesis, stated in the beginning of the present chapter.

In order to fulfil this requirement, two sub-requirements are stated. First, multiple CTs must be considered, so the same use-case should be implemented at least two times, with different communication protocols, without losing functionality. Furthermore, the migration between the CTs should be made as simple as possible, with no changes in the client's source code. The sub-requirements are summarised in Table 3.2.

Table 3.2: Technology-independent concept – Sub-Requirements.

R4	Description	Evaluation
a.	Different technologies implemented	Same use-case implemented with at least two communication technologies.
b.	Straightforward migration	Transition between technologies should require a small number of modifications in system's code.

3.4 Comparison to Legacy Approach

The second hypothesis of this thesis - comparison to a legacy architecture - should also be considered when establishing the requirements. The scope of the comparison must be clearly defined, since focus is given on effort and complexity.

It should be taken into account that, in classical architectures, the integration of data is a complex and sometimes unfeasible task, that requires great effort. Therefore, the last requirement - R5 - is formulated: lower complexity and effort when compared to a classical approach.

These two terms, complexity and effort, can be considered synonyms. Usually a more complex deployment implies extra effort, and more effort in a certain task can be a measure of increasing complexity. In respect to this requirement, complexity is applied at a conceptual level, the complexity of the concept, while effort is related to the necessary time for a certain task, in what concerns to coding, for

instance.

To ensure the evaluation of this requirement, four sub-requirements must be stated, that can be divided in three perspectives: the user, the system and the technical perspectives.

From the user perspective, the most important requirement in a data acquisition architecture that aims at data integration is to have all data available. This is a measure of complexity of the system, since the transparency of all data allows data analysers to easily extract knowledge from the plant. On the other hand, if data is not directly available, the user needs to make changes to the system and ensure that the connection to the desired data sources exist. Hence, the first sub-requirement is to have transparent data access in the architecture.

According to the system perspective, as discussed in Section 2.6, a way to measure the complexity of an architecture is by the number of communication lines. Therefore, the second sub-requirement is to have a reduced number of connections.

For the technical perspective, the final sub-requirements are related to accessing the effort needed for the implementation and for the migration. Implementation means the initial deployment of the different data sources and consumers and migration refers to a transition to a different middleware realisation or communication protocol. In both cases, the use of code metrics can be a method of evaluation, adding that, in the migration scenario, the relative number of actions needed to succeed in the transition should also be taken into account.

Summing up, compliance with R5 means that, comparing with a classical P2P approach, the architecture requires less complexity, by having data transparently accessible and fewer communication lines, and less effort, due to a reduction in implementation and maintenance effort. The sub-requirements are presented in Table 3.3.

Table 3.3: Lower effort and complexity compared to legacy approach – Sub-requirements.

R5	Description	Evaluation
a.	Transparent data access	All data is available.
b.	Reduced number of interfaces	Smaller number of communication lines.
c.	Reduced effort for implementation	Code analysis metrics.
d.	Reduced effort for migration	Code analysis metrics. Number of actions necessary.

3.5 Implementation Goals

The derived requirements are summarized in Table 3.4. The implementation goals of this thesis can now be understood in a more clear way: to design and implement a technology-independent architecture applied to *Industrie 4.0*, ensuring interoperability of the connected devices and that, in comparison with a P2P approach, ensures a decrease in complexity and effort.

In the following chapter, an analysis to state of the art architectures will be undergone, evaluating how other authors fulfill the derived requirements to find a gap in the literature.

Table 3.4: Derived requirements for a technology-independent data acquisition architecture.

ID	Description
R1	**Applied to *Industrie 4.0*** Data integration should be performed in a automated production system.
R2	**Implementation on real environment** Implementation and validation of the architecture in industrial plants.
R3	**Interoperability of data sources and systems** Different systems, using different technologies, should work together. All devices must be interconnected through a standard interface. This requirement keeps in mind brown-field implementations with legacy devices.
a.	Standard Interface
b.	Support legacy devices
R4	**Technology-independent concept** Implementation and migration between different communication technologies.
a.	Different technologies applied
b.	Straightforward migration
R5	**Lower effort and complexity when compared to a legacy approach** A comparison based on complexity and effort to highlight the improvements accomplished. Data transparently available and the reduction of the number of interfaces leads to less complexity. Less effort for deployment and migration is also desirable.
a.	Transparent data access
b.	Reduced number of interfaces
c.	Reduced effort for implementation
d.	Reduced effort for migration

Chapter 4

State of the Art

In the present chapter, state of the art data integration architectures for industry are evaluated in comparison with the requirements defined in Chapter 3.

4.1 Existing Architectures for Data Integration

The SOCRADES project introduces an architecture for integration in the shop floor, as presented in the project's website [35], where base information, publications and several implementations can be found (R2). The goal was to develop a design, execution and management platform for automation systems. A description of the architecture and its requirements is presented in [36], based mostly on agile manufacturing and smart objects as web-services to develop a service-oriented architecture. This project ended before the term *Industrie 4.0* was coined, but a lot of principles are similar, such as data integration in different automation levels.

The presence of a middleware (R3-a) in the architecture ensures that all data is available (R5-a) and the number of communication interfaces is smaller than in a P2P layout (R5-b). Presence of legacy devices is also accounted (R3-b), as discussed in [37]. The SOCRADES project is focused on platform-independence, which is guaranteed by using web-services, but technology Independence in the MOM is not mentioned.

To cope with the dynamic environment in manufacturing, where unexpected changes can occur without warning, Marín et al. [38] suggest a conceptual architecture based on intelligent services and a ESB: the intelligent Enterprise Service Bus (iESB). An intelligent service is defined as a software system that produces results by itself or using another intelligent service. Based on this independent piece of software, the architecture aims at fulfilling five requirements: include human initiative with HMIs, distributed decision making, coordination of schedules with negotiation-based approaches, legacy systems connected (R3-b) and ability to update schedule and planning.

Due to its age, this architecture did not contemplate the concept of *Industrie 4.0*, nonetheless, data integration in industry is an objective of the iESB, along with smart decision-makers to improve produc-

tion, so the first requirement can be considered partly fulfilled. The intelligent services are integrated with a ESB, so a standard interface is present (R3-a). The architecture uses an upper-level P2P network, however, the communication is mostly interfaced by the ESB, so all components can communicate with each other (R5-a). Since a comparison was not made, it is not possible to know if the number of interfaces and the effort would be reduced. An execution sample where three instances of the architecture are used is presented, but a prototypical implementation is missing. Technology independence was also not considered.

The iESB is defined in the scope of the Adaptive Production Management project. Using similar principles, and in the scope of the same project, Leitão et al. [39] present a multi-agent architecture for strategic planning in small lots manufacture of complex products (airplanes, for instance). Thus, the feasibility of applying the iESB architecture in different production contexts is proven, even though a prototypical implementation is still missing.

The Cloud Collaborative Manufacturing Networks project, C2NET, creates a cloud-based collaborative network for supply chain interactions of ERP data, as presented by Qureshi and Agostinho in [40]. This architecture enables the connection of management systems from different organisations, putting special emphasis on legacy systems (R3-b). The C2NET data collection framework is composed by two main components: the company middleware and the cloud component, where virtualisation is attained. The cloud can be considered as a standard interface (R3-a) between the different companies' systems. With this framework, small and medium enterprises in the same supply chain can share resources in a common ecosystem, allowing them to collaborate and save resources.

Although this architecture applies *Industrie 4.0* principles, only data from the ERP is obtained, so the first requirement of this work can only be considered partly fulfilled. Two use-cases are implemented, that consider different legacy systems (R2). Technology independence is not considered and a comparison is not done, although it can be argued that the cloud component allows data transparency (R5-a) and a lower number of interfaces (R5-b), compared to a deployment where each enterprise would need to communicate directly with the others.

Perez et al. [41] designed a CPPS-based architecture for *Industrie 4.0* that focus on vertical integration of data in the production process (R1). It aims at making all data accessible (R5-a) and at extracting knowledge using technologies from big data analysis.

The architecture is prototypically implemented, however, it does not take into account the different levels of the automation pyramid. It uses an information model containing logical process nodes, each node related to a certain device. Consequently, a standard interface is not present. A technology-independent concept is never mentioned.

Ismail and Kastner [42] propose an architecture for a distributed gateway service bus that aims to achieve vertical integration in *Industrie 4.0* (R1). This architecture uses an international standard based on function blocks to model the control logic, encapsulating logic algorithms and defining events and data

for other function blocks. The information and information exchange modelling is guaranteed by using a Machine-to-Machine (M2M) communication protocol. Device interoperability, accounting for legacy systems (R3-b), is ensured with a hybrid solution of protocol translation and tunnelling .

A decentralised middleware is preferred to avoid a single point of failure. The coordination between different nodes of the gateway service bus is done using a P2P network, so the number of interfaces is not reduced if it was to be compared with a legacy approach. Accessibility is simplified with the use of semantically standardised data, although it is unclear if data is made transparently available. Technology independence and a prototypical implementation are not discussed.

Fleischmann et al. [43] proposed a RAMI 4.0 based architecture that was developed following previous works on Socio-CPSs [44, 45]. This kind of systems take into account the human user as an important factor in the decision-making process, with valuable inputs based on different knowledge or experience. The proposed architecture is a modular web-based framework for condition monitoring systems and fault diagnosis, created to support technical maintenance. Integration is achieved by using the RAMI 4.0 hierarchy levels and layers as the basis of the architecture (R1).

The architecture is decentralised and only relevant data is integrated in cloud modules, that communicate using a manufacturing service bus. Thus, improvements in terms of complexity and effort in comparison to a legacy approach can not be asserted. The integration of existing machinery is considered as a requirement (R3-b). Technology independence is not discussed and the architecture is implemented in a test cell (R2).

The communication architecture from BaSys 4.0 [46], an open source project focused on enabling *Industrie 4.0* scenarios, is evaluated. Its main focus is on changeability, allowing mass individualisation while reducing downtime. All relevant systems have an asset administration shell, a digital twin that enable access to information, including from different automation levels (R1). Different implementations from different enterprises are also presented (R2).

One of the pillars of BaSys 4.0 is a middleware named virtual automation bus. This serves as a central point for communication between devices (R3-a), including legacy components (R3-b). This middleware can be considered technology-independent, since a standard interface can be implemented with different CTs. Nonetheless, the implementation with different technologies uses two MOMs with different protocols connected through a gateway [47], so it does not fulfil this work's fourth requirement. With the virtual automation bus, end-to-end communication is eased and all systems can access each others data (R5-a). Compared with a P2P approach, the number of interfaces would be reduced, since systems can communicate through the middleware (R5-b).

The LISA project [6] is an event-driven service-oriented architecture that relies on an ESB (R3-a) to avoid point-to-point connections. The events are created with a prototype-based approach and represent one of the basis to ensure loose coupling of devices. The architecture uses a simple message format and integrates heterogeneous and legacy devices with communication endpoints (R3-b).

LISA is an architecture for *Industrie 4.0* that focus on integration between all levels (R1). The architecture has been evaluated in an industrial plant (R2). Focus is not given to technology independence, even though the authors state that it is a requirement to achieve the full potential of service-oriented architectures. Once again, the presence of a middleware ensures the fulfilment of the first two sub-requirements in R5, although a clear comparison was not described.

The PERFoRM [15] project, Production harmonisEd Reconfiguration of Flexible Robots and Machinery, is focused on the seamless reconfiguration of aPS. It aims at aggregating a network of smart and heterogeneous components and enhancing planning, simulation and operational features. It positions itself in *Industrie 4.0* through the use of Industrial CPSs in the digitalization of the shop floor and by the integration of all automation levels (R1). Its architectural elements include a MOM (R3-a), standard interfaces, that ensure transparent interconnection of heterogeneous components (R5-a), technological adapters, which in turn ensure the presence of legacy systems (R3-b), human integration and advanced tools.

The concept is not dependent on the CT, nonetheless, effort is not put in abstracting it, has seen by the data adapters that are strongly dependent on the technology used in the MOM. The authors do not formally compare the PERFoRM architecture with a legacy P2P approach, although it can also be argued that a reduced number of interfaces is achieved (R5-b). Implementations of some components of the PERFoRM project can be found in the literature [48, 49], however a full-range deployment is still missing.

Trunzer et al. [50] introduce an architecture to unify data transfer in aPS. This architecture relies on a MOM (R3-a) and a common information model in order to achieve data integration in all levels of automation (R1), taking into account brownfield deployments and a gradual migration to *Industrie 4.0*, supporting legacy devices (R3-b). Focus is also given on processing data from both historic and near real-time sources, different analysis tools and ensuring data is shared in a secure way. The architecture was validated with a prototypical implementation (R2), where it was also named UDaTA [2].

Even though the technologies used in UDaTA's implementation are considered of secondary relevance, only one CT was used and migration between communication protocols was not taken into account, so the architecture can not be considered technology-independent. Comparing with a legacy approach, all data is made available (R5-a) and the number of interfaces is reduced by the use of a MOM. A comparison of implementation and migration was not described.

Schel et al. [51] developed a manufacturing service bus based architecture, that uses an enhanced ESB for integration, extending a first proposal from Minguez [52]. This is applied to *Industrie 4.0* by ensuring the vertical and horizontal integration in a cloud-based platform (R1). A set of functional and non-functional requirements are developed for this architecture, with special focus given on security and on the support for multiple communication protocols, although technology independence is not in focus.

The basis for the implementation of the manufacturing service bus was described, but a prototypical

use-case is missing. The ESB serves as a standard interface (R3-a), that also integrates legacy systems (R3-b). An harmonised communication model and the presence of multiple communication standards ensure data transparency (R5-a). The number of interfaces is certainly reduced with the ESB (R5-b), but a formal comparison was not performed.

Kirmse et al. [53] propose a lightweight architecture that uses big data analytics and machine learning to achieve optimised integration and harmonisation of heterogeneous and distributed data sources. The concept focus on data accessibility, having raw data transparently available (R5-a) in a managed data pool. This data is used by machine learning analysers to optimise processes, accounting for all systems in the automation pyramid as data sources, as required in *Industrie 4.0* (R1). The focus is on data analysis, so technical features such as communication speed or presence of real-time data are not considered. Nonetheless, the architecture is able to reintegrate analysis results in the production process, providing real-time improvements.

Integration of legacy systems is considered (R3-b) using a modular structure that encapsulates similar groups of data sources, without a standard interface. Implementation is mentioned, but only a description of technologies for the different components is given, being inconclusive if the architecture was implemented and in which use-case. Technology independence is not mentioned.

