NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

DAVID RODRIGUES FERREIRA
Master in Electrical and Computer Engineering

MARKETPLACE-DRIVEN FRAMEWORK

FOR THE DYNAMIC DEPLOYMENT AND INTEGRATION OF DISTRIBUTED, MODULAR INDUSTRIAL CYBER-PHYSICAL SYSTEMS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2024

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

MARKETPLACE-DRIVEN FRAMEWORK

FOR THE DYNAMIC DEPLOYMENT AND INTEGRATION OF DISTRIBUTED, MODULAR INDUSTRIAL CYBER-PHYSICAL SYSTEMS

DAVID RODRIGUES FERREIRA

Master in Electrical and Computer Engineering

Adviser: André Dionísio B. da Silva Rocha
Assistant Professor, NOVA University Lisbon

Examination Committee

Chair: Name of the committee chairperson
Full Professor, FCT-NOVA

Rapporteur: Name of a rapporteur
Associate Professor, Another University

Members: Another member of the committee
Full Professor, Another University

Yet another member of the committee
Assistant Professor, Another University

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2024

Marketplace-Driven Framework
for the Dynamic Deployment and Integration of Distributed, Modular Industrial Cyber-Physical Systems

---
This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.10) [1].

To my friends and family.

Acknowledgements

Acknowledgments are personal text and should be a free expression of the author.

However, without any intention of conditioning the form or content of this text, I would like to add that it usually starts with academic thanks (instructors, etc.); then institutional thanks (Research Center, Department, Faculty, University, FCT / MEC scholarships, etc.) and, finally, the personal ones (friends, family, etc.).

But I insist that there are no fixed rules for this text, and it must, above all, express what the author feels.

"The most important step a man can take.
It's not the first one, is it?
It's the next one.
Always the next step (...).
"

— Brandon Sanderson, Oathbringer

Abstract

The concept of Industry 4.0 revolutionized the manufacturing sector. It called for the use of Cyber-Physical Production Systems and the digitalization of many manufacturing processes. Even though the implementation of Industrial Multi-agent Systems as Cyber-Physical Production Systems comes with a lot of advantages, it has not seen much use by the industry.

As a consequence of this, Multi-agent Systems have not grown out of their infancy and haven't evolved like other technologies through their practical use. This may be because there is still some scepticism surrounding the concept of Multi-agent Systems, and whether or not they can perform to the same efficacy when compared to already existing systems.

In this work, an architecture which helps solve this problem is proposed. More precisely, this platform would allow for the flexible integration of agents with their respective hardware by proposing a method that selects one or more generic libraries based on the kind of hardware that is being integrated into the industrial Multi-agent System. Through this method, more devices are able to be integrated in less time and with less work, contributing for the flexibility and scalability that is characteristic of Multi-agent Systems.

This platform would facilitate the adoption of industrial Multi-agent Systems, because not only would it make integrating them easier, it would also mean that anyone could write these libraries, making it easily adaptable to already existing systems, which would reduce costs in the adoption of Multi-agent Systems for industrial applications.

Keywords: Industry 4.0, Cyber-Physical Production System, Multi-agent System, Manufacturing Systems, Hardware Integration

Resumo

O conceito de Indústria 4.0 revolucionou o setor de manufatura. Uma das características deste conceito é o uso de Sistemas de Produção Ciberfísicos e a digitalização de processos de manufatura. Apesar da implementação de Sistemas Industriais Multi-agente como Sistemas de Produção Ciberfísicos trazer várias vantagens, não tem sido muito adotada por parte da indústria.

Como consequência, Sistemas Multi-agente não passaram da sua fase inicial e não evoluíram através do seu uso prático como outras tecnologias. Isto pode ser atribuído ao ceticismo do qual o conceito de Sistemas Multi-agente ainda sofre, e à incerteza sobre se estes conseguem operar com a mesma eficácia quando comparado com sistemas já existentes.

Neste trabalho, é proposta uma arquitetura que poderá ajudar a atenuar este problema. Mais precisamente, esta plataforma permite a integração flexível de agentes com o seu respetivo hardware, propondo um método que seleciona uma ou mais bibliotecas genéricas baseadas no tipo de hardware que está a ser integrado no Sistema Multi-agente industrial. Com isto, mais dispositivos são capazes de ser integrados em menos tempo e com menos trabalho, contribuindo para a flexibilidade e escalabilidade que é característico dos Sistemas Multi-agente.

Esta plataforma facilitaria a adoção de Sistemas Multi-agente industriais, porque não só faria a sua integração mais simples, mas também permitia que qualquer desenvolvedor pudesse criar uma destas bibliotecas, tornando o sistema adaptável a sistemas já existentes, reduzindo custos na adoção de Sistemas Multi-agente para aplicações industriais.

Palavras-chave: Indústria 4.0, Sistemas de Produção Ciberfísicos, Sistemas Multi-agente, Sistemas de Manufatura, Integração de Hardware

Contents

List of Figures

List of Tables

Acronyms

AGV Automated Guided Vehicle *(p. 4)*
API Application Programming Interface *(p. 8)*

CA Component Agent *(p. 17)*
COM/DCOM Microsoft Windows Distributed Component Object Model *(p. 10)*
CPPS Cyber-Physical Production System *(pp. 1, 2, 4–7, 11–15, 17, 31)*
CPS Cyber-Physical System *(pp. 1, 4, 5, 31)*

DA Deployment Agent *(pp. 15, 16, 19, 20, 22, 24, 29)*
DF Directory Facilitator *(pp. 20, 21, 25)*
DWPS Device Profile Web Services *(pp. 15, 17)*

FIPA Foundation for Intelligent Physical Agents *(pp. 2, 10, 12, 16, 20, 21, 24)*

HTTP Hypertext Transfer Protocol *(pp. 26–30)*

IntA InterfaceAgent *(p. 13)*

JADE Java Agent DEvelopment Framework *(pp. 10, 12, 16, 17, 24)*

MAS Multi-Agent System *(pp. ix, 2, 4, 6–8, 10–12, 14–16, 19, 22, 24, 26, 29, 31)*
MQTT Message Queuing Telemetry Transport *(pp. 26, 27, 29)*

OEE Overall Equipment Effectiveness *(p. 5)*
OPC Classic Open Platform Communications Classic *(p. 10)*
OPC UA Open Platform Communications Unified Architecture *(pp. 7, 10, 11, 13, 14, 26–28, 30)*

PA Product Agent *(pp. ix, 12, 15, 16, 19–21, 24, 25)*

PLC Programmable Logic Controller *(pp. ix, 7, 10, 11, 14, 15, 17)*
PM Product Manager *(pp. 19, 24, 29)*
PMA Production Management Agent *(pp. 16, 17)*
PSA Prime System Agent *(p. 16)*

RA Resource Agent *(pp. ix, 12, 13, 15, 16, 19–21, 24–26, 29)*

SMA Skill Management Agent *(p. 17)*

TA Transport Agent *(pp. ix, 15, 16, 19–21, 24–26, 29)*
TCP-IP Transfer Control Protocol - Internet Protocol *(p. 11)*

UDP User Datagram Protocol *(p. 11)*

WS4D-JMEDS Web Services 4 Devices - Java Multi Edition DPWS Stack *(p. 15)*

XML eXtensible Markup Language *(pp. 12, 27)*

1

Introduction

1.1 Motivation

In 2011 the term Industrie 4.0 was introduced for the first time during a German conference. This term was later translated into Industry 4.0 and it quickly became synonymous with the fourth industrial revolution. With it, came the appearance of Cyber-Physical Production Systems (CPPSs). A CPPS is essentially, as the name implies, a CPS capable of operating in a production line. A Cyber-Physical System is a physical system that has a digital representation. Both the physical and digital counterparts exchange data to maintain coherence and work in tandem to achieve a goal. These CPPSs brought about the digitalization of the manufacturing sector, and despite there being different models, most of them follow these design principles [2]:

- Services offered through an online platform
- Decentralized, enabling autonomous decision-making
- Virtualized, to allow interoperability
- Modular, making them flexible to changes in the system
- Real-time capabilities
- Ability to optimize processes
- Communication is done through secure channels
- Cloud service for data storage and management

This new way of operating a production chain revolutionized the industry, because it brought about the changes needed to make manufacturing processes more flexible, adaptable and re-configurable, bringing with it a solution for the ever increasing complexity needed to address the rapidly changing costumer demands. In recent years, multiple companies have made the transition to this model. They have made diversified changes, from the way they analyze and process data to the way they manufacture and distribute products, with very positive results as we can see in [3].

In light of this, Cyber-Physical Production Systems became a popular research subject. These CPPSs would allow for more efficient, robust and flexible systems, equipped with Big Data analyzing algorithms, Cloud storage to easily access data, service oriented manufacturing and interoperability.

This research was mainly done on the topic of Multi-Agent Systems [4] [5]. These MASs are a coalition of agents, all part of one single system but completely independent of each other. They are intelligent, social and capable of performing tasks on their own, however due to their social capabilities, they can also perform tasks cooperatively, making them powerful tools in goal-oriented networks. Because this system is, by nature, decentralized, these agents can leave and join a coalition of agents as needed, to complete selfish or collective objectives. Evidently, the system is also highly flexible and robust, since agents can be taken out of commission and new agents can be introduced as needed, either because the overall system specifications need to change or simply because an agent has become faulty [6].

