SCHOOL OF ENGINEERING

CONTROL & AUTOMATION ENGINEERING DEGREE FINAL PAPERWORK

New Technologies Applied to Production Systems

Author: João Ricardo Cardoso

Advisor: Prof. Dr. Marcelo Götz

Porto Alegre, July 2017.

Summary

Thanks

This is just an attempt to show my appreciation for all those who have been by my side during all these years. This work would not be possible without the support and unconditional love of my parents Osni and Neiva, my brothers Lázaro and Manoel, grandma Cleci, uncle Nelson and my girlfriend, Manu. I would also like to thank my roomates Gui, Jeff and Nico for all the good moments, and my professional colleagues for the given lessons and authentic collaboration for this work ABC, Moa, Posser and Zani. And last but not least, my advisor Dr. Marcelo Götz for having this work guided by his vast scientific knowledge.

I believe in the power that the circumstances have in life, and I'm very privileged to say that those people have provided a comfortable, exciting and, above all, inspiring environment so I could develop this work with the maximum of my sanity, and finish my academic trajectory with the desire to contribute to the well-being of society through technological advances.

"What we usually consider as impossible are simply engineering problems...

There is no law of physics preventing them."

Michio Kaku

Abstract

The concept of Industry 4.0 was recently introduced as a reference of the fourth industrial revolution, which has its origin mainly at the recent changes in the dynamics of consumption. Those changes demand more efficiency in the production's processes, as well as more diversity and flexibility on the list of products offered by companies. One of the central elements in Industry 4.0 is the Internet of Things paradigm, which has been intensively discussed due to, among other reasons, the mass diffusion of devices capable of connecting to the Internet, and the many, and still not explored, possibilities of application, including the industrial area. The purpose of this work is to develop a platform that is capable of integrating devices, operators and production demands into a flexible manufacturing system. The rules governing the system are based on the concepts of Internet of Things, Multi-Agent Systems and Service Oriented Architecture. The platform is developed based on programming languages HTML and JavaScript, uses a tool called Node, and uses the MQTT as communication protocol. In addition, an application of the supervisory software Elipse E3 controlling a Dexter µDx100 PLC is used, representing the interaction of industrial equipment with the system; To represent the interaction of embedded devices it is used the prototyping platform Arduino UNO equipped with a Wi-Fi Shield. At the end, the functionalities obtained with this system, its limitations and the aspects that could be improved in future works are discussed.

Keywords: Industry 4.0, Internet of Things, Multi-Agent Systems, Service Oriented Systems, MQTT, Web programming.

List of figures

List of abbreviations and acronyms

CoAP – Constrained Application Protocol

CPS – Cyber-Physical Systems

CRM – Customer Relationship Management

ERP – Enterprise Resource Planning

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IoT – Internet of Things

IVACE – Instituto Valenciano de Competitividad Empresarial

JSON – JavaScript Object Notation

MAS – Multi-Agent System

MES - Manufacturing Execution System

MIT – Massachusetts Institute of Technology

MQTT – Message Queue Telemetry Transport

MRP – Manufacturing Resource Planning

M2M – Machine to Machine

OEE – Overall Equipment Effectiveness

OPC – Object Linking and Embedding for Process Control

PLC – Programmable Logic Controller

RAM – Random access memory

RFID – Radio-Frequency Identification

ROM – Read only memory

SOA – Service Oriented Architecture

TCP – Transmission Control Protocol

URI – Uniform Resource Identifier

XML – Extensible Markup Language

XMPP – Extensible Messaging and Presence Protocol

1 Introduction

Society has recently witnessed a transition from the times when computing was generaly performed almost exclusively by desktop machines, to the times when the vast majority of computers on the planet is available in versions that fit in the palm of people's hands. In projection to this confirmation, it is possible to infer that in the near future there will be microcomputers integrated with many of the things that are part of our daily lives.

These changes in the computer use dynamics imply in the generation of a huge amount of data that must be stored, processed and presented to users. It is at this point that one of the great challenges arises: performing all these tasks efficiently and presenting the results concisely and clearly.

