NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DAVID RODRIGUES FERREIRA

BSc in Electrical and Computer Engineering

MARKETPLACE-DRIVEN FRAMEWORK

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

Dissertation Plan
MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
Draft: *January 27, 2024*

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

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

BSc in Electrical and Computer Engineering

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

Dissertation Plan
MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
Draft: *January 27, 2024*

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 skepticism surrounding the concept of Multi-agent Systems, and whether or not they can perform to the same efficacy when compared to the already existing systems.

In this work, an architecture which helps solve this problem is proposed. More precisely, this platform allows 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. By doing this, 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 v

List of Tables vi

Acronyms vii

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Problem and Solution . . . 2

2 State of the Art 3
- 2.1 Cyber-Physical Production Systems . . . 3
- 2.2 Multi-Agent Systems . . . 4
  - 2.2.1 Best Practices and Common Architectures . . . 5
  - 2.2.2 OPC UA . . . 8
  - 2.2.3 Modbus . . . 9
- 2.3 Practical Uses . . . 9
  - 2.3.1 Bottling Plant . . . 10
  - 2.3.2 Agent-based Plug and Produce CPPS . . . 13
  - 2.3.3 PRIME . . . 14

3 Preliminary Work 17
- 3.1 Base Framework . . . 17
  - 3.1.1 Design . . . 19
  - 3.1.2 Implementation . . . 19
  - 3.1.3 Test Libraries . . . 19
  - 3.1.4 Validation and Testing . . . 19
  - 3.1.5 Writing the dissertation . . . 19

4 Conclusion 20

Bibliography **21**

List of Figures

2.1 Industrial Multi-Agent System 5
2.2 Tightly coupled Hybrid interface. Source: Adapted from [11] 7
2.3 Loosely coupled Hybrid interface. Source: Adapted from [11] 7
2.4 Tightly coupled On-device interface. Source: Adapted from [11] 8
2.5 Loosely coupled On-device interface. Source: Adapted from [11] 8
2.6 Interactions between PLCLinkAgent and Programmable Logic Controller (PLC). Source: Adapted from [7] 12
2.7 Plug and Produce Resource Agent (RA). Source: Adapted from [18] 13

3.1 System Architectures 18

List of Tables

3.1 Project timeline 18

Acronyms

AGV	Automated Guided Vehicle *(p. 3)*
API	Application Programming Interface *(p. 6)*
CA	Component Agent *(p. 15)*
COM/DCOM	Microsoft Windows Distributed Component Object Model *(p. 9)*
CPPS	Cyber-Physical Production System *(pp. 1–5, 10, 11, 13, 15, 17, 19)*
CPS	Cyber-Physical System *(pp. 1, 3)*
DA	Deployment Agent *(pp. 13, 14)*
DWPS	Device Profile Web Services *(pp. 13, 15)*
FIPA	Foundation for Intelligent Physical Agents *(pp. 2, 8, 10, 14)*
IntA	InterfaceAgent *(p. 11)*
MAS	Multi-Agent System *(pp. 2–4, 6, 8–10, 12–14, 17, 20)*
OEE	Overall Equipment Effectiveness *(p. 4)*
OPC Classic	Open Platform Communications Classic *(pp. 8, 9)*
OPC UA	Open Platform Communications Unified Architecture *(pp. 5, 8, 9, 11, 12)*
PA	Product Agent *(pp. 10, 13, 14)*
PLC	Programmable Logic Controller *(pp. v, 5, 9, 12, 13, 15–17)*
PMA	Production Management Agent *(p. 15)*
PSA	Prime System Agent *(p. 15)*
RA	Resource Agent *(pp. v, 10, 11, 13, 14)*
SMA	Skill Management Agent *(p. 15)*

TA Transport Agent *(pp. 13, 14)*

TCP-IP Transfer Control Protocol - Internet Protocol *(p. 9)*

UDP User Datagram Protocol *(p. 9)*

WS4D-JMEDS Web Services 4 Devices - Java Multi Edition DPWS Stack *(p. 13)*

XML Extensible Markup Language *(p. 10)*

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. The concept of Cyber-Physical Production System (CPPS), is essentially a Cyber-Physical System (CPS) that is thought of from a production perspective. A CPS is a physical system that has a digital representation. They both 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 [1]:

- 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 [2].

In light of this, CPPSs became a popular research subject. This research was mainly done on the topic of Multi-Agent System (MAS) [3] [4]. 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 [5].