Multi-Agent Systems (MASs) have actually been around for decades and as such, a lot of research and standardization already exists, like the Foundation for Intelligent Physical Agents (FIPA) specifications. FIPA as an organization have ceased operations, however their standard are still put to use in MASs nowadays [7]. An industrial MAS follows similar requirements as a non-industrial one, although it needs to take into consideration other factors, such as hardware integration, reliability, fault-tolerance and maintenance and management costs.

Despite all the advantages an MAS has for the manufacturing sector, it has not seen much success outside research fields. This could be due to the fact that there is still skepticism surrounding agent-based systems for industrial production [8]. Because it never left the prototyping stages, real-world applications never evolved past their infancy, therefore never gained much momentum in the industry at large [9]. Another consequence of this is the difficulty in designing a robust and scalable interface, that allows, for example, the addition and removal of agents without system reconfiguration.

1.2 Problem and Solution

In this work, a new approach to developing interfaces for an Industrial MAS is proposed. It consists of a framework that allows the creation of industrial agents through the selection of generic libraries for the interface between agent and device. These libraries are picked based on the type of communication protocols and because they are generic, they can be re-used with minimal reconfigurations. The aim of this solution is to simplify the process of creating new agents for the system, enabling them to interface with any kind of hardware.

2

State of the Art

In this chapter we'll explore how researchers have dealt with the challenges of creating a Multi-Agent System (MAS) based Cyber-Physical Production System (CPPS). We'll start by examining the concepts of CPPS and MAS in more detail and by taking a look at the most commonly used designs and tools, as well as the recommended practices for an industrial MAS. Finally, we'll do a brief analysis on some prototypes that were made to showcase the usefulness of an MAS in an industrial setting.

2.1 Cyber-Physical Production Systems

As stated before, a Cyber-Physical System (CPS) is a system composed of two main entities, the physical system that contains all the hardware and resources and the cyber system that represents the physical system in the digital world. These two systems work together, the physical system sends data taken from sensors from the environment to the digital system and the digital system processes and stores this data and instructs the physical system on what actions to perform through the use of actuators. As such, this system works in a loop, constantly exchanging data and instructions to achieve a set goal.

A Cyber-Physical Production System is, as the name implies, a Cyber-Physical System (CPS) that operates in a manufacturing setting. The physical system is composed of hardware like robot arms, AGVs, conveyor belts and specialized machinery for manufacturing. The cyber system might be as simple as a computer program or as complex as a full-on three dimensional model of the physical system. Vogel-Heuser and Hess [2] proposed that a CPPS should:

- Be service oriented, meaning that it should offer its services through the internet
- Be intelligent, with the ability to make decisions on its own
- Be interoperable, by having the capacity to aggregate and represent human-readable information and by providing a virtualization of the physical system
- Be able to flexibly adapt to changes in requirements and scale

- Have Big Data algorithms capable of processing data in real time
- Be capable of optimizing processes to increase Overall Equipment Effectiveness (OEE)
- Be able to integrate data across multiple disciplines and stages of the products life cycle
- Support secure communications to allow partnerships across companies
- Be capable of storing and accessing data in the Cloud

In summary, these CPPSs abstract the hardware resources in the physical environment into their digital counterparts, giving them a measure of autonomy and intelligence, which in turn allows for the fulfillment of most requirements. The remaining requirements are fulfilled by integrating the CPPS with other existing tools such as Big Data processing platforms, Cloud storage services and data-sharing services.

These systems have a virtual representation of the physical system, that updates in real-time according to the information received from the physical world. This virtualization allows for a more efficient and realistic analysis of the real world, enabling more complex behaviors, better error correction and the optimization of industrial processes.

Although advantageous in comparison to existing industrial systems, CPPSs are still fairly recent and therefore creating one of these systems from the ground up still has its challenges. Leitão, Colombo, and Karnouskos [10] have compiled key challenges, as well as their difficulty and priority.

For these reasons, companies have not made a huge push to create CPPSs from the ground up, instead opting to adapt their already existing systems to integrate CPPS elements and characteristics. This is essentially an integration of a CPS in an already existing production line. many of these adaptations depend on creating interfaces capable of interacting with many types of hardware, to allow an easier and more streamlined integration of new entities into the system.

Some examples of CPPS retrofitting can be seen in Cardin [11] and Arjoni et al. [12]. Cardin [11] have done an analysis on the whole process of retrofitting a robotic arm. Before the retrofit, this arm was controlled through a remote controller or a serial interface. After the retrofit, the arm was able to receive commands through a new interface, which was able to communicate through standard Wi-Fi and Ethernet protocols, sending sensor data and receiving commands to and from the network. This network also integrated a cloud service to allow for the storage and access of information. Performance-wise, the arm showed better energy efficiency and less operation latency after the retrofit and all values were below the thresholds for the standards in an Industry 4.0 CPPS.

Arjoni et al. [12] created a small manufacturing plant using old devices. They consisted of two robotic arm, a CNC machine and a welder arm. All three devices communicate through different protocols, but by using off-the-shelves hardware like an Arduino UNO and a Raspberry Pi 3, Arjoni et al. were able to give these devices the ability to be integrated into a more complex system. Although they recognize the system could still be optimized, they considered the result promising in the retrofit of old devices into a CPPS.

2.2 Multi-Agent Systems

To understand what should be the best practices in an MAS based CPPS, we first need to understand its base characteristics. An MAS is, in essence a system composed by many entities called agents that have their own capabilities. These agents communicate and collaborate together by exchanging data among themselves and acting on that data to achieve a common goal. They are capable of adapting their behavior and of autonomous decision-making to determine the best course of action. This system is by nature decentralized and has no hierarchy, making it highly flexible and modular. At any point an agent can leave or join the system, without significant changes in architecture [6].

Industrial agents inherit all the qualities of software agents, like the intelligence, autonomy and cooperation abilities, but in addition are also designed to operate in industrial settings, and need to satisfy certain industrial requirements such as reliability, scalability, resilience, manageability, maintainability and most important of all, hardware integration [13]. An example of a base architecture for an MAS can be seen in Figure 2.1.

These requirements are generally tough to fulfill, especially so because despite the theoretical potential MAS have shown in supporting them, there aren't a lot of agent-based production systems outside the prototyping phase. This has stopped the growth of Industrial MAS due to the lack of practical knowledge in this field [14]. Another problem seen in Industrial MAS is the lack of models that can represent these systems. One of the key elements of an MAS are the changes made in structure and logic as the system operates. Thanks to the decentralized nature of the system, it is possible to add, remove or reconfigure modules freely to better adjust it to the systems needs [14].

Now that we have an idea of the characteristics and requirements for Industrial MAS, we can take a look at the most commonly used architectures, followed by what is recommended by the IEEE Standard.

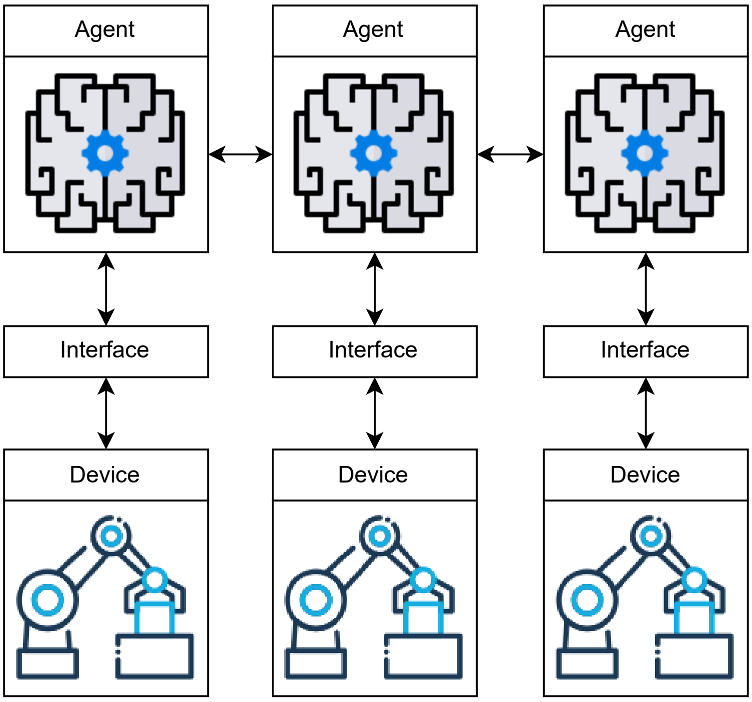


Figure 2.1: Industrial Multi-Agent System

2.2.1 Best Practices and Common Architectures

Leitão et al. [13] analyzed the IEEE 2660.1 Standard for the recommended practices in integrating software agents and low-level automation functions. They described the use of an MAS as a CPPS, where control is decentralized, emerging from the interactions of agents that are part of the system. And as we've discussed before, one of the biggest problems is creating an interface between the agent and the device associated with it. Because of this, the IEEE 2660.1 Standard was created, defining the best practices in designing an appropriate interface.

As an example, the authors mention three main types of interfaces. An interface for a smart sensor, to acquire measurements. An interface for a PLC, to control simple devices like conveyor systems. And finally, an interface for a robot controller to control more complex functions in the CPPS. These three interfaces present different challenges on a development level, because each requires consideration on which architecture to follow, with different consequences to the evolution of the manufacturing plant over time.