The COCOP message bus mediator [54] is a data and event-driven architecture that aims at easing the development of plant-wide industrial process monitoring and control. With the purpose to avoid P2P communications, the architecture uses a loosely coupled approach that allows scalability and performance. New control functions and reactive control applications are added to the legacy systems previously present, using a communication platform as a mediator (R3-a). Existing control systems are integrated by the use of adapters (R3-b).

Even tough the architecture is applied to aPS and integration between different automation levels is considered, the architecture is not contextualised for *Industrie 4.0*, so the first requirement is not considered fully fulfilled. Technology independence is not contemplated and a comparison is not implemented, even tough discussion between the advantages of using a middleware over a point-to-point architecture is present (R5-b). The architecture is implemented in a laboratory use-case, although it is not clear if it considers different automation levels or legacy devices, since the discussion is more focused on messaging, message formats and communication patterns.

Longo et al. [55] propose an *Industrie 4.0* architecture that implements a service-oriented digital twin, developed in the same publication. The focus is on data representation and augmented reality, to allow knowledge acquisition and use. The architecture represents a distributed system of networked components, using an ESB implemented in a central server (R3-a) and accessed through web-services. The central component is the CPPS, which is reflected in real-time in the digital twin. Management level systems are also integrated (R1).

The architecture was implemented in two use-cases, one in a *Industrie 4.0*-ready company and the

other in a small enterprise (R2). Both in the concept and in the implementations, the integration of legacy devices was not described. The architecture relies on one communication protocol in the ESB, so it is not technology-independent. A comparison was not made, although data is transparently available through the ESB (R5-a) and the number of interfaces is smaller than in a P2P architecture that would consider the same components (R5-b).

4.2 Comparison and Research Gaps

The comparison between different state of the art architectures is summarised in Table 4.1.

Table 4.1: Classification table for existing data integration architectures.

Concept	**R1**	**R2**	**R3**	**R4**	**R5**
SOCRADES [36]	○	+	+	-	○
iESB [38]	○	-	+	-	○
C2NET [40]	○	+	+	-	○
Perez et al. [41]	+	○	○	-	○
Ismail and Kastner [42]	+	-	○	-	○
Fleischmann et al. [43]	+	+	○	-	-
BaSys 4.0 [46]	+	+	+	-	○
LISA [6]	+	+	+	-	○
PERFoRM [15]	+	○	+	-	○
UDaTA [50]	+	+	+	-	○
Schel et al. [51]	+	-	+	-	○
Kirmse et al. [53]	+	-	○	-	○
COCOP [54]	○	○	+	-	○
Longo et al. [55]	+	+	○	-	○

+ Fully Fulfilled, ○ Partly Fulfilled, - Not Fulfilled

Regarding the first requirement, almost all architectures consider vertical integration of all automation levels. This can be seen even in architectures presented prior to the introduction of *Industrie 4.0*, showing the importance of this industrial revolution.

Half of the studied architectures still lack a fully prototypical implementation that considers *Industrie 4.0* constraints. Although conceptual architectures are relevant, a functioning implementation is important to validate the architecture, and is something that should be achieved by new concepts.

All the studied architectures consider interoperability of systems, although some of them do not fulfil this work's third requirement. Data is integrated in almost all architectures by the means of a standard interface, a middleware in most cases. Only two architectures do not consider the inclusion of legacy devices.

Independence of the communication protocol used the middleware is stated as an important requirement in some works, but is only implemented by one architecture. However, this requirement focus on the implementation of a single use-case with one technology and then migrating without losing functionality, whereas in [47] the use-case implements two technologies at the same time and does not

consider migration. Thus, the fourth requirement represents a clear gap in the literature, since no architecture conceptualised the same use-case with different CTs, implemented that concept in an industrial deployment and migrated between the protocols without losing functionality.

Finally, even tough some middleware implementations discuss its advantages, a formal comparison with legacy P2P approach is never executed. According to the fifth requirement, the table 4.1 can be misleading. The requirement is considered partly fulfilled because it is possible to assert a reduction in complexity when a MOM is used. Nonetheless, no concept pointed this out by comparing with a legacy P2P use-case with the same functionality, and a effort comparison was never made.

In conclusion, one can assert that two gaps in the literature exist for *Industrie 4.0* data integration architectures: technological independence and a a complexity and effort comparison with legacy approaches. The main hypothesis of the present work consist on filling these gaps.

Chapter 5

Concept of a Technology-independent Architecture

In this chapter, the concept for a technology-independent data acquisition architecture is developed. The concept considers not only the architecture but also a new communication protocol to ensure CT independence. Afterwards, the prototypical use-case to be implemented is deployed. A comparison between different communication protocols is also present.

5.1 Concept Development

The concept of the architecture will be developed according to the requirements defined in Chapter 3. To start, the fourth requirement is considered: concept independent of the communication technology. To fulfil this requirement, and before detailing the architecture, a new communication protocol that abstracts the CTs must be defined (Section 5.1.1). The concept of the architecture that implements this protocol, achieving the desired technology independence, is defined afterwards (Section 5.1.2).

5.1.1 Technology-independent Abstraction Protocol

As stated before, in order to abstract different CTs, a new communication protocol should be developed, capable of implementing the different logic behind each technology. Accordingly, the Technology-independent Abstraction Protocol (TIAP) is introduced.

To understand the concept, a class diagram developed using the Unified Modelling Language (UML) is presented in Figure 5.1. This diagram represents only the classes and the relationships existing between them, considered the most relevant features in order to understand the logic behind the new CT. The fields and methods will be presented in the implementation (Chapter 6).

The most important component of this protocol, where technology independence is achieved, is the interface *ICommTech*. This interface defines the communication functions as methods, such as a method for publishing and another for subscribing. The interface is then implemented by different classes, one

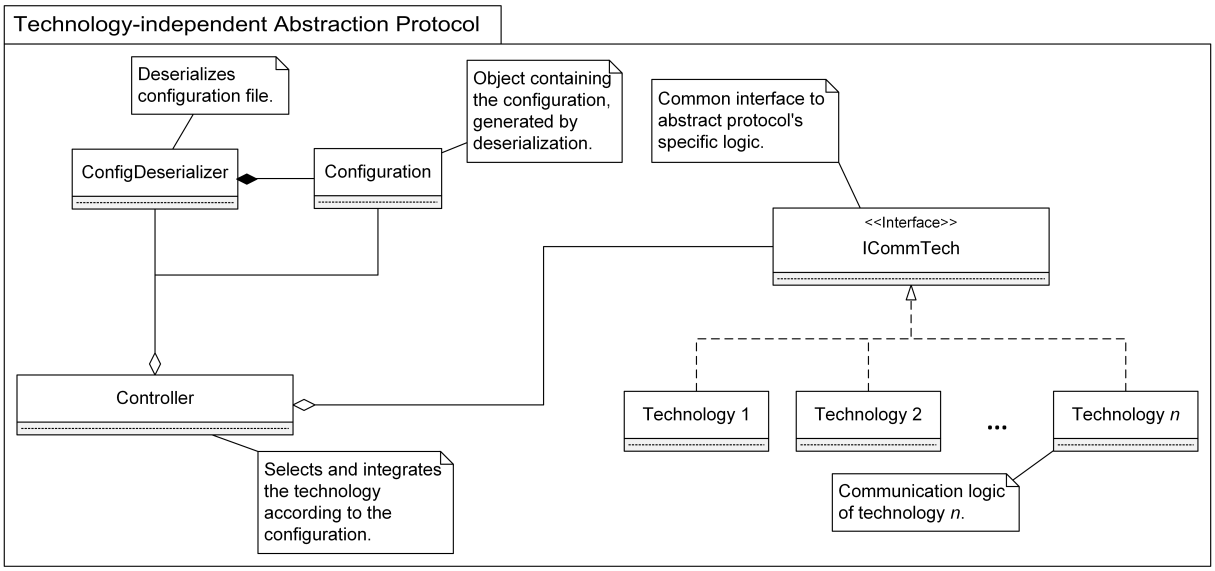


Figure 5.1: Simplified class diagram representing the Technology-independent Abstraction Protocol.

for each CT, where the specific logic of the protocol is used.

To increase modularity of the solution, configuration files are used. This minimises the amount of changes needed to the static source code in case of reconfiguration, the technology is selected using an external component. The configuration file defines not only the CT used, but also relevant information, such as the broker's location. The class *ConfigDeserializer* deserializes the configuration file to an object of the class *Configuration*.

The last component is the *Controller*. This class aims at integrating the previous components, by encapsulating the configuration and communication logic. By creating an instance of the *Controller*, the configuration file is read and the CT selected. The class also defines the same methods as the *ICommTech*, using polymorphism to implement them independently of the communication protocol defined in the configuration file. Since the configuration and communication logic are abstracted by this class, the client remains unchanged independently of the CT used.

The developed protocol takes into account the sub-requirements R4-a and R4-b, respectively, by using the interface *ICommTech* to implement different CTs and by using a configuration file to define the CT used. The presence of the *Controller* also takes into account sub-requirements R5-c and R5-d. This will be further discussed in Chapter 7.

5.1.2 Technology-independent Data Acquisition Architecture

The concept of the technology-independent data acquisition architecture must be derived from the requirements and defined in order to be applied in diverse use-cases.

Considering the first requirement, the architecture must be applied to *Industrie 4.0*, meaning that data should be obtained not only from industrial plants and machines but also from management-level clients, in order to ensure integration in all levels of the automation pyramid (Figure 1.1). Data analysis and presentation should also be taken into account by the inclusion of analysers and Human-Machine

Interfaces (HMIs).

The architecture should be MOM-based, to account for the third requirement. The presence of a middleware guarantees that there is an interface connecting all devices (R3-a) and is also an important component to achieve sub-requirements R5-a and R5-b. The support of legacy devices (R3-b) is ensured with the presence of data adapters, that take the payload from the legacy communication protocol and translate that information to the middleware's protocol (and vice-versa). Evidently, the CT used in the middleware must be the TIAP.

Since the second requirement is a prototypical implementation, it is not necessary to discuss it in this section. Regarding R5, although this requirement can only be evaluated after the implementation, decisions like using a MOM in the architecture will directly influence the complexity, and that is the reason it was considered.

In Figure 5.2, a graphical representation of the developed architecture is presented, based on the UDaTA architecture [50]. The arrows represent data exchange and it should be noticed that all the clients communicate through the MOM, except for the dashed arrows in the HMI, that represent data visualisation by a worker. Components with different functions are aggregated in four sections, represented by the grey rectangles in the background.

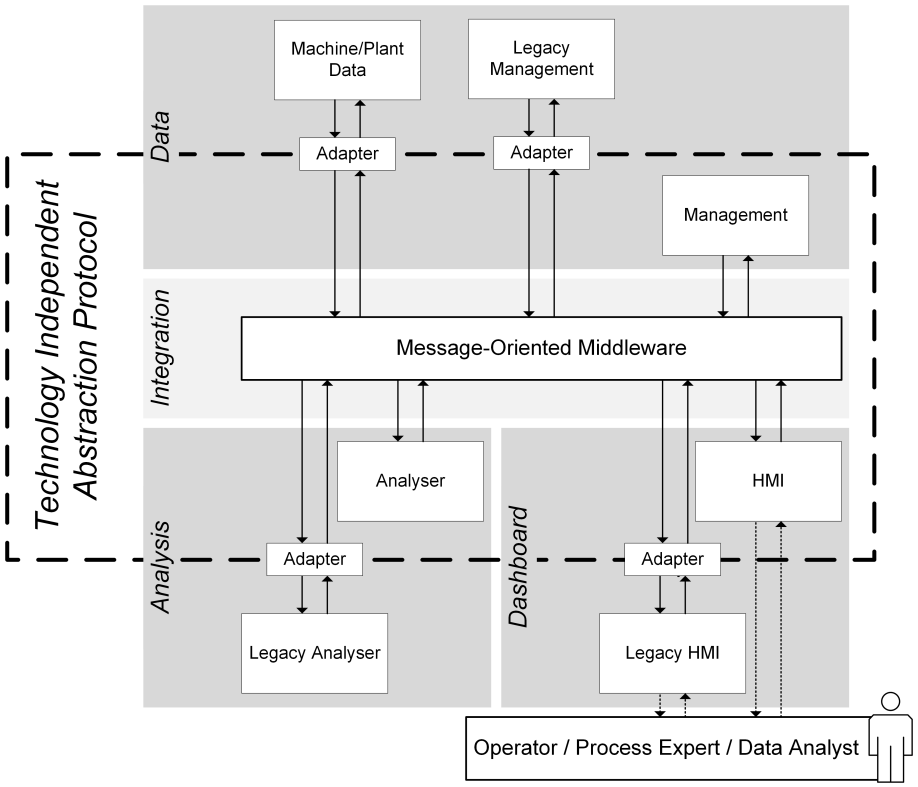


Figure 5.2: Graphical representation of the technology-independent data acquisition architecture (TIAP) with message-oriented middleware, adapters, and TIAP-compliant clients, as well as legacy systems. Adapted from [50].

The section *Data* contains the clients that handle data, producing or consuming. In the *Integration*, a MOM is present, to assure data integration between all systems. The *Analysis* contains systems to take knowledge from data, while the *Dashboard* contains systems for data visualisation.

The architecture is useful in the presence of legacy devices, integrating them by means of data adapters, but also for new devices that already communicate with the TIAP. In a brownfield deployment, all legacy devices can be integrated and new greenfield devices can be added with ease. For the case of *Machine/Plant Data*, new devices were not considered, since this communication protocol was developed during the present work, so there are no data sources that communicate with it originally. In the future, greenfield *Machine/Plant Data* systems could be added.

It should also be noted that clients present in Figure 5.2 are just a reference for a class of clients that can be present in a certain implementation. For instance, the block *Legacy Management* does not imply that there will be a legacy MES or ERP client present in the use-case, or the block *Analyser* does not mean that there is only one analyser per use-case. This should be taken into account in the next section, where the use-case to be implemented is developed.

5.2 Prototypical Use-Case

As discussed in the State of the Art (Chapter 4), there is still a considerable number of architectures to be validated in a real industrial implementation. This leaves an open-ended question of whether an architecture is feasible for *Industrie 4.0* or not. It was along these lines that the second requirement, the implementation of the developed architecture in a real environment, was derived.

