NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DAVID RODRIGUES FERREIRA

Master/BSc in Name of Previous Degree

A VERY LONG AND IMPRESSIVE THESIS TITLE WITH A FORCED LINE BREAK

SOME THOUGHTS ON THE LIFE, THE UNIVERSE, AND EVERYTHING ELSE

Dissertation Plan
MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
Draft: *January 19, 2024*

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

A VERY LONG AND IMPRESSIVE THESIS TITLE WITH A FORCED LINE BREAK

SOME THOUGHTS ON THE LIFE, THE UNIVERSE, AND EVERYTHING ELSE

DAVID RODRIGUES FERREIRA

Master/BSc in Name of Previous Degree

Adviser: Mary Doe Adviser Name
Full Professor, NOVA University Lisbon

Co-advisers: John Doe Co-Adviser Name
Associate Professor, NOVA University Lisbon

John Doe other Co-Adviser Name
Full Professor, NOVA University Lisbon

Dissertation Plan
MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
Draft: *January 19, 2024*

Abstract

Regardless of the language in which the dissertation is written, usually there are at least two abstracts: one abstract in the same language as the main text, and another abstract in some other language.

The abstracts' order varies with the school. If your school has specific regulations concerning the abstracts' order, the NOVAthesis LaTeX (novathesis) (LaTeX) template will respect them. Otherwise, the default rule in the novathesis template is to have in first place the abstract in *the same language as main text*, and then the abstract in *the other language*. For example, if the dissertation is written in Portuguese, the abstracts' order will be first Portuguese and then English, followed by the main text in Portuguese. If the dissertation is written in English, the abstracts' order will be first English and then Portuguese, followed by the main text in English. However, this order can be customized by adding one of the following to the file `5_packages.tex`.

```
\ntsetup{abstractorder={<LANG_1>,...,<LANG_N>}}
\ntsetup{abstractorder={<MAIN_LANG>={<LANG_1>,...,<LANG_N>}}}
```

For example, for a main document written in German with abstracts written in German, English and Italian (by this order) use:

```
\ntsetup{abstractorder={de={de,en,it}}}
```

Concerning its contents, the abstracts should not exceed one page and may answer the following questions (it is essential to adapt to the usual practices of your scientific area):

1. What is the problem?
2. Why is this problem interesting/challenging?
3. What is the proposed approach/solution/contribution?
4. What results (implications/consequences) from the solution?

Keywords: One keyword, Another keyword, Yet another keyword, One keyword more, The last keyword

Resumo

Independentemente da língua em que a dissertação está escrita, geralmente esta contém pelo menos dois resumos: um resumo na mesma língua do texto principal e outro resumo numa outra língua.

A ordem dos resumos varia de acordo com a escola. Se a sua escola tiver regulamentos específicos sobre a ordem dos resumos, o template (LaTeX) novathesis irá respeitá-los. Caso contrário, a regra padrão no template novathesis é ter em primeiro lugar o resumo *no mesmo idioma do texto principal* e depois o resumo *no outro idioma*. Por exemplo, se a dissertação for escrita em português, a ordem dos resumos será primeiro o português e depois o inglês, seguido do texto principal em português. Se a dissertação for escrita em inglês, a ordem dos resumos será primeiro em inglês e depois em português, seguida do texto principal em inglês. No entanto, esse pedido pode ser personalizado adicionando um dos seguintes ao arquivo `5_packages.tex`.

```
\abstractorder(<MAIN_LANG>):={<LANG_1>,...,<LANG_N>}
```

Por exemplo, para um documento escrito em Alemão com resumos em Alemão, Inglês e Italiano (por esta ordem), pode usar-se:

```
\ntsetup{abstractorder={de={de,en,it}}}
```

Relativamente ao seu conteúdo, os resumos não devem ultrapassar uma página e frequentemente tentam responder às seguintes questões (é imprescindível a adaptação às práticas habituais da sua área científica):

1. Qual é o problema?
2. Porque é que é um problema interessante/desafiante?
3. Qual é a proposta de abordagem/solução?
4. Quais são as consequências/resultados da solução proposta?

Palavras-chave: Primeira palavra-chave, Outra palavra-chave, Mais uma palavra-chave, A última palavra-chave

Contents

List of Figures

List of Tables

Acronyms

novathesis NOVAthesis LaTeX *(pp. i, ii)*

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 Systems, or CPPSs for short, is essentially a Cyber-Physical System that is thought of from a production perspective. A Cyber-Physical System 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 Systems (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 MAS based 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 Cyber-Physical Production System (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 Vehicles (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 [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 MAS based CPPS, we first need to understand its base characteristics. A Multi-Agent System 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 many 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].