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 [6]. An MAS in an industrial 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 [7]. 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 [8]. 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.

1.1.1 Problem and Solution

In this work, a new approach to developing an interface 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. This would simplify the process of creating new agents for the system, making the addition of new agents way more flexible and less time consuming. It would also allow the creation of more generic agents that could be used in to integrate many types of hardware.

2

State of the Art

In this chapter we'll explore how researchers have dealt with the challenges of creating an 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 from the environment to the digital system and the digital system processes this data and instructs the physical system on what actions to perform. As such, this system works in a loop, constantly exchanging data and instructions to achieve a common goal.

A CPPS is, as the name implies, a CPS that operates in a manufacturing setting. The physical system is composed of hardware like robot arms, Automated Guided Vehicle (AGV)s, 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 [1] 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 algorithm 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

2.2 Multi-Agent Systems

To understand what should be the best practices in an Multi-Agent System (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 [5].

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 [9]. 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 [10]. 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 to the systems needs [10].

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.

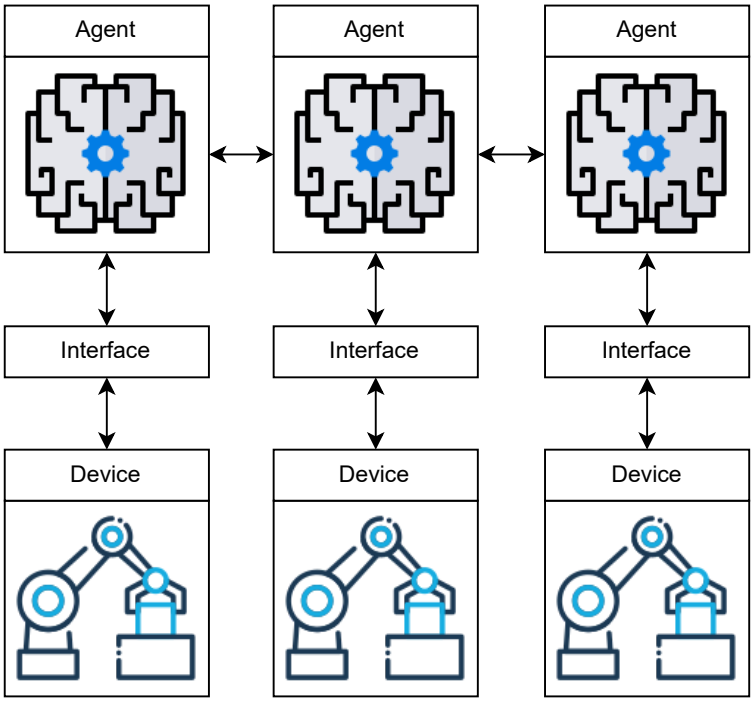


Figure 2.1: Industrial Multi-Agent System

2.2.1 Best Practices and Common Architectures

Leitão et al. [9] 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 one 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 Programmable Logic Controller (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 Open Platform Communications Unified Architecture (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 [10], 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. [11] analyzed a study performed under the IEEE P2660.1 Working Group [12] 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 [11]:

- 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 [11]:

- 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 Application Programming Interface (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 [11].

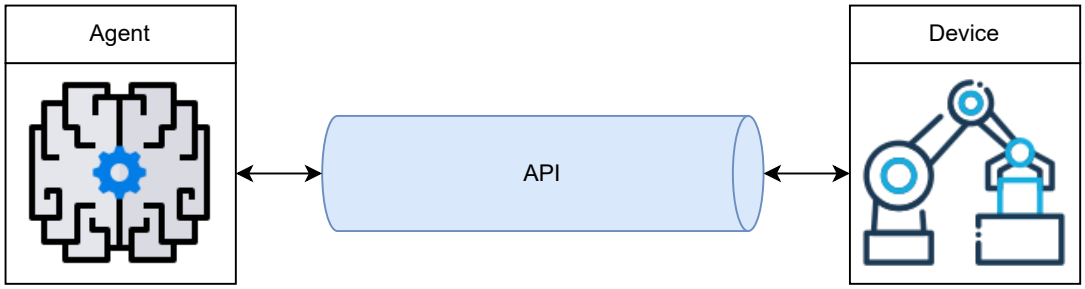


Figure 2.2: Tightly coupled Hybrid interface. Source: Adapted from [11]

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. 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 [11].