To account for this requirement, the prototypical use-case to be implemented is derived. The architecture is used in an anomaly detection system, where critical components' data is acquired from industrial plants and is analysed to prevent potential failure and to apply predictive maintenance. The discussion focus on the systems that are used, in conformity with the different sections present in Figure 5.2.

It must be noted that the complexity of the use-case is not a requirement of the present work. This section describes a simple use-case, with simple components, that was created to prove the feasibility of an MOM-based architecture for data acquisition independent of the CT used in the middleware. This use-case could be scaled to different and more complex deployments, further discussed in Chapter 7.

5.2.1 Data

Regarding the data sources and management, three components are used: two prototypical industrial plants and one MES system. The plants work as legacy devices while the MES is a newly developed system to be used as a representation of different automation levels.

Industrial Plants

The first plant integrated is the Modular Production System (MPS) [56]. As seen in Figure 5.3(a), this is an educational plant composed by many different modules, created to teach students technology fundamentals and to work with different automation processes.

The module used was the handling station, that uses a pneumatic gripper to move a working piece between different locations, with eight digital inputs and outputs. The data collected is the instantaneous value from eight sensors, while the data to be analysed is the time it takes for the gripper to drive the working piece from the initial to the final position, and also the time it takes to come back to get a different working piece.

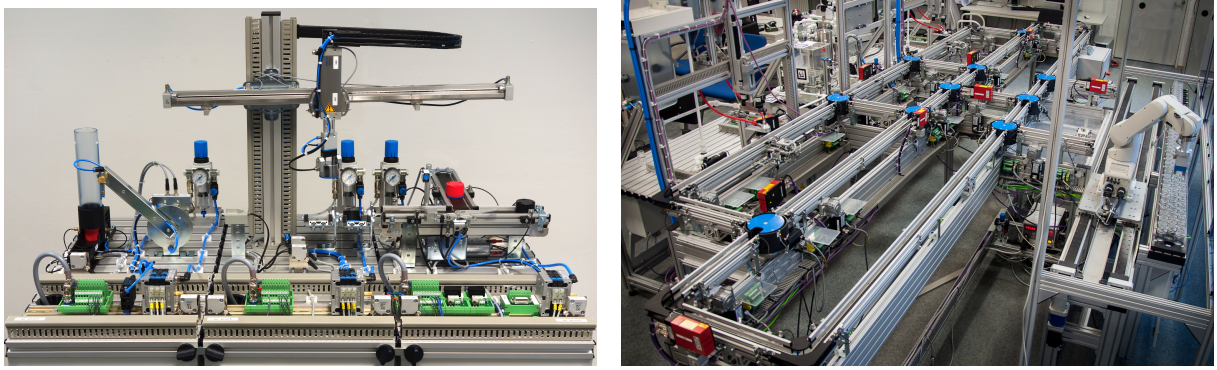
The data is obtained directly from the PLC, using a TCP-direct connection in a RR pattern.

The second prototypical plant is the *MyJoghurt* [57], as seen in Figure 5.3(b). It is a demonstrator platform for *Industrie 4.0* that displays the coupling and interconnection of distributed systems. It presents a diverse number of systems, such as filling stations and a 6-axis robot, and uses 45 bus couplers in a tree topology, accounting for several hundred input and output signals.

The critical components that are analysed by the system are four Beckhoff AM8100 synchronous servomotors [58] that drive four of the conveyor belts. The data acquired is the instantaneous torque and speed of each motor, although only the first one is analysed. The speed is simply for visualisation and comparison with the torque.

The CT used to get data from the plant is the Open Platform Communications Unified Architecture (OPC UA), a communication protocol that is going to be further discussed in Section 5.3. The data is obtained with a PS messaging pattern and sent from the plant's PLC.

Both plants are considered legacy devices in the use-case, since they use different CTs, requiring the implementation of two data adapters. A more thorough description of both plants' used components is given in the implementation (Chapter 6).



(a) Modular Production System [59]

(b) *MyJoghurt [57]*

Figure 5.3: Prototypical industrial plants used in the implementation.

Manufacturing Execution System

The MES has two main objectives: store data, so that it can be further visualised by the HMI, and update the threshold used in the analysers to define if a certain value represents an anomaly. Thus, it receives data from the two plants and also receives and stores the anomaly reports of the analysers. In reverse, it sends the updated thresholds to the analysers and the plant values to the HMI.

This system was developed in the current work, so the communication is made using the TIAP. Thus, it is not a legacy component and does not require a data adapter.

5.2.2 Analysis and Dashboard

In respect to data analysis and dashboard, four components were developed: three analysers and one HMI. As with the MES, all systems were designed for the current use-case, so they use the developed CT to send and receive data.

Analysers

The first analyser, *Time Interval*, receives data from the MPS plant every time the value of the sensors of the gripper position changes, computing the time difference between two consecutive occurrences. The time value used for the computation is sent in the message payload by the MPS, and corresponds to the time it noticed the change in the sensor's value. The data is then sent to the second analyser.

The goal of the second analyser is to perform outlier removal with an Hampel filter. In this filter, the value being analysed is considered an outlier and replaced by the median if the absolute value of difference to the median is bigger than the Median Absolute Deviation (MAD), multiplied by a constant [60]. In this case, filtering considers a window of previous values, since it must be done when the value is received, so future values do not exist. The size of the window and the constant that multiplies with the MAD must be tuned.

The *Hampel Filter* receives the time interval values from the first analyser, *Time Interval*, and also the torque values directly from the *MyJoghurt* plant. The filtered value is sent to the last analyser.

The final analyser, *Anomaly Detection*, was created with the goal of detecting anomalies, from both the MPS's gripper movement and the servomotors in the *MyJoghurt* plant. It receives filtered values from the *Hampel Filter* and compares them with thresholds received from the MES. It transfers the data to the HMI and sends periodical anomaly reports to be stored in the MES.

Human-Machine Interface

The designed HMI is a simple dashboard system that allows the user to visualise data. It is composed of three main components: the anomaly report, the anomalies visualisation and a display of all plant values.

The anomaly report consists of a *Status* box indicating the system where the last anomaly was detected. The anomalies visualisation consists of a graphical representation of the values checked for anomalies, which are differentiated from the remaining data. The display of plant values is also a graphical representation, but of all values stored in the MES.

This component receives the anomaly data from the *Anomaly Detection* analyser and plant values from the MES.

5.2.3 Layout

With the systems and their relationships introduced, a graphical representation of the layout is presented in Figure 5.4. The most important points must be emphasised.

This use-case is a brownfield deployment, due to the presence of legacy devices that are integrated: the two industrial plants that use different CT. All communications between the components are performed using the developed TIAP, mediated using a MOM. The data exchange between the plants' PLCs and the MES ensures that data integration occurs between different automation levels, while the presence of the middleware guarantees that data is available to all components. Finally, all areas of the architecture are represented with clients in the use-case, so its implementation also represents an implementation of the developed architecture.

Despite being representative of all areas of the developed architecture, this use-case represents a real industrial application, mostly due to the presence of the two plants. Although this is not a a full enterprise with hundreds of systems integrated, this use-case represents a possible sub-system for fault detection and predictive maintenance, that is crucial in industrial environments. Thus, the developed use-case is representative of industrial applications.

In Chapter 6, a legacy P2P approach that respects the functionality of the use-case is discussed, to carry out the necessary comparison, according to the fifth requirement.

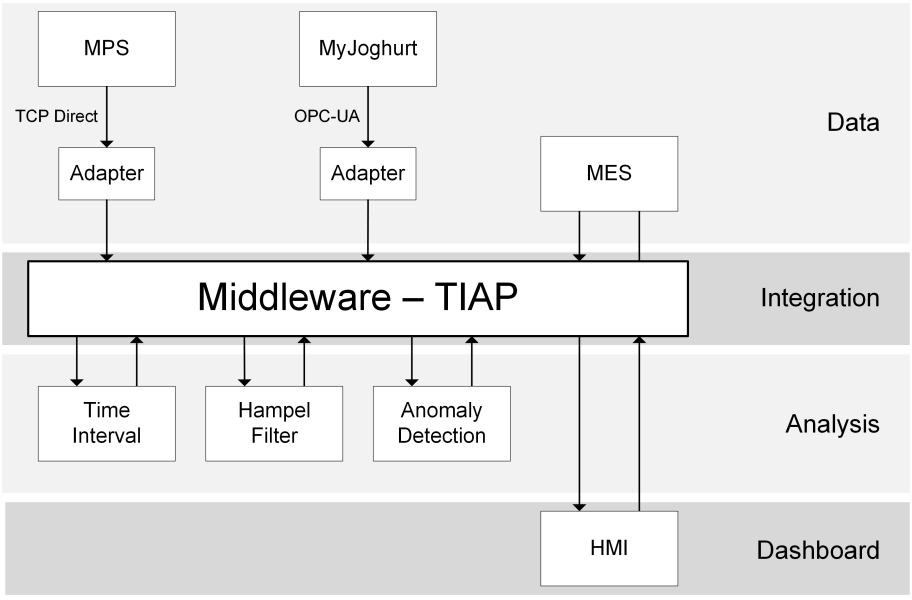


Figure 5.4: Structured, graphical representation of the implemented use-case, using the developed architecture.

5.3 Communication Technology in the Middleware

For the use-case developed in the previous section, and considering the requirement for technology independence, it is necessary to use the TIAP with at least two different CTs. Thus, eight communication protocols for MOM are considered, along with a description of the most important features and a

comparison between them.

5.3.1 Comparison of Relevant Communication Technologies for *Industrie 4.0*

As previously mentioned, a comparison between different CTs is relevant to define the specific protocols that will be abstracted by the TIAP and implemented in the use-case. Although previous comparisons are available in the literature [3, 61–64], they do not consider all the CTs that were intended to be evaluated and miss some of the most relevant features for an application in *Industrie 4.0*. Thus, a new comparison is relevant also to fill this gap.

Below, the relevant features to be compared are presented, followed by a thorough description of each CT. All the discussed protocols are suitable for M2M communications, making them suitable for the deployed use-case.

Relevant Features

The features that are evaluated for each CT are: *Message Pattern*, *Protocol*, *Architecture*,*QoS*, *Security*, *Solution Providers*, *Property* and *Languages Supported*.

The parameter *Messaging Pattern* respects to patterns like RR and PS, *Protocol* is the transport layer protocol and *Architecture* evaluates the architectural pattern of the broker. The case of QoS is different, since this is a qualitative parameter that characterises the CT according to the integration of this mechanisms. For example, a $-$ indicates that the protocol does not consider QoS, while a $++$ points out that QoS is a major concern of the CT.

Security evaluates if the CT has built-in security features. Main focus is given to authentication and encryption.

In *Solution Providers*, the existence of open source solutions and the number of available solution providers are taken into account. The purpose is to qualify the availability of the CT for a deployer. When there are open source solutions and when the number of solution providers is bigger than 5, the CT is considered to be very available and is ranked with a $++$. If, on the other hand, there are no open source solutions available, the CT is ranked with a $-$. The purpose is to identify the availability of the CT for deployments and the chance of a vendor lock-in.

Languages Supported, reflects the number of programming languages that are supported by the CT. If the principal programming languages (like Java, C based languages or Phython) are supported, it is ranked with a $+$. On the other hand, if there are few clients available, it will be ranked with a $-$. This feature is important mostly in an implementation viewpoint, since supporting more programming languages makes the CT flexible for more users.

The last parameter, *Standard Owner*, indicates the organisation that standardises the communication protocol. This parameter is relevant since a standardised protocol is considered more mature, and is usually a more reliable solution. If it is an organisation, it is presented with the holder of the standard. If it is non-standardised and open source, it appears *Open*. It can also appear the standard name and number, if there is one.

DDS

Data Distribution Service (DDS) is a CT standardised by the Object Management Group (OMG) to allow communication and integration of distributed applications [65]. It is described by a PS messaging pattern and a decentralised architecture, in order to avoid single points of failure. It is a CT that puts a lot of emphasis on QoS, with policies that focus on, for example, data availability or data delivery [66].

In terms of transport layer protocol, the DDS Interoperability Wire Protocol [67] is defined to exchange messages over different protocols, although it should always guarantee implementation on top of User Datagram Protocol (UDP). In respect to security, the OMG defines a security specification [68] that constitutes the security model of DDS applications. It allows for customisation of security related features, such as authentication and encryption, enabling interoperability between different DDS applications.

DDS has more than one open source solution and there are 10 solution providers [69]. A large number of programming languages is also supported.

Kafka

Apache Kafka is a middleware software and communication protocol introduced by *LinkedIn*, a social network website, in order to process hundreds of gigabytes of new log data that are generated each day [70]. It was developed with a distributed architecture and uses a PS pattern on top of a high-performance Transmission Control Protocol (TCP) protocol for building a real time streaming platform [71].

Contrasting with other CT, the consumption information is kept not by the broker but by the consumers. Thus, despite reducing overhead and complexity in the broker, this design principle implies that QoS are not supported. Regarding security, and although it is disabled by default, connection can be made over Transport Layer Security / Secure Socket Layers (TLS/SSL), ensuring authentication and encryption of messages.

Although developed by *LinkedIn*, it was open sourced and there is only one solution available for all community, that is constantly being improved. Clients have been created for all of the principal programming languages [72].

AMQP

Advanced Message Queuing Protocol (AMQP) is a standardised [73] CT that originated from the financial field [74] but is being used by more than 500 companies in a variety of areas [75]. It works with a centralised broker and a PS messaging pattern on top of TCP [76].

It supports QoS, with special focus on message acknowledgements, and uses TLS/SSL to ensure security features, with authentication and encryption included.

There are more than 10 solution providers for AMQP [75], with clients available in the most used programming languages.

OPC UA

OPC UA is a standard protocol [77] by OPC Foundation. It is focused on all industrial levels, from sensors and actuators to ERP systems [78]. The standard version uses a RR communication pattern and it has recently been complemented with a PS specification [79]. Since the PS specification is not standardised, this analysis will focus the two specifications of OPC UA in a distinct way.

The standard OPC UA has a decentralised architecture, with clients and servers communicating with each other over TCP. The support for QoS is not discussed in the specification. Authentication and encryption are addressed in the security specification [80], where emphasis is put on secured data exchange between applications.

This CT is a widely used protocol in industry, with a big number of solution providers, including open source solutions [81]. As seen in the previous list, a set of different programming languages is supported.

Regarding the PS model, it can be deployed in two different settings: with broker, a centralised setting, and without broker, where communications are UDP-based [79].