The authors then created multiple scenarios, one of them being factory automation. They then proposed that the most valuable criterion was the response time of the system. As a secondary criterion scalability was chosen, but with a lesser importance. From this scenario the authors then concluded that a Tightly coupled Hybrid OPC UA interface was preferable according to the IEEE 2660.1 standard. This means that the interface should have a client-server approach and be running remotely, a Tightly coupled Hybrid approach. However the authors also mention that this setup has a relatively low score, meaning that many of the other proposed practices are still viable, with testing needed to be done in order to pick the best one based on each specific scenario.

In [14], it is proposed that one of the key requirements in the design of interfaces for MAS is interoperability. This comes with other challenges associated, like re-usability and scalability. In an MAS, the authors identified two main types of interfaces, the interface between agents, which normally is provided through the framework of the agent-based system, and the interface between agent and device.

Leitão et al. [15] analyzed a study performed under the IEEE P2660.1 Working Group [16] and concluded that most approaches followed a two-layer convention. The upper layer contained the agents part of the MAS and the lower layer the hardware associated with the physical production system. These two layers can interact in two ways [15]:

- Tight coupling, where the two layers communicate either through shared memory or through a direct network connection. This communication is synchronous and more direct.
- Loose coupling, where the two layers communicate through a queue or a pub/sub channel. This communication is asynchronous and less direct.

These layers can also be hosted in different setups [15]:

- Hybrid setup, where the two layers run in different devices.
- On-device setup, where the two layers run in the same device.

This means that there can be four different interfaces, Tightly coupled Hybrid, Loosely coupled Hybrid, Tightly coupled On-device and Loosely coupled On-device.

A Tightly coupled Hybrid interface (Fig. 2.2) is characterized by having the upper layer where the agent operates running remotely and accessing the lower layer through an API. This API is responsible for translating the instructions given by the agent into commands the hardware can interpret. It is also responsible for the opposite, translating the hardware output, such as error codes or function results into data the agent can use. This approach is limited by the channel through which both layers communicate since both agent and device operate on two different computing platforms. This channel is affected by the amount of traffic in the network, more connections implies a lesser quality of service, namely in response time [15].

A Loosely coupled Hybrid interface (Fig. 2.3) also sees both agent and device running on different computing entities. The difference is that instead of each agent having a direct connection to the corresponding device, they communicate through a message broker.

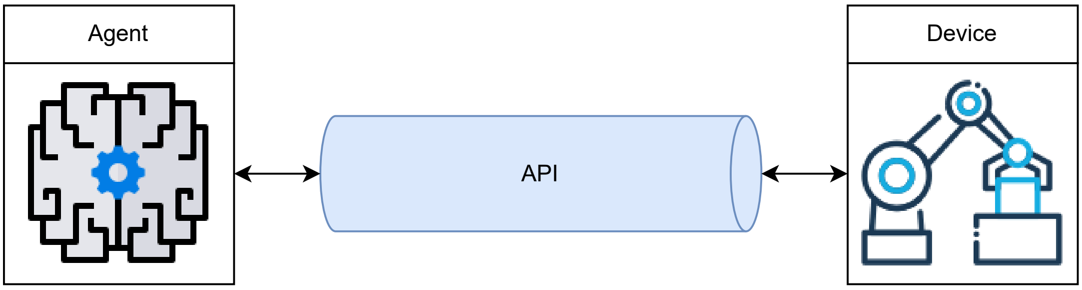


Figure 2.2: Tightly coupled Hybrid interface. Source: Adapted from [15]

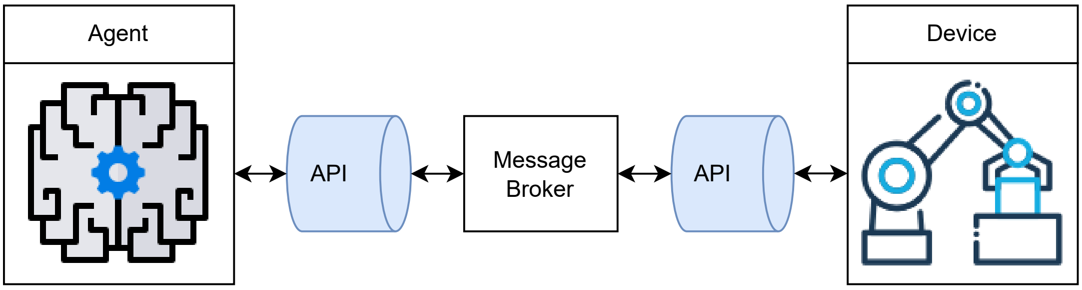


Figure 2.3: Loosely coupled Hybrid interface. Source: Adapted from [15]

Since the system still runs on two different computers it still suffers from the quality of the connection between layers, making this somewhat inappropriate for systems highly dependent on real time action. However, this approach sees better results in complex systems, where the agent layer needs to publish information to a large amount of devices at once. It also sees good results when it comes to scaling the system, since both layers are very independent of each other [15].

A Tightly coupled On-device interface (Fig. 2.4) on the other hand follows an architecture where both devices share the same physical platform and can be done in two different ways. The first one, and far less common, has both agent code and device code compiled into a single binary running in the same computing element. This solution provides far better results in very demanding real time applications, however it also removes some flexibility from the system and is far more complicated to design due to the lack of development tools. The second option has the computational resources shared through a software library, where communication is done through software functions while abstracting some elements. This option still holds good results in real time control, but not as good as the first option [15].

Finally, a Loosely coupled On-device interface (Fig. 2.5) is characterized by having the agent embedded in the device and communication is done through a broker. Both layers share a physical unit but do not share computational resources. The utilization of a broker between the two layers offers some flexibility, since the agent and hardware are less dependent of each other. This comes with the caveat that the real time response of

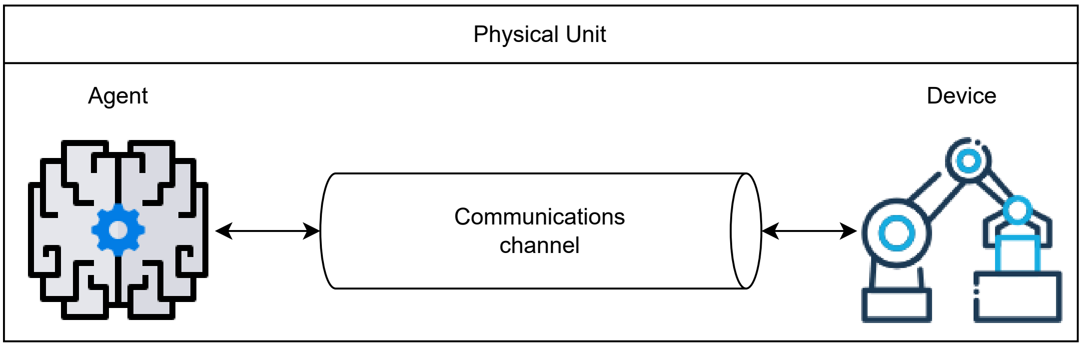


Figure 2.4: Tightly coupled On-device interface. Source: Adapted from [15]

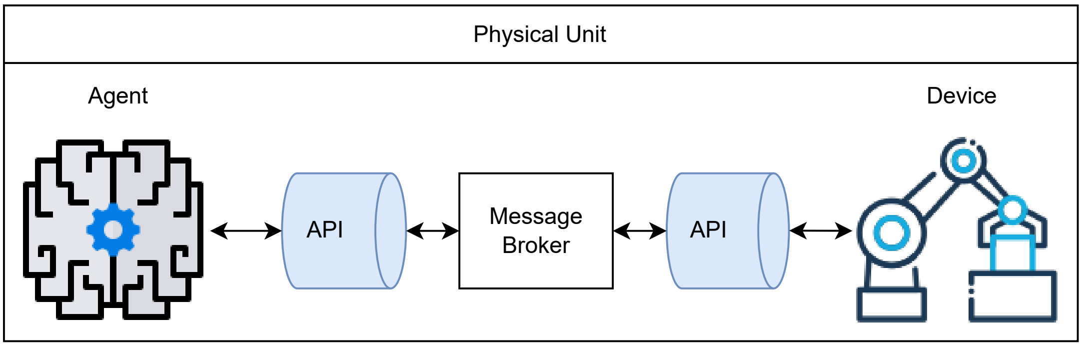


Figure 2.5: Loosely coupled On-device interface. Source: Adapted from [15]

the whole unit is dependent on the performance of the broker [15].

The most common programming language to codify agents is Java, most likely due to the existence of JADE followed by C++ [15]. JADE helps developers in the implementation of MASs with the FIPA specifications. It also allows deployment for different machines, due to Java supporting multiple devices [17]. For the device part, preexisting hardware is used in the majority of cases because it can be integrated into an MAS by using protocols such as OPC UA [15], which is a platform independent data exchange standard. It allows for both server-client and publish/subscribe communications as we'll see in the following section [18].

2.2.2 OPC UA

Another way to integrate hardware into an MAS it through the use of already made libraries. The Open Platform Communications Unified Architecture (OPC UA) was created based on Open Platform Communications Classic (OPC Classic), which in turn is based on the COM/DCOM for the exchange of data. OPC UA is an evolution of OPC Classic but with extra functionalities, like added security, extensibility and platform independence. This last feature allowed it to run on many more platforms such as cloud-based servers, micro-controllers and PLCs [18]. As mentioned in Section 2.2.1, it is a recommended way to implement an interface for an MAS. OPC UA can be used with as a tightly coupled or loosely coupled interface, because it supports both direct client-server connections and pub/sub message transmissions, making it a strong option to integrate hardware into an MAS.

