Guilherme Alexandre Rolo

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Improve the Performance of Industrial Agents using Fog Computing

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: José António Barata de Oliveira, Professor Doutor, Universidade Nova de Lisboa

Co-orientador: André Dionísio Rocha, Doutor, CTS-UNINOVA

Júri

Presidente: Prof. Doutor João Carlos da Palma Goes
Arguente: Doutor Ricardo Alexandre Fernandes da Silva Peres
Vogal: Prof. Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Improve the Performance of Industrial Agents using Fog Computing

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

"Change is the law of life. And those who look only to the past or present are certain to miss the future."
John F. Kennedy

Acknowledgements

The conclusion of this project makes me achieve a personal milestone that is very important for my future.

I would like to thank Professor José Barata for giving me the chance of working with such equipment and develop the project at my own pace.

I would like to thank Dr. André Rocha for helping me throughout the whole project, taking his time to guide me even if it meant staying after hours. I hold nothing but a deep respect for the devotion shown on teaching and guiding.

I would like to thank all the colleagues in the research centre for their helpful insights.

To my friends, Gonçalo Biscaia and Bruno Ameixieiro, for all these years in University and for all we've been through. It would have been a long way without you. Thank you.

To my family for all the support and advice. Thank you very much.

Abstract

In the last decade, the market requirements have been increasing by demanding numerous different products being highly customizable. Given this need, the necessity for dynamic and flexible production lines are a high priority to meet this change.

A traditional approach is not enough to meet the market demand and due to this, several paradigms have been coined out to try and solve this problem. The proposed approach is related to communication between the shop-floor modules in order to create different products.

This work proposes an architecture where an integration layer will join a Multiagent System capable of the more recent production paradigms with legacy hardware that is present in the more traditional factories in order to have different products being produced in the same production line.

This architecture that revolves an interface that can be used by the agents in the factory in order to use the hardware modules to create a different product if need be. The main features of this project is the fact that by using datamodels and an interface created, it can be easily plugged new stations with different tools to modify the product thus increasing the amount of products that can be created.

Keywords: MultiAgent Production System, Flexible Manufacturing, Production, Integration.

Resumo

Na última década, as exigências do mercado têm vindo a aumentar pela procura crescente de diferente tipos de produto altamente costumizáveis. Dado isto, a necessidade de criar linhas de produção dinâmicas e flexíveis são de grande prioridade.

Uma abordagem tradicional não é suficiente para satisfazer a pressão do mercado. Devido a isto, vários paradigmas de produção industrial foram criados para tentar resolver este problema. A abordagem proposta é relacionada com a comunicação entre os módulos do chão-de-fábrica para criar produtos diferentes.

Este trabalho propõe uma arquitetura onde uma camada de integração irá juntar um sistema de multiagentes capaz de executar os mais recentes paradigmas de produção industrial com hardware legacy que está presente nas fábricas tradicionais para que se possa ter vários tipos de produtos diferentes a serem produzidos.

Esta arquitetura gira em torno de uma interface que pode ser usada pelos agentes na fábrica para que possam usar os modulos do hardware criar diferentes tipos de produto se necessário. As principais características deste projeto é o uso de modelos de dados e uma interface para poder acoplar novos estações com ferramentas diferentes para poder modificar o produto e assim aumentar o número de produtos que podem ser criados na fábrica.

Palavras-chave: Sistemas multiagente de produção, Manufatura flexível, Produção, Integração.

Contents

List of Figures

List of Tables

Listings

Acronyms

ASk	Atomic Skills.
BMS	Bionic Manufacturing Systems.
CFP	Call for Proposals.
CLA	Coalition Leader Agent.
CMfg	Cloud Manufacturing.
CPPS	Cyber Physical Production Systems.
CPS	Cyber Physical Systems.
CSk	Complex Skills.
DA	Deployment Agent.
DDS	Data Distribution Service.
DF	Directory Facilitator.
EPS	Evolvable Production Systems.
FIPA	Foundation for Intelligent Physical Agents.
FMS	Flexible Manufacturing Systems.
HMS	Holonic Manufacturing Systems.

I4.0	Industry 4.0.
ICT	Information and Communication Technologies.
IoT	Internet of Things.
MA	Mechatronic Agent.
MAS	Multiagent Systems.
MRA	Manufacturing Resource Agent.
PA	Product Agent.
RA	Resource Agent.
RAMI4.0	Reference Architecture Model 4.0.
RMS	Reconfigurable Manufacturing Systems.
ST	Structured Text.
TA	Transporter Agent.

Chapter 1

Introduction

At the beginning of the industrial production, there were very few models and thus the factories did not have to adapt to change since it was easier to just have different lines on the shop-floor creating different models.

Over time, with the increase need of customised and the huge quantity that it was demanded, each model had to have very different characteristics. This change came due to the amount of products that could be chosen by the customer.

Given the constant change on products being produced and the amount being produced, there was the need to create a product that was flexible and dynamic.

With the traditional approach, the shop-floors became obsolete and unable to cope with this new reality. This led to having different paradigms being proposed to meet these needs and evolve the factory to something that would fit the needs and desires of the client.

A lot of different approaches were suggested, some more subtle and some very evolutionary. These new approaches were suggested to the use of Information and Communication Technology (ICT) in the Industry.

In this architecture, on the top side, lies the Multiagent system where the means of transporting the products are abstracted as well as the skills that will be executed. The information regarding the product is exchanged between these abstractions. Whenever a new product is launched, it will communicate with the rest of the agents which skills it wants to have executed, after this communication, the agents will sort themselves and move the product accordingly. On the bottom side of the layer architecture lies the edge level, in this level the hardware was programmed in modules so that the agents in the multiagent level can access it without running the whole code. This is done through the Integration layer. This layer main purpose is to bridge the gap between those two layers, enabling their communication. Through this, it also bridges the gap between the

emergent production paradigms and legacy hardware.

1.1 Research question and hypothesis

Taking into account the problem above mentioned, a question can be asked:

- What kind of architecture can be created to join legacy hardware with Multiagent Systems?

It is proposed as a hypothesis to design an integration layer that connects to the agents in the system and communicates with the legacy hardware. This architecture is capable of running different modules in the hardware in accordance to the multiagent system in order to create the desired product.

1.2 Work overview

The proposed architecture consists of an integration layer. To interact with the integration layer there is an interface with three methods, one that is called by agents in charge of abstracting the movement of the products, one that is called by the agents in charge of abstracting the changes in the products and one that tells the transporter which transporter will be the next one in the occasion if a transporter has two next transporters.

The factory is defined firstly on a datamodel. From this datamodel, it is taken the information regarding the products, stations and movements and that is adaptable to any factory. The only entity added in runtime is the product, the transporters know which transporter is after them and the resources know where the changes in the product take place.

Inside the integration layer, the methods that are unique to each factory are present and it should be the only code that should be changed to adapt to the factory.

1.3 Contribution

The principal contribution that this thesis provides is:

- A way to bridge the gap between the emergent production paradigms and legacy hardware.

By creating an integration layer that bridges the gap between legacy hardware and the new production paradigms, it potentializes the use of these concepts in factories.

1.4 Thesis outline

The thesis is organized in the following way: Chapter 2 presents the main theoretical aspects used in the development of the work followed by their uses. Chapter 3 presents

important technological concepts. Chapter 4 presents the architecture of the system. Chapter 5 presents the implementation of all the layers in the system. Chapter 6 presents the results obtained. Chapter 7 wraps up the thesis and comments on possible improvements and future work.

Chapter 2

Literature Review

This chapter presents the literature review on three main topics. Firstly, it is presented an explanation on the fourth industrial revolution and its oncoming emergence. Secondly, it is presented the concept of Cyber Physical Production Systems (CPPS), outlining its importance in this new era of industrialisation. Finally, it is presented the emergent production paradigms, namely, Flexible Manufacturing Systems(FMS), Bionic Manufacturing Systems (BMS), Holonic Manufacturing Systems(HMS), Reconfigurable Manufacturing Systems (RMS) and Evolvable Production Systems(EPS), and their impact in production systems.

2.1 The fourth industrial revolution

Throughout the centuries, there have been some industrial revolutions, mostly when there was a new invention capable of optimising the process of manufacture. At first, due to the steam engine, the industrialisation managed to increase the speed of production. Afterwards, due to electrical and automated production lines, it simplified the human work while complicating the process. Finally, there has been a lot of talk about the fourth Industrial revolution(Industry 4.0). It started in 2011 [1], it is clear that in this decade, this subject is an upcoming trend with hundreds of research articles being published every year.

In Figure 2.1, each revolution has bullet points that were the main inventions that enable the change in the manufacturing systems. As for the first Industrial Revolution, Mechanical manufacturing and the steam engine enable such progress. In the second Industrial Revolution, due to the conception electrical production lines, it was achieved mass production. In the third Industrial Revolution, integration Information Technology and Electronics, it was achieved automated manufacturing lines. In I4.0, by integrating

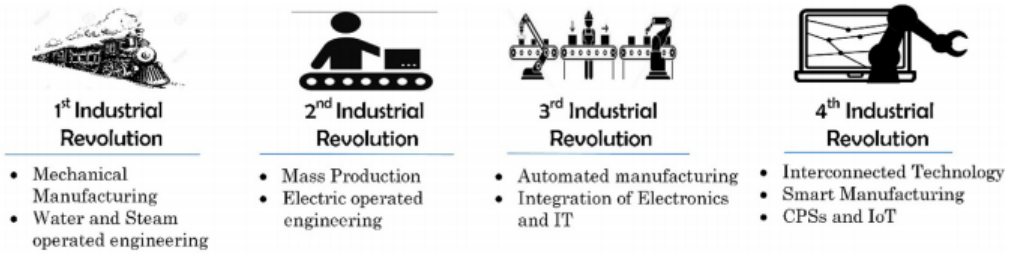

Figure 2.1: Evolution of industrial revolutions.[1]

the Internet of Things (IoT) and Cyber-Physical Systems (CPS), it is achieved adaptive manufacturing.

The Industry 4.0 was a concept created in order to transform regular machines to self-aware and self-learning machines so their overall performance and maintenance improves. The goal of I4.0 is to achieve real time data monitoring and tracking the status of the products[2].

The key element that enables this revolution is the deep change in the manufacturing systems. Due to newly Information and Communication Technologies, such IoT and Big Data and by merging them with CPPS, it creates a whole new environment where it is easy for machines to communicate with one another thus reducing the complexity of each operation[3].

The outcome of this revolution should deliver greater flexibility and robustness together with high quality standards in manufacturing. This will lead to the development of dynamic, self-organising value chains that can be optimised by varying some criteria such as cost, availability and resource consumption[4].

2.2 Cyber-Physical Production Systems

There are a number of different definitions in regard to CPS. The first one, coined out in 2006 by [5] defines it as: *"Cyber-Physical Systems are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa."*. This definition is accurate if it looked at with a 2006 perspective, however there have been published new definitions.

[6] defines it as cooperative systems with decentralised control, merging the physical and virtual world while operating autonomously only dependant on the context they are set in.

A broader concept was conceived by [7]: *"Cyber-Physical Systems (CPS) are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the internet."*.

A cyber-physical production system is the implementation of a CPS specifically in a manufacturing environment.

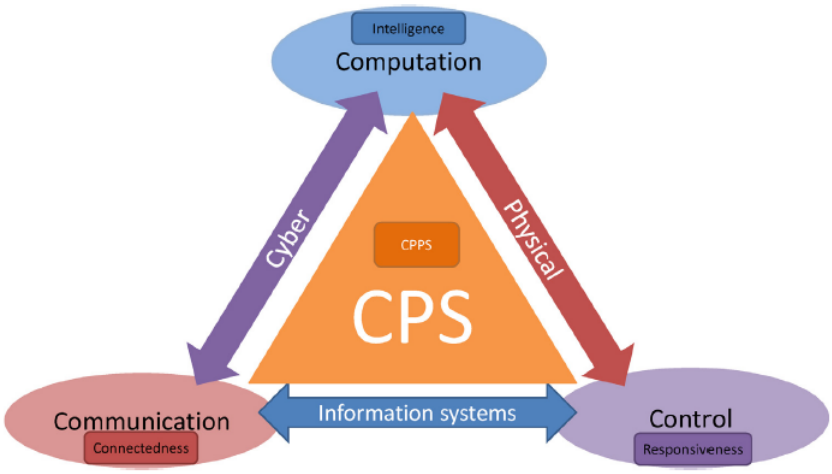

Figure 2.2: The three main characteristics of a CPPS.[8]

In Figure 2.2, the three main characteristics a CPPS has to have,according to [9], are presented.

- Intelligence, the elements present in the system must be able to acquire information from their surroundings and use it to operate autonomously.
- Connectedness, the elements taking part in the system must be capable to connect and communicate with their counterparts, which includes human beings and the services they take part in the Internet, for the sake of cooperation and collaboration.
- Responsiveness to changes, given external or internal stimuli, the CPPS must be able to respond to those changes and adapt to what those changes request.

One of the primary objectives of a CPPS is to abstract and interoperate different modules in a heterogeneous environment. To achieve that, Plattform Industrie 4.0 developed the Reference Architecture Model 4.0 (RAMI 4.0) [10]. This new model presents a structured manner to approach the development of distributed systems.

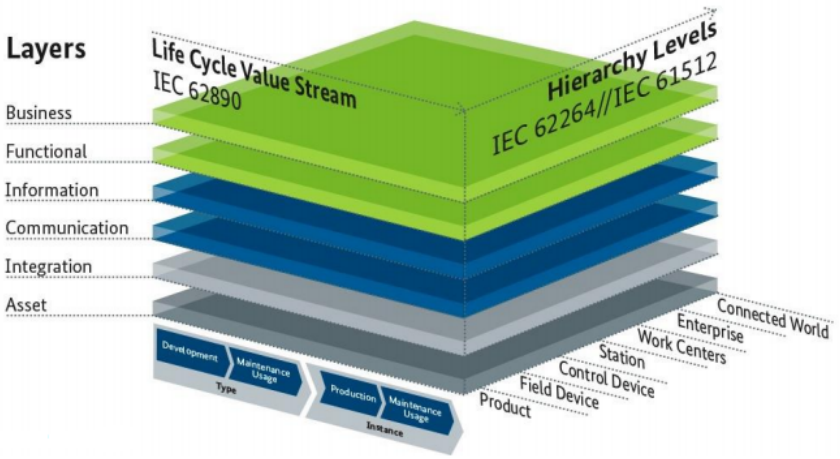

Figure 2.3: Reference Architecture Model for Industrie 4.0. [10]

RAMI 4.0 is a service-oriented architecture that combines all the IT components and elements present in a layer and life cycle model.[11]

As presented in Figure 2.3, this model consists of three axes[11]:

- Hierarchy levels. This horizontal axis contains all the flexible systems and machines with functions that are distributed throughout the network. These components can interact and communicate across all hierarchy levels.
- Product life cycle and value stream. This layer represents the life cycle of the components, be it physical resources, products or the facility.
- Layers. This axis represents the degree of abstraction. The *business layer* has functions that map,regulate and rule the business processes. The *functional layer* has functions that formalise the model and integrate services. The *Information layer* has functions that pre-process events, execute rules, analyse data and assure the quality. The *communication layer* supports the communication between devices. The *integration layer* provide asset information and enables control and component virtualisation. The *Asset Layer* has fuctions that are performed by a specific component.[12]

The CPPS together with I4.0 has enabled the creation of several production paradigms which aim is to adapt to the market demand. These Emergent Production Paradigms will be addressed in the next section.

2.3 Emergent Production Paradigms

2.3.1 Flexible Manufacturing Systems

Due to the high demand of customised products, the manufacturing market has suffered from high fluctuations of product demand.[13] This leads to shorter product life cycles which increases the number of new models being produced,thus creating a volatile market. Taking the consumer satisfaction into consideration, it is clear that flexibility is a vital characteristic in manufacturing systems.

"The flexibility of a manufacturing system is determined by its sensitivity to change."[14] The lower the sensitivity, the higher its flexibility.

According to Chryssolouris in [15], there are three main forms of flexibility.

- Product flexibility, this is the ability a system has that enables it to make different product parts with the same equipment.
- Capacity flexibility, it is the ability of a system to adapt to volume demand changes while remaining profitable. This reflects the ability of expanding or contracting easily.
- Operation flexibility, it is the ability to be able to produce different set of products using different machines and operations. It implies that a factory can make different types of products without having to stop the production to apply changes to the manufacturing line.

The ORCA-FMS is an architecture applied to FMS. It is a dynamic architecture having two functioning modes, normal and disrupted. If there is a perturbation in the system, the disrupted mode is activated thus enabling optimizers to fix the disruption. ORCA was tested in a environment with a loading/unloading machine, three assembly machines and an automated inspection machine. This architecture can be used in system like health care and logistics since it is focused on scheduling. [16]

2.3.2 Bionic Manufacturing Systems

Due to the increasing demand of customised products, there are three basic properties to be provided in a Bionic Manufacturing Systems. Those are *autonomous distribution,self-growth and harmonised integration*. Concurrent properties for time and spatial localised properties are very important points in BMS because these properties enhance the flexibility of the system. In [17], the author notes that the basic concept of approaching to BMS is learning biological organisms and how to coexist with them. In a manufacturing approach, it means the need to harmonise with people and machines and locate the manufacturing system in the global ecosystem.

In [17], the author coins out the concept of DNA-based BMS. This concept aims to enhance the flexibility and autonomy by integrating substance and information. Each

component of the manufacturing system, tools, robots,etc are imitated by autonomous organism with DNA-type information. Furthermore, the product is regarded as a growing substance, from raw material to final products. By spreading evenly the information on the machine or system side, the system is highly versatile and can deal with abnormal situations.

The BIOSOARM is a BMS based architecture created for highly dynamic manufacturing environments. In this architecture, mating is used as an example given that animals try to attract their counterparts to reproduce. In BIOSOARM resources attract different parts that after the execution phase generate a new part. This is repeated until the final product is achieved. There are male entities that adapt their behaviour to be the most attractable to female entities that move to the most attractive male. The male entities are resources in the manufacturing domain while the part and transportation are the female entities.[18]

2.3.3 Holonic Manufacturing Systems

This paradigm is derived from the concept of Holon, which was coined by the philosopher Arthur Koestler. He defined a holon as being a whole consisted by parts and a part which is a part of a whole. In a manufacturing system, a holon consists of a physical processing part and an information processing part. It is autonomous and cooperative inside a system, being able to transform, transport and validate information. To fully understand what a Holonic Manufacturing System is, the following concepts were created by the HMS consortium.[19]

- Autonomy: A holon has to be able to create and control the execution of its own plans.
- Cooperation: Multiple holons have to be able to work towards the same plan.
- Holarchy: Multiple holons cooperate to achieve the same goal or objective, thus this concept depends on the two above-mentioned.

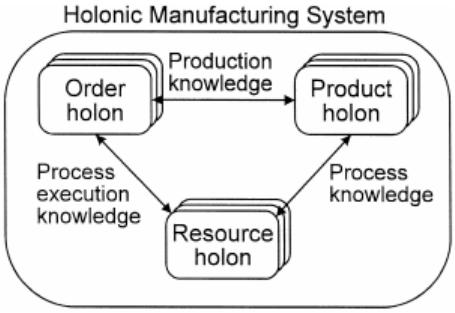

Figure 2.4: PROSA: reference architecture.[20]

According to [20], the typical structure of a HMS (PROSA) is built around three types of holons. The order holons,which are responsible for the logistics of the system, the

product holons,which are responsible for the technological planning, and the resource holons which are responsible for the resource capabilities.

A resource holon contains a physical part, a production resource of the system and an information processing part for the control of the resource. Due to the prior concepts, the holon offers its functionalities to the surrounding holons.

A product holon is responsible to ensure the quality of a product. The holon contains up-to-the-minute information on the product, such as, requirements, design, process plans, materials and so on. This holon acts like the information server to the other holons in the HMS.

An order holon is responsible for performing the tasks on time. It manages the products and all the logistic information. It can be tasked to repair resources.

As it can be seen in Figure 2.4, the holons exchange knowledge between one another. Order and resource holons exchange process execution knowledge,resource and product holons exchange process knowledge and product and order holons exchange production knowledge.

Process execution knowledge contains the information and methods in order for the execution process to progress.It is knowledge on how to manage the production process, such as how to start, stop,interrupt,etc.

Production knowledge has the information on how to produce the products. It is knowledge on how the production process is being executed, the sequence, the structures, how to access information of the process plan,etc.

Process knowledge contains the information and methods on how to perform processes on resources. It is knowledge about the resource, what it can perform and the process quality.

The ADAptive holonic COntrol aRchitecture (ADACOR) is an architecture built upon a set of autonomous and cooperative holons. Its aim is to support the distribution of skills and knowledge and to improve the adaptive capability in a environment change. The holons are representations of physical resources or logic entities. ADACOR holons are aware of the environment and they react to changes in it, even though their behaviour is mostly reactive, the holons still take initiative in elaborating product plans or predicting future occurrences of disturbances. This architecture defines four holons, product, task, operational and supervisor. The product holon, task holon and operational holon are similar to the product, order and resource holons above-mentioned due to being inspired in the PROSA architecture. The supervisor holon is in charge of coordination and global optimisation and is also responsible for the formation and coordination of groups of holons.[21]

The ADACOR2 is another holonic architecture that aims to enhance the above mentioned ADACOR architecture by implementing self-organisation and chaos principles in order to achieve an evolvable system. This architecture, at a micro level, focuses on the behavioural self-organisation. To achieve this, each holon is permanently monitoring its state and the environment, looking for an opportunity to evolve or waiting for an external

evolution trigger. Depending on the trigger type, the holon self-organises by selecting the right behaviour from set behaviours. Given this, the ADACOR2 self-organisation can be defined as *"the change in the internal state of the holon using a set of internal rules and mechanisms, triggered in response to a plan deviation or a new evolution opportunity, with the aim of reestablishing normal functioning or improving performance."*[22]

At a macro level the focus of this architecture is structural self-organisation. In the presence of a change, the holons have the means to re-arrange their relationships and their organisation. In ADACOR2, structural self-organisation is defined as *"'the change in the relationships between holons, and consequently the change in the holarchy structure, which is triggered in response to a deep impact plan deviation, thus promoting a drastic response that aims to re-establish normal system functioning or improve its performance"*. There are three levels of re-organisation:[22]

- Level 0 (emergence): Due to self-organisation, the relations between the holons change.
- Level 1 (logical structural self-organisation): Each holon is trying to optimise its place in the holarchy. Due to this constant optimisation, it is possible for the holon to change holarchy, to belong to several holarchies at the same time or to be a freelancer and work autonomously.
- Level 2 (physical structural re-organisation): In this level, the holons not only change their relations with other holons but they physically change their place aswell.