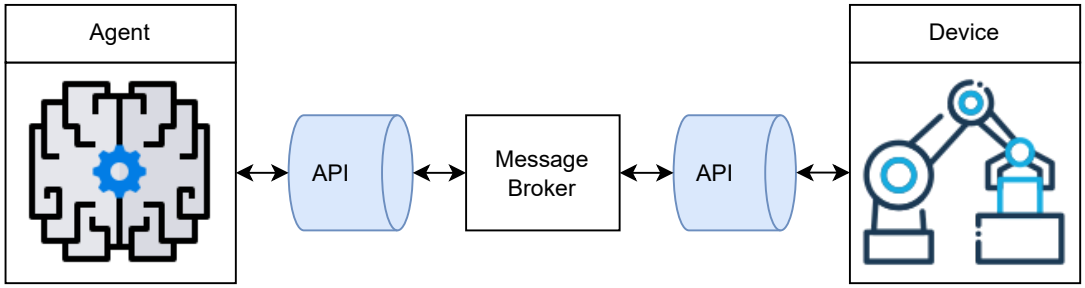


Figure 2.3: Loosely coupled Hybrid interface. Source: Adapted from [11]

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 but abstracting some elements. This option still holds good results in real time control, but not as good as the first one [11].

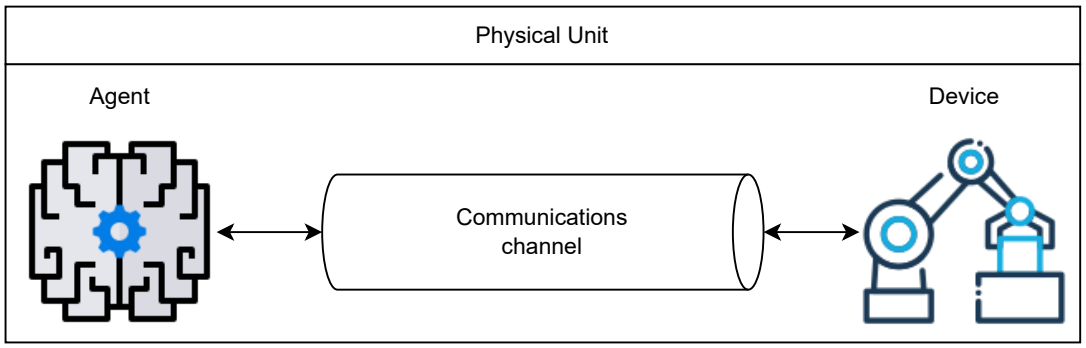


Figure 2.4: Tightly coupled On-device interface. Source: Adapted from [11]

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 the whole unit is dependent on the performance of the broker [11].

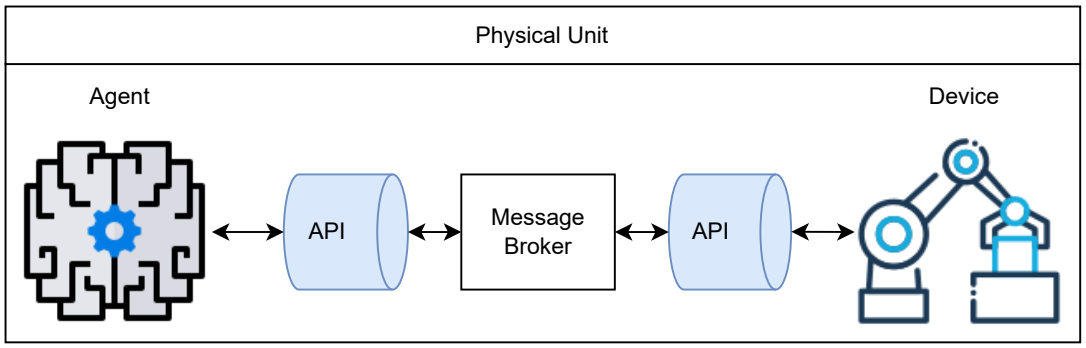


Figure 2.5: Loosely coupled On-device interface. Source: Adapted from [11]

The most common programming language to codify agents is Java, most likely due to JADE, an agent-based framework, followed by C++ [11]. This framework helps developers in the implementation of MASs with the Foundation for Intelligent Physical Agents (FIPA) specifications. It also allows deployment for different machines, due to Java supporting multiple devices [13]. 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 [11], which is a platform independent data exchange standard. It allows for both server-client and publish/subscribe communications as well see in Section 2.2.2 [14].