Because OPC UA already implements communication protocols and communication infrastructure, as well as an information format, all participants in the network know how to communicate. This means that the MAS can be running on any kind of platform or be programmed in any kind of language, increasing system flexibility. Many modern PLCs already provide an embedded OPC UA server facilitating their integration into a new system [19].

2.2.3 Modbus

Another way to integrate hardware into an MAS is by using Modbus. Modbus is an industrial communications system that has been around since 1979. It follows a master/slave architecture, where the master is usually an application capable of acquiring data and the slave a PLC. This master application can be substituted for an agent from an MAS, integrating it with the slave hardware. Modbus supports standard TCP-IP and UDP. It is simple, has high levels of compatibility and is decentralized due to the use of master/slave communications, making it flexible. Modbus could be integrated with any programming language of choice, by using software libraries to interpret and send commands to the Modbus layer [20].

2.3 Practical Uses

Adoption of industrial oriented MAS has been slow. According to Karnouskos and Leitão [9], the technology was still in its infancy almost two decades ago, with an incremental progress at best being made since then. In [14], Karnouskos et al. claim that agent-based applications in the industry is still limited. This is because despite the potential shown, these systems have not been implemented in real-world applications, where they would have the chance to evolve and leave the prototyping phase as new research is being done to make them more suitable for these applications.

There are, although, many research prototypes of MAS used for an industrial applications, and we'll take a look at some of them in this section.

2.3.1 Bottling Plant

For the design of the bottling plant in [8], a research paper [21] was first done by Marschall, Ochsenkuehn, and Voigt, with the collaboration of multiple partners from different backgrounds. This was done to define the base requirements for the CPPS as well as the associated MAS. These requirements are:

- Easy Scalability and Functional Expansion
- Manufacturer-neutral Resource Representation
- Robust Production Control by the Product
- Lot Size One without Identification

Summarizing, the system needed to be easily scalable and it must be able to expand its functions to handle changes in production, it must be able to represent the resources in the CPPS as a generic and manufacturer independent representation, products must be able to handle their own production steps, handling errors and reacting robustly to those errors and finally be able to produce lot sizes of one without needing an identification system for each individual product.

In consequence of these requirements, Marschall, Ochsenkuehn, and Voigt designed a base solution consisting of two generic agents, a Product Agent (PA) that represents individual products being manufactured, in this case the beverage bottles, and a Resource Agent (RA) that represents the resources used to manufacture the beverage bottles.

The MAS was made using JADE, and as such all agents in this system are FIPA compliant and communicate through FIPA Requests and the FIPA Contract Net.

Each agent inherits from a generic class called Agent. The RA and PA then inherit from this class, with added functionalities and variable fields. The PA is an agent with the capabilities of performing autonomous and goal-driven behaviors. The RA is an agent capable of controlling the physical resources by using sensor data and messages with hardware instructions.

A PA differs from the generic Agent by implementing a configuration file ProductConfig. This file is in the XML format and has information on the product represented by it. It also has a PlanningModule, which is essentially a StateMachine with all the products processing instructions.

Like the PA, the RA also mirrors the Agent class. It must be generic in order to be implemented on multiple resources, so for each ResourceType there exists a ResourceConfig file which is loaded onto the RA in order to implement its interface. To give an example, if the resource has the ResourceType Machine, the ResourceConfig file MachineConfig must be loaded.

The authors identified five different ResourceTypes by classifying many resources from renowned machine manufacturers. RAs not only include robots, transport systems and stations but also databases, as a data management method. The authors recognize that not all types of possible resources for a CPPS have been considered. This means that for every new resource type that needs to be added to the system, a new ResourceConfig must be created in order to integrate the new RA into the system. In addition, a new resource that communicates differently from the ones already in the system would need a new interface, and thus a new ResourceConfig would also need to be created, even if the new resource performs similar functions in the manufacturing plant.

In [8], Marschall et al. created the service oriented bottling plant using the industrial agent system designed in [21], with positive results. To allow for extensibility of the system by additional resources a new agent class was created called InterfaceAgent (IntA). The ResourceConfig files for each type of RA now have a new field which contains an InterfaceAgentConfig. This new IntA is instantiated by the RAs following the configurations of their respective InterfaceAgentConfig and it extends either a DBLinkAgent, used to connect to a database, and a PLCLinkAgent, used to connect to a OPC UA client. All the information, like server address, port and authentication, is stored in the InterfaceAgentConfig file. By using OPC UA, the system can now interface with the hardware in the manufacturing plant, through a Loosely coupled Hybrid interface. The authors recognize that according to the IEEE P2660.1 Working Group the scalability is considered weak, although they claim the flexibility is worth this loss in scalability.

To implement a new IntA, the following must be provided:

- The communications protocol and the address, ports and authentication
- The way the interactions between the agent and resource should proceed
- The rules for the interpretation of internal resource states

Communication between the PLCLinkAgent and the machine is done through the use of simple commands. They specify the command to perform and the program type to use and must also include a unique name, the NodeID which identifies the type of the given input on the OPC UA server, the data type, all possible values, the read/write direction and the description. The program type is chosen based on the size of the bottle and the command has to be one of the five commands already defined:

- NoCommand, in case no action is needed
- ProductInPosition, in case the product is in position and ready
- ExecuteProcess, in case the process should start
- ProductRemoved, in case the product needs to be removed from the machine/station
- PrepareMachine, in case the machine needs to be prepared for a process to start

All commands have a number code associated to them to facilitate communications and at any time the PLCLinkAgent can retrieve the state of a machine by observing a NodeID with the machine state code. In Figure 2.6 we can see the state machine representing the interactions between the PLCLinkAgent and a machine. To handle manufacturer specific machine state codes, the authors mapped said codes onto generic MAS states, with the possibility of having multiple codes mapped to a single state. This was done to make the system capable of handling machines from different manufacturers without sacrificing flexibility in the MAS.

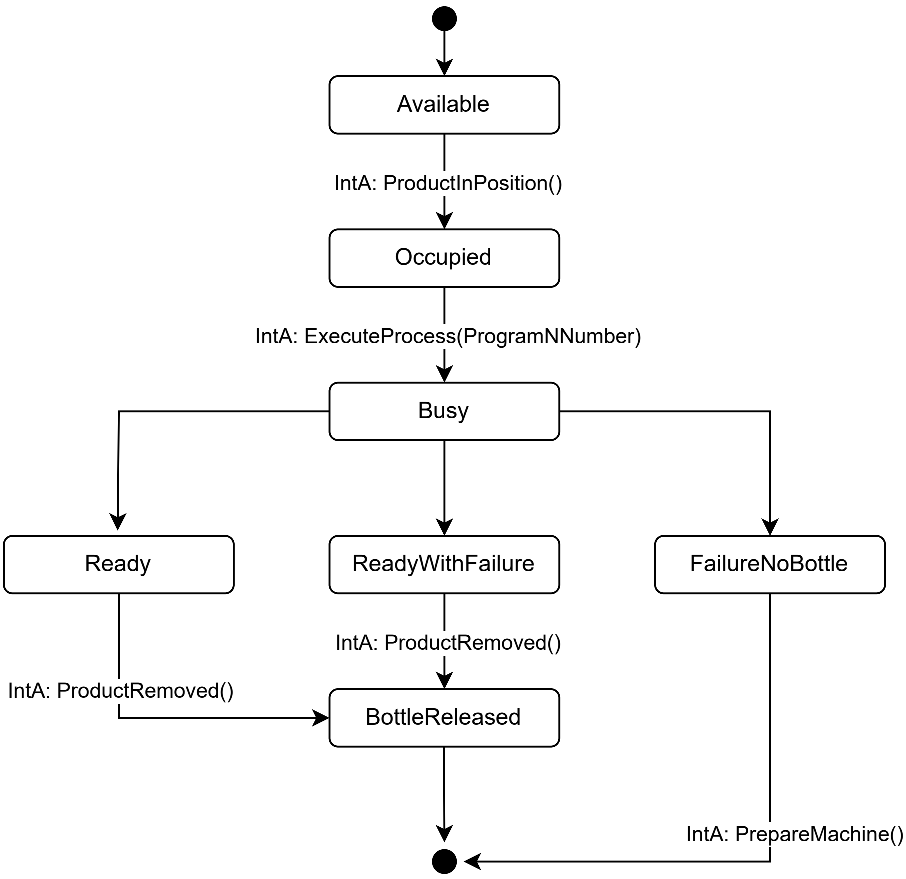


Figure 2.6: Interactions between PLCLinkAgent and PLC. Source: Adapted from [8]

The use of OPC UA makes this system very flexible. New hardware that needs to be implemented in the system can be configured to use these commands through the use of a PLC. The use of generic states make it scalable and configuration files also help fulfilling this necessity. However, it is still dependent on these files, with individual configurations needed for different manufacturers and machine types. The system also needs to be adapted to recognize the machine states from the new implemented hardware.

Nevertheless this system performed as intended, producing customized beverage bottles according the customers specifications. It is a good example of an MAS based CPPS, flexible, scalable, robust, with decentralized control and autonomous intelligent agents.

2.3.2 Agent-based Plug and Produce CPPS

Rocha et al. [22] have created a Plug and Produce CPPS. It is capable of integrating new agents on the fly, as the system operates. First, the authors divided the system into three layers. The upper layer is where the MAS operates, called the fog layer. The middle layer is where the interface between the MAS and the hardware is, called the edge layer. And finally, the lower layer is where the hardware components of the CPPS are, called the physical layer.