Watching the latest strategies and the destination of the investments of large technology companies, the computational infrastructure that will support these changes will be provided by remote servers - cloud computing. According to a survey conducted by Professor Octavian Prosperous Sanchez of Fundação Getúlio Vargas, the adoption of cloud computing is seen as an opportunity to reduce the investments in information technology (SANCHEZ, 2012). Through cloud infrastructure it will be possible to connect monitoring devices, storage devices, production systems, analytical tools, command and viewing platforms. This model will consist of a wide range of services provided by virtual system agents, similar to what occurs with the traditional service market, where each service has attributes such as cost and delivery time (GUBBI, 2013). These services will be negotiated via M2M (machine-to-machine) communication and will provide access to on-demand applications to users from anywhere.

According to KAGERMANN et al., 2013, in the future companies will establish global networks incorporating their machinery, warehouse and production facilities in the form of CPS (Cyber-Physical Systems). The CPS concept was first defined in 2006 by Dr. James Truchard, CEO of National Instruments, based on the virtual representation of a manufacturing process. In the production environment, these CPS includes machines, inventory and production systems capable of exchanging information between themselves autonomously, allowing production to be controlled independently. The rigid centralized control systems grant now their place to decentralized intelligence, with M2M communication on the shop-floor (BRAGA, 2014).

These new technologies fall into the concept of Industry 4.0, that have the goal of bringing more manufacturing flexibility for mass customization, besides ensuring higher quality and productivity for manufacturing systems. However, to achieve these attributes, as mentioned at DAVIES, 2015, companies will need to invest in equipment, information and communication technologies, as well as data analysis tools and integration of data flows across the value chain.

One of the indications that companies worldwide, including in Brazil, are already investing in the future predicted by these new approaches, is the initiative of the Brazilian company Elipse Software, which has recently initiated a project to develop a system able to provide to the industry the benefits brought to the methods and technologies present

in the concept of Industry 4.0. This project has as one of its contributors the author of this work.

Analyzing the content exposed so far, it's natural to think that the industry requires theoretical contribution, and it is in this context that fits one of the academic production's goals, which is to provide knowledge to generate benefits to society through technological advance.

1.1 Goals

The general goal of this work is to develop a platform for partial management of the production of a flexible manufacturing system in a simulated environment, using some of the concepts that are usually present in 4.0 Industry prospects, as Internet of Things, Multi-Agent Systems and Service-oriented Architecture. Complementing the scope of this work, there is the task of exploring the challenges encountered during the modeling and development of this platform, validating the used concepts, evaluating its features and identifying the limitations of the developed system.

The specific goals of this study are: to develop a software that provides communication service among the other elements of the system; develop a software that is able to carry out production orders; develop a firmware for embedded systems that represent an industrial equipment of the manufacture system; develop a software for users to view and control the features of the equipment; and, finally, evaluate the performance of the system, its limitations, and identify the challenges during its development.

1.2 Structure of the work

Chapter 2 presents a bibliographical review about the topics which the project is based on: 4.0 Industry; Internet of Things; Multi-Agent Systems; and Service-oriented Systems. Chapter 3 presents the methods and tools used for the implementation of the platform proposed in this work. Chapter 4 describes the designed architecture and the development of each of the system components. Chapter 5 contains the presentation of the developed platform and results obtained with the project. Finally, Chapter 6 contains an assessment related to the achievement of the goals, the conclusions about the work and the proposals for future work.

2 Literature review

2.1 Industry 4.0

The term "Industry 4.0" was created by a working group convened in 2012 by the Federal Ministry of Education and Research of Germany to develop strategic recommendations for technological innovation for the German industry. The Group was led by Prof. Dr. Henning Kagermann, current Director of the National Academy of Science and Engineering (acatech) and Dr. Siegfried Dais, from Bosch, amongst several researchers of German institutions such as the Fraunhofer Institute, the Technical University of Munich, and several members of companies such as BMW, ThyssenKrupp, and Festo. The result of the Working Group was the document entitled "Recommendations for implementing the strategic initiative INDUSTRIE 4.0". According to DAIS, 2014, Industry 4.0 is currently one of the most discussed topics among scholars of Germany.

Powerful stand-alone microcontrollers (embedded systems) have been increasingly connected to each other and to the internet. This is the result of the convergence between the physical and the virtual world in the form of CPS. In the territory of manufacturing systems, those technological developments can be described as the fourth stage of industrialization, or Industry 4.0. Figure 2.1 illustrates the four stages of industrialization, according to KAGERMANN et al., 2013.