When a broker is used, a standard communication protocol (such as AMQP) is used for the middleware to send messages over TCP. QoS are defined and mapped according to the broker's implementation of QoS. In terms of security, it depends on the broker's implementation as well. A TLS connection is the mentioned case, with end-to-end security being necessary in the connection to the broker. No implementations of this specification were found.

The UDP-based can be well suited in environments where small amounts of data are frequently transmitted. In this case, the security features are built in, with focus on authentication and encryption of data. Only one solution provider was found, the open source *open62541* [82].

MQTT

Message Queuing Telemetry Transport (MQTT) is a standardised [83] communication protocol that is light weight, open and easy to implement. It runs over TCP and features characteristics such as PS as messaging pattern, flexible content formats and three QoS: ”at most once”, ”at least once” and ”exactly once” delivery.

In MQTT, all clients subscribe or publish in topics located in a centralised broker [84]. Authentication can be passed using a MQTT packet, while encryption is handled by the TLS/SSL [85].

Information regarding solution providers can be found in [86], with a large number of solutions, including open source. A list of clients that includes more than 25 programming languages is also included.

REST WS

Representational State Transfer for Web Services (REST WS) is a decentralised architectural style employed for creating web services. It differs from the other CT, since it is not a application level protocol, but a framework for communication. REST WS uses Hyper Text Transfer Protocol (HTTP) methods, on

top of TCP, to provide a simple synchronous RR messaging system [87]. Although it is not a communication protocol, it is included here since it is widely used in M2M communication [61].

Security can be handled by TLS/SSL, including encryption and authentication. Since the only concern is providing functional interfaces, QoS are ignored by REST WS [88].

Since this is an architectural style, it does not make sense to speak of solution providers or programming languages. Technically, REST WS can be applied with every programming language, as long as an HTTP library is present.

CoAP

Constrained Application Protocol (CoAP) is an Internet Engineering Task Force (IETF) standard [89] for M2M communications in constrained (for example, low power) environments. It runs on top of UDP, a more lightweight protocol, and uses a RR messaging pattern. CoAP was designed according to the REST WS architecture, making mapping between the two feasible.

QoS is present in four possible message types that ensure reliability. CoAP is secured using Datagram Transport Layer Security (DTLS), another IETF standard [90]. This is a protocol based on the TLS that provides similar security features.

Implementations of CoAP can be found in large number and for a diverse group of programming languages, with commercial and open source solutions available [91].

MTConnect

MTConnect is a standard by the MTConnect Institute [92] for data acquisition in manufacturing environments. It uses a decentralised architecture based on entities called *MTConnect Agents*, responsible for collecting, rearranging and sending data. This CT support both PS and RR communication pattern and uses HTTP, on top of TCP, as a lower-level protocol.

QoS, although already tested in [93], are not included in the specification. The same happens with security, that is not addressed in the specification. The use of security based on, for example, TLS/SSL is recommended [64].

MTConnect is an open source standard and with a big number of agents and clients available in a variety of programming languages [94].

Results

The results of the comparison are presented in Table 5.1.

Most design choices are a trade-off between advantages and disadvantages, so it is impossible to say that one is better than the other. Nonetheless, some tendencies can be identified.

The majority of the CT evaluated use a PS messaging pattern, due to advantages like the simplicity for one to many communications. There are also protocols that can already implement both patterns, like MTConnect, and there is the case of OPC UA, where the standard still only implements RR but is developing solutions that also consider PS.

Most protocols run over TCP, since it ensures more reliable communications than UDP. Both fully implemented protocols that use UDP, DDS and CoAP, balance the eventual packet loss by using some form of QoS.

Some protocols already consider QoS, even though only DDS focus on a wide range of policies. Contrary, some CTs don't consider QoS, since they increase complexity and decrease bandwidth.

Security is a common factor among all protocols, with the exception of MTConnect, although it is something that can be added. When security is considered, authentication and encryption are always focused.

With exception for the contemporary PS specification of OPC UA, it must be noted that all CTs present a big number of open source solutions, which facilitate implementation, dissemination and improvements. One implementation of PS OPC UA with UDP already exists but it still lacks validation. Kafka was well rated despite only one implementation, since it is a widely used and validated solution that is evolving with the input of its users.

All protocols with good results in the *Solution Providers* parameter also support the most important programming languages.

Table 5.1: Comparison of communication protocols for industrial communication.

	DDS	Kafka	AMQP	OPC UA			MQTT	REST WS	CoAP	MTConnect
				Standard	PS					
					Broker	Direct				
Message Pattern	PS	PS	PS	RR	PS & RR		PS	RR	RR	PS & RR
Protocol	UDP	TCP	TCP	TCP	TCP	UDP	TCP	TCP	UDP	TCP
Architecture	DC	D	C	DC	C	DC	C	DC	DC	DC
QoS	++	-	+	-	+	-	+	-	+	-
Security	DDS Specification	TLS/SSL	TLS/SSL	Built in	TLS + EE	Built in.	TLS/SSL	TLS/SSL	DTLS	No
Encryption	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-
Authentication	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-
Solution Providers	+	+	+	+	-	o	+	NA	+	+
Open Source Solutions	Yes	Yes	Yes	Yes	No	Yes	Yes	NA	Yes	Yes
Number	5-10	1	>10	>10	0	1	>10	NA	>10	>10
Languages Supported	+	+	+	+	-	-	+	++	+	+
Standard Owner	OMG	Open	ISO/IEC 19464	IEC 62541	OPC Foundation		ISO/IEC 20922	Open	IETF RFC7252	MTC Institute

NA – Not Applied; C — Centralised; DC — Decentralised; D — Distributed; EE — End-to-end Security

Chapter 6

Implementation

The implementation details of the concepts derived in Chapter 5 are presented in the this chapter. It starts with the implementation of the Technology-independent Abstraction Protocol (TIAP), considering the technical details and the technologies implemented. Afterwards, the implementation of the prototypical use-case is delineated, followed by a description of the deployment.

The end of the chapter is dedicated to the comparison with a legacy approach. First, the use-case is implemented using a P2P architecture, with the details of the implementation being presented. In the end, the evaluation metrics used for the comparison are determined.

6.1 Implementation of the Communication Protocol

In this section, the TIAP's implementation is described, with focus on the development environment, the definition of the communication protocols to be used and the most important details for the implementation of the concept defined in Chapter 5. In the end, the most important steps to implement a TIAP's client are described.

6.1.1 Development Environment

With the goal of ensuring reusability and a modular structure in the clients, the TIAP was developed as a Dynamic Link Library (DLL). This proved to be an efficient design choice, since the deployment required continuous testing and upgrades, that are facilitated by this kind of libraries, and since it allowed clients to implement the protocol with little effort.

The library was developed using the framework .NET Core 2.2 [95], that has the advantage of being independent of the operating system, very important for an application to be applied in heterogeneous scenarios, according to *Industrie 4.0* requirements. C# was the programming language used and the development environment was Visual Studio Community.

After developing TIAP's concept, presented in Figure 5.1, and after deciding on the development environment, an important design choice is necessary: defining the CTs to implement.

6.1.2 Selection of Technologies

The selected technologies are two of the previously mentioned in the comparison: Kafka and AMQP. The choice was based in the fact that both share some common ground, like a PS messaging pattern, but also have a lot of differences that make the migration challenging.

One of the most important differences is related to QoS policies. While Kafka focus more on storing the data in the topic, being the user responsible for correctly consume and publish, AMQP takes into account QoS, having more control on message handling. On the other hand, by reducing complexity in the broker, Kafka can present higher throughput. This design choices represent a trade-off between relevant features.

Another interesting aspect is that both brokers are originally from different fields. Kafka originated in information technologies and big data, while AMQP came from the financial field. Nonetheless, both have already been implemented in industrial practices. Furthermore, the aggregation of technology from different fields is one of the drivers of *Industrie 4.0*, so using these CTs is interesting in the context of this work.

It is important to define the implementation of these CTs that will be used as a broker. For AMQP, RabbitMQ 3.7.10 [96] was used. This software was built based on AMQP 0-9-1, but can support AMQP 1.0 via a plugin. For Kafka, the broker used is the widely used and exclusive solution: Apache Kafka 2.1.0 [97].

To ensure communication with both brokers, the TIAP must implement clients containing AMQP's and Kafka's connection logic. For the RabbitMQ, the package used is the official .NET package: RabbitMQ.Client (v5.1.0) [98]. In the case of Kafka, the client used was developed by Confluent and it is the most commonly used: Confluent.Kafka (v0.11.6) [99].

6.1.3 Implementation Details

According to the diagram presented in the previous chapter (Figure 5.1), the communication logic of both protocols must be programmed in two different classes, that should implement the *ICommTech* interface. This interface defines the methods that all the technologies should have.

Figure 6.1 presents the implementation of the TIAP that was used in the use-case. The specific AMQP logic for communicating using the RabbitMQ broker is contained in the class *RabbitMQBroker*. Similarly, the Apache Kafka's logic is programmed in the class *KafkaBroker*.

The methods contained in the interface *ICommTech*, that are implemented by both technologies, are mostly self descriptive. For example: the method *SearchBroker* returns true if the broker is available; the method *HasMessages* returns true if the defined topic has messages; the method *ConsumeAll* reads all messages from an array of topics and returns a *List* containing arrays of strings; and the method *SeeNext* returns the next message that will be received, ignoring messages in line. Even though most methods are self descriptive, some important concepts must be discussed.

First, when a method consumes from more than one topic, if the return type is a *List* containing arrays of strings, the first position in each array is the consumed message and the second position is the

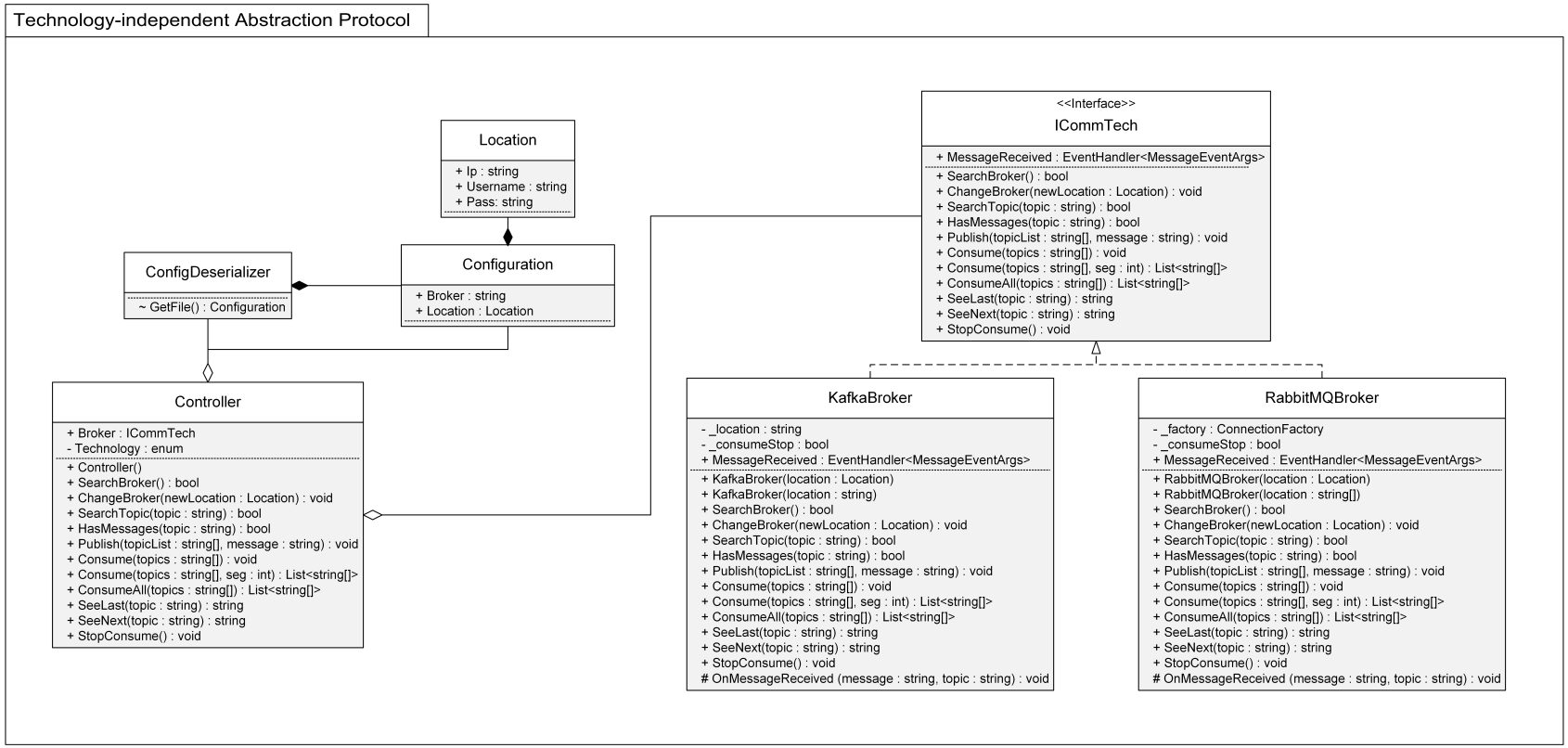


Figure 6.1: Class diagram representation of the implemented Technology-independent Abstraction Protocol.

topic of origin. Since the TIAP allows multiple subscription, there must be a way to differentiate between the origin of the messages, which explain this solution.

To increase flexibility for the data consumers, since it is one of the most important processes in communication protocols, there is an overload in the method *Consume*. The first one, that returns void, is a method for continuous consumption in a different thread, that fires the event *MessageReceived* when a message is consumed. The consumption must be stopped by the user with the *StopConsume* method.

The second overload is a method for synchronous consumption, that consumes according to a certain time. The consumption time, in seconds, is one of the inputs of the method.

Another important thing that should be noticed is that all messages consumed or published are strings. This means that the TIAP only guarantees integration in a syntactical level. The user is responsible for the semantic interoperability.

The most important difference between Figure 5.1 and 6.1 is the definition of the fields and methods and the specification of the two defined CTs. Nonetheless, the TIAP is a generic concept, which is the main reason for a distinction between the concept and the implementation. For different use-cases, the addition of more CTs implementing the *ICommTech* interface is not only viable but desirable.

The remaining classes were already introduced in the concept. Nonetheless, visualizing the implementation in Figure 6.1 helps to better understand the functional logic.