The ADACOR2 holons act as swarms, constantly trying to build a cohesive group, maintaining distances and imposing crow management.It was introduced a central authority to increase the optimisation levels of the architecture.

The CRIO (Capacity - Role - Interaction . Organisation) model is a framework to design holonic multiagent systems. This model is based on four main concepts:[23]

- Capacity: A capacity is a description of a service. The description contains a name to identify the capacity and the input and outputs variables.
- Organisation: It is a set of roles and their interactions pattern to define an organisation in a domain.
- Role: A role is the abstraction of a behaviour and confers a status within the organisation.
- Interaction: An interaction links two roles in which an action in one causes a reaction in the other.

The NCRIO is a normative extension of CRIO. In this framework, there are *internal norms* and *external norms*.The internal norms are in charge of the internal functioning of a

holon. These norms address the protection of private data, the management of databases and the confidence of the data transmitted and received. The internal norms are used to guide and control the performance of actions in a specific context. It can forbid, obligate or permit a behaviour of a holon. The external norms are in charge of the actions between holons. These norms publish the capabilities and functionalities of each holon allowing them to interact.[24]

2.3.4 Reconfigurable Manufacturing Systems

In the era of globalisation, the call for customised products are on the rise, therefore the need of changing the production line rapidly and with a low cost is a requisite.

A Reconfigurable Manufacturing System is designed when the product demand calls for a change in software or hardware components so it can rapidly adjust to the new products.[25].A RMS should feature the following six features[26]:

Modularity, which is having different operational functions compartmentalised into units in order to be alternated in between different production schemes for optimisation. *Integrability*, it is the ability to connect above-mentioned modules rapidly by a set of interfaces in order to facilitate communication and integration. *Diagnosibility*, the system should be able to assess its current state to detect and diagnose possible causes for product defects, thus correcting them. *Convertibility*, it is the ability to change its own functionality to be able to join a new production system. *Customisation* has two main aspects, customised control, which is obtained through the integration of modules to provide the actions needed, and customised flexibility, which is building the machines around family parts that only provides the flexibility needed to produce specific parts. *Scalability*, the ability to modify the production capacity through adding or removing resources.

The core component of a RMS is the Reconfigurable machine tool. These tools possess several modules. The basic modules are things such as, base and slide ways. The auxiliary modules are the tools in themselves, such as spindle heads and tool changers. Due to these modules, machines can easily change their configurations by adding, changing or removing their modules.[27]

In [28], RMS were used to produce Load-Haul-Dump machines. These machines are a combination of loading machines and dump trucks. Both machines have been used to exploit mineral deposits but at a high cost. The Load-Haul-Dump machines were created by putting the other two machines in a module that could be attached to a single vehicle. This means that each module could vary in size depending on the customer without needing to buy a different vehicle in a whole.

2.3.5 Evolvable Production Systems

The Evolvable Production System paradigm emerged to fill the gap that paradigms like HMS had left untouched. According to [29], the EPS is the only paradigm that achieves

fine granularity. *Granularity* is the level of complexity of the component in a manufacturing system. If there are cells which are modules that can be plugged, that is called *thick granularity*. However, if what can be plugged in and out are sensors, or grippers, etc, that is the *fine granularity*.

Change is what drives EPS in the evolution path, not the already known scenarios. This paradigm focus on the unpredictable changes that may occur within a very limited product range, thus creating a very specific and limited first solution. However, if successful, it applies its solution to other products. Hence, the name Evolvable, it doesn't create a generic solution but a specific approach that can be adopted by other products in the line.[29]

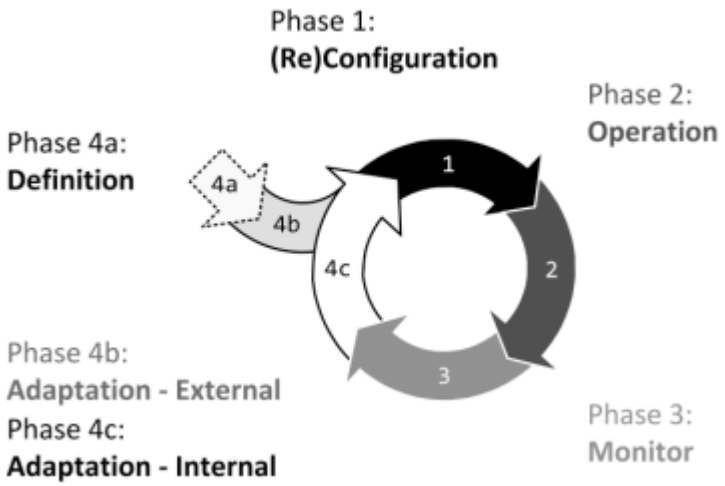

Figure 2.5: The four phases of an EPS.[30]

This paradigm is a four-phase cycle[30]. The first phase is *(Re)Configuration*, it represents either the first configuration of a production line or its subsequent reconfigurations. The reconfigurations can be either physical or software based changes.

The second phase is *Operation*, it represents the execution of the production line. Its configuration and the resources are fixed during this phase.

The third phase, *Monitor*, occurs while in Operation. This phase is focused in recording information about the production line. This information is then processed and its output can be used in three different ways. Firstly, it can be used to inform the system operators of the current system state and the performance of the production line. Secondly, to record relevant details of the operations performed on quality control. Thirdly, to record the system's performance for use in the next phase, the *Adaption*.

The fourth phase is Adaption, and it consists of three components. The *Definition*, in this part, external evolutionary pressures are translated into a language utilised by the system. These external pressures might be the requirements of the products, the need to change the capabilities of the resources or to change the priorities in the system. The second component is *Adaptation-External*, it is in this phase that is planned the response

to external stimuli defined in the Definition phase. This component aims to produce a set of reconfiguration changes in order to respond to the pressure. The third component is *Adaptation-Internal*. This component handles the information gathered in the Monitor phase, if there is enough pressure to apply a change, the system will recommend it. The recommended changes are fed back to the (Re)Configuration phase for the user to deploy.

In [31], it is said that a EPS targets the following four points:

- Optimised functionality. This point focuses in keeping the equipment as simple as possible to facilitate the connection between cells or systems.
- Optimised orchestration, by using a multi-agent based and distributed approach the control system is agile.
- Adaptability, by being modular, the system is highly upgradeable while being economic. This enables the system to adapt to minor changes.
- Robustness, the equipment is dedicated to its own function. Some modules can be reconfigurable. Due to the control system being goal-oriented, this results in a dedicated system based on an adaptable concept.

The IDEAS architecture has a Human Machine Interface that is linked to controllers. At the bottom of the control infrastructure, the controllers and production components are linked via Input/Output connections or a computer network. This architecture is divided in the Human Machine Interface layer, a high-level multiagent system and the hardware communication layer.

To allow flexibility, it is mandatory that agent share information and interact with one another in order to change configuration according to the needs. When a new resource is detected, a message is sent to the agent that is in charge of the sub system. When that message is received, the agent initiates a verification to check if the new resource is already abstracted. If it does not exit, a new message is sent to create that new resource abstraction. After creating this, the agent responsible for the new tool announces its presence, thus being available to use whenever needed.[32]

The Coalition Based Approach for Shop Floor Agility(CoBASA) is a multiagent based EPS architecture. To create a coalition, the user has to choose depending on the logical structure that is to be created. After creating them, in the operation phase the coalitions execute the skills due to produce the product. If an agent does not answer of refuse to address a request, it is removed from the coalition. Whenever a change is due, the coalitions are to change their organisation to meet the needs of the factory. A coalition can be dissolved if the system is dismantled or if it is being reengineered.[33]

2.4 General Conclusions

Given the extensive demand of ever-changing products, the factories have struggled to keep up with the customers needs. The Industry 4.0 was a concept created facilitate interoperability between the tools in the shop-floor level of a factory.By linking Cyber Physical Systems with manufacturing lines with the help of the Internet of Things, Cyber Physical Production Systems were created enabling possible solutions to the problem that is customer demand.

As it can be seen in the literature review, in order to create a solution to this problem, several paradigms emerged. However, these paradigms are heavily dependent on computational power that most of the devices on the shop-floor level can not handle. Thus, the objective of this thesis is to bridge the gap between these paradigms and the hardware limitations in order to have a working I4.0 factory.

CHAPTER 3

SUPPORTING CONCEPTS

This chapter presents the supporting concepts on topics such as, Cloud and Fog Computing. Afterwards, it presents the concept of MultiAgent System and several Industrial Protocols. These protocols are used to communicate between edge devices and software.

3.1 Cloud Manufacturing

The term Cloud Manufacturing was formally introduced in 2010, even though the concept of Manufacturing as a service had already been created in 1990. That concept was set back by the initial state the technology was at at that point. CMfg through different contributions, is defined as: *"Cloud Manufacturing refers to the Service oriented Model, infrastructure as a service, with the goal of Manufacturing as a Service, which is grounded in Cloud Computing. It offers a ubiquitous, on-demand network access to a shared pool of manufacturing resources and capabilities by provisioning of manufacturing resources, services and capabilities through virtualisation."*[34]

The core idea of CMfg is to provide manufacturing resources as services, by linking the consumers' needs with the resource providers, enabling them to fulfil those needs during the product life cycle. The key characteristics of CMfg are[34]: *Networked environment and collaboration among users*, *Service and requirement orientation*, *Interoperability among systems*, *Effective realisation of intelligence by Knowledge and Data*, *Virtualisation principle* and *Scalability*.

According to [35], an important goal of cloud manufacturing is to provide on demand services for the resources and capabilities needed through the internet. Manufacturing enterprises have to deal with large amount of different resources. After virtualising and modelling these distributed and heterogeneous resources, it is needed to efficiently

manage and coordinate services in a centralised way to guarantee the best quality, performance and security. In order to overcome the complexity of the manufacturing system, multi-agent approaches are being used to support collaborative management of resources and services.

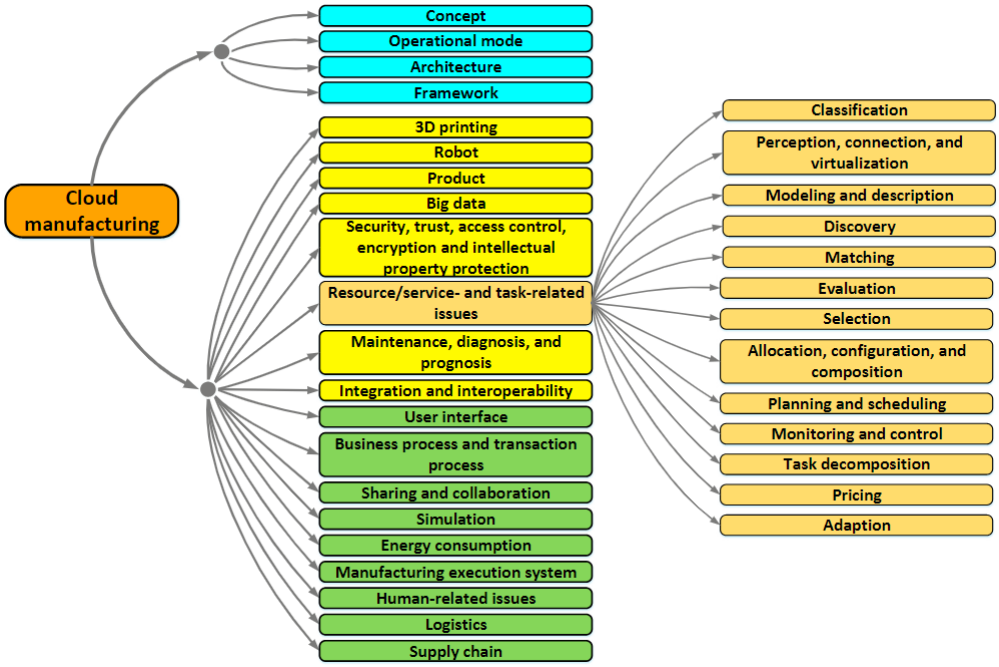

Figure 3.1: Research topics on cloud manufacturing.[36]

Research on CMfg has covered diverse topics. In Figure 6, there are listed some topics which are being researched under this concept. In [36], the author reviews these topics. In the context of *Robot*, it is being developed robot control-as-a-service and cloud robotics. Its objective is to try and enable technologies for a distributed control approach for assembly tasks. Under *Product*, some researches include customisation, configuration and planning.

In *Resource/Service - and task-related issues*, there are several topics being researched. *Resources Classification* is due to the amount of different resources present in a manufacturing system, thus to efficiently process information, it is needed a good classification to each resource. In *Modeling and description*, its aim is to solve data consistency problems. *Evaluation*'s objective is to create a quality and capability assessment due to its important role in cloud manufacturing. In *Selection*, it is being researched a process to determine services that suit the task at hand. In *Allocation,configuration and composition*, Allocation wise, it is a priority to create a procedure to meet on-demand services, there has been several proposals, such as, energy-aware allocation and virtual resource allocation. In service composition, most approaches encompass Quality of Service aware methods. In *Monitoring and Control*, it has been discussed remote monitoring to determine resource availability. It has been introduced real-time status monitoring and the use of intelligent

and distributable Function Blocks decision modules.

In the topic of *Integration and interoperability*, given the different amount of services being provided and due to the fact that these services are developed in different platforms, interoperability is one key factor that has to be addressed. Therefore, different researchers are developing frameworks to address this issue.

3.2 Fog Computing

Cloud computing is the most prominent paradigm in the present day, however, its concept conflicts with the principles imposed by Industry 4.0. The conflict lies in the lack of a decentralised structure and real-time control. The Cloud services can support these characteristics,but in the end, it is all stored in the cloud server. The distributed agents all depend on a connection to the cloud.[37]

Given these problems, the Fog Computing paradigm was created. This paradigm is a highly virtualised platform that enhances devices with computation,storage and connection services, to the cloud. These resources are the building blocks of both fog and cloud computing.

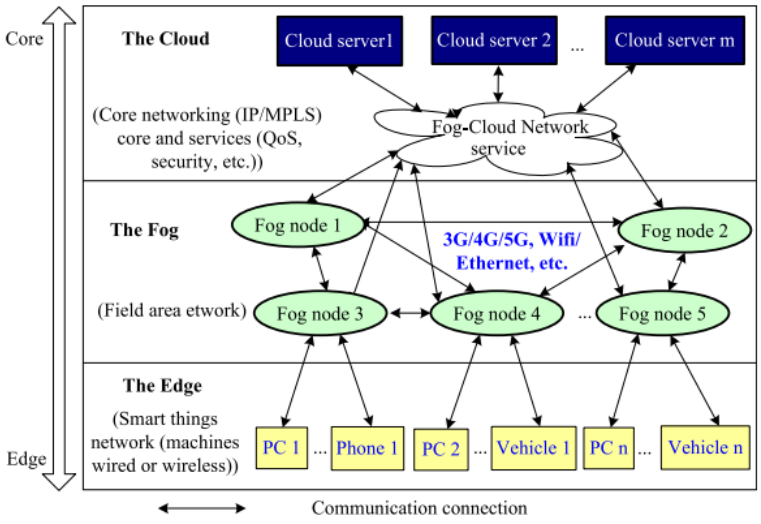

Figure 3.2: Fog computing architecture.[38]

The fog network is composed by interconnected nodes. There are 5 different type of nodes.[39] The *Servers*, which are geo-distributed and deployed at common places, such as bus terminals, parks. These servers are virtualised and able to storage, compute and network with other equipment. These nodes enhance the computation and storage capacity of the edge components in this paradigm. The *Networking Devices*, devices like routers and switches act as an infrastructure for Fog computing. The *Cloudlets*, these nodes are viewed as micro-clouds and extend Cloud based services. The *Base Stations*, these nodes exist to create a seamless communication between wireless networks. The

Vehicles with computation facilities can work as Fog nodes, they create a distributed and scalable Fog environment.

These nodes have three collaboration techniques.[39]*Cluster*, by forming cluster among themselves, the nodes create a collaborative environment. These clusters can be formed based on location or the activity performed. *Peer to Peer*, this type of collaboration can be hierarchical or in a flat order. Based on proximity, this communication can be classified as home, local or non-local. *Master-Slave*, in this type of collaboration, the master controls all the functionalities, resource management and data flow of underlying slave nodes.

In [40], several characteristic of Fog Computing were defined.

- Geographical distribution. In contrast to the cloud, in Fog computing, services are to be widely distributed.
- Large-scale sensor networks in order to monitor the environment.
- Fog applications must communicate directly with mobile devices, thus supporting mobility techniques.
- Real-time interactions.
- Predominance of wireless access.
- Heterogeneity. The nodes come in different forms and are deployed in different environments.
- Interoperability. For seamless communication, cooperation is required from the providers, which leads to the need to interoperate between different components.
- Support for on-online analytic and interplay with the cloud. Depending on the amount of information the node has to process, the computation work should be sent to the cloud.

The Fog computing paradigm saves bandwidth and can increase computing speed because the amount of data needed to be moved to a remote server decreases. Therefore, by working with devices at the edge of the network, it guarantees low latency and it focuses on location awareness.[41]

3.3 Multiagent Systems

In the Literature Review, it could be seen that the current production paradigms require a decentralised control and autonomy. To tackle that issue, several architectures were coined out based on these systems. In [42], a multiagent system is defined as: *"is a complex system consisting of multiple intelligent agents which can operate themselves independently, yet can still act in a consistent way due to the information connections among agents."* These agents are defined as: *"An autonomous component that represents physical or logical objects*

in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it does not possess knowledge and skills to reach alone its objectives."[43]

In [33], some characteristics that are considered important are listed:

- Autonomy - This characteristic is viewed as the most important because it gives the agent control of its own without needing external intervention.
- Reactivity - It is the capability to react to changes in the environment.
- Proactiveness - This represents the ability to take the initiative towards the goal it is set for the agent and not just react to outside stimuli.
- Social Ability - This is the ability to interact with other agents which can have different designs and objectives.
- Adaptability - The capability to adapt to the environment.
- Continuity - The agent should be a long-term addition to the environment.
- Mobility - Some agents should be capable to change environment without losing their internal status and knowledge.

To implement agent-based manufacturing systems, there are key issues that must be addressed. In [44], the author mentions some of these issues.

In *Agent encapsulation*, there are two different approaches to the issue, functional decomposition and physical decomposition. In the *functional decomposition*, the agent represent modules, such as order acquisition, process planning, scheduling, transportation management and product distribution which have no relationship with physical entities. The issue in this approach is the amount of state variables that the agents need to share, often leading to consistency problems or unintended interactions. In the *physical decomposition* approach, agents represent physical entities, such as operators, machines, tools, products and parts. In this case, there is an explicit relationship between the agent and the entity.

Another issue mentioned is the *Agent Organisation*, there are three different approaches to take into consideration in these systems. The *Hierarchical approach*, the *Federation approach* and the *Autonomous Agent approach*.

A *hierarchical approach* can be seen in a typical manufacturing enterprise where a number of semi-autonomous units are physically distributed, each with a degree of control over local resources. Albeit the centralised appearance, it is still used this approach.

In the *Federation approach*, some variations have been created, Facilitators, Brokers and Mediators. In the Facilitator approach, agents are combined into a group and the communication between those agents are through an interface called facilitator. The facilitator is in charge of providing an intermedium between local agents and remote agents. The Brokers approach is similar to the facilitator, the main difference between

the two is, while the facilitator is in charge of a designated group, the broken can be contacted by any agent in order to find the service agents to complete a task. The Mediator approach, in addition to the facilitator and the broken, it assumes the role of system coordinator, promoting cooperation between agents and learning from the agents. Federation architectures aim to ensure the stability of the system, provide scalability and reduce overheads.

The *Autonomous Agent approach* has a set of characteristics according to [44]. The agent is not controlled or managed by any other agent or human being; It can communicate/interact directly with any other agent and with external systems; It has knowlodge about other agents and its own environment; It has its own goals and a set of motivations. This architecture is useful to develop complex and dynamics systems that are composed by a large number of resource agents.

Agent negotiation is another issue to be addressed. Negotiation is a method commonly used in agent-based manufacturing systems. The negotitation protocols include a voting mechanism, Contract Net or its modified versions, such as Extended Contract Net, Market-Driven contract net, etc.

System dynamics is an issue due to the highly dynamic nature of a manufacturing environment. To address this, it can be used a dynamic scheduling to adapt to situations such as insertions and machine failures.

Another issue,*Tools and standards* has to be addressed because to develop agent technology, it depends on the availability of the development tools and platforms. In return, these tools usually have a standard of functionality. The Foundation for Intelligent Physical Agents (FIPA) has created a set of specifications that agents have to comply to. There are also three well-known agent platforms: JADE, ZEUS and FIPA-OS.

ADACOR[21] is a holonic architecture which concepts were implemented by a multiagent system. The ADACOR prototype was developed in Java Agent Development Framework (JADE). Each holon is an agent in this framework.

In [45], the author introduces the notion of skill in an agent based architecture. A skill includes an interface that contains all the information for an agent to execute it. This information is, the skill identification, the parameters and their types, a reference to the agent that will execute it and variables to control its execution and type. There are two types of skills, *Atomic skills* (ASk) and *Complex skills*(CSk). The ASk use I/O mappings and other system integration details to associate the skill information with the execution of the skill in a specific controller. The CSk implement processes. They support work-flow like description that comprises other skills and define if the execution is to be parallel or sequential.

In CoBASA[33] architecture there are three basic control agents:

- Manufacturing Resource Agent (MRA) - This agent abstracts the shopfloor entities that cannot be further decomposed as lower order agents. The functions these

entities perform are atomic. The MRA implement all the control logic to handle an ASk.

- Coalition Leader Agent (CLA) - This agent abstracts compositions of CLA's and MRA's by hosting complex skills. Therefore, each CLA is able to process CSk by managing their execution flows.
- Product Agent (PA) - This agent is the top level agent in the COBASA architecture and it manages its production plan and it is responsible to optimise its time on system by interacting with the transport agents.