2.2.2 OPC UA

Another way to integrate hardware into an MAS it through the use of already made libraries. Open Platform Communications Unified Architecture (OPC UA) was created based on Open Platform Communications Classic (OPC Classic), which in turn is based

on the Microsoft Windows Distributed Component Object Model (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 [14]. 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 model, 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 [15].

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 Transfer Control Protocol - Internet Protocol (TCP-IP) and User Datagram Protocol (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 [16].

2.3 Practical Uses

Adoption of industrial oriented MAS has been slow. According to Karnouskos and Leitão [8], the technology was still in its infancy almost two decades ago, with an incremental progress at best being made since then. In [10], 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 [7], a research paper [17] 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 Extensible Markup Language (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 acquisition 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 [7], Marschall et al. created the service oriented bottling plant using the industrial agent system designed in [17], 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.

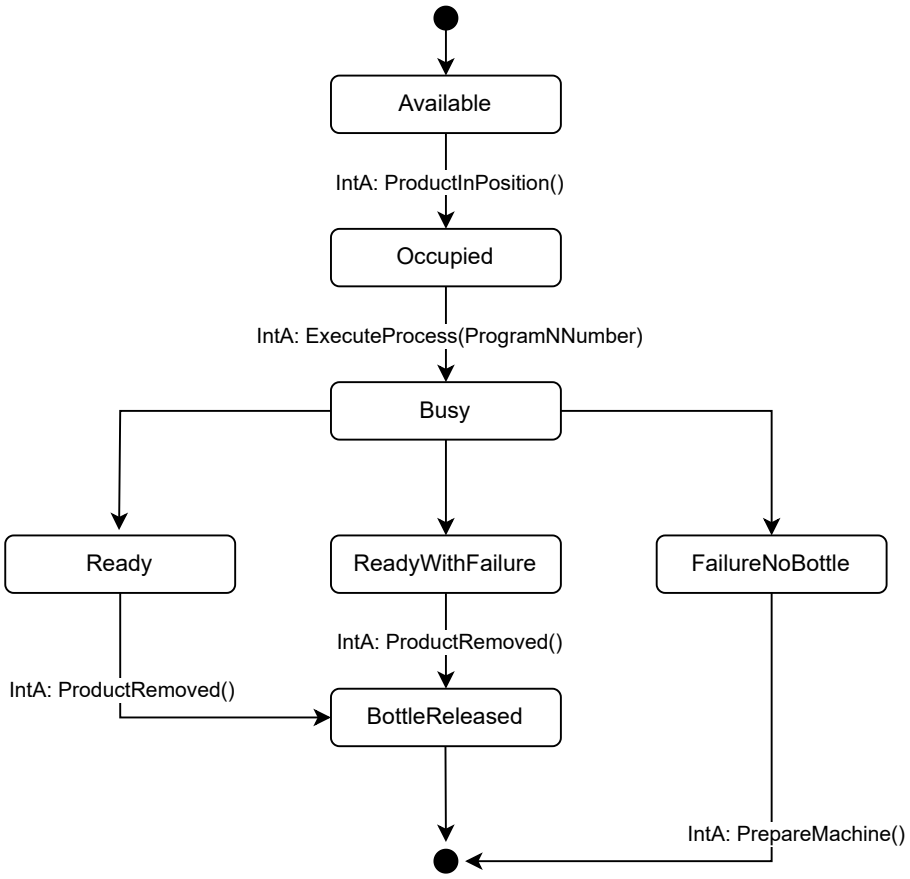


Figure 2.6: Interactions between PLCLinkAgent and PLC. Source: Adapted from [7]

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. [18] 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 Product Agent (PA) and Resource Agent (RA), performing similar functions to the ones in Section 2.3.1 and in addition the authors also considered Transport Agent (TA)s 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.