The *Controller* is the object that a user of the TIAP is going to create. The constructor will create a *ConfigDeserializer* and use its only method, the *GetFile*, to obtain an object of the type *Configuration*.

The method *GetFile* reads the information of a JavaScript Object Notation (JSON) file with the name and extension *configuration.json* that must be present in the working directory. Afterwords, the file is deserialized to create an object of the class *Configuration*. The framework used for the deserialization is the package Newtonsoft.Json (v12.0.1) [100]. This is the framework used every time there is the need to handle JSON files throughout the implementation.

The obtained object, of the class *Configuration*, contains the most important information for starting the communication with the middleware: the name of the CT, field *Broker*, and, in an object of the class *Location*, the broker's location in the network and authentication settings (if necessary). The class *Configuration* can be adapted to meet different CTs specific requirements, with only minor changes needed in the rest of the implementation.

With this information, the *Controller* can initialize its field *Broker*, that is an object of the *ICommTech* interface. Using an enumeration named *Technology*, containing the names of the communication protocols available, and a *switch* statement, the *Broker* is initialized with the corresponding class.

The rest of the functioning is straightforward, since the remaining methods are similar to the ones of *ICommTech*. Every method uses the field *Broker*, after it is initialised with the communication protocol, and simple applies the respective method from the corresponding class.

It can be argued that the *Controller* should implement the interface *ICommTech*. Although that would be logical in a technical perspective, in a conceptual perspective is not desirable, since the *Controller* is not a CT, it is the class that allows the user to control the TIAP. Thus, not implementing the interface

was a design choice, to ensure a logical deployment and avoid implementation mistakes.

TIAP's development considered the need of adding new CTs in the future, and that this inclusion should be performed with as little effort as possible. The most time consuming step is the creation of a new class that implements the *ICommTech* interface, since implementing a client of a new protocol can be a complex task. Afterwards, all that there is to do is create a new configuration file, eventually with some minor changes in the *Configuration* class, adding the name of the new broker to the enumeration *Technology* and adding a new *case* in the *switch* statement that the *Controller* uses to differentiate between the brokers.

6.1.4 Implementation of the TIAP in a Client

A brief explanation of the most important details for implementing the TIAP in a new client will be described. The most important steps are presented with the assumption that the middleware is already well configured and that communication with it is possible, something that might require some effort to accomplish in certain occasions.

The first step is to make sure the necessary packages are installed, that the TIAP's DLL is referenced and that the configuration file is well structured and in the working directory. Following, an object of type *Controller* must be initialised. Then, one must implement the methods using the correct syntax.

Using the developed library is straightforward, since emphasis was put on simplicity. Nevertheless, some comments must given regarding the two most used methods: *Publish* and *Consume*.

Since the *Publish* method is synchronous, it is advisable to call the method in a different thread. This applies mostly in CT where some kind of acknowledgement is expected, nonetheless it is a good practice in a general way, to avoid blocking the client.

As explained before, there are is a method overload in the *Consume*. One is a timed consumption while the other consumes indefinitely until the user calls the method *StopConsume*, firing the event *MessageReceived* every time a message is received. It is the second *Consume* that deserves two uncomplicated remarks: one must first subscribe the event and in the end of the program, or in another location the user finds relevant, the consumption must be interrupted with the appropriate method. Failure in this last step will result in blocks and errors that might be critical in a real industrial application.

6.2 Implementation of the Prototypical Use-Case

This section aims at clarifying the most important details of the implementation. First, an analysis on the development of the use-case clients is done, followed by the description of the deployment in terms of clients distribution in the network. The basis of the implementation is the prototypical use-case present in Figure 5.4.

6.2.1 Development of the Clients

Before discussing the implementation of the clients, the way data was passed between them must be explained. As mentioned before, the TIAP ensures only syntactic integration, so semantic interoperability must be guaranteed by the user.

To ensure integration, communication is based on strings in JSON format. Since focus was not on a common data format, most clients had their own classes for deserialization, with different fields. This presents the disadvantage of increasing complexity, but, on the other hand, ensures that only relevant data is present in every message, decreasing network usage. Nonetheless, applying a common data format to the developed protocol could be a feature to add in the future.

All clients where developed using the .NET Core 2.2 framework, for the same reason of the TIAP. The only exception was the HMI, as will be further discussed.

Industrial Plants

Regarding the MPS, the goal was to obtain data from eight digital outputs corresponding to the sensor values, so a new client was created. The plant is interfaced via USB and a FESTO EasyPort [101] that provides a virtual COM port for communication with the plant and a proprietary serial protocol. A software developed in previous projects translates the data between the virtual COM port and a raw TCP connection with a server.

To obtain the data from the sensors, a TCP client was used, that serves as a legacy adapter to obtain data from the plant's TCP server in a RR pattern. After obtaining the sensor values, data was converted to the right format and sent to other systems using the TIAP. This logic was contained in a new MPS client, created in the scope of this work for the developed use-case.

Regarding the *MyJoghurt*, it has an OPC UA server directly on a Beckhoff CX2040 PLC with TwinCAT3 TF6100 [102]. The PLC is connected to the plant via EtherCAT realtime Ethernet.

Data was obtained from the Beckhoff AM8111 servo drives, used to move four conveyors. Connection is made using the following client: OPCFoundation.NetStandard.Opc.Ua (v1.4.354.23) [103], which is the reference implementation of the OPC Foundation. This software is used as an adapter, in order to obtain data from the plant's PLC. Afterwards, the important payload is sent using the TIAP to the other clients in the architecture. The logic of the OPC UA adapter and of the communication using the TIAP is contained in a client developed for the prototypical use-case. The developed client can be located in a separate machine, as long as communication is possible.

In both cases, to take into consideration real-time industrial constraints, data acquisition must be performed with high frequency. In the motors from the *MyJoghurt*, anomalies might cause instant torque increases, that are quickly normalised. In the MPS, the object of analysis is time, so longer data acquisition time intervals represent an increase in measurement error. Thus, data was obtained in periods of around 100 ms.

Manufacturing Execution System

The MES receives data from both plants and the *Anomaly Detection* analyser. The goal is to store this data, and, in the case of the plants' data, send it to the HMI. The database for storage was merely illustrative, and, for simplicity, was implemented using a *List* from the *Collections.Generic* namespace.

The MES also had the management function of updating the thresholds for detecting anomalies. These values could be tuned by the user before starting the client and would then be sent to the *Anomaly Detection* analyser.

Analysers

The developed analysers were already explained in the previous chapter, and, regarding the *Time Interval* and the *Hampel Filter*, the implementation can be considered straightforward. These analysers consume the values, perform the necessary computation (computing time difference and filtering, respectively) and send the values to the next analyser.

Regarding the *Anomaly Detection*, the implementation also included a heartbeat report to be presented on the screen every minute. It is called *StatusUpdater*, since the goal is to update the user on the anomalies detected: where those anomalies came from, the part of the plant that presented the anomaly and the time of the anomaly.

This last analyser sends data to the MES only when there is an anomaly, reporting the anomaly so that the management level can access all the anomalies that happened in the system. Since the HMI intends to present a data visualisation of the values with and without anomalies, data is sent to this client every time a value is analysed, even if it is not an anomaly.

Human-Machine Interface

The HMI was develop using Windows Presentation Foundation (WPF), a graphical subsystem to develop user interfaces. It is the only client that was not developed with the .NET Core 2.2 framework, since it is not yet supported. The framework used was .NET Framework 4.6.1.

In Figure 6.2 the layout of the HMI is presented. Two text boxes are used, one to show the status, where the last anomaly detected is presented, and another present to the user how the data exchange is being handled. Also, the HMI displays four buttons, divided in two areas and, inside each area, one for the MPS and one for the *MyJoghurt*.

In the *Check Anomalies*, a graphical representation of all the values evaluated by the *Anomaly Detection* analyser is presented, with a clear distinction between anomalies and non-anomalies. This is useful, since it allows the user not only to check the anomalies and where they are located, but also to have a clear perception of how the values are evolving.

The other area, *Check Plant Values*, displays the raw data from the plants, as it is stored in the MES. Once again, this is important since the operator might want to check a value from a different variable or just have a perception of how the whole system is evolving, and not only the parts that are in focus for the anomalies.

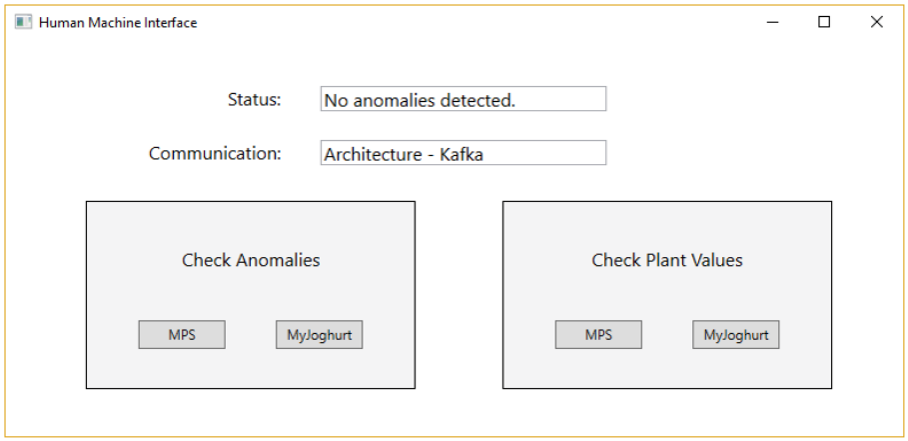


Figure 6.2: Layout for the developed HMI, in a case where the middleware was Apache Kafka and there were no anomalies detected.

The graphical representations were created using a open-source plotting library named Oxyplot, specifically the packages OxyPlot.Core (v1.0.0) and OxyPlot.Wpf (v1.0.0) [104].

In the next section, the hardware deployment of the different clients is presented.

6.2.2 Deployment of the Components

Since the focus of the study is *Industrie 4.0* and the integration of legacy systems, it is important to implement the use-case with a perspective on distributed systems. Thus, the setup was developed using machines in different locations.

For RabbitMQ and Apache Kafka, a machine running on Windows 10 with Hyper-V installed created two Virtual Machines (VMs) with Ubuntu 18.04.1 LTS. Each VM contained only one broker, either RabbitMQ or Apache Kafka, and no other clients were installed in the VMs. Thus, every client that connected to a broker was communicating from a different machine.

Besides the machine holding the VMs, two more were added, all within the same ethernet network. The setup was the following:

- 1st Machine - VM with broker, HMI and *MyJoghurt*;
- 2nd Machine - MPS and *Hampel Filter*;
- 3rd Machine - MES, *Time Interval* and *Anomaly Detection*.

In order to validate technological independence, the setup remains unchanged independently of the broker being used. The development of the setup also considered the P2P approach, that will be discussed in the following section.

6.3 Legacy Peer-to-Peer Approach

To perform the comparison defined in the fifth requirement, it is necessary to implement a use-case that guarantees the same functionality, but where the communication is performed in a legacy P2P

way. Additionally, using a RR messaging pattern, that better characterises legacy applications, is also desirable.

Along this line, direct TCP connections with RR pattern were implemented, with clients sharing the same internal logic as in the TIAP implementation. This is a low-level and simple to apply protocol, that is a good representation of legacy architectures.

The adapted use-case is represented in Figure 6.3.

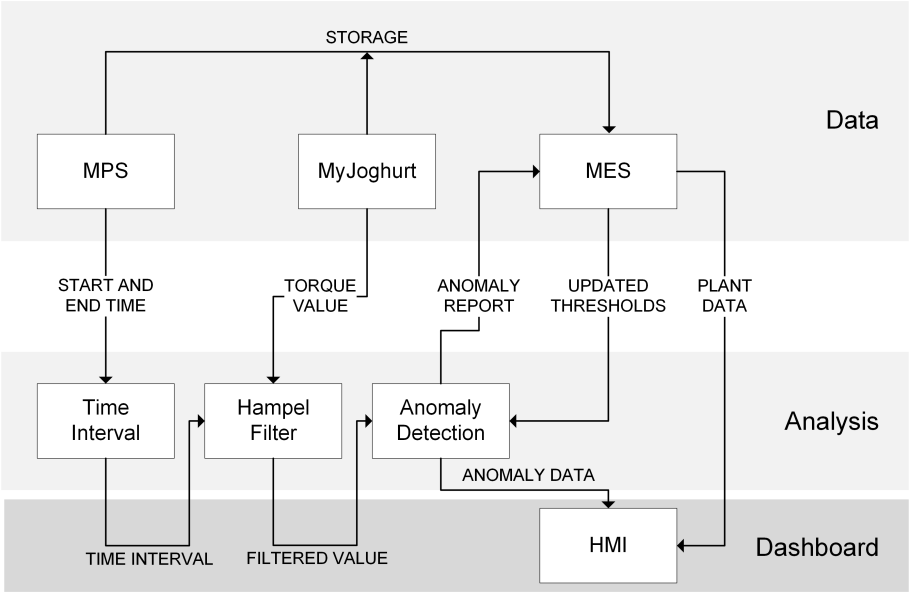


Figure 6.3: Structured, graphical representation of the implemented use-case, using direct TCP connections (the connection with the legacy plants is not represented, for simplicity).

This protocol was implemented using the classes *TcpListener* and *TcpClient* from the *Net.Sockets* namespace.

The *TcpClient* is a client that sends data to a defined address and port. It waits for the response of the server before finishing the task, and requires that a server is always listening. This can lead to the application blocking for some seconds while the *TcpClient* is trying to establish a connection. Thus, using a different thread is advantageous and can prevent critical problems.

The *TcpListener* is a server that continuously listens in a certain port and that sends a response to the client when a message is received. It was programmed similarly to the *Consume* method that consumes indefinitely from the TIAP. It fires an event when a message is received and there is also a method used to stop consumption, which should always be done at least in the end of the program developed for the respective system.

Each client present in the use-case implements a *TcpListener*, a *TcpClient*, or both. The implementation in the different clients was programmed with the original logic and without encapsulation. It can be argued that it would require less effort to, just like in the TIAP, create a DLL that would be referenced by each client. This makes sense in a deployer point of view, nonetheless, it is important to take into account that the comparison is between the usage of the developed protocol and the procedures of the legacy TCP-based communication, which is the reason why it is implemented in a legacy way, instead of being abstracted in a library. Besides, most of the times the systems were added in a intermittently,

so it can be argued that effort was not put in optimising code abstraction but on connecting the singular clients in the architecture. Thus, the comparison only makes sense when the developed protocol is put side-by-side with a legacy protocol, implemented in a legacy way.