The IDEAS Mechatronic Multiagent Architecture (IMMA) is a multiagent based architecture applied to an EPS.[46] In this architecture, all the agents are extensions of the Mechatronics Agent (MA) Concept in order to facilitate self-organisation since the MA gathers the main descriptive attributes of all types of agents in the system. Taking this into consideration, the architecture provides adequate support to the interoperability, such as:

- Functionality Representation - Agent's functionalities are viewed as skills as mentioned above.
- Yellow Pages Service Interaction - the MA provides the description for each agent, making public and discoverable the available skills.
- Messaging - the MA implements two FIPA communication protocols. The FIPA request to be used during skill execution and the FIPA Contract Net for skill execution negotation.

3.4 Industrial Protocols

In order to keep up to the demanding and ever changing market,an improvement in integration and interoperability between devices was necessary, several Industrial Protocols have been trying to address this issue.

3.4.1 OPC-UA

The OPC-UA communication protocols is a bridge between business applications and automation hardware. OPC-UA introduces the object oriented paradigm to the address space, thus the objects being able to be defined as variable and methods. Every object in this protocol is represented as a node in the address space. A node can be from eight different classes, object,variable,method,view,objectType,VariableType, referenceType and dataType.[47]

In [48], the author explains the impact OPC-UA has in CPPS. OPC-UA was named a candidate for communication aspects in RAMI4.0 due to its use in control and monitoring

but also for the increase in device connectivity it provides via standard communication. This approach abstracts data from the network and software application and offers a generic communication interface. It is considered as one of the key technologies for transparent data representation between heterogeneous system components. However OPC-UA deals with more than simple data access, it takes into account other aspects such as, security, reliability, access control or historical data. These aspects show some building blocks which OPC-UA can carry out. These building blocks are as follows:

- Separation of information model and data base structure: Hiding hierarchical data structures behind object-oriented models in OPC-UA in order to be independent of the fixed structure of the data base.
- Minimal OPC-UA client functionality: By creating an OPC-UA client, the developers do not have to deal with OPC-UA itself.
- Graphical information model definition: Creating OPC-UA servers with end users based on a graphical representation of OPC-UA models.
- Common info view: Data acquisition from OPC-UA for management view being accessible from different sources, such as web interfaces, that run in parallel to the real-time communication.
- Aggregating OPC-UA server functionality: Having OPC-UA servers that combines different underlying servers and their information.
- AutomationML or other description model as base/data source: The use of external model descriptions as source for information models.

In Figure 3.3 it can be seen the relation between the building blocks and the vertical axis of RAMI4.0.

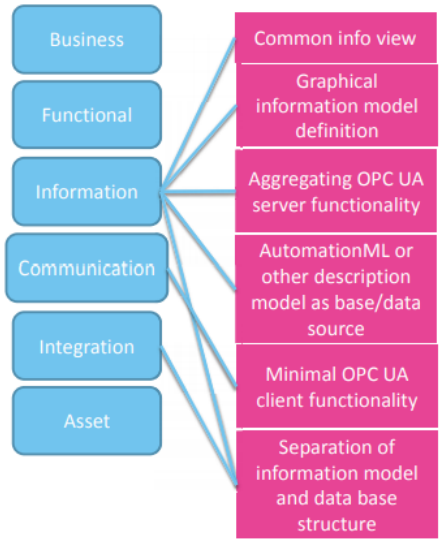

Figure 3.3: OPC-UA building blocks(on the right) relation to the interoperability layers of RAMI4.0(on the left).[48]

3.4.2 Data Distribution Service (DDS)

"The Data Distribution Service (DDS) is a middleware protocol and API standard for data-centric connectivity from the Object Management Group."[49]

The uniqueness in DDS in comparison with other middleware standards is *Data Centricity*, which is ideal for the IoT. DDS knows what data to store and how to share the data. DDS has also a local data storage called the *"global data space"*. Given the fact that this protocol deals with data in motion, each application has local storage, however the fact that there is a permanent exchange of data, there is always access to all information. DDS also provides *Dynamic Discovery* of publishers and subscribers which makes the DDS application extensible. In other words, this means the application does not have to configure endpoints because they are discovered with this method. This method also discovers what the endpoint is doing and the type of data they are using.[49]

In the official document for DDS [50], the purpose of this protocol is described as enabling *"Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time."*

The expected application domains require the DDS to be a high-performance and predictable protocol and efficient in the use of resources. Therefore, the interfaces were design in a way that:

- Middleware can pre-allocate sources in order to reduce dynamic resource allocation to a minimum.
- it should avoid properties that have hard-to-predict resources.
- Minimise the copy of data.

To do this, DDS uses typed interfaces, which are interfaces that take into account actual data types. The advantage to use these interfaces are, their simplicity in use since the programmer can directly manipulate the data; safe to use given the fact that verifications can be performed at compile time; efficient since the execution code relies on the knowledge of the exact data type in advance.

3.4.3 Modbus

Modbus protocol is a messaging structure developed by Modicon in 1979. It is a standard Industrial network protocol to establish master-slave or client-server communication between devices. This protocol is mainly used in master-slave applications in order to monitor and program devices and to communicate between them.

Being TCP/IP protocol the most used communication protocol, the Modbus TCP/IP protocol was created to integrate the factory with the corporate intranet that supports it. Due to the use and knowledge on the TCP/IP protocol, the Modbus TCP/IP is:

- Simple: This protocol takes the regular Modbus instructions and sends it through TCP/IP. The development costs are low and it requires next to none hardware to implement.
- Standard Ethernet: By being TCP/IP, there is no need to have different ethernet cards on the computer in order to use a device.
- Open: Modbus TCP/IP is free to use.
- Availability: Different companies created devices that are compatible with the Modbus protocol.

Due to being an Ethernet protocol, Modbus can transfer data at tremendous speed (60% efficiency) depending on the connection the user has.[51]

Chapter 4

Architecture

In chapter 2, it can be seen that there are several different methods to create a flexible and adjustable system. This thesis proposes and architecture for an integration layer in order to combine a multiagent system with legacy hardware to fulfil the production paradigms that were reviewed.

The proposed architecture used the concepts in chapter 3 to connect to legacy hardware and used the production paradigms as a starting point of developing the multiagent system.

The multiagent system has four different agents, the Deployment Agent (DA), the Transporter Agent (TA), the Resource Agent (RA) and the Product Agent (PA).

These agents are in charge of abstracting the factory. The PA abstracts the product being made, the TA abstracts the movements done by the PA and the RA abstracts the skills that are used to change the product. The DA launches all the agents in the factory by loading them through the data model.

The edge level was created in a modular way. Each action made in the factory is a module in the Controller. That module is launched by reading Flags that were set by the MAS.

In order to connect the MAS with the edge level, a Integration Layer was created. This Layer has an interface with two main methods, executeSkill and executeMove. These methods are called by the respective agents to be able to perform the requested action in the edge level. The TA will call executeMove to be able to move the product in the factory whereas the RA will call for the executeSkill to be able to make the desired change to the product.

It is in the Integration Layer that the system connects to the edge level and where the flags are managed to run each edge level module. Depending on the agent that calls for the executeSkill and executeMove methods, a different flag is set to true and when read

on the edge level will trigger different modules. This way, the connection between the MAS and the edge level is simplified.

This layered architecture main purpose is to bridge the gap between the emergent production paradigms and legacy hardware. This type of hardware is powerful when performing simple tasks, however with these production paradigms, it is no longer possible to keep everything simple. Thus, by creating modules in the edge level, simplifying each task and by managing it through the Integration Layer, it is made possible to use this type of hardware when applied to these new paradigms.

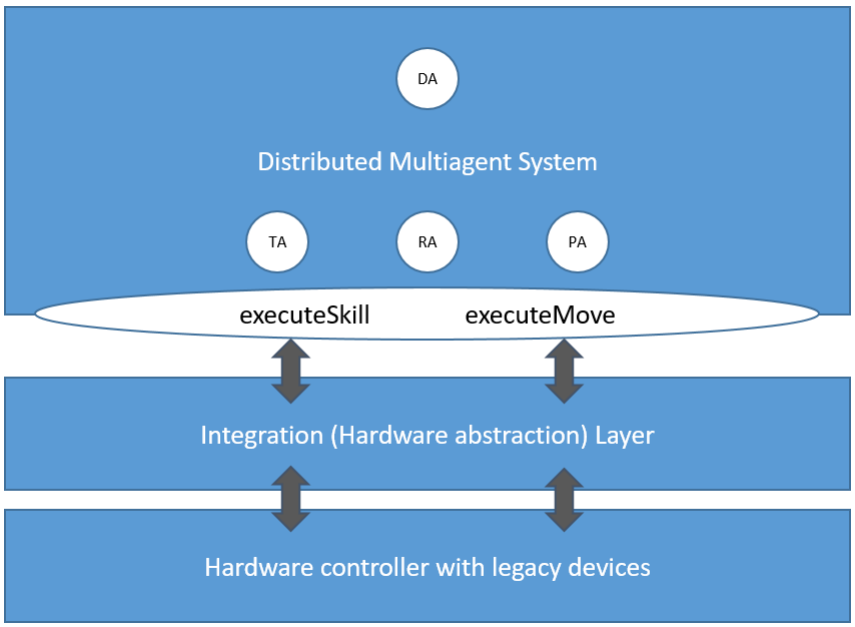

Figure 4.1: Thesis' architecture.

This chapter will be divided in three sections, the multiagent architecture, the legacy systems architecture and the integration layer architecture.

4.1 MultiAgent Architecture

The multiagent architecture is where the agents are setup and communicate between one another.

In the multiagent environment, there are four different types of Agents. These agents are divided according to their role. There are the TA, PA, RA and the DA.

The TA is launched by the DA and the amount of TA's in the system depends on the factory model. After launch, the TA is on standby waiting for a product. Whenever a product arrives in the TA, the agent will ask the following TA if it is occupied or free. If the Agent is free, it will execute the executeMove method from the Integration Layer, if it is occupied, it will wait for the next TA to be free before moving the product. When the Integration Layer is called, the agent will wait for the layer to tell the TA that the product has arrived in the next position.

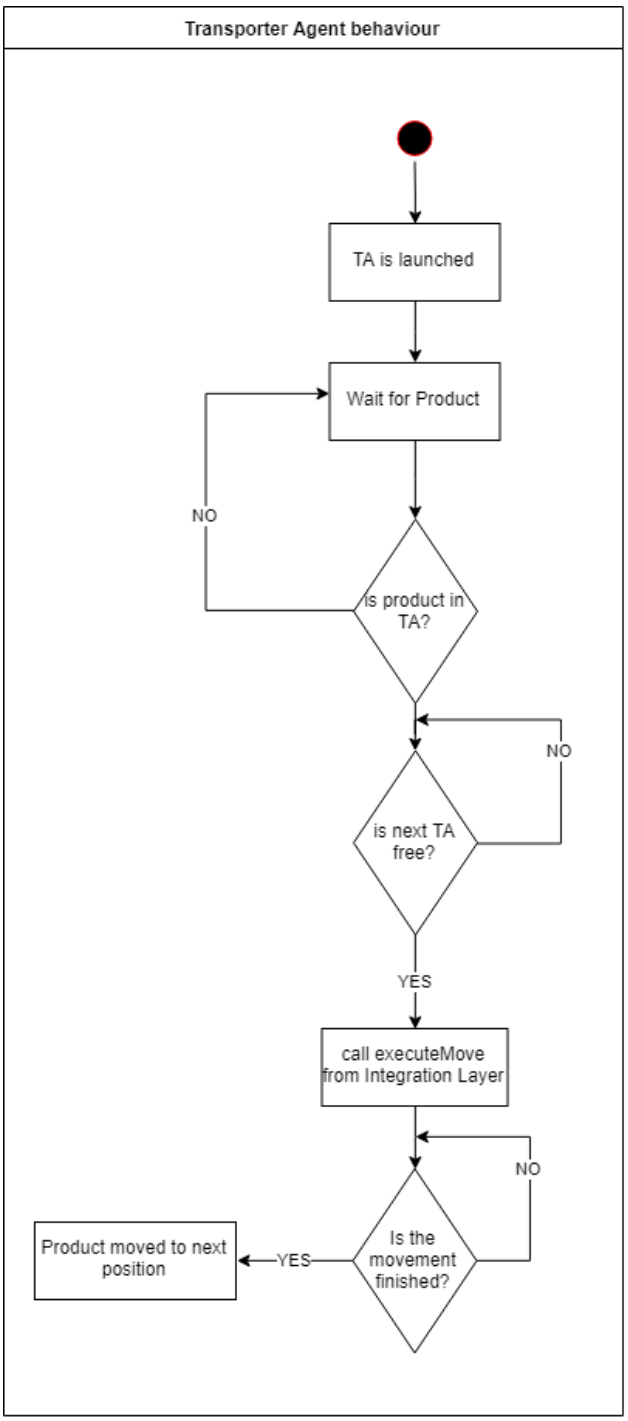

Figure 4.2: Transporter Agent's behaviour.

The RA is launched by the DA, the number of RA's depend on the amount of skills the factory can do. The RA is contacted by the PA to execute a skill, the RA will inform the PA where that skill is. When the product arrives in that skill's location, the RA will call the method executeSkill that was developed in the Integration Layer. When that happens, the RA will wait for the layer to inform that the skill was done. When it does, if there is no more skills due, the PA will move to the exit, else it will repeat the cycle.

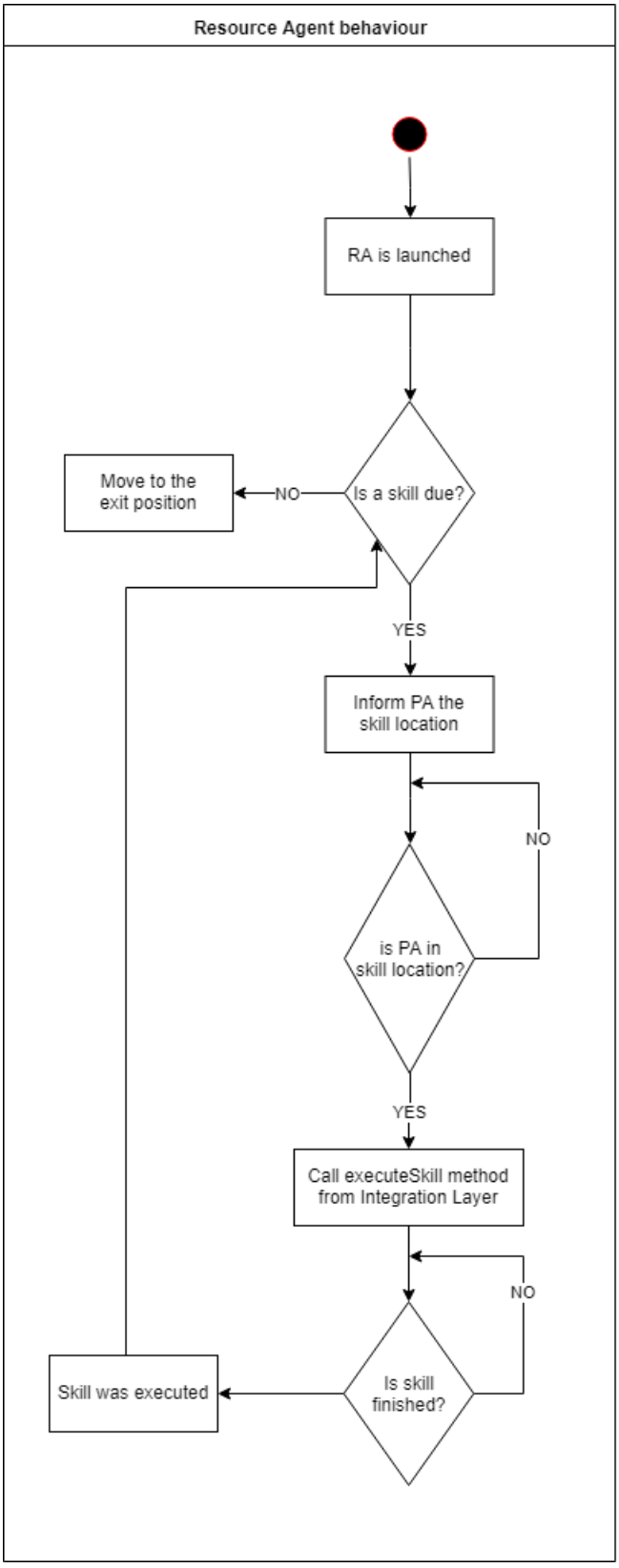

Figure 4.3: Resource Agent's behaviour.

When the PA is launched, it will have skills assigned to itself. Depending on those skills, the product will take its route in the factory. Firstly, the PA will have to know where the skills are, thus contacting the RA. If the product is not in the right position, it will ask the TA to move it to the correct one and then requesting the RA to execute the skill. This cycle repeats itself until there are no more skills.

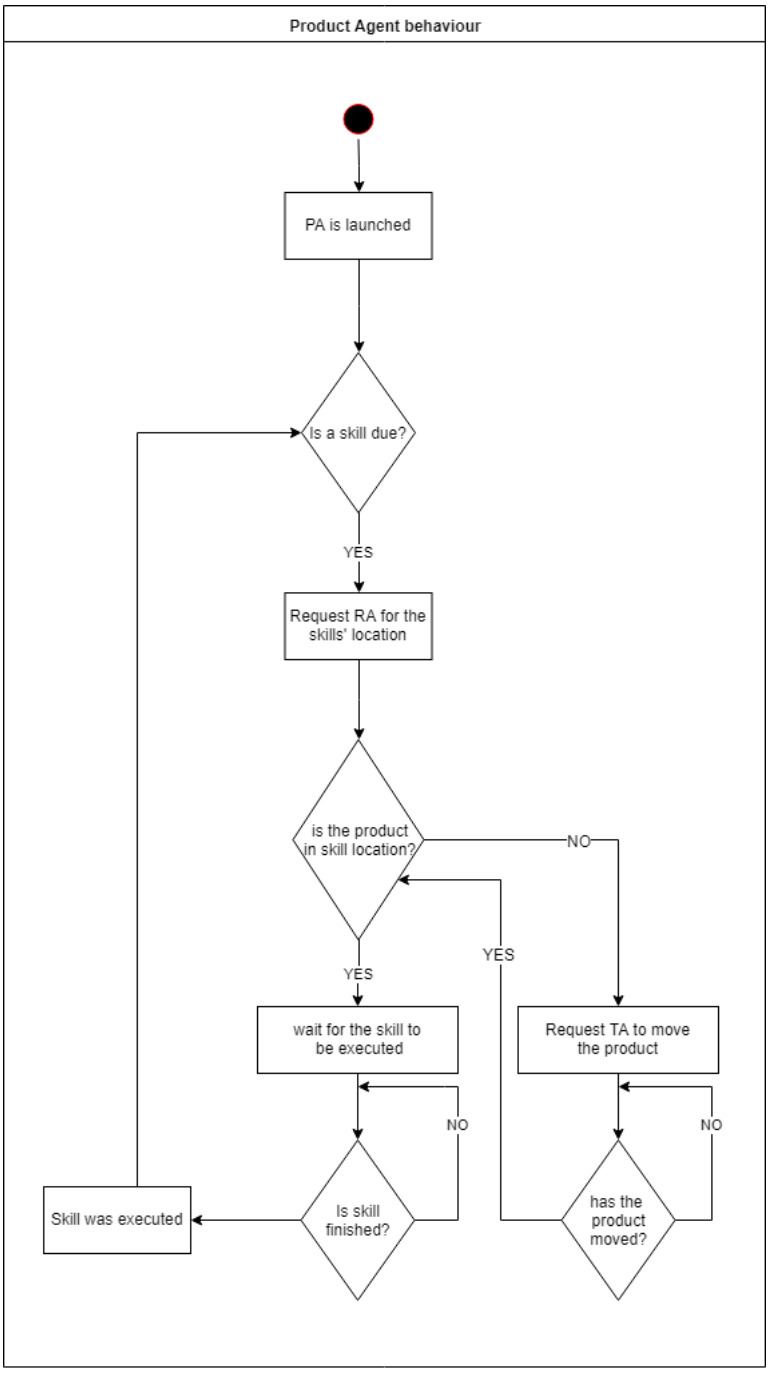

Figure 4.4: Product Agent's behaviour.

In the following table there is a summary of the agents and their roles.

Table 4.1: The system's agents and their role.

Agent	Role
Transporter	This agent moves the product from one station to another and it is also responsible for checking if the next station does not have a product in it.
Resource	This agent is in charge of selecting the skill that will be executed and informing the TA to where it has to move the product.
Product	This agent is the one that request the skills to be done according to its type. It requests from the resource agent the location of the skill and it requests to the TA to be moved to that position.
Deployment	This agent is in charge of launching the agents that are instanced in the data model. It launches the TAs, PAs and RAs.

The agents are in charge of communicating between one another in order to complete the task. Since they are autonomous, they have to figure out what to do after each action. When PA is launched, it has to ask the other agents where the action he wants to perform is, afterwards, the TAs speak among themselves to decide which TA will move PA to the place he requested. The RA is the one that tells the PA where the location of the requested skill is.

4.1.1 Concept of skill

In this system, the agents express their functionalities as skills as defined in [33]. The skills are for an agent like a method for an object. In the architecture there are atomic skills which are directly associated with the manufacturing components' functionalities. These skills include an interface that have the information needed for their execution. In that information there is:

- Name and ID of the skill.
- The name of the agent that will execute it.
- The description of what is executed.
- The parameters and their data types.

4.1.2 Data model

The agents are launch via DA. This agent has the data on all other agents and their properties and interactions. This data is uploaded from the factory's data model. In the data model, every skill, resource and transporter are defined. As well as, defining which transporters are next to the transporter being used. The data model has entities which are the Product, Resource, Skill and Transporter and each entity has instances according to the layout of the factory.

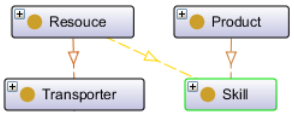

Figure 4.5: Entities present in the architecture.