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.

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. 2.1) 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].

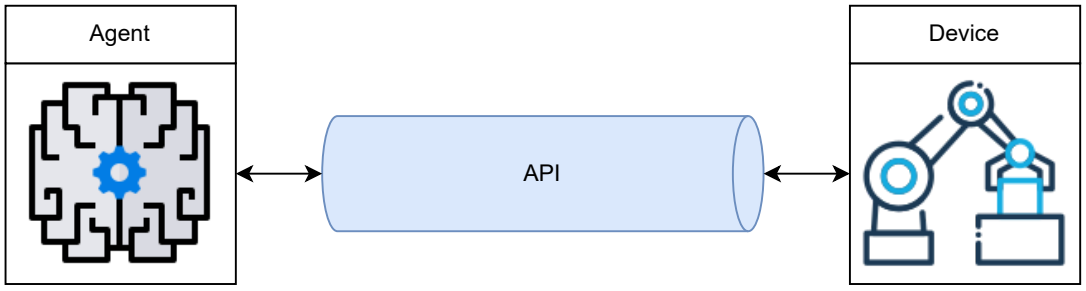

Figure 2.1: Tightly coupled Hybrid interface. Adapted from [11]

A Loosely coupled Hybrid interface (Fig. 2.2) 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].

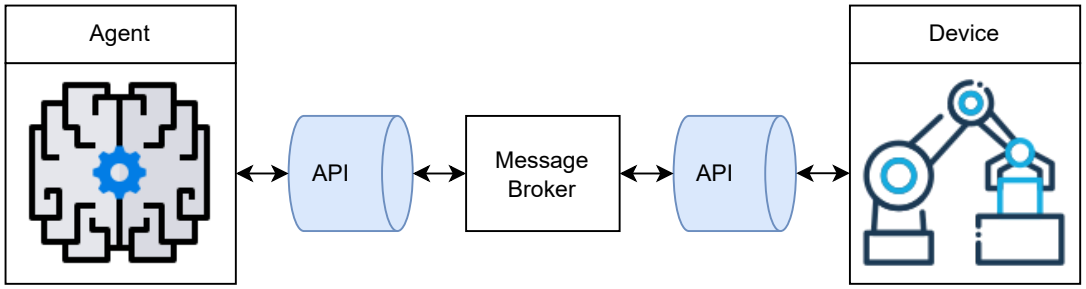

Figure 2.2: Loosely coupled Hybrid interface. Adapted from [11]

A Tightly coupled On-device interface (Fig. 2.3) 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].

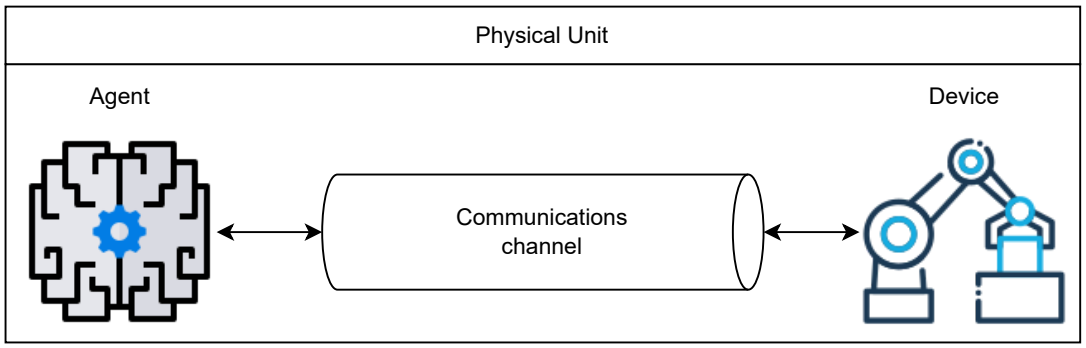

Figure 2.3: Tightly coupled On-device interface. Adapted from [11]

Finally, a Loosely coupled On-device interface (Fig. 2.4) 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].