Further details of the implementation and setup are the same as for the technology-independent data acquisition architecture's use-case, thus a more complete description of this use-case would be redundant.

6.4 Evaluation Metrics

In this section, the metrics used for computation of effort will be introduced. The first three metrics analysed are included in the Visual Studio environment, and must be presented according to the way they are used in that software [105]. The last metric is a commercial software from ProjectCodeMeter [106].

The metrics might be of complexity, of effort or both. Complexity metrics evaluate the complexity of the source code. Effort metrics evaluate the time it takes to develop it. In Chapter 3 a distinction between complexity and effort was discussed, when applied to the comparison. It is important to clarify that, when defining the metrics that will be used to compare the programming effort needed in both approaches, both complexity and effort metrics can be important. In a general way, a more complex code requires more time to be developed, thus requiring more effort.

The first metric is the Lines of Code (LoC). It is, originally, a size metric, that can be used to have a first idea on complexity and effort. There are some variants of this metric, but, in Visual Studio, it simply counts the number of executable of code. When the number of LoC is high, the source code is probably too complex and should be divided for simplicity. This is a useful metric, that gives a clear first impression of the complexity of the code.

Another metric available is the Cyclomatic Complexity (CC). This is a complexity metric based on the number of decision logic statements in the source code. For example, if a *switch* statement with four possibilities is added in the code, the CC would increase by four units. A high value indicates high complexity. By measuring the number of linearly independent paths in the code, it can compute its complexity, giving a valid notion of effort.

Maintainability Index (MI) is a metric that is computed from other metrics, but with a divergent meaning. It focus on the simplicity for implementing modifications in the system, something that conjugates perfectly with *Industrie 4.0*, which needs smart systems that can easily change according to different use-cases. Focusing solely on the developed architecture, that is independent of the technology, maintainability is also important, mostly in a migration scenario. The result of this metric is a percentage, and the code is considered easier to maintain, therefore less complex and requiring less effort, if the value is closer to 100%.

Weighted Micro Function Points (WMFP) is an effort metric that breaks the source code in micro functions and computes a final effort score from complexity and volume metrics. Comparing to the three metrics derived by Visual Studio, it offers a different type of comparison, since the result is an estimate

of number of hours for coding, debugging and testing. Thus, this is an interesting metric that brings a new perspective to the comparison, and is used for that reason.

Summing up, four metrics are going to be applied: LoC, CC and MI from Visual Studio [105] and WMFP from ProjectCodeMeter [106].

Chapter 7

Evaluation and Discussion

The purpose of the present chapter is to evaluate the two main hypothesis of this thesis, introduced in Chapter 1. It focus on the proof-of-concept and on the comparison between the developed architecture and a legacy P2P approach, respectively the first and second hypothesis. Finally, the fulfilment of the requirements introduced in Chapter 3 is summarised.

7.1 Proof-of-Concept Results

The goal of the present section is to evaluate if a data acquisition architecture can be independent of the technology used in the middleware's communication, thus verifying the concept of the architecture defined in Chapter 5.

The evaluation should start by focusing on the conceptual requirements and on the operating results in terms of communication. This is relevant since a technology-independent architecture where data acquisition is not correctly achieved is not an important contribution to scientific research and industry. The first three requirements will be considered.

After validating the concept for data acquisition, the migration between different communication protocols should be evaluated. in order to discuss technological independence. In this case, the evaluation will take into account if the migration was successfully accomplished, according to the fourth requirement, and if it represented a loss in functionality.

7.1.1 Communication and Data Acquisition

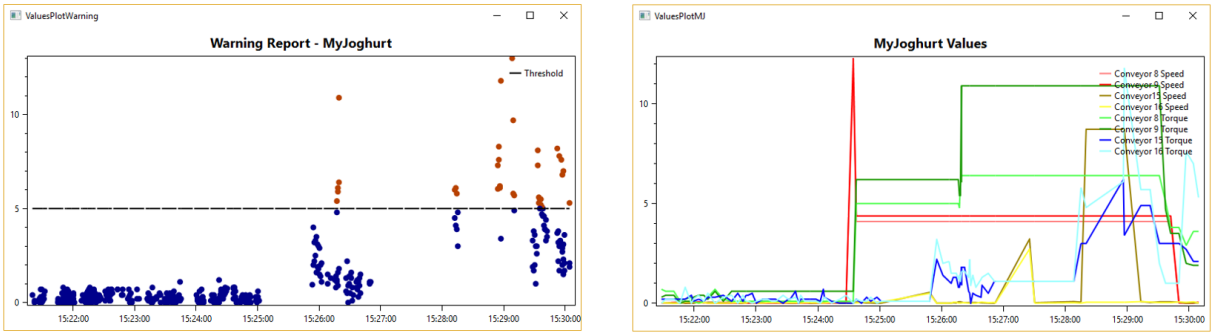
For both scenarios, with Apache Kafka and RabbitMQ, all messages where delivered in real-time and without message loss, so proof-of-concept in a prototypical implementation (R2) was achieved. Additionally, the analysers were able to extract knowledge from the data and to display it graphically in a HMI, which also presented historical data stored in a MES system. Thus, the use-case considered in this contribution reflects a typical, although simplified, *Industrie 4.0* environment with numerous heterogeneous systems in a connected aPSs (R1).

Examples of the graphics obtained with the HMI are shown in Figures 7.1 and 7.2.

Regarding interoperability, all data was transparently available to all connected systems (R3-a), since every system was able to consume or produce to topics that were available in the MOM to all the other systems, without the need to create a new communication line. Legacy systems could also be integrated, by using data adapters and communicating over the middleware using a common programming interface (R3-b). This was the case of the two prototypical plants.

It must be emphasised that the important contribution of this paper is not the successful implementation of the CT's logic, but the integration of the technologies in a common protocol that abstracts their functioning and creates a new technology-independent protocol. In fact, the technologies used - Apache Kafka and AMQP - have been successfully implemented in a diverse set of applications.

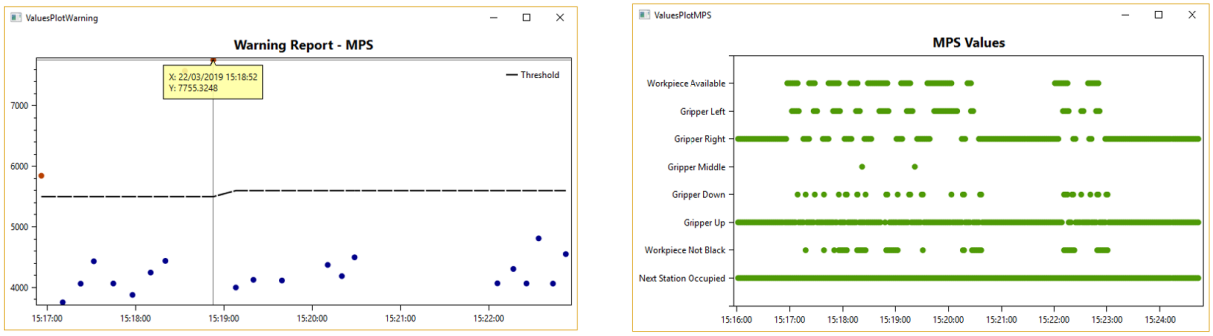
Despite this being a simple use-case, an important point can be made regarding scalability. Since all technologies used were implemented in real and more complex environments, it can be said that a more complex use-case would also be successfully implemented with the TIAP.



(a) Values evaluated for anomalies

(b) Raw plant values

Figure 7.1: Graphical output from the HMI for the *MyJoghurt* plant.



(a) Values evaluated for anomalies

(b) Raw plant values

Figure 7.2: Graphical output from the HMI for the MPS plant.

7.1.2 Migration between Communication Protocols

A technology-independent concept, as explained in the fourth requirement, is validated by having a use-case successfully implemented with two CTs with a straightforward migration. A straightforward migration is achieved when the change of CTs is made with as little effort as possible.

In the implementation, the architecture was first deployed with the TIAP abstracting the AMQP

and communicating over a RabbitMQ broker. Afterwards, the configuration files of each client where changed, and the scenario was successfully migrated to Apache Kafka. Despite changing the base communication protocol and making communication over a MOM in a different location, the functionality of both scenarios remained unchanged (R4-a).

During the migration process, no changes on the source code of the clients were necessary as the reconfiguration could be handled by changes to the configuration file of each application only, with zero changes in the source code of each client. Therefore, migration between different technologies using the TIAP is straightforward as the architecture is developed in a technology-independent way using the common programming interface (R4-b).

The effortless migration between technologies can be important in many deployments. For example, a company that uses Apache Kafka but wants to gradually make a change to DDS due to concerns with QoS, can use this architecture to make improvements in the new deployment while keeping the previous untouched, easily migrating between both CTs. Moreover, a company can have distinct setups that benefit from different technologies, so a straightforward migration in the systems that participate in both use-cases is important. By using the developed architecture, based on the TIAP, these companies could greatly reduce the effort for migration

To conclude, the implementation was successful in terms of migration between CTs, since it was possible to change the communication protocol by updating a configuration file, without changing a single line in the source code and without losing functionality (R4). The first research gap identified in Chapter 4 is considered filled.

7.2 Comparison with Classical Approach

The comparison with a classical P2P approach, as stated in the fifth requirement, is based on complexity and effort. As previously explained, the comparison evaluates the conceptual complexity and the implementation effort. Table 3.3 gives a more clear explanation of what should be evaluated.

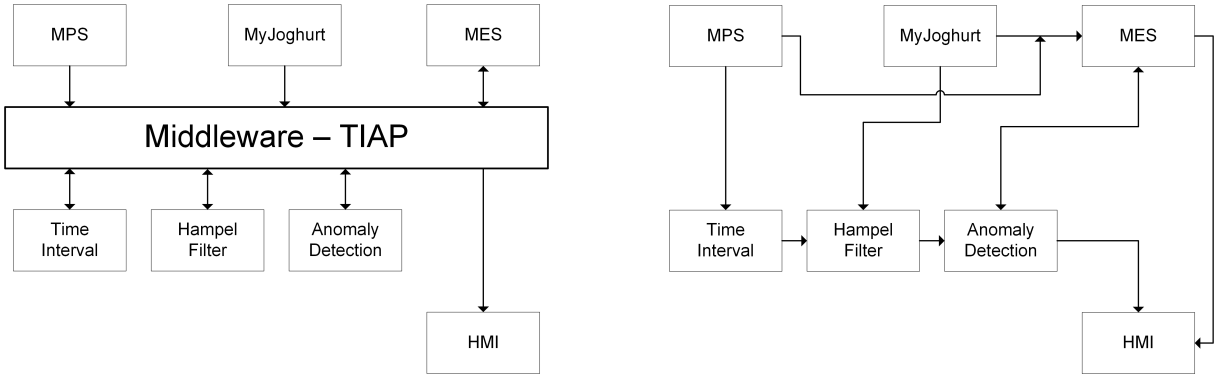
In this section, the complexity will be evaluated, followed by the effort comparison. It finishes with a discussion about how the two approaches could be compared in different scenarios.

7.2.1 Complexity of the Concept

The comparison with the classical P2P approach, as stated in R5, is based on the conceptual complexity and the implementation effort.

According to the sub-requirements, the complexity's comparison is based on two main factors: transparent data access and a reduced number of interfaces. Figure 7.3 clarifies clarifies that transparent data access is only achieved in the MOM-based implementation, since every client can subscribe or publish to another client's topic (R5-a).

Regarding the number of communication channels, there are seven connections necessary for the given use-case, while the P2P implementation accounts for nine. For a fully connected system (each



(a) TIAP's Implementation

(b) TCP-based Implementation

Figure 7.3: Schematic representation of the two scenarios. TIAP middleware approach with 7 point-to-point connections (left) and the classical peer-to-peer approach with 9 point-to-point connections (right).

system connected to each other), the number of channels is n for the TIAP approach and $n(n-1)/2$ for the P2P approach, as can be seen in Figure 2.2. Hence, the middleware based approach also shows less complexity regarding the number of interfaces (R5-b).

Despite these two sub-requirements, another disadvantage was found in the P2P approach during the implementation, that reflects conceptual complexity. If one client changes its address in the MOM-based architecture, as long as it is not the middleware, only his connection to the middleware must be updated. In a TCP-direct deployment, every component that communicates with that system must at least update its address to the new one. In a system with hundreds of clients, this task is not only impractical but can even be unachievable if the system is too complex and not well documented, as probably happens in a lot of companies.

7.2.2 Effort for Implementation and Migration

The comparison of effort is based on two factors: reduced effort for implementation and migration. To evaluate effort, as explained in Chapter 6, code metrics are going to be used.

In Table 7.1, the results of the code metrics analysis for each client are presented.

Table 7.1: Raw values of code metrics for each connected system for both TIAP and P2P scenarios.

	TIAP				P2P			
	LoC	CC	MI (%)	WMFP (h)	LoC	CC	MI (%)	WMFP (h)
MPS	284	49	87	109	321	49	83	117
MyJoghurt	538	83	87	197	573	83	84	204
MES	151	16	81	62	260	27	79	97
Time Interval	110	20	87	43	219	29	85	75
Hampel Filter	207	29	80	88	317	38	79	119
Anomaly Detection	223	43	86	94	335	54	84	126
HMI	1,124	161	82	631	1,245	173	81	668

Table 7.2: Comparison of the relative code metrics for each connected system. Table summarises the ratios between code metric for TIAP scenario divided by code metric for P2P scenario.

	TIAP/P2P			
	LoC	CC	MI	WMFP
MPS	88%	100%	105%	93%
MyJoghurt	93%	100%	104%	97%
MES	58%	59%	103%	64%
Time Interval	50%	69%	102%	57%
Hampel Filter	65%	76%	101%	74%
Anomaly Detection	67%	80%	102%	74%
HMI	90%	93%	101%	94%

As explained lines of Code (LoC) is the number of functional lines in the code, so more lines of code imply more effort. In the case of Cyclomatic Complexity (CC), an higher number also means more effort, as explained in the previous chapter. The Maintainability Index (MI) is a number between 0 and 100, where a higher number indicates higher maintainability, so less effort.

The results of WMFP are presented in programming hours. The presented results are not to be taken into real consideration in terms of raw value, but are important as a comparative reference between the effort necessary to deploy a solution that uses each CT. The values do not represent reality, mostly due to the fact that the code was developed in a modular way generic parts could be reused in different clients.