The agents were also categorized into PA and RA, performing similar functions to the ones in Section 2.3.1 and in addition the authors also considered Transport Agents (TAs) which abstract all resources whose function is to move a product from point A to point B. They also considered a Deployment Agent (DA) tasked with managing the existence of all other agents. This DA should create a new agent or remove an existing one whenever a physical resource is plugged into or unplugged from the system.

To accomplish this the authors considered the grouping of a physical resource with its agent a Module. Modules are the components of the whole production system, and they can be plugged and unplugged at any time. The system need to be able to operate to the best of its abilities at all times. In Figure 2.7 we can see an example of one of these Modules.

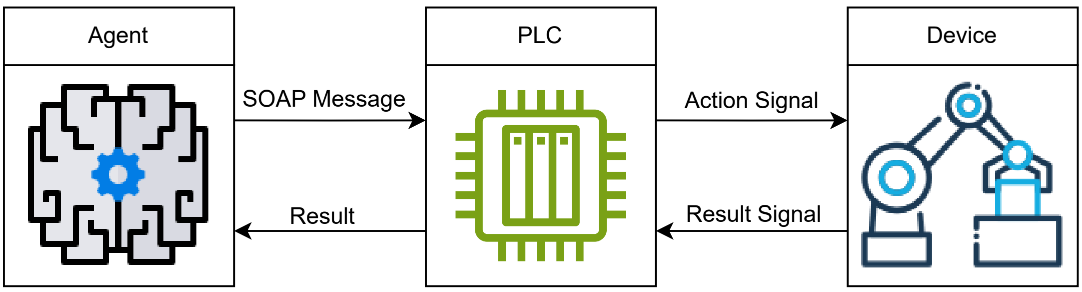


Figure 2.7: Plug and Produce RA. Source: Adapted from [22]

For the interface between hardware and agent, PLCs were chosen. These PLC are able to communicate using Device Profile Web Services (DWPS). For the DA, a Java class was implemented using the Web Services 4 Devices - Java Multi Edition DPWS Stack (WS4D-JMEDS) framework, which searched for the devices in the network and obtained information about them. It was able to detect all devices connected to the network and add and remove them as needed.

Each resource is able to perform certain procedures on the products, represented in the MAS as skill which RAs perform on to PA. This is how the authors simulate the system virtually, whenever a product needs a certain procedure to be performed on itself, the corresponding PA asks an RA for that skill. This is relevant because the system may or may not have a resource able to perform a specific skill. In the case of a PA that needs an non-existing skill to be performed, it waits until an RA capable of performing the needed skill to be plugged into the system.

The MAS was developed in the JADE framework, all agents are therefore FIPA compliant and communicate through FIPA Requests and FIPA Contract Net. More precisely, the FIPA Requests are used for PA-TA communications and the FIPA Contract Net used for PA-RA communications. Whenever a new PA enters the system, it asks through the FIPA Contract Net which RAs are can perform the needed skill. After getting responses from all the RAs, the PA picks the best one and receives its location. It then sends a FIPA Request to the TA to transport it to this location. After arriving, it then requests that the skill be performed to the RA.

This system was used to simulate a simple conveyor belt line with brushing capabilities. The authors where able to successfully plug and unplugged Modules from the system during its operation. The DA correctly connected and disconnected hardware components from the system and deployed and kicked agents from the system accordingly. This shows that an MAS built from the ground up with these functionalities is very powerful in its scalability, and although this was only a prototype it shows a lot of promise for a dynamic system.

2.3.3 PRIME

Rocha, Barata, and Santos [23] have done a demonstration on the PRIME architecture as an agent based framework with plug and produce functionalities. PRIME is a project developed thanks to the European FP7 program, and it proposes a solution to allow plug and produce using any kind of computational devices. This means that it is no longer needed to restrict a manufacturing plant to specific controllers to provide to it some sort of reconfigurability and flexibility. It allows any kind of controller or to be integrated into the system while still using the same framework.

All agents in this framework were developed using JADE, therefore all agents are FIPA compliant. This adds to the plug and produce functionality of the system. The PRIME agent system is composed of eight different agents:

- Prime System Agent (PSA) is the highest level agent in the framework. It manages the current state of the system.
- Production Management Agent (PMA) is responsible to combine all resources and tasks in the same space to abstract certain functionalities

- Skill Management Agent (SMA) works in tandem with a PMA, combining the lower-level skills into higher-level ones according to pre-defined rules, that the associated PMA then provides to the system
- Component Agent (CA) is the agent that abstracts the hardware in the CPPS. It is able to read and write data to and from the hardware
- Local Monitoring and Data Analysis

All devices to be integrated into the system were categorized into three groups according to their characteristics:

- Fully intelligent resources, capable of running the necessary agents locally, typically a machine with the ability to run the Java environment, setup in a Tightly or Loosely Coupled On-device architecture
- Semi-intelligent resources, capable of announcing their existence to the system and of reconfiguring themselves but without the ability to run the Java environment, setup in a Tightly or Loosely Coupled hybrid architecture
- Passive resources, without any computational abilities and dependent on a controller to connect them to the main system

For the first type of device, PRIME was able to very easily connect to it. The device ran all the necessary components to enable communications and to expose itself to the network. It ran the necessary agents related to the hardware locally and the main framework was able to configure the hardware by interacting with the local agents, which in turn interfaced with the device. This is the type of architecture preferred by PRIME, since it provides all services autonomously [23].

For second category of devices an INICO PLC capable of running DWPS was needed to connect the device to the framework since they can't run the agents locally. An auxiliary computer running the DWPS software was responsible for detecting the PLC and connecting to it. The necessary agents running on this computer launched the hardware related agents whenever they detected a new device running DWPS, and these local agents were able to connect to the main PRIME network [23].

The last category of devices is the most complicated one because they have no easy way to interface with the main system. In this case a more primitive PLC was used to simulate this limitation. An auxiliary computer running the JADE framework with all necessary agents is needed to create this interface and the system communicates with the PLC on a case-by-case scenario, with each PLC possibly needing different configurations. When the relevant agent detect a new device, it launches a new agent to interface with this device. The new agent then uses a case specific library to interface the PLC using standard protocols [23]. This last option is not flexible and uses a case-by-case configuration, making it the least desirable in a system.

3

Architecture

In this section, the developed architecture is explained. It has been split into two parts, the first presents the structure of the Multi-Agent System (MAS) that was developed as a prototype to showcase the framework, and the second explains how the framework fits into the agents part of the MAS.

3.1 Multi-agent System Architecture

As we have seen in Chapter 2, MAS can have many different architectures. The main objective of this work is to create an interface between agent and hardware, so a fairly simple structure for the system was designed. It consists of three main types of agents:

- A Product Agent (PA), that represents the products that are currently being manufactured.
- A Resource Agent (RA), that represents the many devices capable of performing some kind of process on the products.
- A Transport Agent (TA), that represents all devices tasked with transporting products from one processing station to another.

All of the above agents were designed to be as generic as possible while also taking into account their specific functions in the overall system. This assures that at any point the system can be adapted to new specifications, like a new kind of product to be fabricated or a new set of work stations to be integrated.

To help with this, a Product Manager (PM) and a Deployment Agent (DA) were also created. They are both agents that have a graphical user interface and do not represent any physical entity in the system. The main functionality of the PM is to launch PAs and list them in a table of products, with ID, product type and list of skills needed to be performed. Similarly, the DA can launch but also terminate RAs and TAs at any point, as long as the system is not processing any product. Whenever an agent is launched through the DA, it receives a marketplace file with all the available libraries, a library type (Link Library) and a configuration file selected by the user. These are what the Module Engine will use to communicate to the hardware represented by the agent, as it will be explained in Section 3.2.

For interactions between agents, the Foundation for Intelligent Physical Agents (FIPA) already has specifications in place. For this work, two interaction protocols were selected. These are the FIPA Contract Net and the FIPA Requests. The FIPA standard also includes a Directory Facilitator (DF), where agents can register themselves with their own capabilities and functionalities. This allows agents to easily find other agents capable of performing specific tasks.

All Resource Agents and Transport Agents will register themselves when they're launched, since these agents are the ones capable of offering some kind of service, be it transportation or an action to be performed on a product. Product Agents are launched whenever a new product enters production, and they will look in the DF for an appropriate agent capable of performing the next step in their individual process.

PAs will then send a message through the FIPA Contract Net to all agents that are able to perform the correct task, and will wait for a response. If there are RAs that are free and capable of performing it, they will answer with a proposal informing the PA as such. Then the PA will pick an RA from the list of proposals and send an acceptance message to it. Finally, the RA informs the PA what their location is. This sequence of events can be seen in Figure 3.1. In the case no proposals are received, or they are refused for whatever reason, the PA will repeat the process until an available RA is found and an agreement is reached.