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 FIPA specifications. It also allows deployment

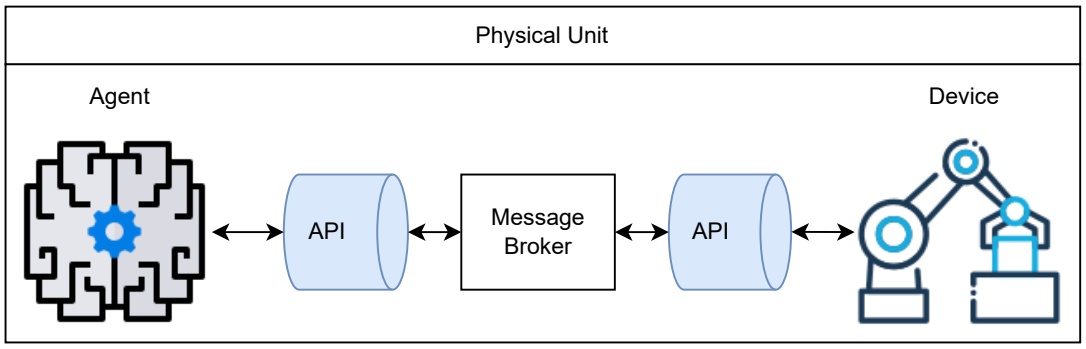

Figure 2.4: Loosely coupled On-device interface. Adapted from [11]

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 adapted into a 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 [14].

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 [15] was first done by Marschall et al., with the collaboration of multiple partners from different backgrounds. This was done to define the base requirements for project, which are:

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

In addition to the requirements, Marschall, Ochsenkuehn, and Voigt also designed a base solution approach. It consists 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.

Since our main focus is on the interface between agent and hardware, we'll mainly focus on the RA. This agent 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 read. The authors identified five different ResourceTypes and recognize that not all types 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 a service oriented bottling plant using the industrial agent system designed in [15], 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 PLC 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 PLCLingAgent and the machine is done through simple commands that specify the type of program to use, depending on the size of the bottle,

and the action to perform. However these commands need to be mapped to the manufacturer specific states through the PLCLinkAgent, making the system less scalable by default.

Nevertheless this system performed as intended, producing customized beverage bottles according the customers specifications.

2.3.2 Agent-based Plug and Produce CPPS

Rocha et al. [16] have created a Plug and Produce CPPS. It is capable of integrating new agents on the fly, as the system operates. They accomplished this by having an agent detecting whenever a new agent joins the system or an existing one leaves. 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.

This system was used to simulate a simple conveyor belt line with brushing capabilities. A product, represented by a PA enters the system and asks for the requisite action performed by sending a request to the FIPA Contract Net. All available agents then respond with their availability and the PA selects a Resource Agent to perform the action. If there is no RA available, either because all of them are occupied or there aren't any RAs with the capability to perform the action, the PA waits until one is available. To test if the system can handle the dynamic movement of agents in and out of the system, some agents were disconnect from the network. The agent responsible for identifying changes in the network acted as expected, by removing agents that were no longer part of the network from the Contract Net, and by adding new agents to it. This is a good example of a system capable of handling changes on the fly, without needing manual action for these changes to happen.

The MAS was developed using the JADE framework, the integration with the hardware done with Device Profile Web Services (DPWS). Whenever a physical action is needed by the system, the agent sends a SOAP message through DPWS to a PLC which in turn activates the corresponding action on the physical component. Although flexible and potentially scalable, this system still requires the individual programming of the PLCs according to the hardware they're integrated with.

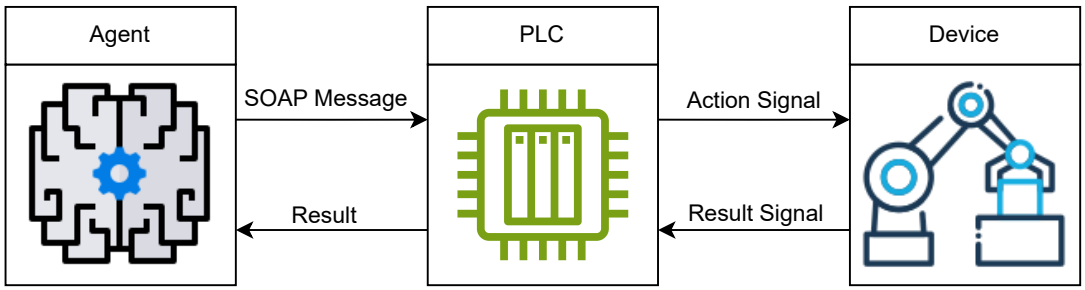

Figure 2.5: Plug and Produce Resource Agent. Adapted from [16]

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, 8, 9).

[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, 8).

[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, 5, 8).

[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. 5–8).

[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. 5).

[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 p. 8).

[15] 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. 8, 9).

[16] 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 pp. 10, 11).