Table 7.2 summarises the ratios between the code metrics for the TIAP scenario divided by code metrics for the P2P scenario, and therefore states the relative effort of the TIAP approach compared to the classical P2P architecture.

All metrics show an increase in effort in the P2P approach compared to TIAP. By abstracting the time-consuming and complex communication-specific logic inside a DLL, the clients are able to communicate with easy to use and abstract methods. Thus, less lines of code are used (lower LoC), there are less linearly independent paths in the code (lower CC) and implementing changes is simpler (higher MI). This results in less programming hours (lower WMFP) and a decrease in effort that can be as much as 50% (R5-c). It should be emphasised that the comparison focused on the clients with all their logic, instead of only comparing the methods for communication, so differences in effort are less evident in complex clients, like the HMI.

When considering migration, the metrics would all point to zero effort in case of a TIAP implementation, since no lines of code are changed. It is clear that changing the legacy protocol used in the P2P scenario to another protocol would always require changes to the code that would be reflected in the code metrics. Regarding the number of necessary steps, TIAP only requires one: updating the configuration file. A migration between the legacy P2P protocol and another protocol keeping all functionality and requiring only one step to be executed can be considered impossible (R5-d).

Therefore, it was proved that an architecture based on the TIAP requires less effort for deployment and migration.

7.2.3 Discussion on Different Use-Cases

The results obtained in the previous sections were very positive for the developed architecture, that fulfilled all sub-requirements. Nonetheless, one can ask if this would also apply in different use-cases, like a greenfield deployment.

Considering the complexity comparison, it should be noted that the only way to achieve transparent data access in a P2P setup is by increasing the number of interfaces to the maximum, by connecting all clients to each other. Thus, only a P2P deployment with two elements, or another with three would have less or equal complexity, respectively, when compared to a MOM-based architecture (Figure 2.2). If there are more than three clients and the legacy architecture aims at having data transparently available, a implementation with a middleware would always have lower complexity.

Regarding effort, it might be argued that, by using a different CT, rather than TCP, the effort for deployment could be lower. Even though the results obtained using the TIAP are very positive, it is indeed possible that some protocols require less effort for implementation. The same does not apply to the effort for migration, since changes in the source code are always needed in order to migrate between different CTs.

7.3 Requirements Fulfilment

A comparison with the requirements developed in Chapter 3 must now be accomplished. A summary of the requirements can be found in Table 3.4.

Regarding R1, *Industrie 4.0* was taken into account mainly considering three factors: all devices present were integrated, the integration was performed between different levels of the automation pyramid (presence of the MES) and data from different plants was also acquired in the same system.

The prototypical implementation of the use-case defined in Section 5.2 ensures that the second requirement was also fulfilled.

The interoperability of data sources and systems (R3) was also fulfilled, according to the sub-requirements defined in Table 3.1. The middleware is a single interface that connects all devices and, by integrating the two plants that used different CTs, the presence of legacy devices was ensured.

The fourth requirement was fulfilled by the use of two different communication protocols in the middleware. This was made possible without a single line of code changed in the clients, only a change of the configuration file was necessary. Consequently, the implementation was completely independent of the technology, so the architecture not only fulfils the fourth requirement but also respects the first hypothesis of this thesis.

Considering the last requirement, R5, the sub-requirements must be considered (Table 3.3). Data was made transparently available by using a middleware: each client connected to the broker could sub-

scribe to any topic, ensuring access to data from every other client. As discussed in previous sections, the number of interfaces was also smaller in the middleware architecture's implementation (7) comparing to the TCP direct deployment (9). While in this implementation the difference might not be substantial, if a new system that needs to consume data from every other client is considered, the difference would be much more relevant: in the architecture there would be one more connection and in the P2P deployment the number of interfaces would double.

The last two sub-requirements, focused more on a technical perspective, were also accomplished. By using the evaluation metrics, the effort for implementation was shown to be lower comparing to the legacy TCP-direct. Regarding migration, it can be argued that, in terms of code metrics, the effort for migration to a different CT is zero. If no line of code is changed, the programmer does not need to do anything, so there is no effort. The number of actions necessary in the architecture for migration is one: changing the configuration file. In terms of the P2P deployment, migration was not implemented, nonetheless, it is clear that the effort could never be inferior than just zero lines of code and only one action, so the sub-requirement is fulfilled. Therefore, the fifth and last requirement was accomplished: the developed architecture presented lower effort and complexity compared to a legacy approach.

In summary, the architecture was successfully implemented in terms of data acquisition and in terms of migration between communication protocols in the middleware in a *Industrie 4.0* environment (R1–R4). Therefore, it was proved that an architecture based on the TIAP requires less effort for deployment and migration (R5). The fulfilment of the requirements R1 to R5 proves the contribution of this work to fill the identified research gaps.

Chapter 8

Conclusions and Future Work

The emergent field of *Industrie 4.0* requires data integration in all automation levels, something that is only possible if the right data acquisition architecture is used. One of the most prominent solutions is the use of architectures based on a Message-Oriented Middleware (MOM), a software entity that acts like a common interface between all systems, allowing for communication between them. The goal of this thesis is to conceptualise and evaluate a data acquisition architecture that is independent of the communication technology (CT) used in the MOM. Two hypothesis were stated and discussed in Chapter 1.

First, it was intended to prove that a data acquisition architecture can be independent of the CT used in the middleware. Technology independence is relevant since it ensures flexibility of the architecture by decoupling concept and implementation, preventing a lock-in to a specific technology. It is also identified in the literature as a requirement for MOM-based architectures.

Additionally, one must prove that a technology-independent MOM-based architecture for data acquisition, when compared with a legacy P2P approach in terms of complexity of the concept, effort in deployment and effort in migrating between different communication technologies, will show improvements. This comparison is relevant since it can highlight the advantages of the developed solution when putting it against the common practice, validating the necessity for such architecture.

To achieve technology independence, the architecture should abstract at least two different CTs behind a common interface and allow an easy migration between these technologies. Regarding the comparison, all data should be available and the number of communication lines should be lower, in order to reduce complexity. To evaluate reduced effort in implementation, code metrics can be used. In the case of migration, the number of actions needed to migrate should be minimised, along with the use of code metrics. Furthermore, the developed architecture should be applied to *Industrie 4.0*, being implemented in a prototypical use-case and ensure interoperability of legacy and non-legacy data sources, completing a set of requirements presented in Chapter 3.

The following sections will focus on the contributions based on the two aforementioned hypothesis, that were successfully proven. Additional contributions are also presented, followed by a few ideas for future work.

8.1 Concept of the Technology-independent Architecture

After an analysis on state of the art data acquisition architectures, in Chapter 4, a research gap related to technology independence was found. No architecture conceptualises the same use-case with different CTs, implemented that concept in an industrial prototype and migrated between the protocols without losing functionality. Thus, no architecture can be considered technology-independent.

According to the identified research gap, and considering the diverse set of CTs available, with different specifications and advantages, a data acquisition architecture is conceptualised independently of the communication protocol. This architecture, presented in Chapter 5, aims at integrating all systems, including legacy devices from all levels of automation in a common MOM. This architecture also accounts for the presence of analysis and dashboard clients.

The developed architecture was implemented in a prototypical use-case, considering two industrial plants communicating over different legacy protocols, one MES system, three analysers and one HMI. The use-case was implemented as a network of distributed heterogeneous systems, reflecting a typical *Industrie 4.0* environment. Communication was done over a MOM, capable of migrating between two scenarios (using different CTs, namely Apache Kafka and AMQP) without losing functionality. The implementation details are presented in Chapter 6.

For both scenarios, all messages where delivered without message loss and proof-of-concept in the prototypical implementation was achieved. During the migration process, no changes on the source code of the clients were necessary, as the reconfiguration could be handled only by changing the configuration file of each application. Thus, the architecture was successfully implemented in terms of data acquisition and in terms of migration between communication protocols in the middleware in an *Industrie 4.0* environment.

The developed architecture allows more flexible deployments, by easing migration between protocols. It can be used for both greenfield and brownfield deployments, with the added advantage of allowing a gradual evolution to *Industrie 4.0* scenarios, due to the use of a MOM capable of integrating legacy devices with new systems.

The implementation of the architecture assures its suitability for data integration in an industrial context, being the only technology-independent architecture already validated. Enterprises that require similar deployments with different CTs would benefit from this architecture, since a straightforward migration reduces downtime and, consequently, increases the OEE, one of the main goals of *Industrie 4.0*.

In conclusion, this work contributes with the concept and implementation of a new architecture for *Industrie 4.0*, independent of the technology used in the MOM.

8.2 Evaluation of the Technology-independent Architecture

In the state of the art analysis (Chapter 4), it was concluded that a formal comparison with a legacy P2P approach is not found in the literature. Although it is possible to assert a reduction in complexity when a

MOM is used, no concept pointed this out with a complexity and effort comparison, so a second gap in the literature was found.

In order to fill the second research gap and confirm the relevance of the new architecture, the implementation of the prototypical use-case previously described was compared with a legacy P2P approach. For this, a new implementation was described (Chapter 6), with the communication based on direct TCP connections, ensuring the same functionality as in the architecture's implementation.

The results of the comparison are described in Chapter 7. Regarding complexity, it was verified that the MOM-based approach allowed for transparent data access and for a reduction in the number of communication lines. Hence, it was confirmed that implementing the developed architecture ensured a reduction in complexity.

Regarding effort in implementation, an analysis based on four different code analysis metrics showed that the effort to deploy the MOM-based use-case is lower than the effort required when implementing the P2P use-case. In the migration scenario, the metrics would all point to zero effort in the architecture's implementation, since no lines of code are changed. Additionally, only one step is necessary for migration: changing the configuration file. For the P2P, it can be said that a migration between the legacy CT and another protocol, keeping all functionality and requiring only one step to be executed, is impossible. Thus, the developed architecture also ensures a reduction in effort, both in implementation and in migration.

This comparison validates the architecture for enterprises wishing to achieve *Industrie 4.0*, by adding a reduction in effort and complexity to other advantages of this revolution. Such reduction is positively reflected in the operating costs, since complex systems are harder to deploy and maintain, while increased effort requires an increase in working hours. Thus, a change to the developed architecture would be beneficial in most cases.

To conclude, this work contributes with a complexity and effort comparison between the architecture's implementation and a P2P use-case that ensures the same functionality.

8.3 Additional Contributions

In order to implement a technology-independent architecture, a new protocol was developed, called Technology-independent Abstraction Protocol (TIAP). It abstracts the specific CT logic behind a common interface, allowing clients to be implemented independently of the underlying protocol actually used for communication.

Although created for the developed architecture, the TIAP can be implemented in a variety of architectures and use-cases, so it is considered a different contribution.

Another contribution of this work is the qualitative comparison of relevant CTs in the field of automation. This comparison focused on significant features for the field, like messaging patterns and security.

This comparison is a useful reference for enterprises that need to decide between different CTs to implement. Additionally, it fulfils a gap in the literature for comparisons of the kind.

Finally, as stated in Chapter 1, a conference paper based on this work was submitted.

8.4 Future Work

Since the two hypothesis of this work were proven and the requirements were fulfilled, the goal of this thesis was achieved, opening the possibility for new developments and improvements. Thus, a few ideas for future work are presented, organised with respect to the main contributions of this work.

First, effort should be made in adding new requirements in the developed architecture, like a common information model and support for enhanced security features. Furthermore, technological development creates the necessity of continuously updating the architecture to ensure that it does not become a legacy solution for *Industrie 4.0* deployments.

Following future improvements in the architecture, new implementations in real industrial environments, with more complex use-cases, would be relevant to verify the advantages of the concept. Further, migration should consider a different number of CT, so effort should be made in adding new communication protocols to the TIAP.

Considering the last point, it should be evaluated if all presented industrial communication protocols can be supported by TIAP or if it first needs to be modified or extended. Moreover, a layered implementation of TIAP could provide better possibilities for tweaking the internals of the protocol, if needed, and provide developers with more possibilities (higher abstraction versus easier customisation).

The comparison with a classical P2P approach can be considered a closed topic, since the improvements were clear. Nonetheless, a prototypical comparison between the developed architecture and other existing architectures for *Industrie 4.0*, as some of the discussed in the state of the art (Chapter 4), could be important. This comparison would lead to a better distinction between the different architectures, and could potentially generate a new and enhanced concept, that would adopt the best features of each architecture.

Finally, and accounting for the constant development of the studied CTs, the comparison between technologies will need to be continuously actualised. Taking a step further in the comparison, for instance with a quantitative approach, would be a good improvement.

Following the submission of the paper for a conference, there is also the intention of applying some of the ideas presented in this section, to create and submit a journal article.

All suggestions for future work keep in mind the transition to *Industrie 4.0*, in the same way this thesis did. The final goal is to keep research going in order to improve human life while creating a new, efficient and sustainable Industry.

Bibliography

[1] H. Kagermann, W. Wahlster, and J. Helbig. Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Final report of the Industrie 4.0 Working Group, 2013. URL www.plattform-i40.de/finalreport2013.

[2] E. Trunzer, S. Lötzerich, and B. Vogel-Heuser. Concept and Implementation of a Software Architecture for Unifying Data Transfer in Automated Production Systems. In O. Niggemann and P. Schüller, editors, *IMPROVE - Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency*, pages 1–17. Springer Vieweg, Berlin, Heidelberg, 2018.

[3] P. Sommer, F. Schellroth, M. Fischer, and J. Schlechtendahl. Message-oriented Middleware for Industrial Production Systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, aug 2018.

[4] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[5] The Association of German Engineers (VDI). VDI/VDE 2657 - 1, Middleware in industrial automation - Fundamentals, 2013.

[6] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson. An event-driven manufacturing information system architecture for Industry 4.0. *International Journal of Production Research*, 55(5):1297–1311, 2017.

[7] S. J. Hu. Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia CIRP*, 7:3–8, 2013.

[8] 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):17–27, mar 2017.

[9] IEC 62264:2013, ISA95 – Enterprise-Control System Integration, 2013.

[10] B. Vogel-Heuser and D. Hess. Guest editorial industry 4.0–prerequisites and visions. *IEEE Transactions on Automation Science and Engineering*, 13(2):411–413, April 2016.

[11] B. Vogel-Heuser, G. Kegel, K. Bender, and K. Wucherer. Global Information Architecture for Industrial Automation. *Automatisierungstechnische Praxis (atp)*, 1(51):108–115, 2009.