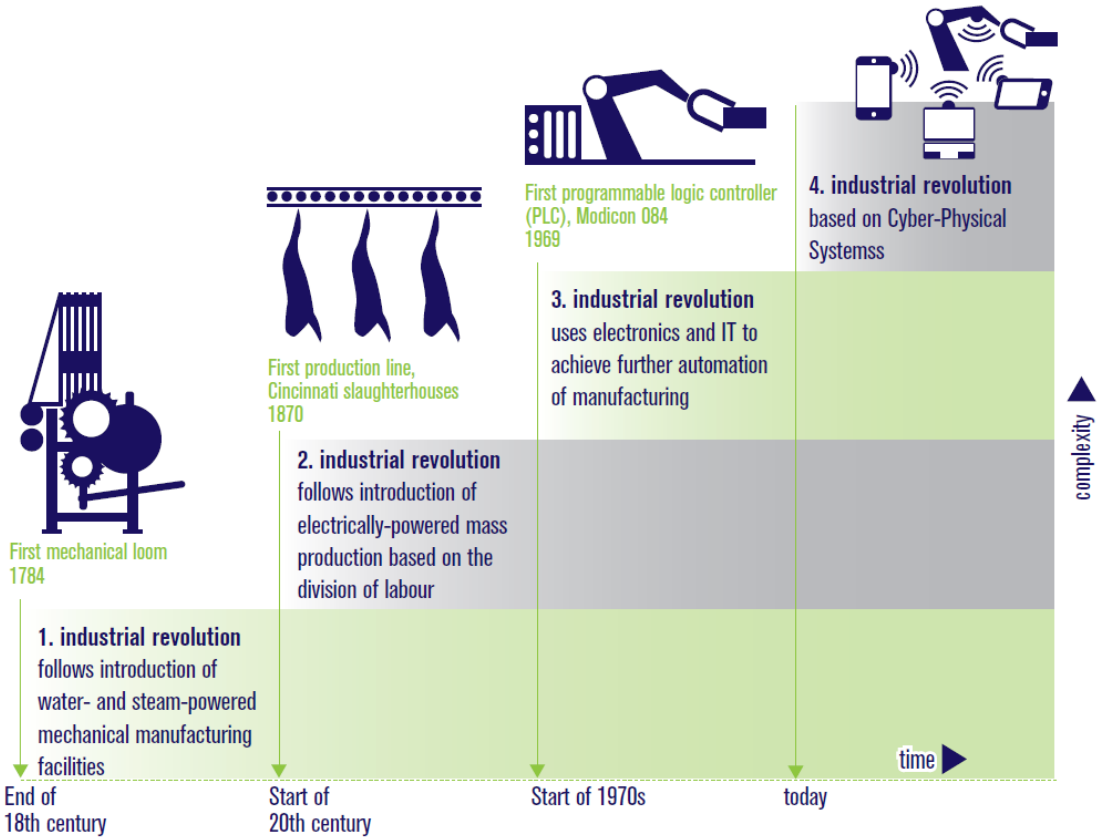

Figure 2.1 – Four stages of industrial revolution – source: KAGERMANN et al., 2013.

According to HERMANN et al., 2015, Industry 4.0 is a collective term for technologies and concepts for value chain organizations. This same work mentions the four elements that are more frequently repeated in the Industry 4.0 definitions found on the literature, they are: Cyber-Physical Systems; Internet of things; Internet of services; and Smart Factories.

In Smart Factories with modular structure, CPS monitors physical processes and carries out decentralized decisions. Via the Internet of Things, CPS communicates and cooperates with each other and with humans in real time. Through its Service-oriented Architecture, organizational processes and production processes are offered and used by participants in the value chain.

Smart factories constitute a completely new approach to production systems, where smart products are universally identifiable, and can be accessed at any time, as well as their own processes, their current state and its history, besides the alternative routes to reach the desired state. Manufacturing embedded systems are connected vertically with production processes under factories and businesses, and can be managed in real time, since the sales order to the output of the product in the logistics (KAGERMANN et al., 2013).