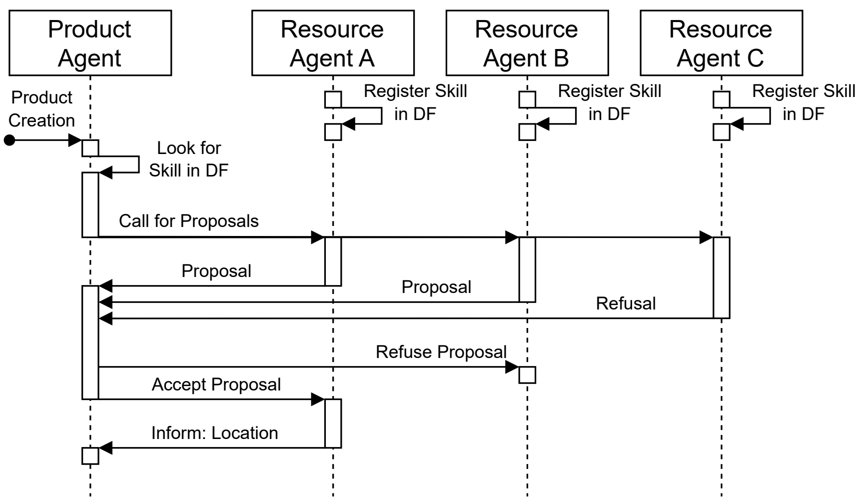


Figure 3.1: Sequence Diagram of Product Agent and Resource Agent Contract Net Communications.

After an agreement is reached between a PA and an RA, the PA then searches through the DF again, this time to look for a TA capable of taking the product to the location provided by the RA. It then sends a request to the TA. After agreeing with the request, the TA proceeds to transport the physical product to the desired location, and after completing the transportation, it will inform the PA. This is represented in Figure 3.2.

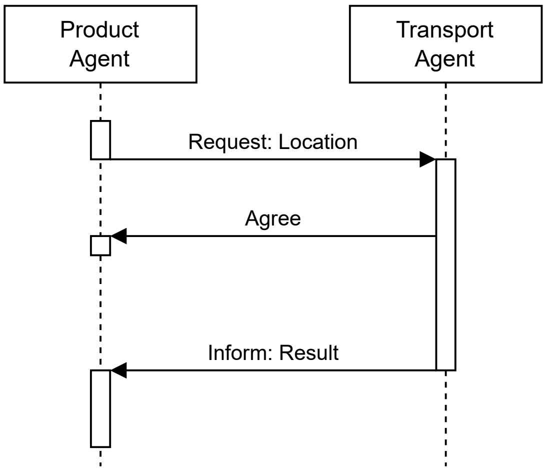


Figure 3.2: Sequence Diagram of Product Agent and Transport Agent Requests.

Finally, the same process is repeated between the PA and the previously chosen RA, this time to request an action to be performed on the product, shown in Figure 3.3.

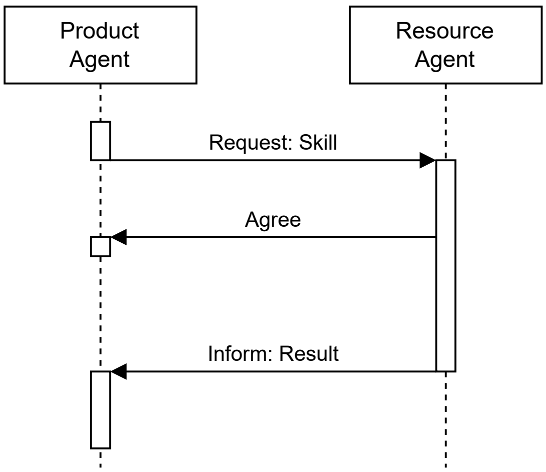


Figure 3.3: Sequence Diagram of Product Agent and Resource Agent Requests.

If this is the last step in the products production sequence, then the PA will request transportation to storage through the same method, at which point it will declare the product as complete and stop its operations. Otherwise, the process will start from the beginning, with the PA asking for a RA through the FIPA Contract Net, and so on. This whole process can be visualized in Figure 3.4.

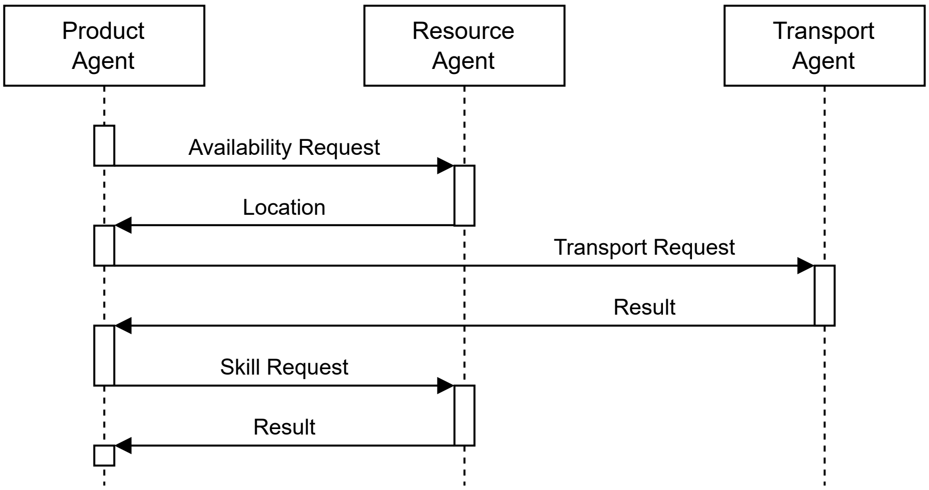


Figure 3.4: Sequence Diagram of Multi-Agent System Operations.

3.2 Module Engine

The Module Engine acts as an interface between the agent software and the Link Library tasked with communicating with its underlying hardware. Whenever an agent is launched, it passes a marketplace file listing all available libraries, a Link Library type and its corresponding configurations given to it by the DA to the Module Engine. The Module Engine will then look up the Link Library type on the marketplace and load it with the configurations provided in order to communicate with the hardware. Each Link Library allows communications through a different protocol, allowing for the flexibility desired. From now on, the Module Engine will act as an intermediary between the agent and the Link Library.

When the Library is loaded, it is capable of immediately establishing a connection to the hardware, if the communications protocol requires it to do so. Whenever the agent needs to make a request of the hardware, it sends the message to the Module Engine. From there it is sent to the Link Library, which in turn sends it through the communications protocol it was designed to use. It then awaits a response from the hardware, which it sends back through the inverse path, pictured in Figure 3.5.

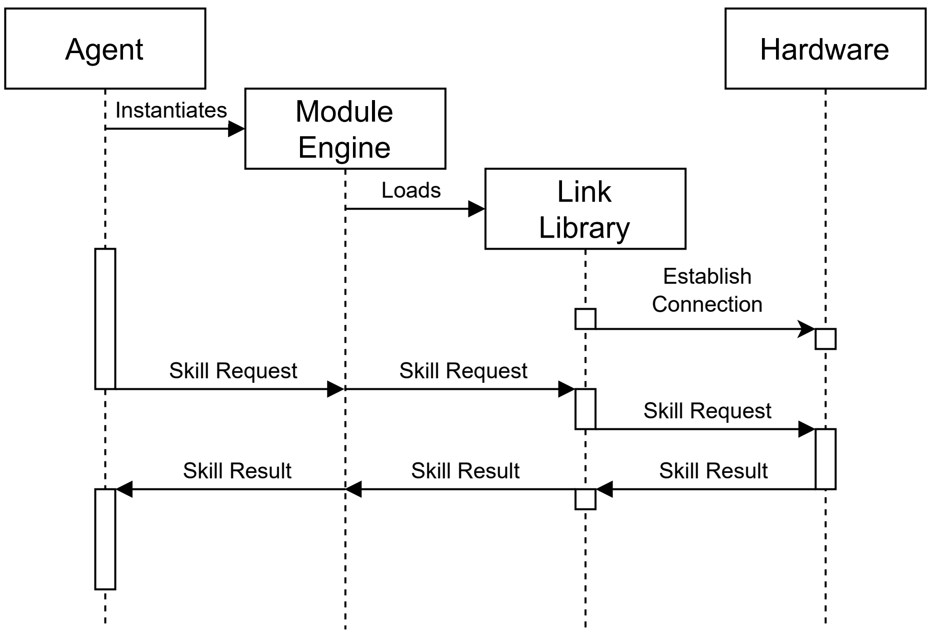


Figure 3.5: Sequence Diagram of Agent-Hardware Communications.

4 Implementation

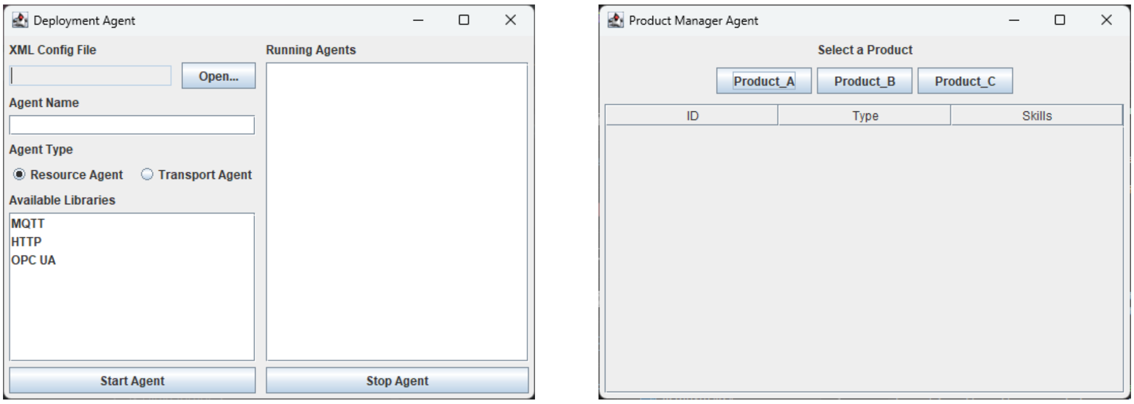
4.1 Agent Management

As specified in Chapter 3, three agent types were developed, along with a Product Manager (PM) and a Deployment Agent (DA) with the purpose of launching the other agent types. Both of these agents have a graphical interface, where it is possible to start the corresponding agents.