The data model class diagram is as follows:

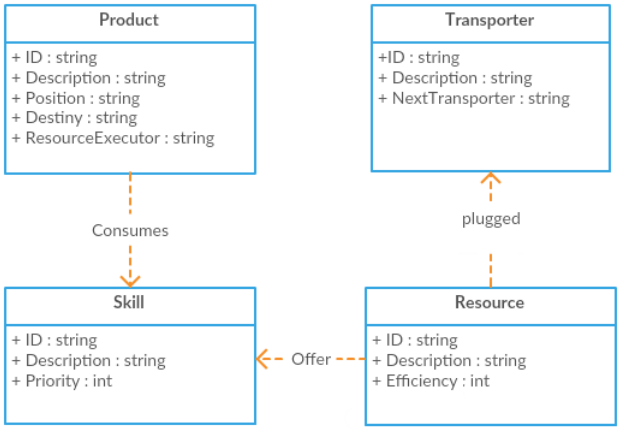

Figure 4.6: Architecture data model.

Figure 4.6 can be translated as Product entity consumes skills. The Resource entity has skills to offer and the Resource is plugged to a Transporter. In terms of the data each entity contains, the Product contains its ID and description, it also contains its actual position, where it is headed and which resource will be executing the skills. The Skill entity has ID and description and its priority as data. The Resource entity, similarly to the skill, has ID and Description and efficiency. Both the priority and efficiency are parameters that define which resource will be executed first and which skill in the resource will be the first to be executed. The Transporter has an ID and Description and it knows what is the Next transporter in the line.

4.1.3 Interactions among agents.

4.1.3.1 Resource Agent

The RA agent communicates with the PA and tells it the location of the skill that the PA wants to execute. This agent has two kind of outputs. It either tells the PA it has to move to a certain station to execute the skill or the PA is in the correct station, the RA executes the skill and tells the PA the skill was completed. The number of RA depends on the factory. A RA is plugged to a station but a RA can have multiple skills.

The way the RA has to communicate with the lower level is through the integration layer developed using the method executeSkill. This way, the action requested by the agent will be performed in the hardware.

4.1.3.2 Transporter Agent

The TA is in charge of all the movement the product makes in the factory. It has 2 different interactions. It can check if the next Transporter is free or it can move the product from one transporter to the following one. Whenever a product is in a transporter, the TA sets itself to occupied while the product is in the transporter.

There is a transporter agent for each movement the product has to make throughout the runtime. This means if there are 5 stations. There is at least one transporter to take the product from station 1 to station 2, another transporter agent to take it from 2 to 3, so on and so forth. Therefore the transporter agent speak with one another to move the product. The transporter with the product has to move forward but to do so it checks if the next transporter is occupied or not.

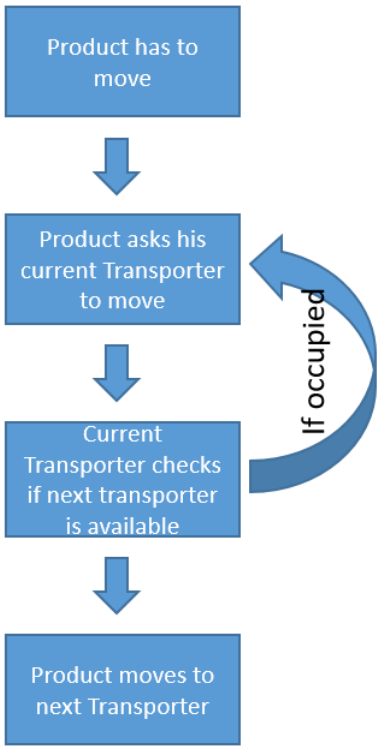

Figure 4.7: Interaction between transporters to ensure the safety of the system.

The method executeMove developed in the integration layer is the way the TA has to operate in the hardware.

4.1.3.3 Product Agent

The PA communicates with the two other agents. The PA starts by communicating with the RA to get a skill to execute. After the RA informs the PA where that skill is located, the PA will order the TA to move him to that location. At arrival, the PA will communicate with the RA to execute the skill.

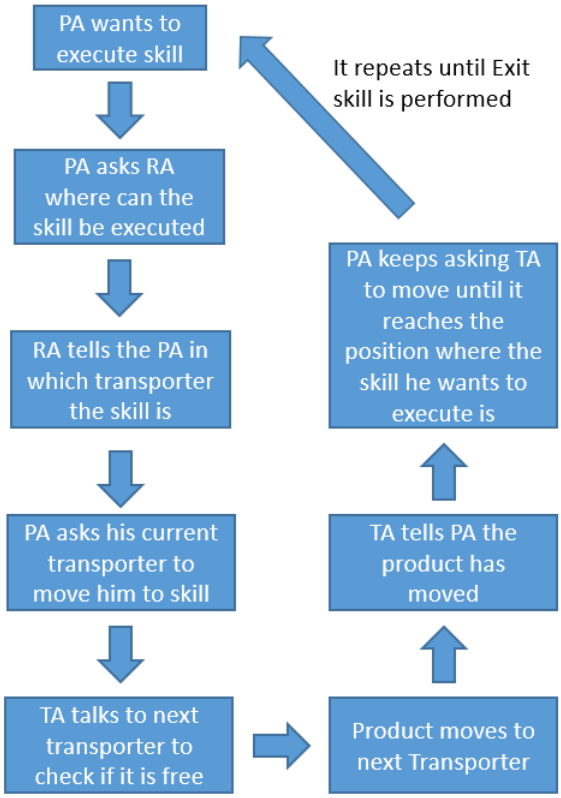

Figure 4.8: Agent interactions.

A standard procedure from the launch to the ending of PA agent is as follows:

- DA launches TA,RA and PA.
- PA looks for the first skill to execute, which is Entry. Then it communicates with the RA to select the best suitor for that skill.
- After knowing the best suitor, it gets its location and asks the TA to move PA to that position.
- Before each move, the TA checks if the next position is occupied. If it is not, it carries on the movement.
- After arriving to the destiny position, it executes the skill associated to the resource in that position.
- When the skill is done, PA looks for the next skill to execute.The RA informs where is the next Destiny.
- After knowing the new destiny, it repeats the movement phase.
- When the product reaches the Exit it removes itself from the system.

4.2 Edge Level

The Edge level is where every action performed by the agents take place. Whether it is a skill or a move, it is all a routine that takes part in the Lower Level.

Every time an agent asks for the Lower Level to execute a routine, a startExecution flag is set to true. When the controller reads it, it starts the routine to perform the requested action. After the routine has ended, the controller sets a finished flag to true. The agent that requested the action which is in standby reads the finished flag and continues its execution.

Each routine has a flag assigned to it and the agents know what flag they have to write and read in order to function. There is a routine for each skill and for each movement. An usual execution of the routine is:

- Agent in Multiagent level sets startExecution flag to true.
- Controller reads startExecution flag.
- The routine associated to the requested execution starts.
- When the routine ends, a finished flag is set to true.
- The agent reads the finished flag.
- The agent continues its execution.

4.3 Integration Layer

The Integration layer was developed to join the Multiagent layer with the Edge layer. This layer enables the use of MAS to run in legacy hardware. In order to do so, this layer has 2 methods. The executeMove and the executeSkill. The executeMove is called by the TA to move the product in the factory whereas the executeSkill is called by the RA to make a change in the product.

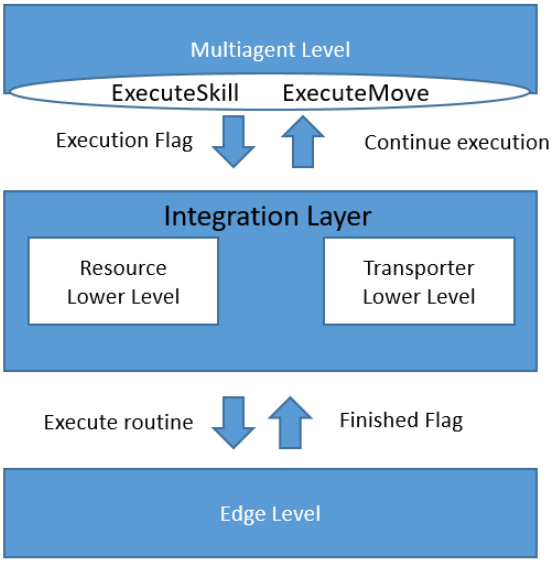

Figure 4.9: Integration layer interactions.

This Layer is divided in two parts, the Resource and the Transporter Lower levels. The Resource part, is in charge of managing the flags that enable the skills whereas the Transporter is in charge of managing the transporting flags. This Layer is always called by the MultiAgent Level through executeMove and executeSkill methods whenever an agent requests it and the first thing it does is connecting to the controller. After connecting, the agent sets the execution Flag to True. The controller is always in standby and when it reads the flag, it executes the routine. When the task is finished, the Edge level sets the finished flag to True and waits for another flag to start a routine.

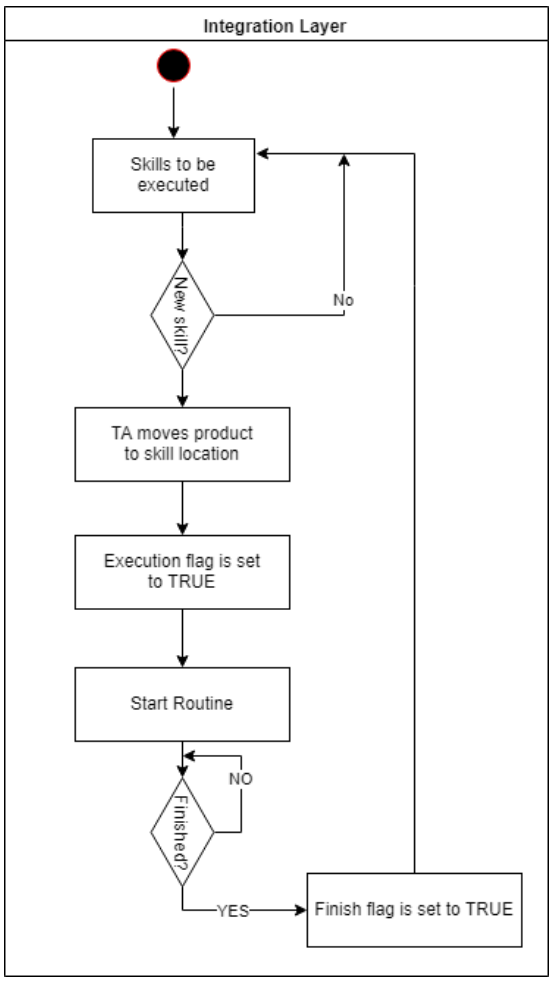

Figure 4.10: Integration Layer flowchart.

4.3.1 The full routine

With the integration of every layer, the runtime of a product in the factory is:

- DA launches all agents.
- PA asks the RA in which Transporter is located the skill it wants to execute.
- RA responds with the location.
- PA asks TA to move to that location.
- TAs talk to one another and call executeMove method.
- In the Integration Layer, the executeMove method sets executeMove flag to True.
- As the Edge Level waits in standby, it reads the flag and executes the routine.
- When the routine finishes, a finished flag is set to true.
- The agent in standby reads the finished flag and proceeds with the executions.

- When the product reaches the correct station, the RA calls executeSkill.
- Integration Layer sets executeSkill flag to True.
- Edge Level routine according to the flag that is enabled.

This sequence repeats itself until there are no more skills to be executed.

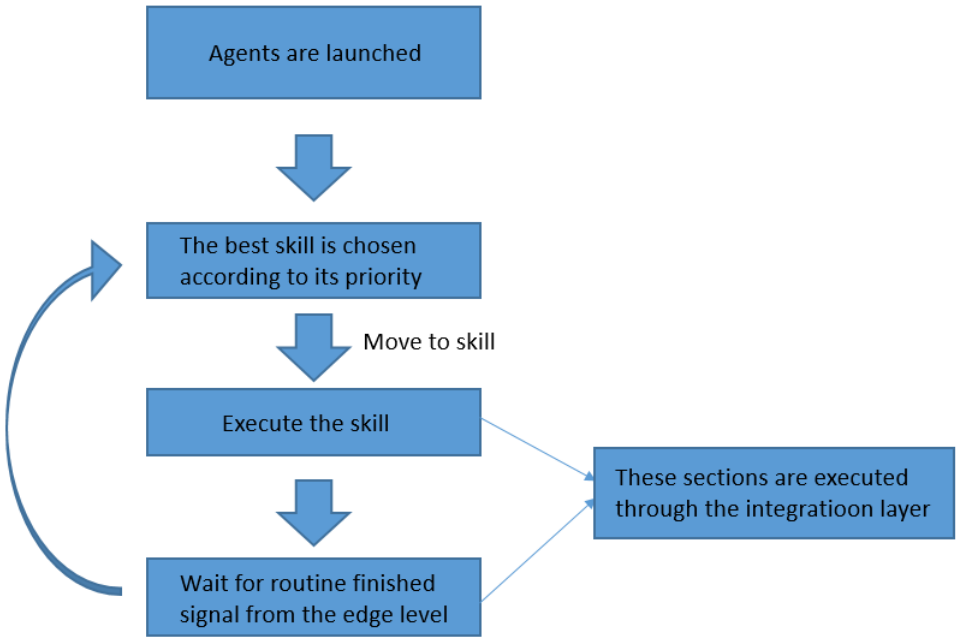

Figure 4.11: Scheme summarising the factory's routine.

In figure 4.12, it is shown a flow chart of a product in the factory. When the product is launched, it will have a skills to be executed. The PA will ask the RA for the skills to be executed and their location, if the PA is not in correct location, the TA will move it to that location if it is not occupied. When the skill is executed, the PA will ask the RA for the next skill. If it was the last one, the PA will remove the product from the factory, else it will repeat the cycle.

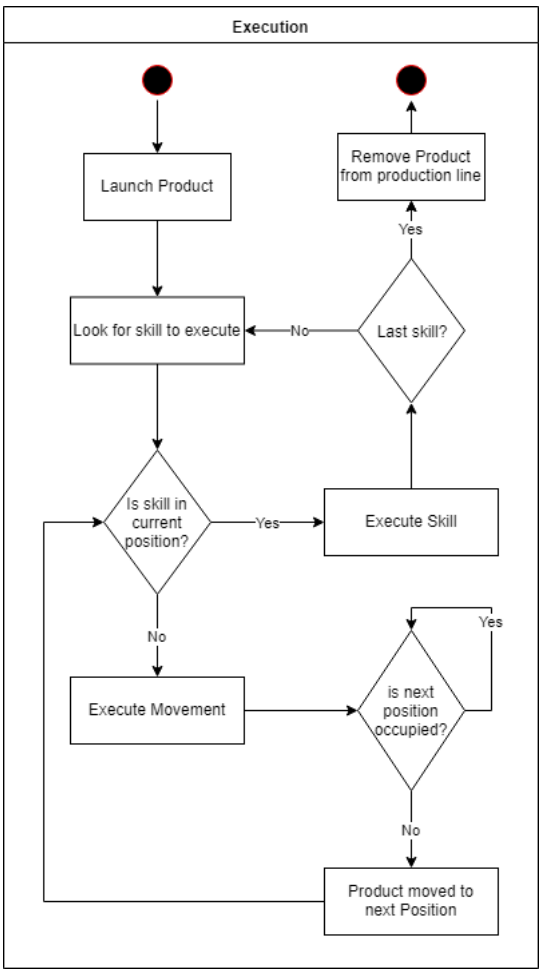

Figure 4.12: Flow chart of the factory routine from the launch to the ending of a PA.

By abstracting products, resources and transporters into agents of a MAS and joining it with a layer designed to communicate with legacy hardware, it is possible to bridge the gap between the emergent production paradigms and legacy edge level devices.

Chapter 5

Implementation

The implementation of the architecture was developed in two different softwares. While the Edge level architecture was developed in Structured Text(ST) in CodeSys IDE, the multiagent system development was in JADE.

5.1 Edge Level Implementation

The Controllers used in the thesis were the combo master and slave Elrest CM211 and CS101. The master has 16 digital I/O signals and the slave is an extension of the master with an additional 32 I/O signals. To connect both PLC, CANBus protocol was used.

Figure 5.1: Master digital inputs.

To setup a project in Codesys, the devices that are used in the IDE are a virtual representation of the hardware setup of the factory. To accomplish this, it was necessary to download the electronic datasheet of the PLC used. This is done in order to match the I/O of the controller with the I/O mapping of the IDE. To add the Slave device to the project, it is needed to use a tool called CANopenManager since master-slave connection is via CANBus. This tool represents the cable that connects both controllers. After adding these devices, it is possible to read and write the inputs and outputs of the factory. Having done this, it is possible to start developing the Structured Text code in the IDE. To be

able to read and write Data from the MultiAgent Level, two new devices had to be added. The first one being an Ethernet device to represent the ethernet cable that is connected between the PC and the controller. The second one is ModbusTCPSlaveDevice which represents the role the PC has in the system. After adding all these devices, the project device tree should look like this:

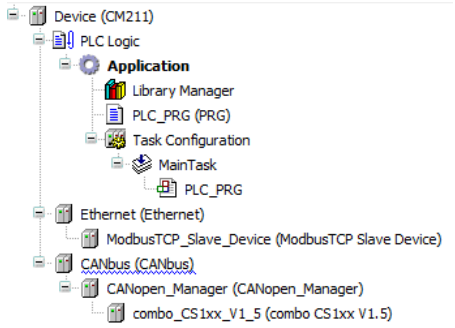

Figure 5.2: The device tree in CodeSys IDE.

To be able to access the sensors and actuators of the factory, they have to be mapped in the I/O mapping tool of the IDE. Due to the eletronic datasheet of the PLC, it is easily done since each bit in the mapping has its description available which matches an entry in the controller in order to recognise which bit is being read or written.

The algorithm developed in the PLC is fairly similar in every routine. The controller reads the execution flag to start the routine then it enables the assigned actuator and from there and with the help of the sensors the routine continues until the end and finally setting the finished flag to true.

Both the execution and finished flags are mapped under the ModbusTCPSlaveDevice since they are the bits that are transferred via ethernet to and from the PC. In that mapping, it could be accessed the modbus holding registers and modbus input registers. The main difference between the holding and the input register is while the input register is read only, the holding is a read and write register. In the code, both had to be mapped because the library used to be able to communicate to higher level would only read from the input register, so they had to be separate.

The actuators and sensors of the master are mapped on the main device whereas the slave mapping is done in the device under the Canbus tree.

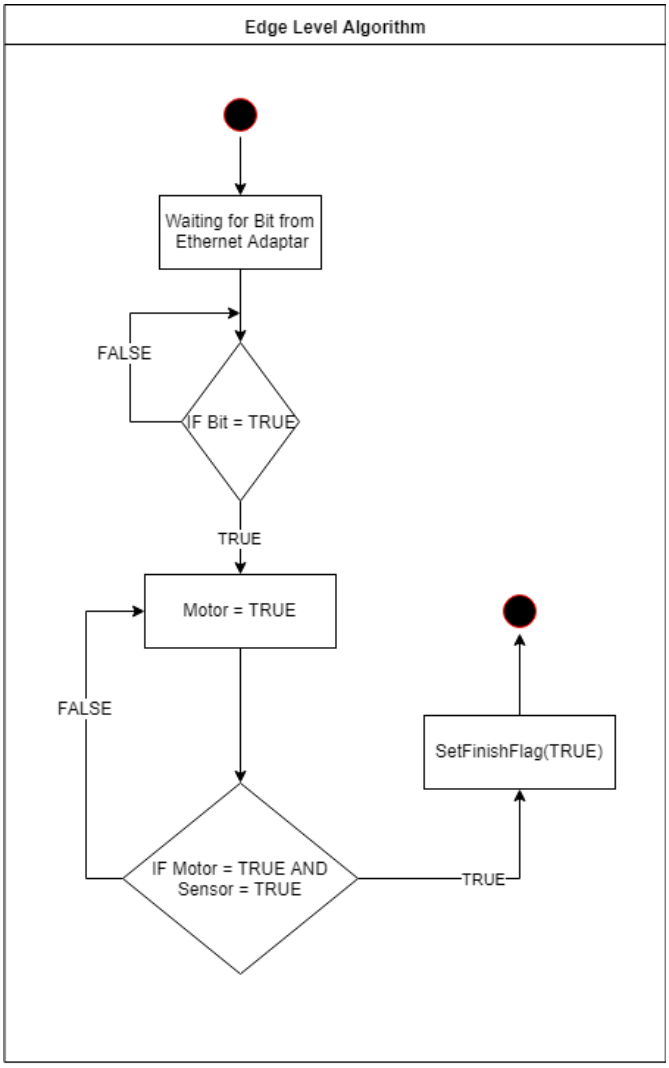

Figure 5.3: Flowchart representing the edge level algorithms.

Each skill and movement is a different algorithm waiting for a different flag and setting a different flag to true. It is important to set control variables if there are modules that share the same sensors to avoid having them start at the same time since the code is run in a thread. These variables tell the system in which module it is currently, disabling the other modules from running without having to.

5.2 Multiagent Level Implementation

As referred in chapter 4, the agents communicate with one another. Two ways of interaction between agents used in this project were the FIPA Request Interaction Protocol and the FIPA Contract Net Interaction Protocol.

The FIPA Request Protocol enables one agent to request another to perform an action. The agent that the action was request has to decide whether to *accept* or *refuse* the request. If it is not *refused*, it is sent an *agree* to the Initiator. After the *agree* is sent, the agent tries to fill the request. If it fails to complete it, it send a *failure*, if it is successful it returns an *inform*.

The FIPA Contract Net Protocol is used when the Initiator has to solicit a number of proposals from other agents, these actions are called *propose* acts. This protocol is used when more than one agent can do the requested task. The proposal sent by the participant agents include the cost, time it takes to do the task. After the deadline passes, the proposals are evaluated. The agents who were selected will be sent an *accept-proposal* whereas the ones that were not will receive a *reject-proposal*. The proposals are a commitment from the agent that proposed so once it is accepted, the task will be performed. Once the task is performed, an *inform* will be sent, however if it fails to complete the task, *failure* will be sent instead.

JADE implements a Directory Facilitator(DF), this is compared to the "Yellow Pages" service. The agents that have services available register themselves in the DF and any agent can search in the DF and look for what agent they want to perform the service they need.

In the agent description of the DF there is:

- AID, the name of the agent.
- Protocols (FIPA Request/FIPA contract net)
- Ontologies (A string that determines the service that is being done).

5.2.1 Transporter Agent

The TA used FIPA requests to communicate. It uses one request to assess whether the next Transporter is available or not, and its action is also requested by the product agent.