[12] T. Sauter. The continuing evolution of integration in manufacturing automation. *IEEE Industrial Electronics Magazine*, 1(1):10–19, 2007.

[13] D. M. Dilts, N. P. Boyd, and H. H. Whorms. The evolution of control architectures for automated manufacturing systems. *Journal of Manufacturing Systems*, 10(1):79–93, 1991.

[14] C. Cimini, R. Pinto, and S. Cavalieri. The business transformation towards smart manufacturing: a literature overview about reference models and research agenda. *IFAC-PapersOnLine*, 50(1): 14952–14957, 2017.

[15] P. Leitão, J. Barbosa, A. Pereira, J. Barata, and A. W. Colombo. Specification of the PERFoRM architecture for the seamless production system reconfiguration. *IECON Proceedings (Industrial Electronics Conference)*, pages 5729–5734, 2016.

[16] S. Vinoski. Do you know where your architecture is? *IEEE Internet Computing*, 7(5):86–88, Sep. 2003.

[17] J. P. A. Almeida, M. Van Sinderen, L. F. Pires, and M. Wegdam. Platform-independent dynamic reconfiguration of distributed applications. *Proceedings - 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 286–291, 2004.

[18] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15): 2787–2805, 2010.

[19] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[20] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, 17(4):18–25, July 2013.

[21] V. Gunes, S. Peter, T. Givargis, and F. Vahid. A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Transactions on Internet and Information Systems*, 8(12):4242–4268, 2014.

[22] J. Lee, B. Bagheri, and H. A. Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[23] S. Izza. Integration of industrial information systems: from syntactic to semantic integration approaches. *Enterprise Information Systems*, 3(1):1–57, 2009.

[24] F. B. Vernadat. Enterprise modeling and integration (EMI): Current status and research perspectives. *Annual Reviews in Control*, 26:15–25, 2002.

[25] D. A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[26] M. . Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797, 2005.

[27] Industrial Internet Consortium. The Industrial Internet of Things Volume G5: Connectivity Framework, 2017.

[28] Deutsches Institut für Normung e.V. (DIN). Reference Architecture Model Industrie 4.0 (RAMI4.0), 2016.

[29] Industrial Internet Consortium. The Industrial Internet of Things Volume G1: Reference Architecture, 2017.

[30] International Organization for Standardization (ISO). Information technology – Internet of Things Reference Architecture (IoT RA), 2016.

[31] J. Gantz and D. Reinsel. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. *IDC iView: IDC Anal. Future*, 2007(December 2012):1–16, 2012.

[32] X. Wu, X. Zhu, G. Wu, and W. Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, Jan 2014.

[33] M. O. Gokalp, K. Kayabay, M. A. Akyol, P. E. Eren, and A. Koçyiğit. Big data for industry 4.0: A conceptual framework. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 431–434, Dec 2016.

[34] H. Xu, W. Yu, D. Griffith, and N. Golmie. A survey on industrial internet of things: A cyber-physical systems perspective. *IEEE Access*, 6:78238–78259, 2018.

[35] Socrades Project 2006-2009, 2019. URL `socrades.net`. Accessed: 2019-04-20.

[36] L. M. S. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. Socrades: A web service based shop floor integration infrastructure. In C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. E. Sarma, editors, *The Internet of Things*, pages 50–67, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[37] S. Karnouskos, T. Bangemann, and C. Diedrich. Integration of Legacy Devices in the Future SOA-based Factory. *IFAC Proceedings Volumes*, 42(4):2113–2118, 2009.

[38] C. Marín, L. Mönch, P. Leitão, P. Vrba, D. Kazanskaia, V. Chepegin, L. Liu, and N. Mehandjiev. A conceptual architecture based on intelligent services for manufacturing support systems. *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, pages 4749–4754, Oct 2013.

[39] P. Leitão, J. Barbosa, P. Vrba, P. Skobelev, A. Tsarev, and D. Kazanskaia. Multi-agent system approach for the strategic planning in ramp-up production of small lots. *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, pages 4743–4748, 10 2013.

[40] K. A. Qureshi, W. M. Mohammed, B. R. Ferrer, J. L. Lastra, and C. Agostinho. Legacy systems interactions with the supply chain through the C2NET cloud-based platform. *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, pages 725–731, 2017.

[41] F. Perez, E. Irisarri, D. Orive, M. Marcos, and E. Estevez. A CPPS Architecture approach for Industry 4.0. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, volume 2015-Octob, pages 1–4. IEEE, Sept 2015.

[42] A. Ismail and W. Kastner. A middleware architecture for vertical integration, April 2016.

[43] H. Fleischmann, J. Kohl, and J. Franke. A reference architecture for the development of socio-cyber-physical condition monitoring systems. *2016 11th Systems of Systems Engineering Conference*, 2016.

[44] H. Fleischmann, J. Kohl, and J. Franke. A modular architecture for the design of condition monitoring processes. *Procedia CIRP*, 57:410 – 415, 2016. Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems.

[45] H. Fleischmann, J. Kohl, and J. Franke. A modular web framework for socio-CPS-based condition monitoring. *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2016-June, 2016.

[46] Basissystem Industrie 4.0: Eine offene Plattform für die vierte industrielle Revolution, 2019. URL `basys40.de`. Accessed: 2019-04-21.

[47] Virtual Automation Bus, 2019. URL `wiki.eclipse.org/BaSyx.VAB`. Accessed: 2019-04-21.

[48] F. Gosewehr, J. Wermann, W. Borsych, and A. W. Colombo. Apache camel based implementation of an industrial middleware solution. *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018*, pages 523–528, 2018.

[49] G. Angione, J. Barbosa, F. Gosewehr, P. Leitão, D. Massa, J. Matos, R. S. Peres, A. D. Rocha, and J. Wermann. Integration and deployment of a distributed and pluggable industrial architecture for the perform project. *Procedia Manufacturing*, 11:896 – 904, 2017. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[50] E. Trunzer, I. Kirchen, J. Folmer, G. Koltun, and B. Vogel-Heuser. A flexible architecture for data mining from heterogeneous data sources in automated production systems. In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 1106–1111, March 2017.

[51] D. Schel, C. Henkel, D. Stock, O. Meyer, G. Rauhöft, P. Einberger, M. Stöhr, M. A. Daxer, and J. Seidelmann. Manufacturing service bus: An implementation. *Procedia CIRP*, 67:179 – 184, 2018. 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 19-21 July 2017, Gulf of Naples, Italy.

[52] J. Minguez. Der manufacturing service bus, 2013. In: Westkämper E., Spath D., Constantinescu C., Lentes J. (eds) Digitale Produktion. Springer, Berlin, Heidelberg.

[53] A. Kirmse, V. Kraus, M. Hoffmann, and T. Meisen. An Architecture for Efficient Integration and Harmonization of Heterogeneous, Distributed Data Sources Enabling Big Data Analytics. In *Proceedings of the 20th International Conference on Enterprise Information Systems*, volume 1, pages 175–182. SCITEPRESS - Science and Technology Publications, 2018.

[54] D. Hästbacka, P. Kannisto, and M. Vilkko. Data-driven and event-driven integration architecture for plant-wide industrial process monitoring and control. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pages 2979–2985, Oct 2018.

[55] F. Longo, L. Nicoletti, and A. Padovano. Ubiquitous knowledge empowers the smart factory: The impacts of a service-oriented digital twin on enterprises' performance. *Annual Reviews in Control*, 2019.

[56] FESTO. Learning Systems: Modular Systems for Mechatronics Training, 2007. URL `festo.com/uslearningsystems`.

[57] Institute of Automation and Information Systems. MyJoghurt Demonstrator, 2019. URL `i40d.ais.mw.tum.de/index/industrie`. Accessed: 2019-04-10.

[58] *Synchronous servomotor AM8100*. Beckhoff, Huelshorstweg, Germany, 2.0 edition, March 2018.

[59] Institute of Automation and Information Systems. Festo-Anlagenpraktikum für die Lehre in der Automatisierungstechnik und Informationstechnik, 2019. URL `ais.mw.tum.de/en/teaching/anlagenpraktikum`. Accessed: 2019-05-20.

[60] R. K. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj. The class of generalized hampel filters. In *2015 23rd European Signal Processing Conference (EUSIPCO)*, pages 2501–2505. IEEE, aug 2015.

[61] V. Karagiannis, P. Chatzimisios, F. Vazquez-gallego, and J. Alonso-zarate. A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*, pages 1–10, 2015.

[62] M. B. Yassein, M. Q. Shatnawi, and D. Al-zoubi. Application layer protocols for the internet of things: A survey. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–4, Sep. 2016.

[63] P. Dobbelaere and K. S. Esmaili. Kafka versus RabbitMQ. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS '17*, pages 227–238. ACM Press, 2017.

[64] J. Jasperneite, A. Neumann, and F. Pethig. OPC UA versus MTConnect. *Computer & Automation*, pages 16–21, 2015.

[65] Object Management Group. Data Distribution Service (v1.4), April 2015. URL `omg.org/spec/dds/1.4/`.

[66] A. Hakiri, P. Berthou, and T. Gayraud. Addressing the challenge of distributed interactive simulation with data distribution service, Aug. 2010.

[67] Object Management Group. The Real-time Publish-Subscribe Wire Protocol (RTPS) DDS Interoperability Wire Protocol Specification (v2.2), December 2013. URL omg.org/spec/ddsi-rtps/2.2.

[68] Object Management Group. DDS Security (v1.1), July 2018. URL omg.org/spec/dds-security/1.1.

[69] DDS Foundation. Where Can I Get DDS?, 2019. URL dds-foundation.org/where-can-i-get-dds/. Accessed: 2019-02-15.

[70] J. Kreps, N. Narkhede, and J. Rao. Kafka: a Distributed Messaging System for Log Processing, 2011.

[71] Apache Kafka. Kafka 2.2 Documentation, 2019. URL kafka.apache.org/documentation/. Accessed: 2019-03-10.

[72] J. Rao. Clients - Apache Kafka, Jun 30 2018. URL cwiki.apache.org/confluence/display/kafka/Clients.

[73] ISO/IEC 19464:2014, Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification, 2014.

[74] S. Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89, Nov 2006.

[75] AMQP Working Group. Products and Success Stories, 2019. URL amqp.org/about/examples. Accessed: 2019-02-15.

[76] AMQP Working Group. AMQP v1.0, Oct 30 2011. URL amqp.org/specification/1.0/amqp-org-download.

[77] IEC 62541:2016, OPC Unified Architecture Specification, 2016.

[78] OPC Foundation. OPC Unified Architecture Specification Part 1: Overview and Concepts (Release 1.04), November 22 2017.

[79] OPC Foundation. OPC Unified Architecture Specification Part 14: PubSub (Release 1.04), February 6 2018.

[80] OPC Foundation. OPC Unified Architecture Specification Part 2: Security Model (Release 1.04), August 3 2018.

[81] open62541. List of Open Source OPC UA Implementations, 2019. URL github.com/open62541/open62541/wiki/List-of-Open-Source-OPC-UA-Implementations. Accessed: 2019-04-26.

[82] open62541: An open source implementation of OPC UA, 2019. URL open62541.org. Accessed: 2019-04-26.

[83] ISO/IEC 20922:2016, Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1, 2016.

[84] Roger Light. MQTT man page, 2019. URL mosquitto.org/man/mqtt-7.html. Accessed: 2019-04-26.

[85] *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta, 29 October 2014. OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[86] Software for MQTT, 2019. URL github.com/mqtt/mqtt.github.io/wiki/software. Accessed: 2019-04-26.

[87] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[88] P. Adamczyk, P. H Smith, R. Johnson, and H. S. Munawar. Rest and web services: In theory and in practice. *REST: From Research to Practice*, 13, June 2011.

[89] The Constrained Application Protocol (CoAP), RFC 7252, June 2014. URL rfc-editor.org/info/rfc7252.

[90] Datagram Transport Layer Security Version 1.2, RFC 6347, January 2012. URL rfc-editor.org/info/rfc6347.

[91] CoAP Implementations, 2019. URL coap.technology/impls.html. Accessed: 2019-04-26.

[92] MTConnect Institute. MTConnect Standard version 1.4.0, March 31 2018. URL mtconnect.org/standard20181.

[93] J. Michaloski, B. Lee, F. Proctor, S. Venkatesh, and S. Ly. Quantifying the Performance of MTConnect in a Distributed Manufacturing Environment. In *Volume 2: 29th Computers and Information in Engineering Conference, Parts A and B*, pages 533–539. ASME, January 2009.

[94] MTConnect Institute Github Repository, 2019. URL github.com/mtconnect. Accessed: 2019-04-26.

[95] Microsoft Docs: About .NET Core, 2019. URL docs.microsoft.com/en-us/dotnet/core/about. Accessed: 2019-04-22.

[96] RabbitMQ is the most widely deployed open source message broker, 2019. URL rabbitmq.com. Accessed: 2019-04-28.

[97] Apache Kafka - A distributed streaming platform, 2019. URL kafka.apache.org/. Accessed: 2019-04-28.

[98] .NET/C# RabbitMQ client library, 2019. URL rabbitmq.com/dotnet.html. Accessed: 2019-04-26.

[99] Confluent's Apache Kafka .NET client, 2019. URL github.com/confluentinc/confluent-kafka-dotnet/. Accessed: 2019-04-26.

[100] Json.NET: Popular high-performance JSON framework for .NET, 2019. URL newtonsoft.com/json. Accessed: 2019-04-26.

[101] EasyPort USB – Interface zum Messen, Steuern, Regeln. Verbindet die Simulation mit der realen Welt, 2019. URL festo-didactic.com/de-de/lernsysteme/trainingspakete/automatisierungstechnik-sps/easyport-usb-interface-zum-messen,steuern,regeln/. Accessed: 2019-04-28.

[102] TF6100 — TwinCAT OPC UA Server, 2019. URL beckhoff.de/default.asp?twincat/tf6100.htm. Accessed: 2019-04-28.

[103] OPCFoundation/UA-.NETStandard, 2019. URL github.com/OPCFoundation/UA-.NETStandard. Accessed: 2019-04-28.

[104] OxyPlot: cross-platform plotting library for .NET., 2019. URL oxyplot.org. Accessed: 2019-04-28.

[105] Code metrics values, 2018. URL docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values. Accessed: 2018-11-19.

[106] ProjectCodeMeter: Timely Source Cost Estimation, 2018. URL projectcodemeter.com/. Accessed: 2018-11-20.