According to HERMANN et al., 2015, there are six Industry 4.0 design principles, which represent the desired attributes by companies that want to adopt such philosophy. These principles guide companies to identify and implement the scenarios forseen in the Industry 4.0. They are:

- Interoperability: the ability of the CPS, the humans and the Smart Factories to connect and communicate with each other through the Internet of Things and cloud computing.

- Virtualization: a virtual copy of the Smart Factories is created by interconnected data sensors (that monitor physical processes) with virtual plant models and simulation models.

- Decentralization: the ability of the CPS to make decisions without human intervention.

- Real-time capability: the ability to collect and analyze data and deliver knowledge derived from these analyses during the operation.

- Service orientation: offering of services (CPS, either human or Smart Factories') through cloud computing.

- Modularity: flexible adaptation for Smart Factories changing requirements through replacement or expansion of individual modules.

A clear case of data application aiming to internally improve a production process would be the predictive maintenance of a production line. In this case the data would be collected from different sensors to monitor certain attributes of the equipment, especially those with high working load. This would facilitate the learning of the patterns that lead to known states of equipment and fire alarms with enough advanced time to apply a planned maintenance action, instead of applying a reactive solution. That way it

would be possible to increase the useful life of equipment components, adjust the stock of spare parts to what is strictly necessary and significantly reduce the percentage of unplanned stoppages in production line. With this is possible to migrate from the break-fix perspective to a predict-prevent approach.

The SAIN4 is a project funded by the Instituto Valenciano de Competitividad Empresarial and the European Union through the Fondo Europeo de Desarrollo Regional. The document entitled "Informe sobre el Estado del Arte de la Industria 4.0" (IVACE, 2016), shows practical examples of implementation of some concepts Industry 4.0 at all levels, including also the concerning of people as one of the principal axes of the industry of the future. Some of these application examples can be seen at the attachments of this paper.

In the analysis performed by IVACE, 2016, the key to this new model business is the inclusion of the client as part of the value chain. This way is a cycle in which the use of information products feeds the production chain, allowing both groups - consumers and producers - to benefit. The consumer will improve the user experience, having access to low-cost products with higher added value, and the producer will have reliable information to improve the design of existing products and create new solutions to reach new markets. However, before implementing any kind of change, especially those involving significant investments, it is necessary to perform a rigorous analysis in relation to the benefits that are wanted and to solutions that serve best to the foreseen scenario. It is extremely important to be cautious because, as mentions DAVIES, 2015, not all observers are convinced of the value that the Industry 4.0 will add to the means of production. Some people think that the Industry 4.0 as a concept is poorly defined and suffers from exaggerated expectations. Others, however, believe that the fully digitized products and the value chains are still a "dream".

2.2 Internet of Things

The concept of Internet of things - IoT - stands for several advances on emerged areas such as embedded systems, microelectronics, information technology and communication (SANTOS et al., 2015). This concept is considered as the third wave of information technology, after the internet and mobile communication network (ZHU et al., 2010).

The term “Internet of things” was coined in 1999 by Kevin Ashton, a Briton who was one of the founders of the Auto-ID Center at MIT. This laboratory has developed pioneering research projects in the area of semantics and identification for networked objects, especially with Radio Frequency Identification (RFID) technologies.

According to a study led by Gartner (GARTNER, 2013), which is a company specialized in researches in the area of information technology, the Internet of things ranks as one of the ten major trends in technological strategies. The Cisco Internet Business Solutions Group projects that by 2020 there will be about 50 billion devices connected to the internet (EVANS, 2011), as illustrated in Figure 2.2.