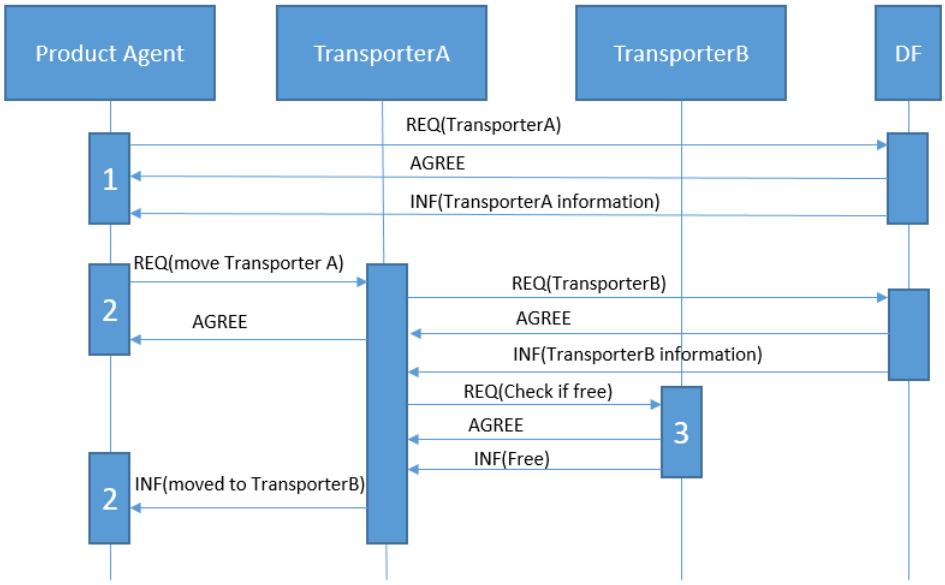

Figure 5.4: Diagram of the TA's communication.

In figure 5.4, there are 2 transporter agents being represented to showcase the communications between TA's. Assuming TransporterA is set before TransporterB, this communication are as follows:

In the box with the number 1, the PA is searching in the DF the information regarding the TransporterA, namely its AID and the protocol needed to communicate with it.

In number 2, the PA requests TransporterA to move, which means, transporterB has to be free. In order to check if the TransporterB is free, the TransporterA has to search in DF transporterB's information. When it receives the information, TransporterA sends a request message with the ontology to check whether it is free or not and the TransporterB informs the agent about its status (box with number 3). Finally TransporterA informs the product that it has moved to TransporterB. This communication is similar between all the TAs.

The TA has information that knows on launch. One of those is that the agent knows its position in the system. It starts unoccupied by default and it knows the destiny of the product that it is transporting. The communication between agents is possible through implementation of behaviours of the type *AchieveREInitiator*. This type of implementation enables the communication with other agents. When the product launches this communication with the Transporter system, the ontology of the message is *EXECUTE_TRANSPORTER*. This way, the agents are sure to communicate with the protocol

FIPA Request. In order to respond to the message, on the TA side, the behaviour implemented is *AchieveREResponder*. This method called *FIPARequestResponderConveyor* receives a message and depending on the ontology does three things:

- If the ontology is *EXECUTE_TRANSPORTER*, it creates a new message with the ontology *CHECK_TRANSPORTER*, and starts another request sequence as in figure 5.4.
- If the ontology is *CHECK_TRANSPORTER*, and if it is occupied, it returns a refuse, else it will inform the conveyor that it is free, and it will execute the movement.
- If the ontology is *REMOVE*, it means that the product has reached the exit and it will be removed from the system, making the transporterE free.

5.2.2 Product Agent

The PA is in charge of choosing the best agent to do a certain task. It is also in charge of initiating all the conversations between PA and RA or TA. Which means, the PA, even though it is the one which physically does the least, it is the one with the most communications.

The PA has set information that comes from the datamodel, such as:

- The skills that needs to be execute.
- Its ID.
- Its description.

The first thing the PA does upon being setup is running an algorithm to determine what is the skill that it will be executed and thus the Resource Agent to be communicated with when doing the *Contract Net*.

Listing 5.1: Next Skill to be executed pseudo code

```
/**
 * Algorithm to know which skill is the next to be executed
 */
 priority = 0;
 //find highest priority
While(iterator.hasNext()){
 priority_max = GetPriority().iterator.next();
 if(priority_max > priority){
    priority = priority_max;
 }
}
//Loop to find the skill to be executed
while(iterator.hasNext()){
    priority_aux = getPriority.iterator.next();
    if(priority_aux <= priority_aux){
```

```
16          priority = priority_aux;
17          bestSkill = next_skill;
18          //Stores the bestSkill to send in the Request
19          PA.setBestSkill(bestSkill);
20      }
21  }
```

After selecting the skill to be executed, a *Call for Proposals(CFP)* will be launched through the behaviour *Contract Net Initiator*. The content for the CFP is the name of the skill and the RAs propose with the efficiency of the skill. When the RA has its proposal accepted, it will send an inform telling the PA in which transporter can that skill be executed. The best proposer AID is saved for further use.

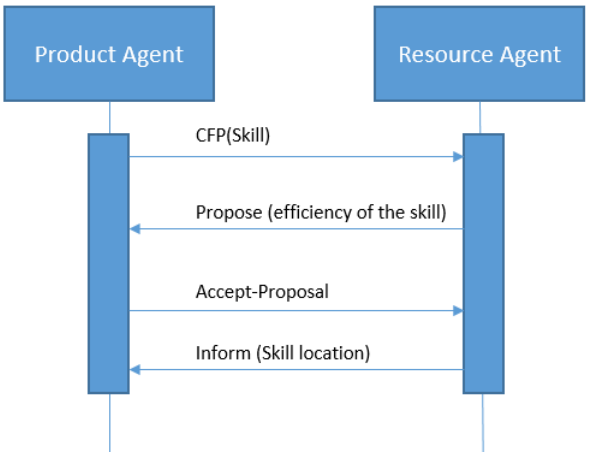

Figure 5.5: Diagram of the contract net protocol.

After the *Contract Net*, the product agent will create a *FIPA Request* with the TA if the product's position is different than the position of the skill it wants to execute by calling the method *FIPARequestInitiatorProduct_Transporter*. The protocol is setup with the ontology *EXECUTE_TRANSPORTER*.

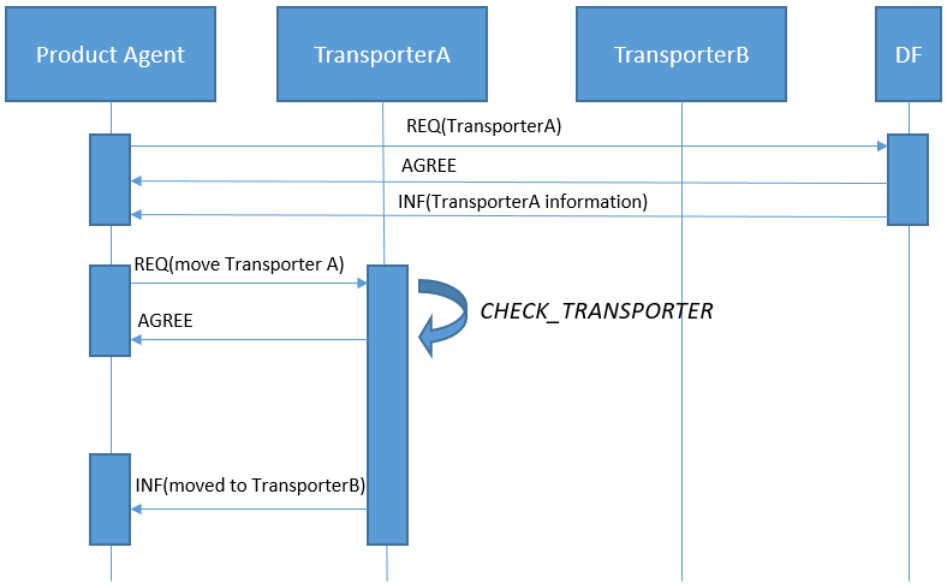

Figure 5.6: Diagram of the communication between the PA and TA.

Every time the product has to move, the following Transporter Agent has to be free, thus the current Transporter creates a *FIPA Request* message with the *CHECK_TRANSPORTER* ontology sent through it.

When the product position is the same as the skill to be executed, *FIPARequestInitiatorProduct_Resource* is called. This method creates a request protocol as well, with the skill to be executed as the content of the message and the receiver the best proposer that was accepted in the CFP. The RA after receiving the request, executes the skill and informs with a *done*. When the PA receives the done, it removes the skill from its list of executes and executes the same routine as in listing 5.1.

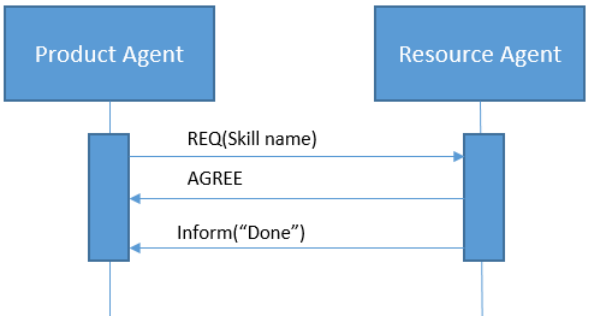

Figure 5.7: Diagram of the communication between the PA and RA.

5.2.3 Resource Agent

The data the RA knows by default comes from the data models as said in section 4.1.2. It knows which skill each resource can execute as well as where they are plugged. On launch, the agent registers itself in the DF with its skill and adds the behaviours *ContractNetResponderResource* and *FIPARequestResponderResource*. As previously mentioned, the *ContractNetResponderResource* proposes to the PA by sending its efficiency and informs it of its location whenever it is the best proposer.

The *FIPARequestResponderResource* receives the request from the PA, reads the content of the request, which is the skill that needs to be executed and executes it.

Listing 5.2: Skill to be executed

```
/**
 * Algorithm to execute a skill
 */
String Skill_ID = request.getContent();
CollectionofSkills Resources = getExecutes();


while(iterator.hasNext()){
    if(Resources.getID().equals(Skill_ID)){
        executeskill();
    }
}
```

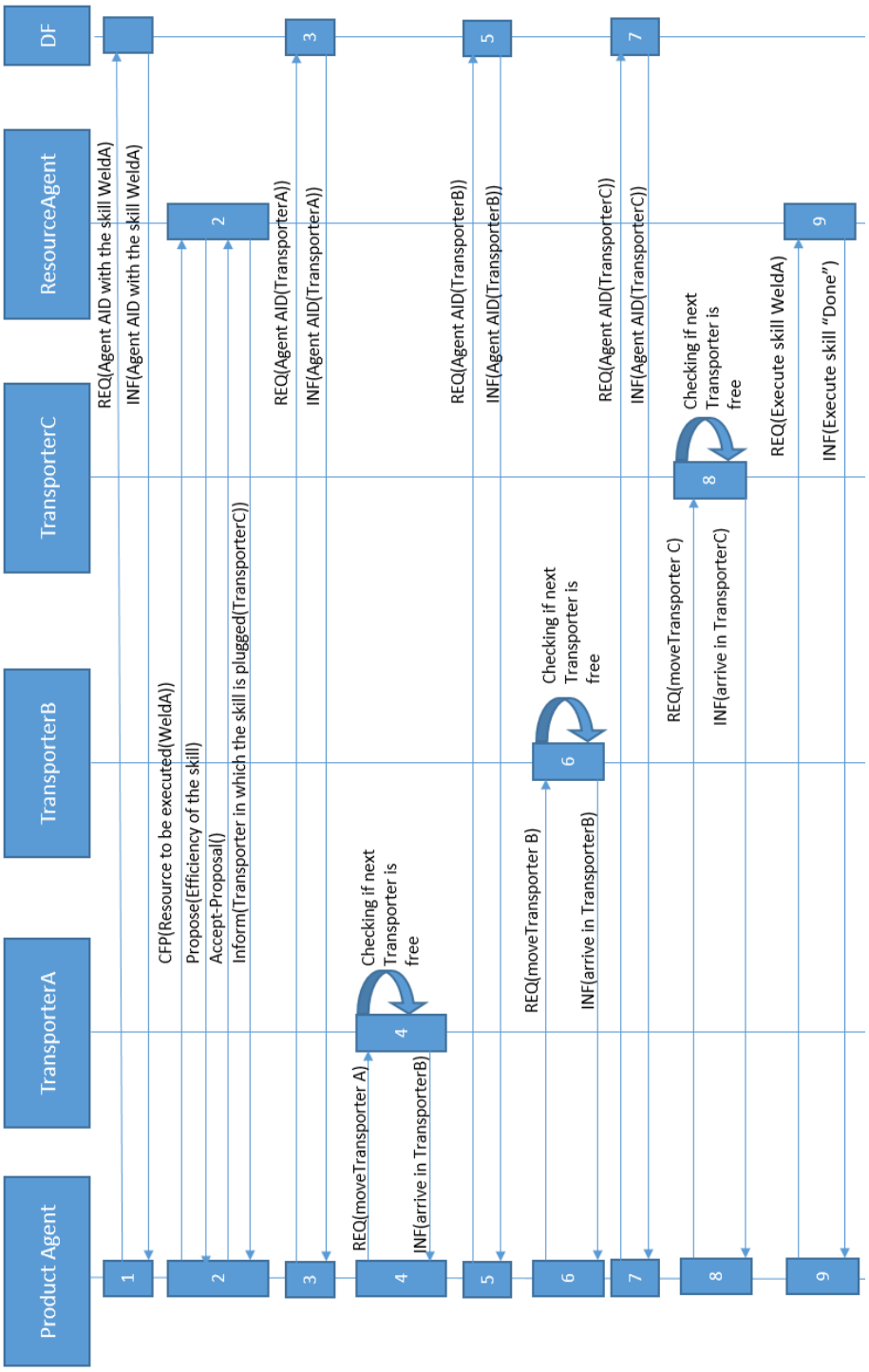

Figure 5.8: Diagram of the communication between all agents to execute a skill.

In box with the number 1, the PA looks in the DF for the RAs that have the ability to execute the skill WeldA. In box number 2, a Contract Net is initialized by the PA to assess which RA is the best, when the best RA is decided, the agent will inform the PA the location of the skill to be executed, in this example, it is located in TransporterC. For the PA to move to that location, it has to go through Transporters A to C, so it starts the execution by searching in the DF for the TransporterA agent as shown in box 3. In box with the number 4, the PA creates a FIPA Request to move the product, to do so, the TransporterA agent will check if the next TA is occupied, in case of it being free, it will move the product and then informing the PA that the product was moved. After arriving in the TransporterB, the PA will again search in the DF for the agent AID of TransporterB. After receiving this information, it will start a FIPA Request with the Agent to move the product to the next TA (box number 6). To do so, the TransporterB will have to check whether the following one is occupied or not and only when free, moving the product and then informing the PA that the product has moved. Box 7 and 8 is a repetition of the movement. On box 9, the PA will start a FIPA Request with the RA in charge of the execution of the skill, in this case weldA. The RA will inform the PA that the skill is done on finish. This communication repeats itself until the product leaves the factory.

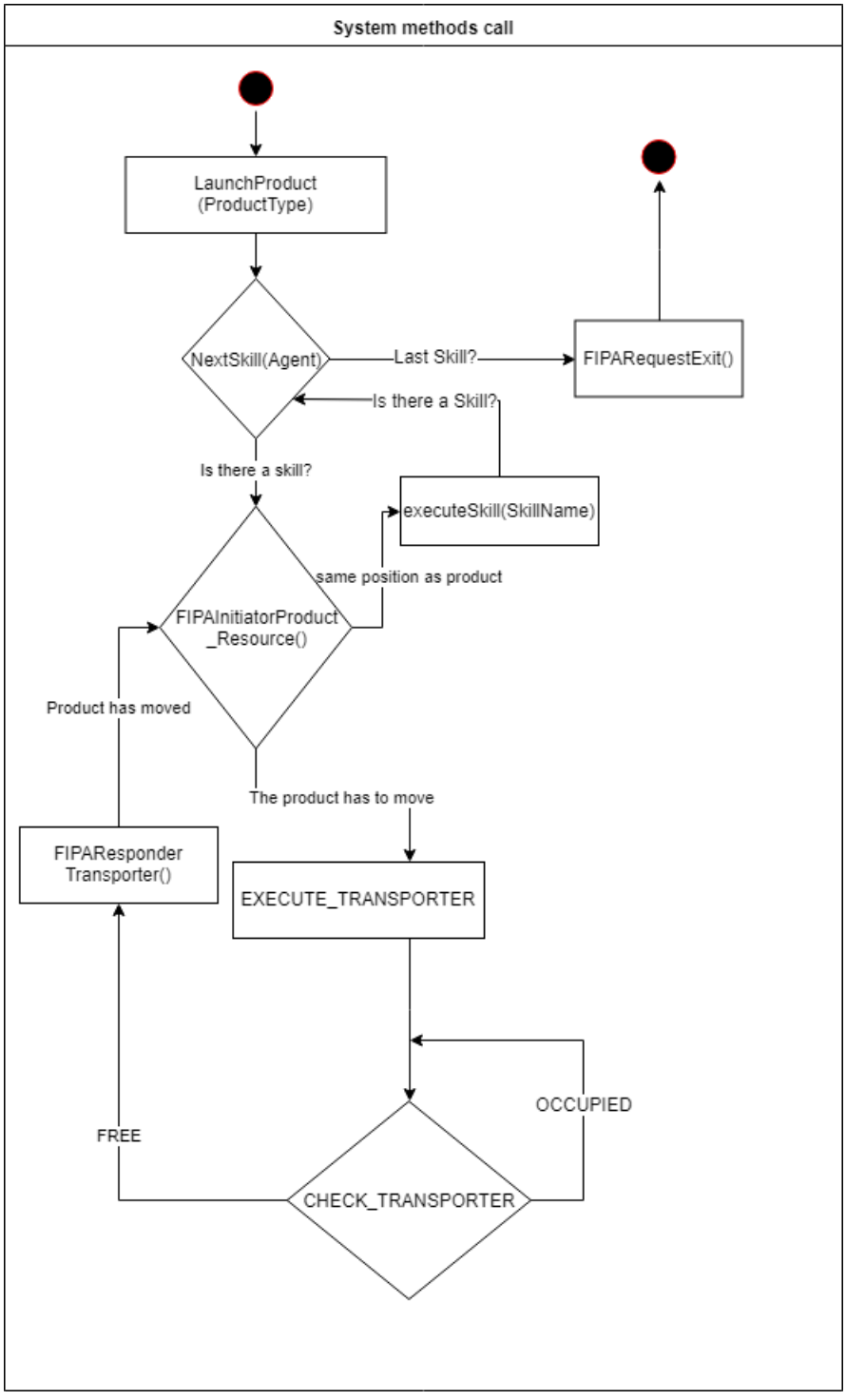

Figure 5.9: Diagram with the states that the system takes.

The figure 5.9 shows the different methods that the system calls. *LaunchProduct* is the agent setup of the PA. The *nextSkill* method is a method developed to know what skill is going to be executed. If there are no more skills, it will create a Request to remove the product from the factory, else it will initialize a Request with the RA that is in charge of the skill to be executed. If the product is in the same position as the skill, it will

execute the skill through the Integration Layer, if not it will set the ontology of the system to *EXECUTE_TRANSPORTER* which will move the product to the correct location. As mentioned, before moving the product, the TA will check if the next Transporter is free by using the ontology *CHECK_TRANSPORTER*, when the product is moved the method *FIPAResponderTransporter* will tell the system that the movement has finished.

5.2.4 Integration Layer implementation

In order to communicate with the controller. It was created a middle level interface in which the coding for every skill and movement is set. It is in this layer that the agents connect to the hardware to execute the skills.

To accomplish this, it was used the openSource library for modbus communication called easyModBus. This library provides the user with different methods for reading and writing in the controller as well as an easy way to connect to the hardware.

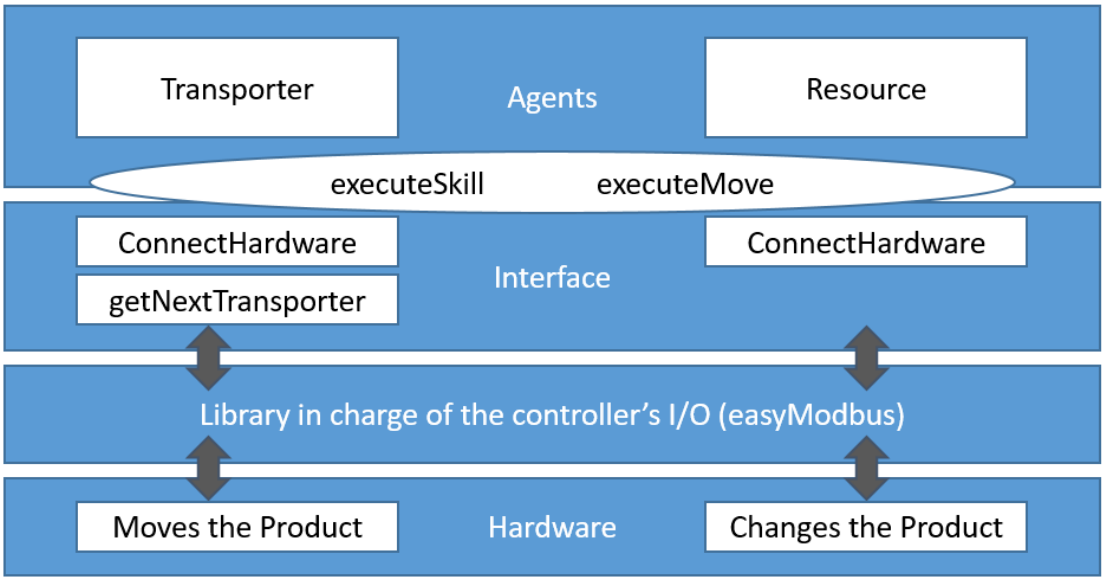

Figure 5.10: Methods available through the integration layer.

The figure 5.10 presents the methods available to the multiagent layer in order to execute the system.

In the interface, the following methods can be called:

- *ConnectHardware*: This method is a boolean that connects the agent to the controller. If it returns *true*, it was successful, else it failed.
- *getNextTransporter*: This method has two parameters, the TransporterName and the product destiny. This method returns a string which is the name of the nextTransporter. It is necessary because in case of TA having two next positions depending on the product being produced, this method will return the correct next position between the two possibilities.

The *ConnectHardware* is called whenever a new agent is launched in order to enable to the controller so it can execute its function. The method *getNextTransporter* is called whenever the product has to decide which Transporter is the next one, this happens when there is more than one possiblity.