For the implementation of the system, the Java programming language was selected. As mentioned in 2.2.1, JADE has been built with the FIPA specifications in mind, so this framework was chosen to implement the Multi-Agent System (MAS).

The JADE framework provides a lot of tools, not only for agent communications but also for agent management. The implemented DA and PM create a container on launch, where their subsequent agents are launched. In the case of the DA, the Resource Agents and Transport Agents are both launched under a "DeployedAgents Container" and in the case of the PM, Product Agents are launched under a "ProductAgents Container". This helps with the management of the MAS and also allows the launch and termination of specific RAs/TAs, useful to demonstrate the functionalities of the Module Engine.

An auxiliary class of constants called "Constants" was defined to store all unchanging variable values during the operations of the MAS, such as available skills and skill sequences for product manufacturing. RAs and TAs are restricted to the names specified in this class, as their skills are picked based on the name of the agent. PA names are also tied to their production sequences, and the PM will present to the user the products specified in the "Constants" class.



(a) Deployment Agent

(b) Product Manager

Figure 4.1: Management Agents Graphical User Interface.

4.2 Agent to Agent Communication

For the DF registration and consultation, another auxiliary class "DFInteraction" was developed, since most of these functionalities are used by all agents.

As soon as a Resource Agent or Transport Agent is launched, they will lookup their skills in the "Constants" class and register themselves using the methods provided in the "DFInteraction" class. They then will wait for requests from a Product Agent.

As explained in Chapter 3, the PAs are the communication starters, they will search the DF for a RA capable of performing the needed skill, and will call for proposals to any RAs that matches the requirements. Upon receiving proposals, they will pick one RA and will accept it. Now the communications are done through simple Requests. The PA will ask one of the TAs for transportation and after arriving, will ask the chosen RA to perform its skill. This process repeats until the production sequence of the PA in question is complete.

4.3 Hardware Implementation

The physical system used to simulate the workshop is made up of two stations capable of performing an action each and a straight, bi-directional conveyor belt with four different positions. The first section of the conveyor belt, referred to as C1, was considered to be the position were an external entity would place a product to be processed. The second and third positions, C2 and C3, would be were the stations could operate on the product, one position reserved for each station, henceforth referred to as Station 3 and Station 4. Station 3 operates on C2 and Station 4 operates on C3. Finally, the last position, C4, is were a product that has finished production would be deposited, to be taken by an external entity. Logically, the conveyor belt is represented by a Transport Agent in the MAS, and both Station 3 and Station 4 are represented by a Resource Agent each. A representation of this system can be found in Figure 4.2

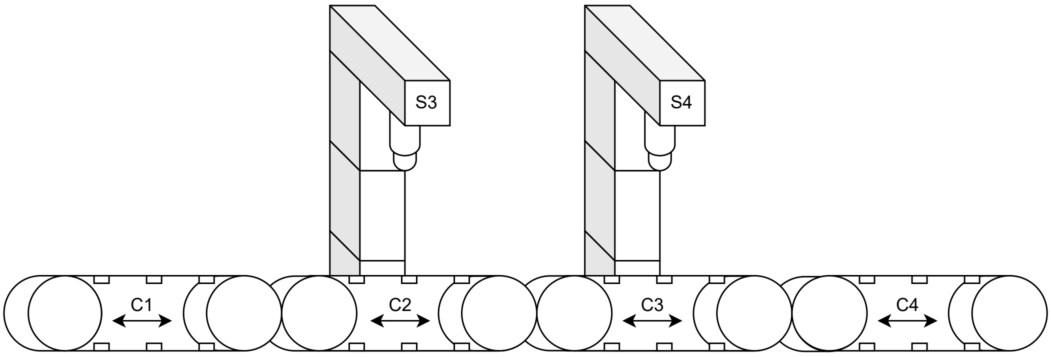


Figure 4.2: Physical Systems.

This system is controlled by a Revolution Pi, or RevPi. The RevPi is a Raspberry Pi adapted to control hardware through hardware interfaces. This controller is running an instance of NodeRED, which is a "programming tool for wiring together hardware devices, APIs and online services in new and interesting ways" [24].

Three NodeRED flows were created to give the conveyor and stations some routines, such as run forward, backward and stop for the conveyor and perform the skill for the stations. These flows receive the instruction through the communications protocol, in this case OPC UA, MQTT or HTTP, and execute the corresponding routine. Both Station 3 and 4 wait perform their action when they receive "Skill_A" and "Skill_B", respectively. More than one skill can be associated to each station, but for the purposes of this system, one was deemed enough. The conveyor belt waits for an instruction in the format "CX#TOKEN#CY" where X and Y correspond to a number from one to four. X represents the starting position for the product and Y the end position. The logic behind the flow is able to determine the direction of the movement and can move the product from any conveyor section to another.

Additionally, a MQTT broker was setup on another machine to allow the use of the MQTT protocol, and a OPC UA server was instantiated on the Raspberry Pi, through the use of a community made library that integrates NodeRED and OPC UA.

4.4 Agent to Hardware Communication

The RAs and TAs are given a configurations file, a marketplace file and the Link Library type on launch. All of these arguments are passed along to the Module Engine. The configurations file has all needed configurations required by each Link Library, they are agent-specific and in the case of this implementation include configurations for all protocols, although it is possible to use different files, as they are user created. The marketplace file consists of the name of all Link Libraries followed by the file path of the corresponding class that implements each Link Library. Finally, the Link Library type simply tells the Module Engine which Link Library to load. All files used are in the XML format.

When an agent is launched, the arguments are passed along to the Module Engine. It will take the marketplace file and transform it into a hash map, to more easily look up the classes of the Link Libraries. The name of each Link Library is used as the key for the hash map, and the path of each class is looked up and loaded as a generic Class object, which is used as the value for the hash map.

An object is then created by looking up the Link Library type in this hash map, and loading it using the Java Reflection, a feature of the Java programming language that allows for the examination of the program itself, even during runtime. This is what allows the Module Engine to load classes and create objects dynamically. When the Link Library object is created, it is given the configuration file in its constructor. The execution of skills is then done by calling a method that should be present, have the same parameters and return the same object types in all Link Libraries. The constructor must also be identical across Libraries.

The chosen Link Library will now look at this configuration file and extract all necessary parameters. As mentioned above, in this particular case, all libraries share a single configuration file that is agent-specific, but the program also support different files for different libraries.

After this, the Library simply awaits method executions. Whenever a skill needs to be executed, the method "ExecuteSkill" is called from the Module Engine. Since all Link Libraries must have this same method, it does not matter which Link Library is currently in use. This method receives a skill in the form of an object of type String and executes the procedures corresponding to the communication protocol that Link Library implements. This method then returns an object of type String with the result of the operation.

For the purpose of this implementation, three different hardware communication protocols were used, hence three Link Libraries developed. The protocols picked were OPC UA, MQTT, and HTTP. All of them need a server/broker address and in the case of OPC UA and MQTT other parameters are needed, such as request and response namespaces.

In the case of the MQTT protocol, the broker handling communications has six topics, two per agent. They are "Conveyor_Req" and "Conveyor_Res" for the Conveyor Belt, "Station3_Req" and "Station3_Res" for Station 3 and "Station4_Req" and "Station4_Res" for

Station 4. Agents publish information on all topics ending in "Req" or "Request" which is subscribed by the hardware. In turn, the hardware publishes information on all topics ending in "Res" or "Response" which is subscribed by the agents. This was done to facilitate development, since no filtering is required to ascertain the direction of the messages.

Similarly, the OPC UA protocol has six variables, with the same names and functionalities, with the caveat that these variables need to be cleared before communication starts. Before the Library sends a message, it clears both "Request" and "Response" variables. Then it sends a message to "Request" and subscribes to "Response".

Finally, the HTTP protocol works a bit different than the others. When a HTTP request is sent, the Library will simply await the HTTP response.

The Link Libraries also have a method "Stop" that doesn't take any parameters. This method is used to disconnect from the server/broker, and must be included even if the protocol in question does not make use of it.

5

Tests

To test the system, both NodeRED and the MAS systems were launched. The Product Manager and Deployment Agent startup and wait for agents to be launched. After this, two Resource Agents are launched, corresponding to Station 3 and Station 4, aptly named Station_3 and Station_4. Along with those agents, a Transport Agent is also launched, corresponding to the Conveyor Belt, called Conveyor. The right configuration files are selected for each agent and they are launched, one by one. To start off, all agents are using the Link Library that communicates through MQTT, and thus the configurations for this protocol are used. These agents are connected to the broker on launch, and will communicate to the corresponding topics described in Chapter 4.

After agents are done with their initial setups, a product agent can be launched. For the purposes of showcasing the systems capabilities, three different Product Types were created. These products are shown in 5.1, along with their production sequences and the sequence of Conveyor Belt sections they need to go to in order to complete their process.

Table 5.1: Product Types

Product Type	Production Sequence	Location Sequence
A	[Skill_A]	[C1; C2; C4]
B	[Skill_B]	[C1; C3; C4]
C	[Skill_B; Skill_A]	[C1; C3; C2; C4]

All three product types were launched, one at a time. The system handled their production without any problems, with all RAs and TA communicating to the hardware through the MQTT protocol.