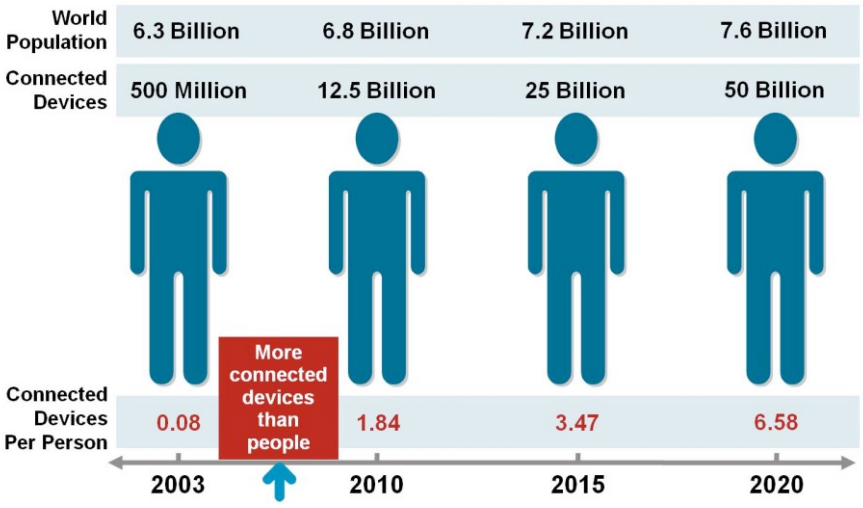

Figure 2.2 – Number of devices connected to the internet – source: EVANS, 2011.

With the launch of the IPv6 protocol in 2012 – that is the current version of Internet Protocol - there are enough available addresses to support direct communication between devices via the internet. This means that, for the first time in the history, it is possible to connect information resources, objects and people, creating the Internet of things. This phenomenon will also strongly impact the industry sector (KAGERMANN, 2013).

It is interesting to note that any real-world entity can be represented in the field of Internet of things, as long as it is able to play any reaction from a stimulus, transmit any type of information or have some attribute that can be measured and reported through an internet-connected device. In this context, an element in the Internet of things is not necessarily the device that connects to the internet, but the entity that it represents. For example, one can represent a fleet of automobiles through their respective location data, using a device in each vehicle that is able to collect the geographic coordinates of the vehicle and report them on the internet. From the point of view of the representation of the automobile, it makes no difference if the device that performs the tasks is the on-board computer, a small embedded system installed in the car, or even some passager's mobile device, in any of these three scenarios the representation of automobile applies only to the set of its location data.

One of the most cited works about IoT is ATZORI et al., 2010, which defines it as the intersection of three paradigms: internet orientation; things orientation; and semantic orientation. It is assumed that this kind of definition is necessary due to the interdisciplinary nature of the subject, but the usefulness of the Internet of things can be triggered only in the intersection of the three paradigms (GUBBI et al., 2013). A representation of this approach can be seen in Figure 2.3.

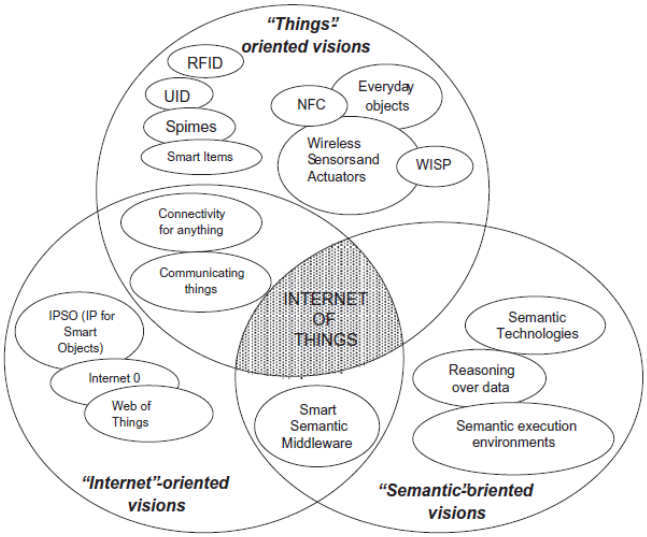

Figure 2.3 – The three paradigms in Internet of things – source: ATZORI et al., 2010.

In the definition of GUBBI et al., 2013, IoT is more user-centric and not restricted to communication protocols. Thus, for this author, IoT is "the interconnection of devices, sensors and actuators with the ability to share information from different platforms through a unified network, developing an operating common vision and so enable innovative applications. This is achieved by the integration of ubiquitous computing, analytical systems and information representation, using cloud computing as a link."

2.3 Multi-Agent Systems

Multi-Agent systems, according to FERBER, 1999, is a software technology that is able to model and implement individual and social behavior in distributed systems. According to PECHOUCEK et al., 2008, the subsystems, now called agents, are active, and can not only post their services and submit requests of its functionalities, but take a proactive role to initiate communication with other agents, suggest negotiations and allocate services.