This layer is the only one that should be changed, in it there are the methods used to execute skills and execute the movements, thus depending on the system, this should be changed.

The modbus library methods allows the user to use methods such as *ReadDiscreteInputs* and *WriteSingleCoil*. The *ReadDiscreteInputs* has two input parameters, the starting bit and the amount of bits it will read, this will have to match the ethernet adapter I/O from section 5.1, in other words, if there is a weld mapped in bit 5, the parameters should be 5 and then 1, since it is only necessary to read one bit. The *WriteSingleCoil*'s parameters are the same as the former method and it is used in the same way, it has to match the mapped I/O.

The *executeSkill* method is called by the RA and has 2 strings as input parameters, one is the Resource name and the other is the skill name. The first parameter is the RA's name and the second one is the skill that will execute. Depending on the skill, the system will then access a method inside the Integration Layer in order to execute the skill.

The *executeMove* method is called by the TAs and has 2 strings as input parameters, the first one is the Transporter name and the second one is the product's destiny position. The first parameter is the TA's name and it is used to then access the correct method to move the product. The second string is used to decide between methods whenever there are 2 different paths that the product can take on the same transporter depending on the destiny of the product.

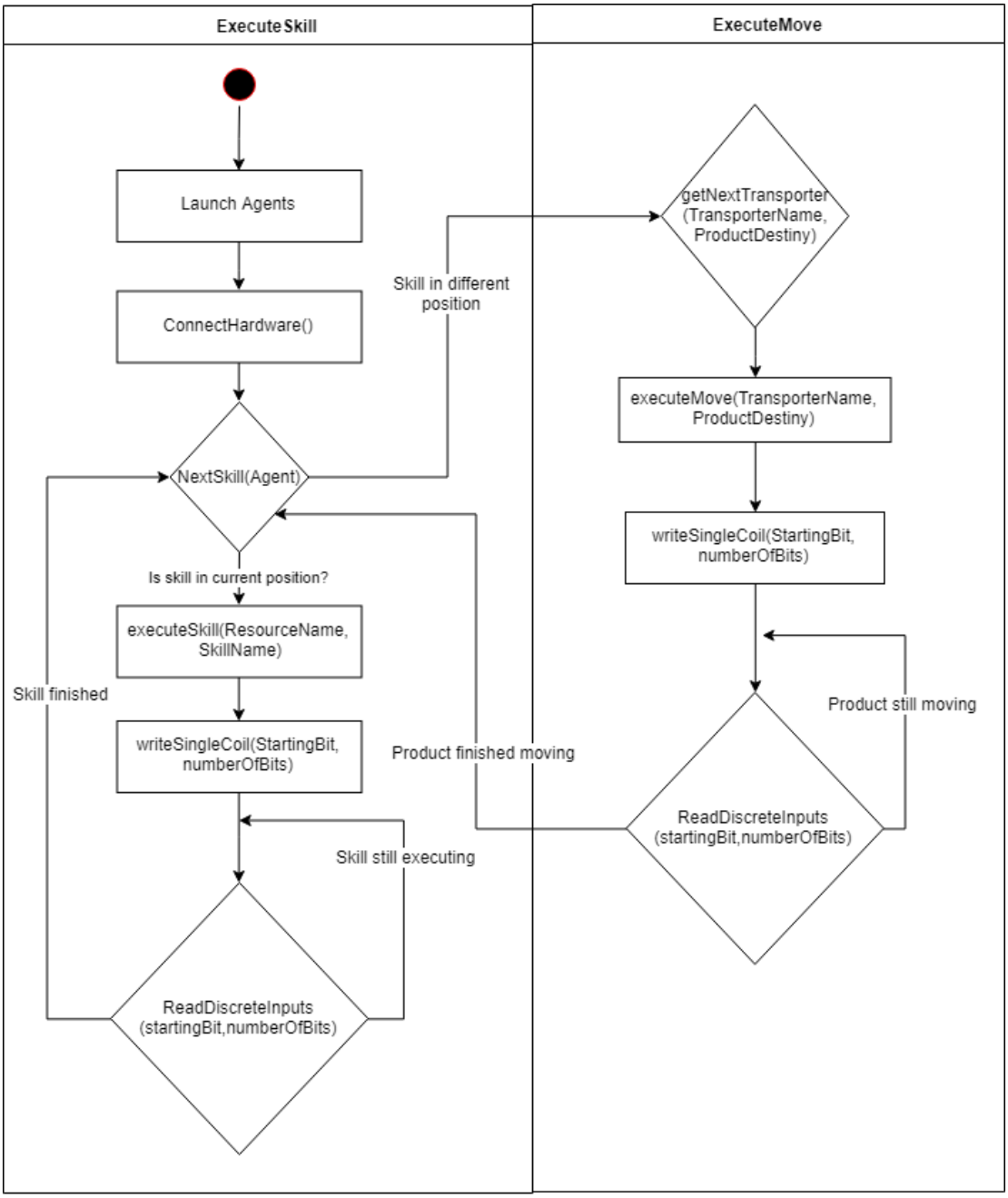

Figure 5.11: Flowchart of the Integration Layer.

In figure 5.11, when the agents are launched they connect themselves to the controller so they're always available to launch modules in the edge level. The NextSkill is called by the product agent that has a list of skills to be executed and by running the algorithm that select which skill is to be executed, it will communicate with other agents to know whether it is in the correct position or not. If it is, it will execute the skill by writing in the coil that is assigned to that skill's module, the coils are mapped in the edge level IDE. If it is in a different location, the TA will move the product to the skill's location by writing in the coil that is assigned to the movement's module. After the movement is initialised, the

agent will wait for the finished flag by reading the bit that is assigned to that flag, like the coils, these are mapped in the edge level IDE. When the skill is being executed, the RA will wait for the finished flag as well, by reading the input dedicated to that end. When it reads it as true, it will inform the product that the skill was executed with success.

In this chapter, it was taken a look in depth on how each level was developed. The edge level was developed in Structured Text and each skill and movement was a module. In the MAS, it was shown how the interactions among agents were made and what they told each other was made clear as well as the states the routine takes. Finally, the Integration Layer was developed through an Interface called by the agents. The interface are the methods that can be called by the agents and are the ones that have to be changed depending on the factory.

Chapter 6

Results

The industrial factory model that was used to execute this project was provided by UNINOVA in Faculdade de Ciências e Tecnologia campus of Universidade Nova de Lisboa.

6.1 The Industrial factory model

The industrial model is made of 2 carts, 2 grippers and a platform which default position is to the right most end of the factory and can move to the opposite end of it to retrieve and move the product. These devices are the ones in charge of moving the product around the factory.

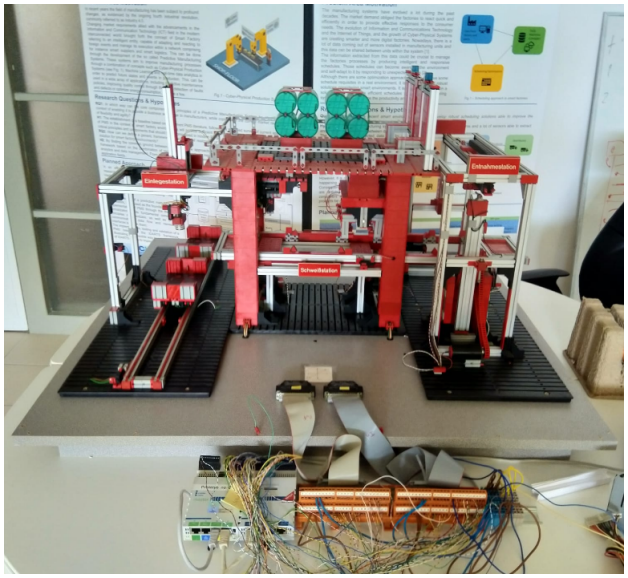

Figure 6.1: Industrial factory model.

In figure 6.1, the entry Skill is executed on the left side. The carts are in one of the edges of the rail, thus enabling the operator to place the product. The carts represent the TransporterA. They will move to place the product to the center of the rail where a Gripper, TransporterB, will descend and take the product from the carts going back up and placing it on the platform. The platform moves from the right to the left, its default position is the most to the right in order to let the product to be lifted from the carts to the platform. The skill WeldA is the first skill in the platform, the TransporterC takes the skill from the Gripper to the WeldA. WeldA welds the product in a line across its width. To move the product from WeldA to WeldB there is the TransporterD. The WeldB is the second position on the platform and it is a timed weld. It will weld the product during 500 milliseconds. The exit skill is executed in the platform's default position and TransporterE is in charge of moving the product from WeldB to the exit position. To execute the Exit, a gripper will go down and take the product to a storage area outside the factory.

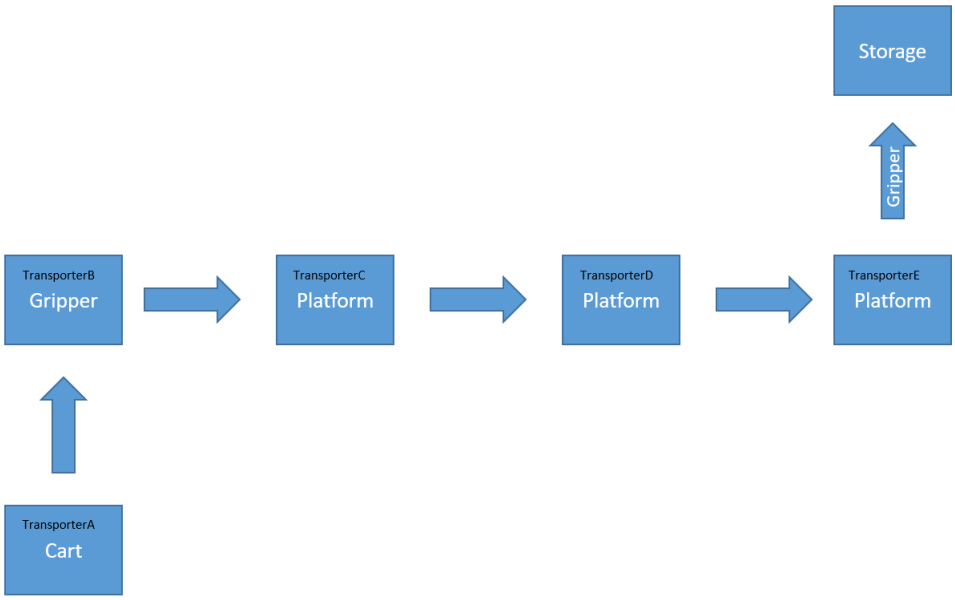

Figure 6.2: Factory's map.

All the agents in the system were developed on a laptop which was connected to two controllers.

To facilitate the launch of every agent in the system, it was developed the DA that deploys all of these with all of their characteristics at start-up. This does not have to be changed after changing the factory since it reads the information from the datamodel which is changed in protegee thus making it simple to modify the factory without having to change the code.

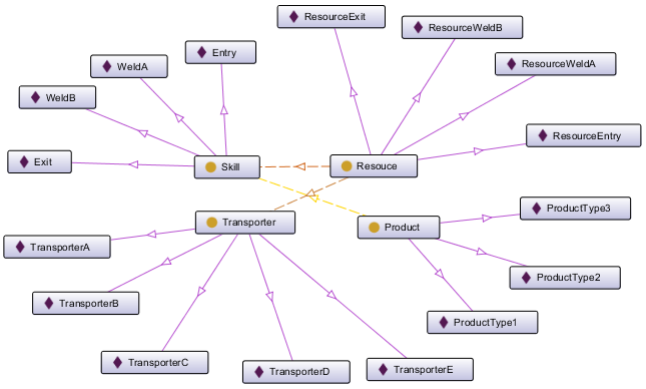

Figure 6.3: Data model instances which are launched via the DA.

DA launches the TA's, TransporterA, TransporterB, TransporterC, TransporterD and TransporterE. It deploys all the RA's, ResourceEntry, ResourceExit, ResourceWeldA and ResourceWeldB and it deploys the PA whenever the user selects a product to be deployed.

The product can be chosen in the following interface:

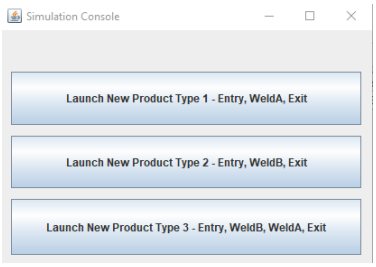

Figure 6.4: User Interface to select the type of product to be produced.

The RA and TA's have to connect themselves to the controller when they want to execute a skill. The PA does not have to connect itself to the controller because the PA is the one request skills from the other agents and it is the other agents that are executing the skills, thus the Agents that connect to the controller are as follows:

Table 6.1: Agents that are connected to the controller.

Controller 192.168.5.237
TransporterA, TransporterB, TransporterC, TransporterD, TransporterE
ResourceWeldA, ResourceWeldB, ResourceEntry, ResourceExit

The system's RAs have only one skill assigned to each, the ResouceEntry executes Entry, ResourceExit executes Exit, ResourceWeldA executes weldA and ResourceWeldB executes WeldB.

The ResouceEntry is plugged to TransporterA, the ResourceWeldA is plugged to TransporterC, the ResourceWeldB is plugged to TransporterD and the ResourceExit is plugged to TransporterE.

It is possible to assign different skills to each Resource if the factory is capable of executing them in the same position. The library being used to communicate through modbus only enables the user to sends bits so tools have to be binary.

This model does not allow to have several PA's at the same time due to the nature of the model itself. If one agent is executing a skill and another is moving a Transporter, they might use the same sensors and actuators which would disrupt either one or the other therefore it is only possible to launch a PA when there are none on the platform.

6.1.1 Factory integration Layer

To make possible the system's execution, it is only needed to change the package LowerLevel on the multiagent system. This package is the one that creates the Integration Layer.

In order to verify that it is possible to use the paradigms reviewed in chapter 2, it was designed three types of product:

- ProductType1: This product consumes the skill WeldA, Entry and Exit thus it starts on TransporterA, executes the skill WeldA on TransporterC and moves to TransporterE to exit the system.
- ProductType2: This product consumes the skill WeldB, Entry and Exit. Therefore it starts on TransporterA, executes the skill WeldB on TransporterD and moves to TransporterE to exit the system.
- ProductType3: This product consumes both WeldB and WeldA skills and Entry and Exit skills. The first skill to be executed is WeldB which is on TransporterD, then it is WeldA which is on TransporterC so the product goes back in the factory to execute this skill and then it goes to TransporterE to exit the system.

Table 6.2: Products and the skills they have to consume.

Product Agent	**Skills**
Type 1	Entry, WeldA, Exit
Type 2	Entry, WeldB, Exit
Type 3	Entry, WeldB, WeldA, Exit

The Integration Layer has the following methods:

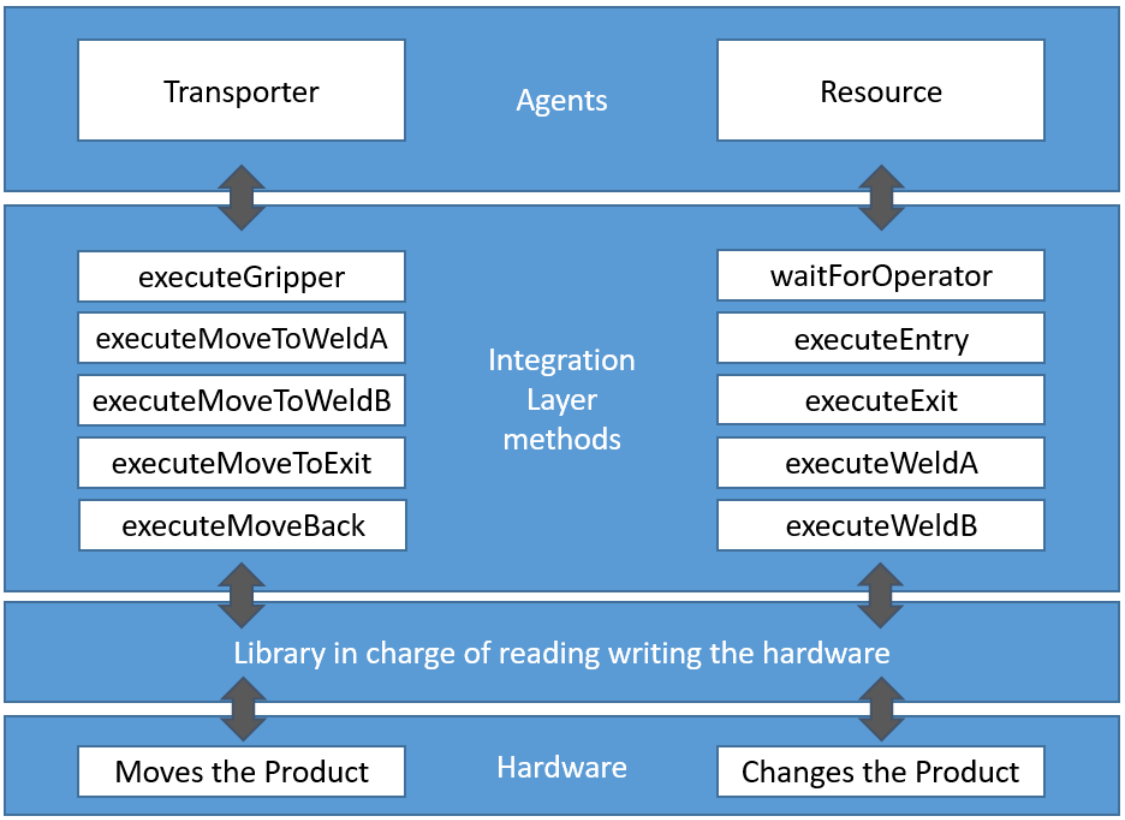

Figure 6.5: Factory specific integration layer.

The Transporter side of the Integration layer methods are:

- *executeGripper*: This method is a boolean. It moves the product from TransporterA to TransporterB. It sets the execution Flag of transporterB to true. It then waits for the finished flag set in the edge level to return *true*.
- *executeMoveToWeldA*: This method is a boolean that moves the product from TransporterB to TransporterC. It sets the execution Flag of TransporterC to *true*. It then waits for the finished flag set in the edge level to return *true*.
- *executeMoveToWeldB*: This method is a boolean that moves the product from TransporterC to TransporterD. It sets the execution Flag of TransporterD to *true*. It then waits for the finished flag set in the edge level to return *true*.
- *executeMoveToExit*: This method is a boolean that moves the product from TransporterD to TransporterE. It sets the execution Flag of TransporterE to *true*. It then waits for the finished flag set in the edge level to return *true*.
- *executeMoveBack*: This method is a boolean that moves the product from TransporterD to TransporterC. This is used when the product being produced is ProductType3 since the first skill that is executed is WeldB and then WeldA so the product

goes to TransporterD first and then back to TransporterC. This method sets the execution Flag of TransporterDMoveBack to *true*. It then waits for the finished flag set in the edge level to return *true*.

The Resource side of the Integration layer methods are:

- *waitForOperator*: This method is part of the Entry skill since it is done by a human operator. It creates an interface for the operator to click after putting a product in the entry position.

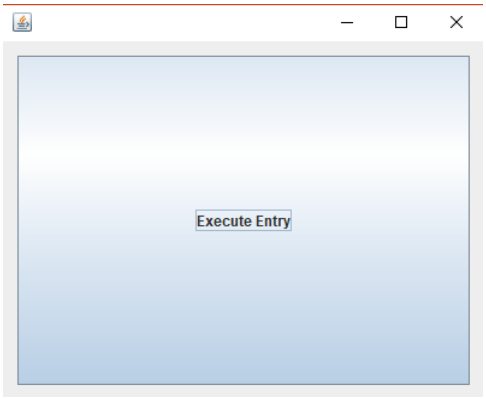

Figure 6.6: Confirmation interface.

- *ExecuteEntry*: This is called after the operator confirms that the product is in the entry position. The entry is done by moving the TransporterA carts to a position where the TransporterB gripper can grab the product. This boolean method returns *true* when the cart stops moving.
- *ExecuteExit*: This method is a boolean. It is called when the product arrives on TransporterE. It sets the execution Flag to *true*. It then waits for the finished flag set in the edge level. This method moves the product from TransporterE to the Storage, removing it from the system.
- *executeWeldA*: This is a boolean method that is called when the product is on TransporterC and the skill WeldA is due. This is true whenever ProductType1 or ProductType3 are the ones queued. It sets the execution Flag to *true*. It then waits for the finished flag set in the edge level.
- *executeWeldB*: This is a boolean method that is called when the product is on TransporterD and the skill WeldB is due. This is true whenever ProductType2 or ProductType3 are the ones queued. It sets the execution Flag to *true*. It then waits for the finished flag set in the edge level.

6.2 Assessing the Results

The factory has 9 edge level modules that are called by the Multiagent system through the integration layer. The 4 skills, entry (TransporterA movement is included in the entry skill since it leaves the PA in the entry position), exit, weldA and weldB and the 4 transporters, TransporterB, TransporterC, TransporterD and TransporterE and moving back from TransporterD to TransporterC. Each of these is a module that is called through the integration layer. Given this, to assess how worthwhile is using a fog approach to Industrial Agents and the current production paradigms, each module was tested 30 times.

These tests were run on a laptop with 8 GB of RAM, with a i7-4500U processor.

For the Entry skill, there were 30 tests measuring 2 different times. In blue, it is measured the time it takes for the Edge Level to execute the entry skill. In orange, it is measured the time it takes for the MAS to execute the skill.

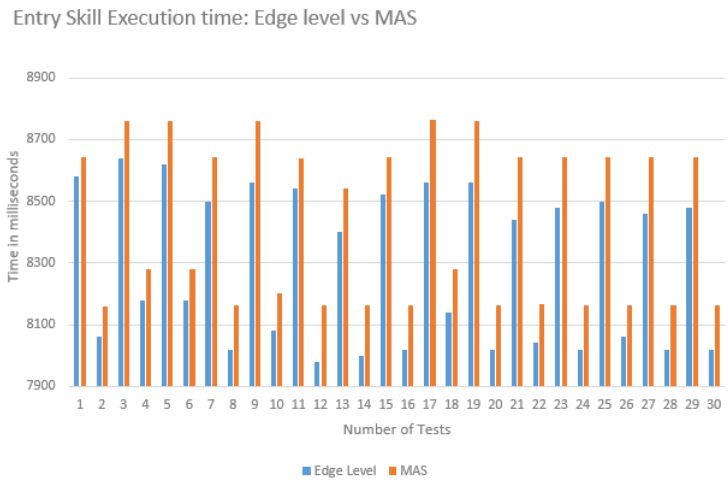

Figure 6.7: Entry skill execution time.