After the last product finished its process, the Resource Agent for Station 3 was stopped and restarted with the HTTP Link Library. The same test was run again, and no significant changes to the production steps were noted, even though Station 3 was now communicating through HTTP Requests to the hardware. Similar tests were performed, where an agent would be stopped and restarted again, with a different Link Library.

The only changes in processing time were noticed when the OPC UA protocol was used, but this is likely due to the server being hosted on the RevPi itself, which could slow down its operation speed somewhat, since multiple processes were running in parallel. This could've easily been mitigated by hosting the OPC UA server on another machine.

To ascertain if there was a difference in processing time in a system using the Module Engine and a system communicating directly to the hardware, a simple test was done. A dummy agent was created for this and in its setup code a simple loop was implemented that would send two HTTP POSTs. The first through the Module Engine and the other directly. This loop ran for a hundred thousand iterations and the time each communication took was taken, and the average of the differences between times was calculated. All average times measured were under a millisecond, which mean that, on average, the system using the Module Engine takes about the same amount of time as system communicating directly to the hardware.

This result was to be expected, since the Link Library used by the Module Engine is loaded like any other Java class and treated like any other Java object. This makes it behave like it was there since the projects compilation, even though it wasn't, which should not impact the time performance significantly, as the test demonstrates.

6 Conclusion

The concept of Industry 4.0 revolutionized the manufacturing sector through the digitalization of manufacturing processes. Industrial CPS or CPPSs enable the transition to the Industry 4.0 standard. These systems are service oriented, can process Big Data and have cloud integration for storage. They are also able to interface with the real world, by using sensors, and to act on it, by using actuators. In essence these systems are composed of two counterparts, a physical one and a digital one. This makes them very robust and efficient, since they rely on the digital version to extract information on how to act on their environment.

Industrial MAS were suggested as a model for the implementation of a CPPS due to the advantages they could bring to the industry, such as the decentralization of the system, allowing for the autonomous behavior of each individual agent to accomplish a common goal. This makes the system very robust and flexible because errors don't propagate throughout the different layers of the system and agents can join and leave the system as needed. They, however, have not seen practical uses outside research prototypes. This may come from the skepticism that they won't perform to the same capabilities as existing systems.

There have been a lot of proposed architectures and methods of interfacing throughout the years to try and make MASs more accessible. In this work a new way of interfacing is proposed, enabling more seamless integration of new industrial agents into already existing hardware through the use of the developed Module Engine powered by the Reflections feature of the Java programming language. This allows for the creation of highly flexible libraries, that can integrate any king of hardware with an MAS. This in turn would help the integration of MASs in industrial settings, increasing their use by the manufacturing sector and consequently allowing for the development of even better technologies due to their practical use.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] B. Vogel-Heuser and D. Hess. "Guest Editorial Industry 4.0–Prerequisites and Visions". In: *IEEE Transactions on Automation Science and Engineering* 13.2 (2016), pp. 411–413. DOI: 10.1109/TASE.2016.2523639 (cit. on pp. 1, 4).

[3] *Industry 4.0 Case Studies (curated)*. URL: https://amfg.ai/2019/03/28/industry-4-0-7-real-world-examples-of-digital-manufacturing-in-action/ (visited on 2024-07-29) (cit. on p. 2).

[4] L. Sakurada and P. Leitão. "Multi-Agent Systems to Implement Industry 4.0 Components". In: *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*. Vol. 1. 2020, pp. 21–26. DOI: 10.1109/ICPS48405.2020.9274745 (cit. on p. 2).

[5] S. Karnouskos et al. "Industrial Agents as a Key Enabler for Realizing Industrial Cyber-Physical Systems: Multiagent Systems Entering Industry 4.0". In: *IEEE Industrial Electronics Magazine* 14.3 (2020), pp. 18–32. DOI: 10.1109/MIE.2019.2962225 (cit. on p. 2).

[6] R. Unland. "Chapter 1 - Software Agent Systems". In: *Industrial Agents*. Ed. by P. Leitão and S. Karnouskos. Boston: Morgan Kaufmann, 2015, pp. 3–22. ISBN: 978-0-12-800341-1. DOI: 10.1016/B978-0-12-800341-1.00001-2 (cit. on pp. 2, 6).

[7] The Foundation for Intelligent Physical Agents. *Welcome to the Foundation for Intelligent Physical Agents*. URL: http://www.fipa.org/ (visited on 2024-07-29) (cit. on p. 2).

[8] B. Marschall et al. "Design and Installation of an Agent-Controlled Cyber-Physical Production System Using the Example of a Beverage Bottling Plant". In: *IEEE Journal of Emerging and Selected Topics in Industrial Electronics* 3.1 (2022), pp. 39–47. DOI: 10.1109/JESTIE.2021.3097941 (cit. on pp. 2, 11, 13, 14).

[9] S. Karnouskos and P. Leitão. "Key Contributing Factors to the Acceptance of Agents in Industrial Environments". In: *IEEE Transactions on Industrial Informatics* 13.2 (2017), pp. 696–703. DOI: 10.1109/TII.2016.2607148 (cit. on pp. 2, 11).

[10] P. Leitão, A. W. Colombo, and S. Karnouskos. "Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges". In: *Computers in Industry* 81 (2016). Emerging ICT concepts for smart, safe and sustainable industrial systems, pp. 11–25. ISSN: 0166-3615. DOI: 10.1016/j.compind.2015.08.004 (cit. on p. 5).

[11] O. Cardin. "Classification of cyber-physical production systems applications: Proposition of an analysis framework". In: *Computers in Industry* 104 (2019), pp. 11–21. ISSN: 0166-3615. DOI: 10.1016/j.compind.2018.10.002 (cit. on p. 5).

[12] D. H. Arjoni et al. "Manufacture Equipment Retrofit to Allow Usage in the Industry 4.0". In: *2017 2nd International Conference on Cybernetics, Robotics and Control (CRC)*. 2017, pp. 155–161. DOI: 10.1109/CRC.2017.46 (cit. on pp. 5, 6).

[13] P. Leitão et al. "Recommendation of Best Practices for Industrial Agent Systems based on the IEEE 2660.1 Standard". In: *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*. Vol. 1. 2021, pp. 1157–1162. DOI: 10.1109/ICIT46573.2021.9453511 (cit. on pp. 6, 7).

[14] S. Karnouskos et al. "Key directions for industrial agent based cyber-physical production systems". In: *Proceedings - 2019 IEEE International Conference on Industrial Cyber Physical Systems, ICPS 2019* (2019-05), pp. 17–22. DOI: 10.1109/ICPHYS.2019.8780360 (cit. on pp. 6, 8, 11).

[15] P. Leitão et al. "Integration Patterns for Interfacing Software Agents with Industrial Automation Systems". In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. 2018, pp. 2908–2913. DOI: 10.1109/IECON.2018.8591641 (cit. on pp. 8–10).

[16] "IEEE Recommended Practice for Industrial Agents: Integration of Software Agents and Low-Level Automation Functions". In: *IEEE Std 2660.1-2020* (2021), pp. 1–43. DOI: 10.1109/IEEESTD.2021.9340089 (cit. on p. 8).

[17] *Jade Site | Java Agent DEvelopment Framework*. URL: https://jade.tilab.com/ (visited on 2024-07-29) (cit. on p. 10).

[18] The OPC Foundation. *Unified Architecture - OPC Foundation*. URL: https://opcfoundation.org/about/opc-technologies/opc-ua/ (visited on 2024-07-29) (cit. on p. 10).

[19] M. Seitz et al. "Automation platform independent multi-agent system for robust networks of production resources in industry 4.0". In: *Journal of Intelligent Manufacturing* 32 (7 2021-10), pp. 2023–2041. ISSN: 15728145. DOI: 10.1007/S10845-021-01759-2 (cit. on p. 11).

[20] F. L. Alejano et al. "Enhancing the interoperability of heterogeneous hardware in the Industry: a Multi-Agent System Proposal". In: *2023 6th Conference on Cloud and Internet of Things (CIoT)*. 2023, pp. 157–162. DOI: 10.1109/CIoT57267.2023.10084891 (cit. on p. 11).

[21] B. Marschall, D. Ochsenkuehn, and T. Voigt. "Design and Implementation of a Smart, Product-Led Production Control Using Industrial Agents". In: *IEEE Journal of Emerging and Selected Topics in Industrial Electronics* 3.1 (2022), pp. 48–56. DOI: 10.1109/JESTIE.2021.3117121 (cit. on pp. 11–13).

[22] A. D. Rocha et al. "Agent-based Plug and Produce Cyber-Physical Production System – Test Case". In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. Vol. 1. 2019, pp. 1545–1551. DOI: 10.1109/INDIN41052.2019.8972169 (cit. on p. 15).

[23] A. D. Rocha, G. Barata Diogo and Di Orio, and J. Santos Tiago and Barata. "PRIME as a Generic Agent Based Framework to Support Pluggability and Reconfigurability Using Different Technologies". In: *Technological Innovation for Cloud-Based Engineering Systems*. Ed. by L. M. Camarinha-Matos, T. A. Baldissera, and F. Di Orio Giovanni and Marques. Vol. 450. Cham: Springer International Publishing, 2015, pp. 101–110. ISBN: 978-3-319-16766-4. DOI: 10.1007/978-3-319-16766-4_11 (cit. on pp. 16–18).

[24] OpenJS Foundation and Contributors. *Node-RED*. URL: https://www.nodered.org/ (visited on 2024-07-29) (cit. on p. 26).