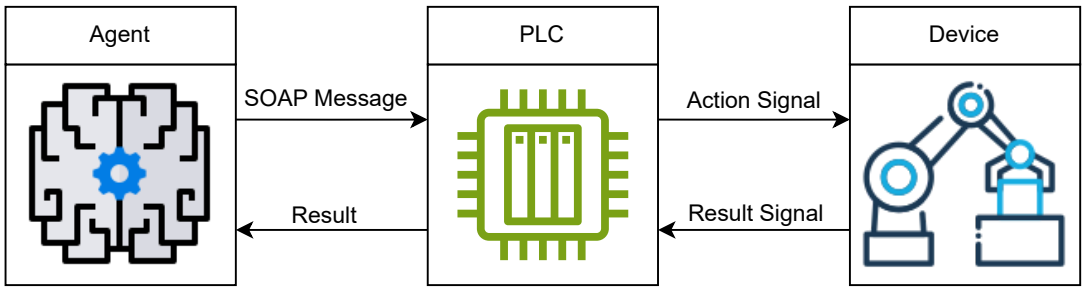


Figure 2.7: Plug and Produce RA. Source: Adapted from [18]

For the interface between hardware and agent, PLCs were chosen. These PLCs 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 PAs. 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 et al. [19] 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 Analyses

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 [19].

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 [19].

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 [19]. This last option is not flexible and uses a case-by-case configuration, making it the least desirable in a system.

3

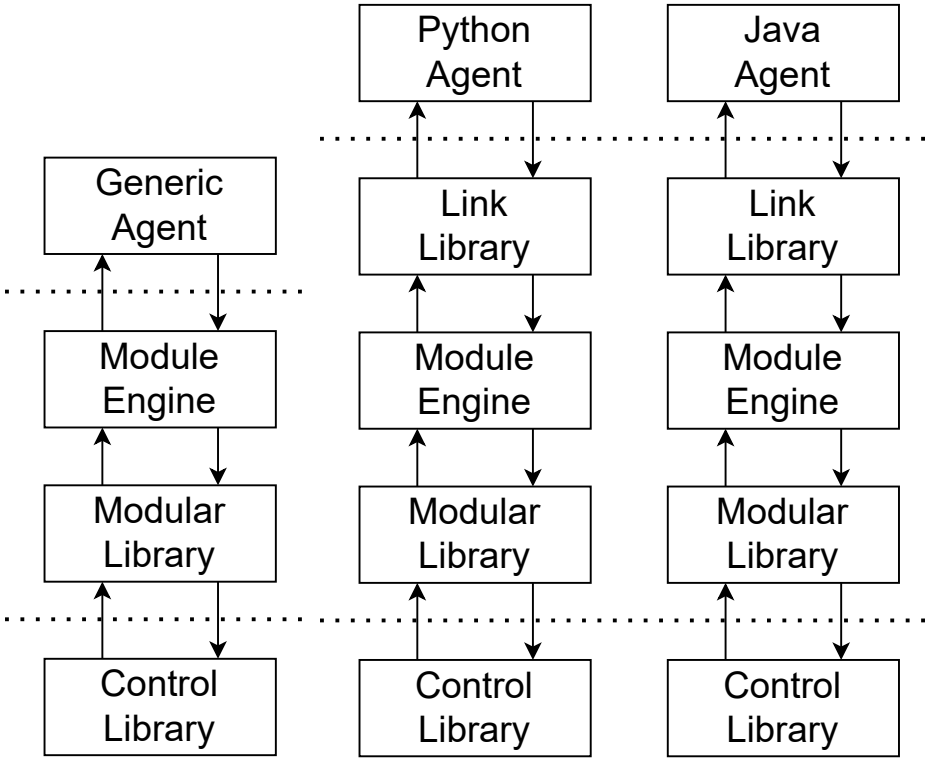
Preliminary Work

As stated in Chapter 1, the adoption of Multi-Agent System (MAS)s as Cyber-Physical Production System (CPPS)s for Industry 4.0 has not been as widespread as desired. It never left the prototyping stage and therefore never evolved in an industrial environment and never improved in a practical setting. This work seeks to alleviate some of this by suggesting a method to more easily create new Industrial Agents, capable of interfacing with industrial hardware.

3.1 Base Framework

In this work we propose a marketplace framework capable of integrating any kind of device, from high level computational interfaces to simple Programmable Logic Controller (PLC)s for hardware interfacing. This system should be capable of easily integrating new hardware by picking from a set of generic libraries. A developer wanting to add new hardware to the system should be able to specify what kind of device it is, and then the proposed framework picks from a database the most adequate libraries for that specific device. The developer should be able to then tweak these configurations to better suit the expected behaviors the new hardware should follow in the production line. An example of the systems architecture can be found in Figure 3.1(a). As an alternative, this system may also be capable of integrating any kind of hardware with any kind of agent, through the use of these generic libraries, an example is shown in Figure 3.1(b).