On the edge level, the execution time on average was 8 seconds and 289 milliseconds whereas the average on the MAS level it is 8 seconds and 431 milliseconds. The difference between each individual test and the distance from the average is shown in figure 6.8. The standard deviation of the time difference was 34,23 milliseconds.

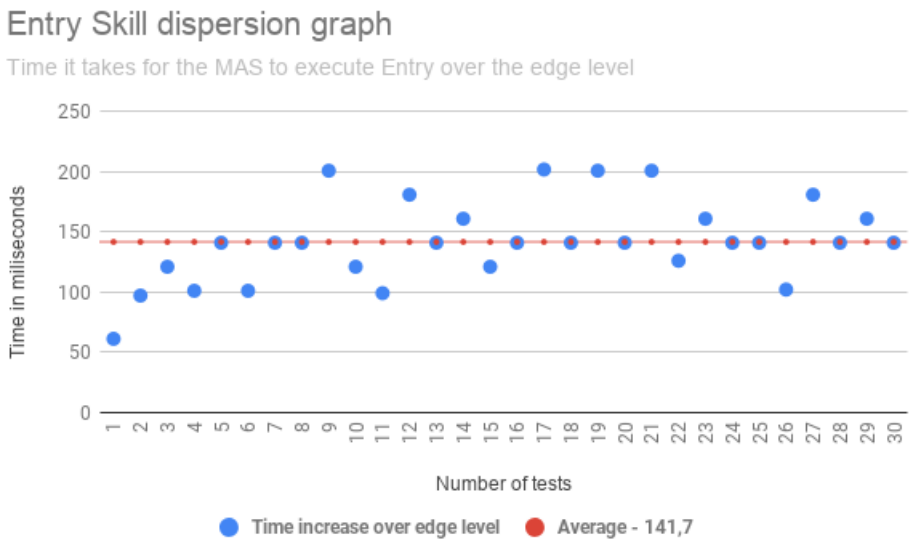

Figure 6.8: Entry skill dispersion graph.

On the 30 tests of the WeldA skill, the average execution time on the edge level was 4 seconds and 898 milliseconds agaisnt 5 seconds and 81 milliseconds on the MAS level.

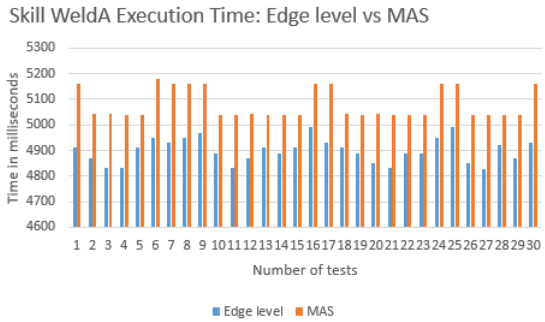

Figure 6.9: WeldA execution time.

The difference between each individual test and the distance from the average is shown in figure 6.10. The standard deviation of the time difference was 36,81 milliseconds.

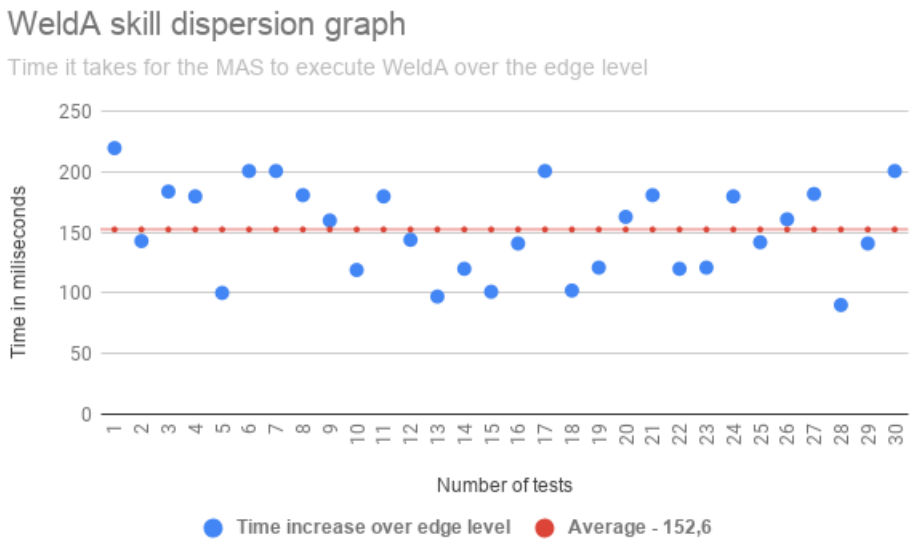

Figure 6.10: WeldA skill dispersion graph.

The average execution time for the WeldB skill for the 30 tests was 500 milliseconds on the edge level and 620 milliseconds on the MAS level. To note, this module was a 500 millisecond timer that activated the weld therefore there's no difference on the edge level between the tests.

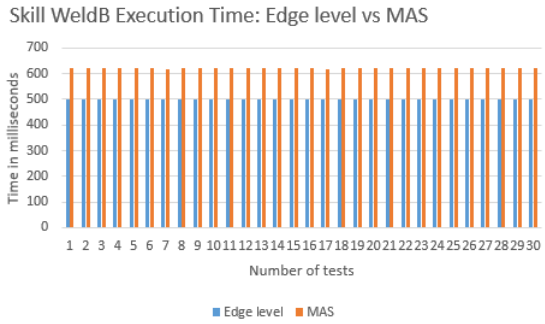

Figure 6.11: WeldB execution time.

The difference between each individual test and the distance from the average is shown in figure 6.12. The standard deviation of the time difference was 1,07 milliseconds.

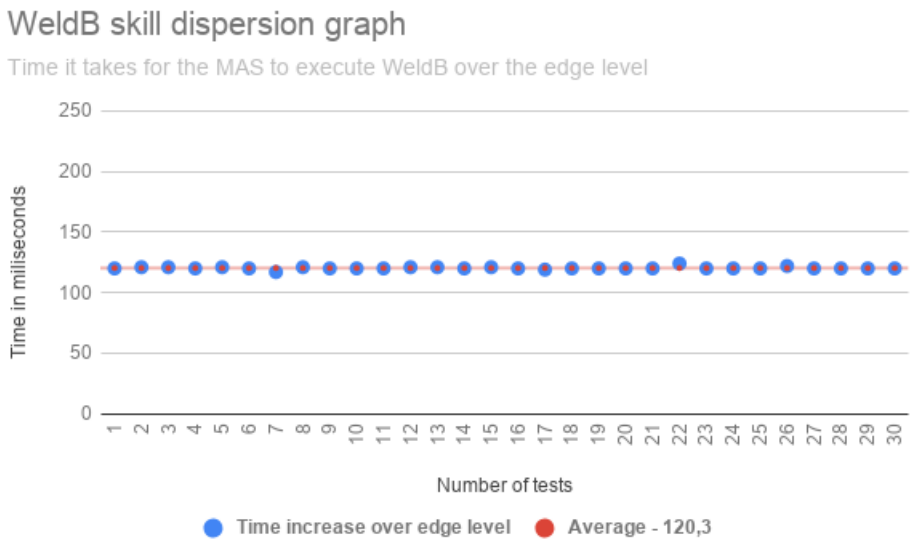

Figure 6.12: WeldB skill dispersion graph.

The average execution time for the Exit skill for the 30 tests was 19 seconds and 658 milliseconds on the edge level and 19 seconds and 815 milliseconds on the MAS level.

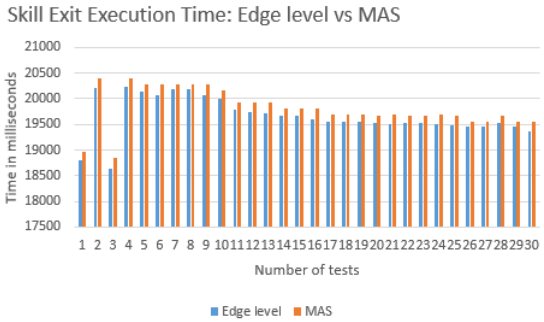

Figure 6.13: Exit skill execution time.

The difference between each individual test and the distance from the average is shown in figure 6.14. The standard deviation of the time difference was 35,05 milliseconds.

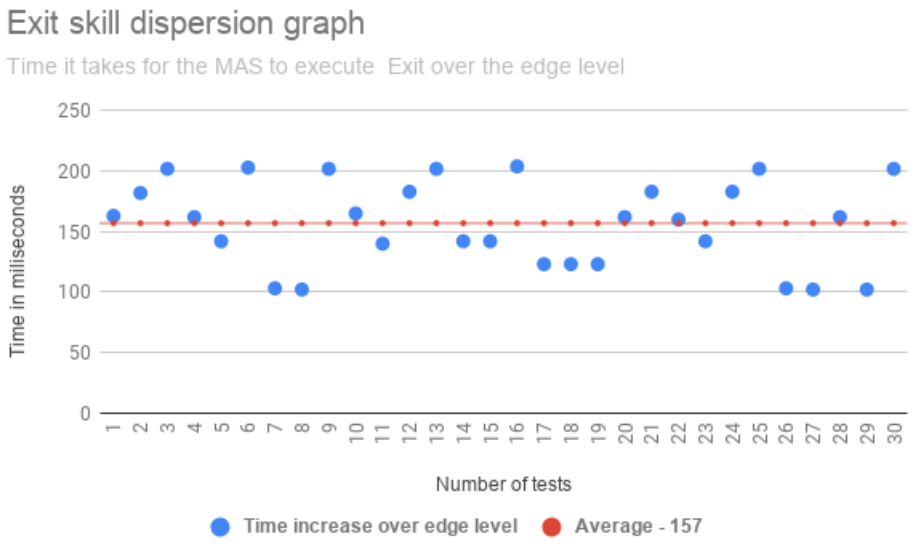

Figure 6.14: Exit skill dispersion graph.

TransporterB is the TA that after the entry Skill is executed moves the product from the Carts to the platform. The average execution time for the TransporterB(Gripper) for the 30 tests was 8 seconds and 92 milliseconds on the edge level and 8 seconds and 282 milliseconds on the MAS level.

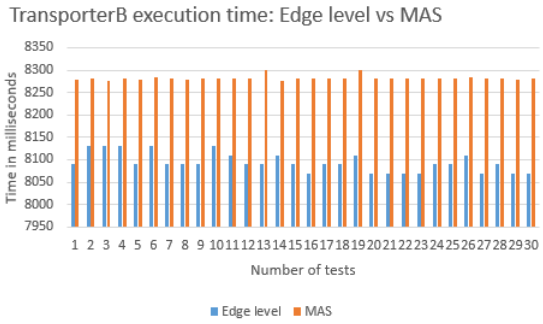

Figure 6.15: TransporterB execution time.

The difference between each individual test and the distance from the average is shown in figure 6.16. The standard deviation of the time difference was 19,87 milliseconds.

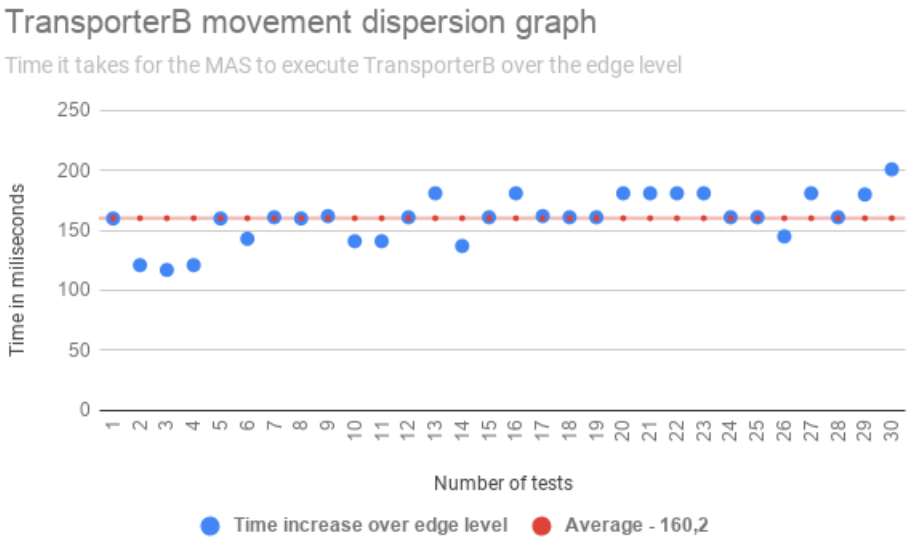

Figure 6.16: TransporterB dispersion graph.

TransporterC retrieves the product from the gripper and moves it to the position where weldA is plugged. The average execution time for the TransporterC for the 30 tests was 12 seconds and 781 milliseconds on the edge level and 12 seconds and 969 milliseconds on the MAS level.

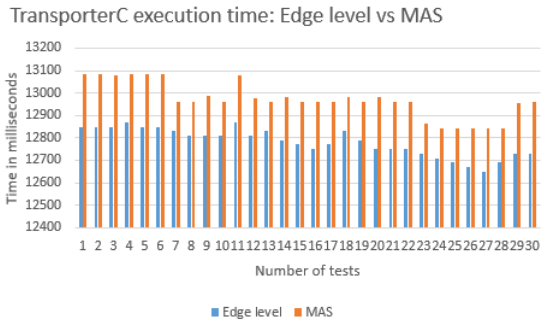

Figure 6.17: TransporterC execution time.

The difference between each individual test and the distance from the average is shown in figure 6.18. The standard deviation of the time difference was 35,83 milliseconds.

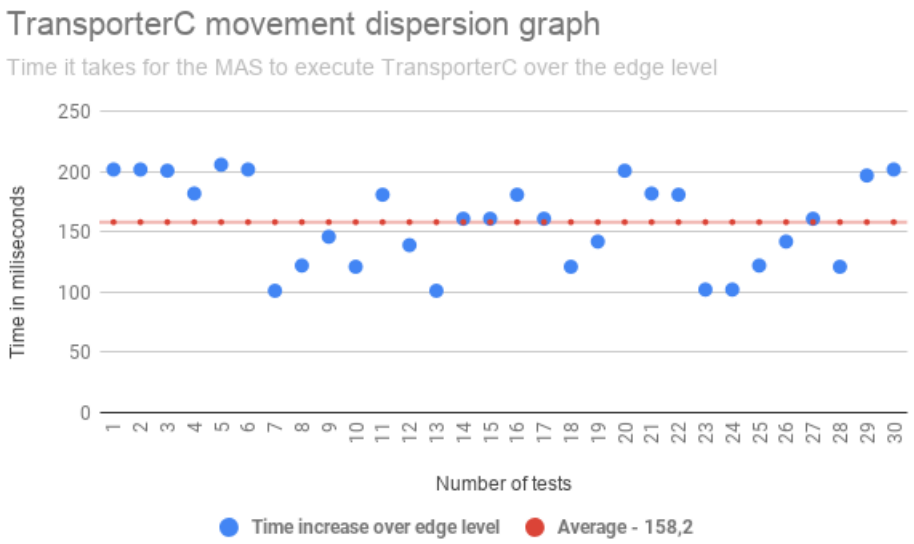

Figure 6.18: TransporterC dispersion graph.

TransporterD moves the product from WeldA position to weldB position. The average execution time for the TransporterD for the 30 tests was 15 seconds and 49 milliseconds on the edge level and 15 seconds and 228 milliseconds on the MAS level.

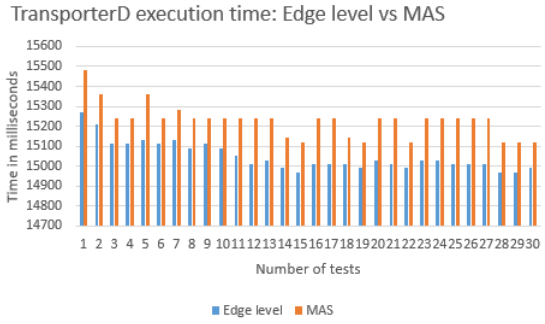

Figure 6.19: TransporterD execution time.

The difference between each individual test and the distance from the average is shown in figure 6.20. The standard deviation of the time difference was 41,54 milliseconds.

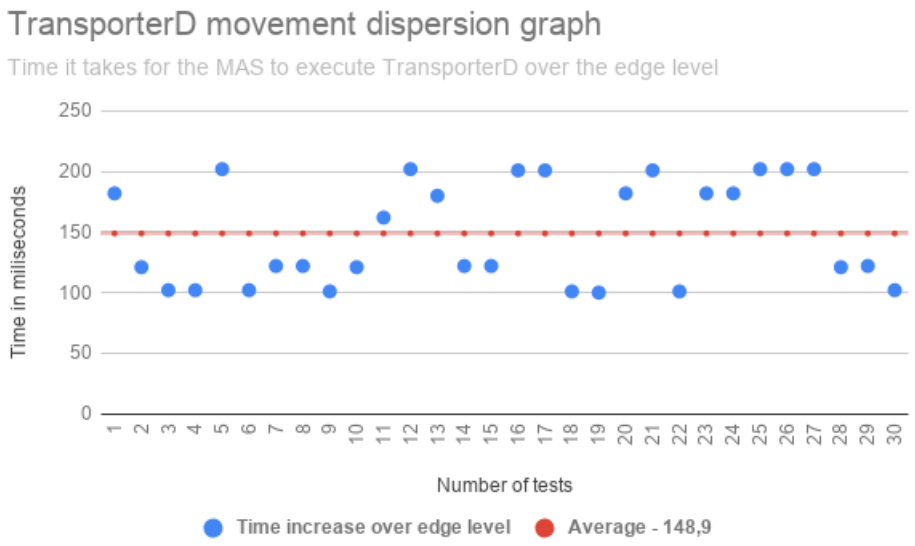

Figure 6.20: TransporterD dispersion graph.

This movement only happens on product type 3, where the product first moves to WeldB and then has to move back to WeldA. The average execution time for the product to make this movement for the 30 tests was 14 seconds and 944 milliseconds on the edge level and 15 seconds and 86 milliseconds on the MAS level.

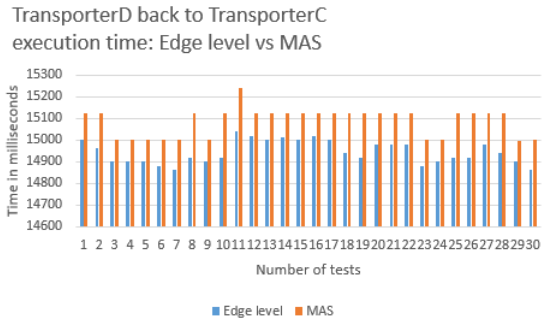

Figure 6.21: TransporterD back to TransporterC execution time.

The difference between each individual test and the distance from the average is shown in figure 6.22. The standard deviation of the time difference was 36,97 milliseconds.

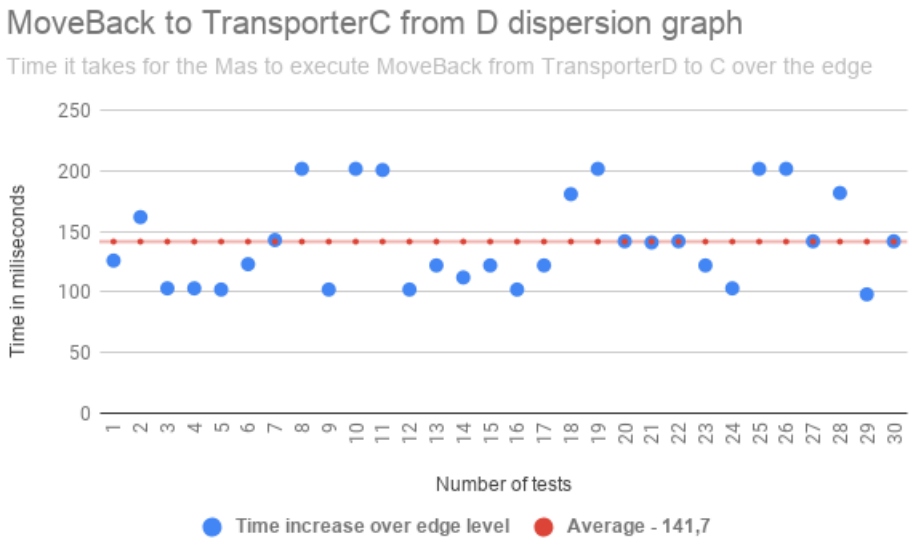

Figure 6.22: TransporterD back to TransporterC dispersion graph.

The TransporterE is in charge of moving the product from WeldB to the exit position. The average execution time for the TransporterE for the 30 tests was 14 seconds and 408 milliseconds for the edge level and 14 seconds and 572 milliseconds for the MAS level.

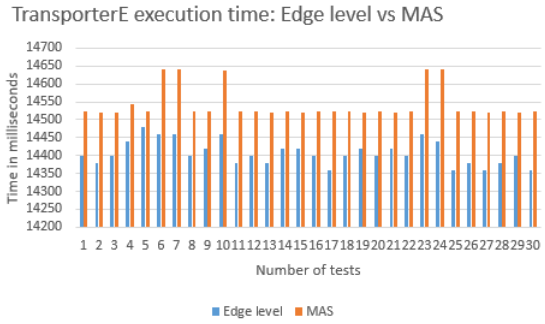

Figure 6.23: TransporterE execution time.

The difference between each individual test and the distance from the average is shown in figure 6.24. The standard deviation of the time difference was 33,18 milliseconds.