The theory of agents aims at understanding what an agent is, its main properties and how to formalize them. In RUSSELL, 2010, there is the following definition: "an agent is all that can be considered able to recognize its environment through sensors and act on its environment through actuators."

Multi-Agent Systems implementations allow arrays in the form of auto-organized systems, in which agents interact with each other and form coalitions to generate productive flows, determine its restrictions, its own monitoring and make decisions autonomously in the production process. The agents adopt this behavior in order to seek together the best solution to meet the demands of production. This happens dynamically, to each product to be rendered, in every stage of production, making decisions and productive arrangements more assertive.

This approach aims to meet a need to quick responses to new demands of the market, where hierarchy systems typically are not able to attend (ONORI et al., 2011). Figure 2.4 presents an abstraction of services trading in a Multi-Agent System.

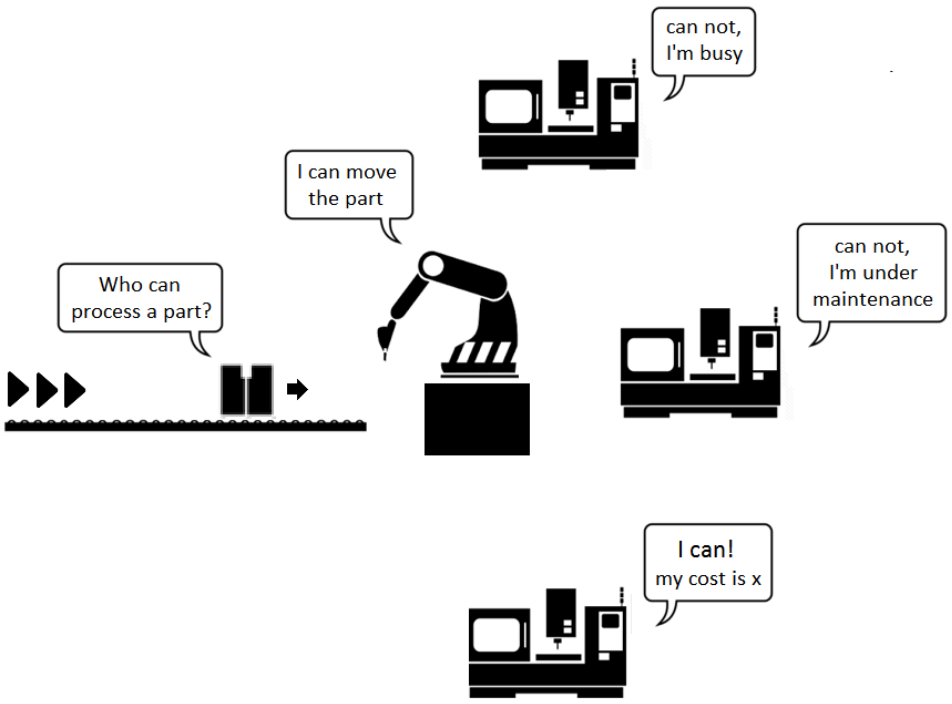

Figure 2.4 – Negotiation of services in a Multi-Agent System.

It is interesting to note that this type of system is characterized as an alternative approach comparing to traditionally hierarchical structures found nowadays in the industry, composed of systems like ERP (Enterprise Resource Planning), MRP (Manufacturing Resource Planning) and MES (Manufacturing Execution System), which sometimes do not interact in a practical way. These alternatives are seeking, among other aspects, to fill this connectivity gap between the various management platforms currently deployed in the industry.

Self-organizing systems are not limited to meet static production flows in the most efficient way, but to meet the need of the product and the production system in a balanced form, without having to worry that it is an element that has already being produced earlier. In short, the self-organization meets the diversity of production aiming to meet, in the limit, the need for products with unique batch.

As quoted by PEIXOTO, 2016, the agents are endowed with social skills, i.e. have the ability to communicate with other agents. This interaction is accomplished through a communication language between agents. In the literature, these languages are referred to as Agent Communication Languages and among the institutions that stand out in the work with the theory of the agents is the Foundation for Intelligent Physical Agents (FIPA). FIPA was founded in 1996 by a non-profit association to develop a collection of rules concerning the technology of software agents. The initiative came from a group of academic and industrial organizations which drew up a set of bylaws to guide the creation of a set of standard specifications for software agent technology. At that time, the software agents were already very well known in the academic community, but they have only received limited attention from commercial enterprises, as well as an exploratory perspective. The consortium agreed to produce standards that would form the basis of a new industry, which should be usable in a wide range of possible applications.