In both Figure 3.1(a) and Figure 3.1(b), the Module Library is the part of the system that selects the correct libraries to use. It then implements them as a Modular Library, that then interfaces with the hardware. Since this systems work from a high layer to a lower layer, it could also work in the opposite direction. The Module Engine could also pick libraries to interface with the agent above and implement them as a Link Library. This would allow the system to integrate any kind of agent with any kind of hardware.



(a) Example Architecture (b) Alternative Architecture

Figure 3.1: System Architectures

The project development plan follows this order, and in Table 3.1 the development timeline is shown:

1. Design the system architecture
2. Implement the platform
3. Implement the test libraries
4. Validation and testing
5. Writing the dissertation

Table 3.1: Project timeline

	February	March	April	May	June	July	August	September
Design architecture								
Implement platform								
Implement test Libraries								
Testing and validation								
Writing Dissertation								

3.1.1 Design

During the design stage of the project, the main features and requirements will be delineated. The base system architecture will be designed and the main programming language with which the system will be made will be chosen.

3.1.2 Implementation

At the implementation stage, the platform will be developed. Its main features will be implemented, such as the way users will configure the libraries. The system architecture may still be adapted at this point to better fit the requirements.

3.1.3 Test Libraries

During this stage, test libraries for the platform will be developed. These test libraries would simulate the database that the platform is dependent on to create the modules for the new industrial agents that will be integrated into the simulated CPPS.

3.1.4 Validation and Testing

Finally, the system will be evaluated with multiple tests to check the robustness and flexibility of the system

3.1.5 Writing the dissertation

This stage is self-explanatory, the dissertation will be written to record the whole process of development, implementation and testing.

4

Conclusion

The concept of Industry 4.0 revolutionized the manufacturing sector by digitalizing manufacturing processes. This allows for the provision of service oriented platforms, better control over production and the customization of products by the customers. For this Industrial Multi-Agent System (MAS) were suggested due to the advantages they could bring to the industry. 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. This work proposes another such method, enabling more seamless integration of new industrial agents into an already existing system. By selecting a collection of generic libraries from a database during agent integration, it would be possible to create more agents more easily in less time, facilitating the adoption of MAS by the industry. This in turn would help the integration of MASs in industrial settings, increasing their use by the manufacturing sector and consequently allow for the development of even better technologies due to their practical use.

Bibliography

[1] 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. ISSN: 15455955. DOI: 10.1109/TASE.2016.2523639 (cit. on pp. 1, 3).

[2] R. editors. *Industry 4.0 Case Studies (curated)*. URL: https://www.rit.edu/advancedmanufacturing/industry40/course/industry-40-case-studies-curated (cit. on p. 2).

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

[4] 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 2020-09), pp. 18–32. ISSN: 19410115. DOI: 10.1109/MIE.2019.2962225 (cit. on p. 2).

[5] P. Leitão and S. Karnouskos. *Industrial Agents*. Elsevier, 2015. ISBN: 9780128003411. DOI: 10.1016/C2013-0-15269-5. URL: https://linkinghub.elsevier.com/retrieve/pii/C20130152695 (cit. on pp. 2, 4).

[6] *Welcome to the Foundation for Intelligent Physical Agents*. URL: http://www.fipa.org/ (cit. on p. 2).

[7] 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, 10–12).

[8] 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, 9).

[9] P. Leitão et al. "Recommendation of Best Practices for Industrial Agent Systems based on the IEEE 2660.1 Standard". In: vol. 2021-March. Institute of Electrical and Electronics Engineers Inc., 2021-03, pp. 1157–1162. ISBN: 9781728157306. DOI: 10.1109/ICIT46573.2021.9453511 (cit. on pp. 4, 5).

[10] 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. 4, 6, 9).

[11] 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. 6–8).

[12] "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. 6).

[13] *Jade Site | Java Agent DEvelopment Framework*. URL: https://jade.tilab.com/ (cit. on p. 8).

[14] *Unified Architecture - OPC Foundation*. URL: https://opcfoundation.org/about/opc-technologies/opc-ua/ (cit. on pp. 8, 9).

[15] 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/FIGURES/10. URL: https://link.springer.com/article/10.1007/s10845-021-01759-2 (cit. on p. 9).

[16] 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. 9).

[17] 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. 10, 11).

[18] 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. 13).

[19] A. D. Rocha et al. "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 et al. Cham: Springer International Publishing, 2015, pp. 101–110. ISBN: 978-3-319-16766-4 (cit. on pp. 14–16).