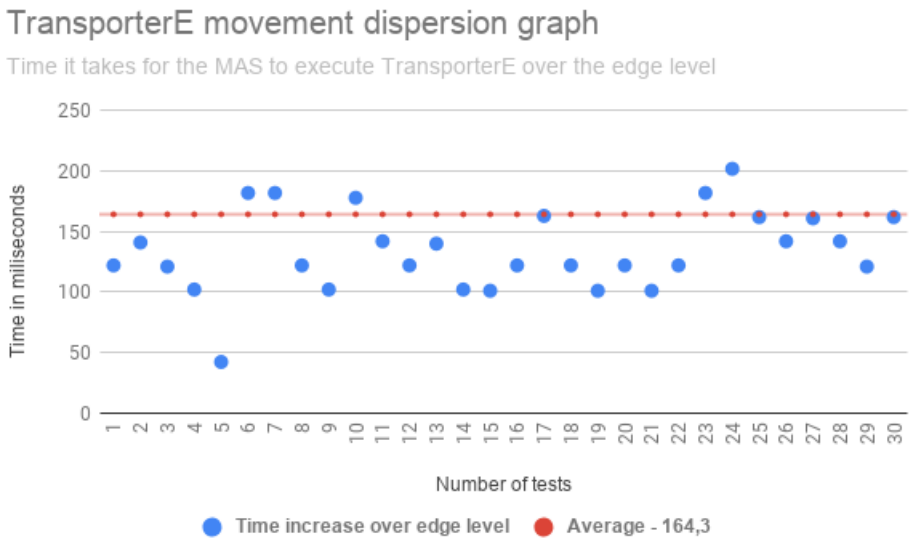

Figure 6.24: TransporterE dispersion graph.

Table 6.3: Average execution times for the 30 tests

	Transporter					Skills			
	B	C	D	E	D back to C	Entry	Exit	WeldA	WeldB
Edge Level average [ms]	8092	12781	15049	14408	14944	8289	19658	4898	500
MAS average [ms]	8282	12969	15228	14572	15086	8431	19815	5081	620
Average difference Edge and MAS [ms]	160,2	158,2	148,9	164,3	141,7	141,7	157	152,6	120,3
Standard deviation[ms]	19,87	35,83	41,54	33,18	36,97	34,23	35,05	36,81	1,07

Table 6.3 shows an average of the 30 tests for each agent. On the first row, it is presented the average of the Edge level, the time it takes since the hardware writes the flag to start the module until it writes the finished flag. The second row shows how long does a module take to execute from the moment the agent calls for the execution until it receives the message that the execution has finished. The third row presents the average difference between the first and second row, which is the time it takes for the message to go through from the computer to the edge level device and back. In the last row, it is presented the average standard deviation of each module.

The total time it takes for the MAS to send and receive a message is on average 150 milliseconds as shown in the graphs above. Taking this time into consideration, given that the modules are several seconds long, it is a very small value that potentializes the use of MAS through fog computing, thus giving life to the current production paradigms with legacy edge devices.

Chapter 7

Conclusion and future work

7.1 Conclusions

With the developed work it can be proven that an Integration Layer is capable of putting legacy hardware to use in the current production paradigms that meet the I4.0 requirements.

The results shown in this thesis suggest that it is possible to improve the current shop-floor level legacy hardware with a layer that enables the use of the state-of-the-art production paradigms. Thus, a factory with no cutting edge hardware has the possibility to meet the product demand with legacy systems.

After assessing the tests, it can be concluded that on average the MAS takes an extra 150 milliseconds to execute a module in the edge level. Each module take on average 10 seconds to execute, the extra milliseconds needed to send and receive a message are a very small percentage of the total execution time but it has to be assessed how valuable that time span is in a real production environment. Modules with very short execution times such as WeldB in this factory model, where the time it takes to send and receive a message is a high percentage of the total execution time, has to be looked upon to see whether it is a viable way to execute it.

The multiagent environment allows to represent several static components of a factory and enables them to communicate with one another to enhance dynamism. This creates the possibility to create products with different skill orders because the product can move freely in the factory matching the demand of the customers. Moreover, by creating an integration layer that allows the MAS to operate with edge level devices enhances the factory, giving the agents the power to enable edge level modules when needed. This freedom allows MAS used in a fog environment to enhance the edge level devices by extending their computation capabilities.

7.2 Future work

As future work, the problem of only using binary tools can be tackled to add different ways to change the product. This is due in order to expand the quantity of tools that can be used through this integration layer, the edge level IDE allows to read every bit one by one. Non binary tools need more than a bit to be read, so to improve the overall capabilities of the system, a way of reading different bits and translate them to the tool in question must be found.

Another point that can be focused is adding more controllers to the edge level and make the integration layer connect to different controllers on need. Given that there ought to be more than one controllers in a factory, it is important that the integration layer is able to connect to several depending on what the agents request.

It has to be studied if the 150 milliseconds delay is a hindrance in real time production and its impact in overall performance of the production. If hundreds of products are being produced every day, a 150 millisecond delay per product might turn out to be a big delay to the overall production. It has to be assessed whether producing less products but having a more flexible factory is worthwhile.

Bibliography

[1] P. K. Muhuri, A. K. Shukla, and A. Abraham. "Industry 4.0: A bibliometric analysis and detailed overview." In: *Engineering Applications of Artificial Intelligence* 78 (Feb. 2019), pp. 218–235. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2018.11.007. URL: http://www.sciencedirect.com/science/article/pii/S0952197618302458 (visited on 12/27/2018).

[2] S. Vaidya, P. Ambad, and S. Bhosle. "Industry 4.0 – A Glimpse." In: *Procedia Manufacturing*. 2nd International Conference on Materials, Manufacturing and Design Engineering (iCMMD2017), 11-12 December 2017, MIT Aurangabad, Maharashtra, INDIA 20 (Jan. 2018), pp. 233–238. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2018.02.034. URL: http://www.sciencedirect.com/science/article/pii/S2351978918300672 (visited on 12/27/2018).

[3] C. Santos, A. Mehrsai, A. C. Barros, M. Araújo, and E. Ares. "Towards Industry 4.0: an overview of European strategic roadmaps." In: *Procedia Manufacturing*. Manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain 13 (Jan. 2017), pp. 972–979. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2017.09.093. URL: http://www.sciencedirect.com/science/article/pii/S235197891730728X (visited on 12/27/2018).

[4] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group*. Forschungsunion, 2013.

[5] E. A. Lee. "Cyber-Physical Systems - Are Computing Foundations Adequate?" en. In: (Oct. 2006), p. 9.

[6] G. Schuh, T. Potente, R. Varandani, C. Hausberg, and B. Fränken. "Collaboration Moves Productivity to the Next Level." en. In: *Procedia CIRP* 17 (2014), pp. 3–8. ISSN: 22128271. DOI: 10.1016/j.procir.2014.02.037. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827114003709 (visited on 01/02/2019).

[7] L. Monostori. "Cyber-physical Production Systems: Roots, Expectations and R&D Challenges." In: *Procedia CIRP*. Variety Management in Manufacturing 17 (Jan. 2014), pp. 9–13. ISSN: 2212-8271. DOI: 10.1016/j.procir.2014.03.115. URL:

http://www.sciencedirect.com/science/article/pii/S2212827114003497 (visited on 01/02/2019).

[8] O. Cardin. "Classification of cyber-physical production systems applications: Proposition of an analysis framework." In: *Computers in Industry* 104 (Jan. 2019), pp. 11–21. ISSN: 0166-3615. DOI: 10.1016/j.compind.2018.10.002. URL: http://www.sciencedirect.com/science/article/pii/S0166361517306231 (visited on 01/02/2019).

[9] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda. "Cyber-physical systems in manufacturing." en. In: *CIRP Annals* 65.2 (2016), pp. 621–641. ISSN: 00078506. DOI: 10.1016/j.cirp.2016.06.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0007850616301974 (visited on 01/02/2019).

[10] P. Adolphs. "RAMI 4.0 - An architectural Model for Industrie 4.0." en. In: (June 2015), p. 31. URL: https://www.omg.org/news/meetings/tc/berlin-15/special-events/mfg-presentations/adolphs.pdf.

[11] K. Schweichhart. "Reference Architectural Model Industrie 4.0 (RAMI 4.0)." en. In: (Dec. 2017), p. 15. URL: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf.

[12] M. Moghaddam, M. N. Cadavid, C. R. Kenley, and A. V. Deshmukh. "Reference architectures for smart manufacturing: A critical review." In: *Journal of Manufacturing Systems* 49 (Oct. 2018), pp. 215–225. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2018.10.006. URL: http://www.sciencedirect.com/science/article/pii/S0278612518301043 (visited on 01/17/2019).

[13] S. Pellegrinelli, C. Cenati, L. Cevasco, F. Giannini, K. Lupinetti, M. Monti, D. Parazzoli, and L. Molinari Tosatti. "Configuration and inspection of multi-fixturing pallets in flexible manufacturing systems: Evolution of the Network Part Program approach." In: *Robotics and Computer-Integrated Manufacturing* 52 (Aug. 2018), pp. 65–75. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2017.05.014. URL: http://www.sciencedirect.com/science/article/pii/S0736584516300412 (visited on 01/03/2019).

[14] G. Chryssolouris, K. Georgoulias, and G. Michalos. "Production Systems Flexibility: Theory and Practice." en. In: *IFAC Proceedings Volumes* 45.6 (May 2012), pp. 15–21. ISSN: 14746670. DOI: 10.3182/20120523-3-RO-2023.00442. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667016331196 (visited on 01/03/2019).

[15] G. Chryssolouris, K. Efthymiou, N. Papakostas, D. Mourtzis, and A. Pagoropoulos. "Flexibility and complexity: is it a trade-off?" en. In: *International Journal of Production Research* 51.23-24 (Nov. 2013), pp. 6788–6802. ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2012.761362. URL: http://www.tandfonline.com/doi/full/10.1080/00207543.2012.761362 (visited on 01/03/2019).

[16] C. Pach, T. Berger, T. Bonte, and D. Trentesaux. "ORCA-FMS: a dynamic architecture for the optimized and reactive control of flexible manufacturing scheduling." en. In: *Computers in Industry* 65.4 (May 2014), pp. 706–720. ISSN: 01663615. DOI: 10.1016/j.compind.2014.02.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166361514000396 (visited on 02/12/2019).

[17] K. Ueda. "A concept for bionic manufacturing systems based on DNA-type information." In: *Human Aspects in Computer Integrated Manufacturing*. Ed. by G. J. Olling and F. Kimura. Amsterdam: Elsevier, Jan. 1992, pp. 853–863. ISBN: 978-0-444-89465-6. DOI: 10.1016/B978-0-444-89465-6.50078-8. URL: http://www.sciencedirect.com/science/article/pii/B9780444894656500788 (visited on 01/19/2019).

[18] J. Dias-Ferreira, L. Ribeiro, H. Akillioglu, P. Neves, and M. Onori. "BIOSOARM: a bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors." en. In: *Journal of Intelligent Manufacturing* 29.7 (Oct. 2018), pp. 1659–1682. ISSN: 1572-8145. DOI: 10.1007/s10845-016-1258-2. URL: https://doi.org/10.1007/s10845-016-1258-2 (visited on 02/12/2019).

[19] V. Botti and A. Giret. *ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems*. en. Springer Series in Advanced Manufacturing. London: Springer-Verlag, 2008. ISBN: 978-1-84800-309-5. URL: //www.springer.com/gp/book/9781848003095 (visited on 01/05/2019).

[20] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. "Reference architecture for holonic manufacturing systems: PROSA." In: *Computers in Industry* 37.3 (Nov. 1998), pp. 255–274. ISSN: 0166-3615. DOI: 10.1016/S0166-3615(98)00102-X. URL: http://www.sciencedirect.com/science/article/pii/S016636159800102X (visited on 01/09/2019).

[21] P. Leitão and F. Restivo. "ADACOR: A holonic architecture for agile and adaptive manufacturing control." en. In: *Computers in Industry* 57.2 (Feb. 2006), pp. 121–130. ISSN: 01663615. DOI: 10.1016/j.compind.2005.05.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166361505001077 (visited on 01/26/2019).

[22] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux. "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution." en. In: *Computers in Industry* 66 (Jan. 2015), pp. 99–111. ISSN: 01663615. DOI: 10.1016/

j.compind.2014.10.011. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166361514001894 (visited on 02/10/2019).

[23] I. Haman Tchappi, S. Galland, V. C. Kamla, and J. C. Kamgang. "A Brief Review of Holonic Multi-Agent Models for Traffic and Transportation Systems." In: *Procedia Computer Science*. The 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018) / The 13th International Conference on Future Networks and Communications (FNC-2018) / Affiliated Workshops 134 (Jan. 2018), pp. 137–144. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.07.154. URL: http://www.sciencedirect.com/science/article/pii/S1877050918311177 (visited on 02/10/2019).

[24] E. Missaoui, B. Mazigh, S. Bhiri, and V. Hilaire. "A Normative Model for Holonic Multi-agent Systems." In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. Boston, MA: IEEE, Nov. 2017, pp. 1251–1258. ISBN: 978-1-5386-3876-7. DOI: 10.1109/ICTAI.2017.00189. URL: https://ieeexplore.ieee.org/document/8372092/ (visited on 02/10/2019).

[25] I. Maganha, C. Silva, and L. M.D. F. Ferreira. "Understanding reconfigurability of manufacturing systems: An empirical analysis." In: *Journal of Manufacturing Systems* 48 (July 2018), pp. 120–130. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2018.07.004. URL: http://www.sciencedirect.com/science/article/pii/S0278612518302036 (visited on 01/13/2019).

[26] M. Bortolini, F. G. Galizia, and C. Mora. "Reconfigurable manufacturing systems: Literature review and research trend." In: *Journal of Manufacturing Systems* 49 (Oct. 2018), pp. 93–106. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2018.09.005. URL: http://www.sciencedirect.com/science/article/pii/S0278612518303650 (visited on 01/13/2019).

[27] D. Prasad and S. C. Jayswal. "Scheduling of Products for Reconfiguration Effort in Reconfigurable Manufacturing System." In: *Materials Today: Proceedings*. 7th International Conference of Materials Processing and Characterization, March 17-19, 2017 5.2, Part 1 (Jan. 2018), pp. 4167–4174. ISSN: 2214-7853. DOI: 10.1016/j.matpr.2017.11.679. URL: http://www.sciencedirect.com/science/article/pii/S2214785317329528 (visited on 01/13/2019).

[28] O. A. Makinde, K. Mpofu, and A. P. I. Popoola. "Review of the Status of Reconfigurable Manufacturing Systems (RMS) Application in South Africa Mining Machinery Industries." In: *Procedia CIRP*. Variety Management in Manufacturing 17 (Jan. 2014), pp. 136–141. ISSN: 2212-8271. DOI: 10.1016/j.procir.2014.02.035. URL: http://www.sciencedirect.com/science/article/pii/S2212827114003679 (visited on 02/10/2019).

[29] M. Onori and J. Barata. "EVOLVABLE PRODUCTION SYSTEMS : MECHATRONIC PRODUCTION EQUIPMENT WITH PROCESS-BASED DISTRIBUTED CONTROL." In: *IFAC Proceedings Volumes*. 9th IFAC Symposium on Robot Control 42.16 (Jan. 2009), pp. 80–85. ISSN: 1474-6670. DOI: 10.3182/20090909-4-JP-2010.00016. URL: http://www.sciencedirect.com/science/article/pii/S1474667015306169 (visited on 01/14/2019).

[30] J. C. Chaplin, O. J. Bakker, L. de Silva, D. Sanderson, E. Kelly, B. Logan, and S. M. Ratchev. "Evolvable Assembly Systems: A Distributed Architecture for Intelligent Manufacturing." In: *IFAC-PapersOnLine*. 15th IFAC Symposium onInformation Control Problems inManufacturing 48.3 (Jan. 2015), pp. 2065–2070. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2015.06.393. URL: http://www.sciencedirect.com/science/article/pii/S2405896315006321 (visited on 01/14/2019).

[31] M. Onori, D. Semere, and B. Lindberg. "Evolvable Systems: An Approach to Self-X Production." In: *Int. J. Computer Integrated Manufacturing* 24 (May 2011), pp. 506–516. DOI: 10.1007/978-3-642-10430-5_61.

[32] A. Rocha, G. Di Orio, J. Barata, N. Antzoulatos, E. Castro, D. Scrimieri, S. Ratchev, and L. Ribeiro. "An agent based framework to support plug and produce." en. In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. Porto Alegre RS, Brazil: IEEE, July 2014, pp. 504–510. ISBN: 978-1-4799-4905-2. DOI: 10.1109/INDIN.2014.6945565. URL: http://ieeexplore.ieee.org/document/6945565/ (visited on 01/26/2019).

[33] J. A. B De Oliveira. *Coalition Based Approach for Shop Floor Agility – A Multiagent Approach*. en. 2003. URL: http://hdl.handle.net/10362/2483.

[34] R. Henzel and G. Herzwurm. "Cloud Manufacturing: A state-of-the-art survey of current issues." In: *Procedia CIRP*. 51st CIRP Conference on Manufacturing Systems 72 (Jan. 2018), pp. 947–952. ISSN: 2212-8271. DOI: 10.1016/j.procir.2018.03.055. URL: http://www.sciencedirect.com/science/article/pii/S2212827118301574 (visited on 01/20/2019).

[35] W. He and L. Xu. "A state-of-the-art survey of cloud manufacturing." en. In: *International Journal of Computer Integrated Manufacturing* 28.3 (Mar. 2015), pp. 239–250. ISSN: 0951-192X, 1362-3052. DOI: 10.1080/0951192X.2013.874595. URL: http://www.tandfonline.com/doi/abs/10.1080/0951192X.2013.874595 (visited on 01/20/2019).

[36] Y. Liu, L. Wang, and X. Vincent Wang. "Cloud manufacturing: latest advancements and future trends." In: *Procedia Manufacturing*. Proceedings of the 8th Swedish Production Symposium (SPS 2018) 25 (Jan. 2018), pp. 62–73. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2018.06.058. URL: http://www.sciencedirect.com/science/article/pii/S2351978918305778 (visited on 01/20/2019).

[37] P. O'Donovan, C. Gallagher, K. Bruton, and D. T. O'Sullivan. "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications." en. In: *Manufacturing Letters* 15 (Jan. 2018), pp. 139–142. ISSN: 22138463. DOI: 10.1016/j.mfglet.2018.01.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S2213846318300087 (visited on 01/24/2019).

[38] P. Zhang, M. Zhou, and G. Fortino. "Security and trust issues in Fog computing: A survey." en. In: *Future Generation Computer Systems* 88 (Nov. 2018), pp. 16–27. ISSN: 0167739X. DOI: 10.1016/j.future.2018.05.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X17329722 (visited on 01/24/2019).

[39] R. Mahmud, R. Kotagiri, and R. Buyya. "Fog Computing: A Taxonomy, Survey and Future Directions." en. In: *Internet of Everything*. Ed. by B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito. Singapore: Springer Singapore, 2018, pp. 103–130. ISBN: 978-981-10-5860-8 978-981-10-5861-5. DOI: 10.1007/978-981-10-5861-5_5. URL: http://link.springer.com/10.1007/978-981-10-5861-5_5 (visited on 01/24/2019).

[40] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog computing and its role in the internet of things." en. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. Helsinki, Finland: ACM Press, 2012, p. 13. ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. URL: http://dl.acm.org/citation.cfm?doid=2342509.2342513 (visited on 01/24/2019).

[41] S. Alamgir Hossain, M. Anisur Rahman, and M. A. Hossain. "Edge computing framework for enabling situation awareness in IoT based smart city." en. In: *Journal of Parallel and Distributed Computing* 122 (Dec. 2018), pp. 226–237. ISSN: 07437315. DOI: 10.1016/j.jpdc.2018.08.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0743731518306130 (visited on 01/29/2019).

[42] M. S. Mahmoud. "Multiagent Systems." en. In: *Advanced Control Design with Application to Electromechanical Systems*. Elsevier, 2018, pp. 313–338. ISBN: 978-0-12-814543-2. DOI: 10.1016/B978-0-12-814543-2.00007-2. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780128145432000072 (visited on 01/26/2019).

[43] P. Leitão. "Agent-based distributed manufacturing control: A state-of-the-art survey." en. In: *Engineering Applications of Artificial Intelligence* 22.7 (Oct. 2009), pp. 979–991. ISSN: 09521976. DOI: 10.1016/j.engappai.2008.09.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197608001437 (visited on 01/26/2019).

[44] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie. "Applications of agent-based systems in intelligent manufacturing: An updated review." en. In: *Advanced Engineering Informatics* 20.4 (Oct. 2006), pp. 415–431. ISSN: 14740346. DOI: 10.1016/j.aei.2006.05.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474034606000292 (visited on 01/26/2019).

[45] L. Ribeiro, A. Rocha, and J. Barata. "A product handling technical architecture for multiagent-based mechatronic systems." en. In: *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*. Montreal, QC, Canada: IEEE, Oct. 2012, pp. 4342–4347. ISBN: 978-1-4673-2421-2 978-1-4673-2419-9 978-1-4673-2420-5. DOI: 10.1109/IECON.2012.6389190. URL: http://ieeexplore.ieee.org/document/6389190/ (visited on 01/26/2019).

[46] L. Ribeiro, J. Barata, M. Onori, C. Hanisch, J. Hoos, and R. Rosa. "Self-organization in automation - the IDEAS pre-demonstrator." en. In: *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*. Melbourne, Vic, Australia: IEEE, Nov. 2011, pp. 2752–2757. ISBN: 978-1-61284-972-0 978-1-61284-969-0 978-1-61284-971-3. DOI: 10.1109/IECON.2011.6119747. URL: http://ieeexplore.ieee.org/document/6119747/ (visited on 01/26/2019).

[47] B. Katti, C. Plociennik, and M. Schweitzer. "SemOPC-UA: Introducing Semantics to OPC-UA Application Specific Methods." In: *IFAC-PapersOnLine*. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018 51.11 (Jan. 2018), pp. 1230–1236. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.422. URL: http://www.sciencedirect.com/science/article/pii/S2405896318315489 (visited on 01/27/2019).

[48] M. Schleipen, S.-S. Gilani, T. Bischoff, and J. Pfrommer. "OPC UA & Industrie 4.0 - Enabling Technology with High Diversity and Variability." en. In: *Procedia CIRP* 57 (2016), pp. 315–320. ISSN: 22128271. DOI: 10.1016/j.procir.2016.11.055. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827116312094 (visited on 01/27/2019).

[49] OMG. *DDS Portal – Data Distribution Services*. URL: https://www.omgwiki.org/dds/ (visited on 01/27/2019).

[50] OMG. *About the Data Distribution Service Specification Version 1.4*. URL: https://www.omg.org/spec/DDS/1.4/ (visited on 01/27/2019).

[51] *Modbus FAQ*. URL: http://www.modbus.org/faq.php (visited on 01/29/2019).