From the reading of some literary works about the topic, it was concluded that one of the most widely used software for implementing Multi-Agent Systems logic from the guidelines of the FIPA is called Jade (JADE, 2017), which is a native system programming Java environment. It is worth mentioning that the platform developed in this paper uses some of the functionality present in Jade, among other mechanisms, as a reference:

a) Agent's identification Service: recognizes each agent that is inserted in the system and identifies it with a unique name, making the list of names for other agents who wish to see which ones are acting on the platform;

b) Yellow-pages service: area where agents can post their services and also see which agents perform a particular service;

c) Messaging transport: allows one to interfere in the messages exchange between agents, serving as a good resource for identifying problems in implementation;

d) Analysis of the communication service: is a verifier of the messages exchange between agents who are on the platform;

e) Interaction FIPA protocols library: provides classes with interaction requirements, facilitating the implementation of the agents.

2.4 Service Oriented Architecture

The question "what is Service-oriented Architecture?" is answered in SCOVINE, 2006, as follows: it is an architecture that enables new applications to be created from a combination of features, so called services. These services may be new developments or created from the exposure of functionality of existing applications. Once created, a service can be reused or recombined in the construction of various systems. For every new system implemented, the number of available components is likely to increase. With the reuse of services, the effort to develop the next systems decreases. A good analogy are the assembling building blocks. It is possible to combine blocks of different shapes and colors to create castles, cars, boats, among others. The services can be compared to the blocks and the combinations generated to join them can be compared to the applications. As well as the same blocks creates a great variety of forms, the same services can create great variety of applications (PEIXOTO, 2016).

Service-oriented Architecture - SOA - is a concept in which a manufacturing system can be seen as a set of systems that offer services that complement each other to form a productive process (MENDES et al., 2008). Each system within SOA provides services independently from the others and, when connected together, they provide services to each other (PEIXOTO, 2016). Figure 2.5 shows an overview of SOA principles, in which there is a service offering publication, as well as a request from this same service.

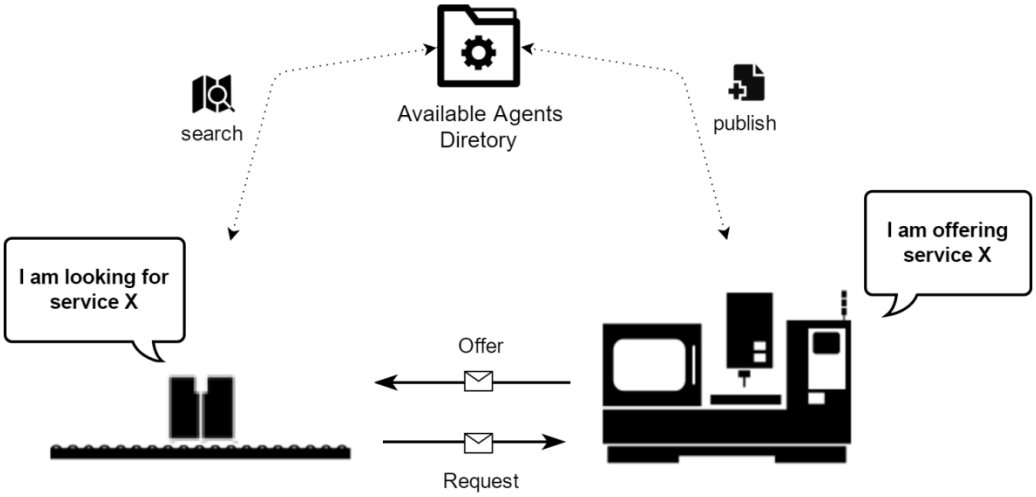

Figure 2.5 – SOA structure.

A very important element in SOA is the discovery mechanism (Available Agents Directory) whose representation can be observed in Figure 2.5. There are several methods to implement this mechanism, of which stands out the Yellow Pages service, which allows agents to be registered and to publish descriptions of one or more services. This way the other agents of the system can easily discover and request them, thus giving continuity to the production flow autonomously according to information collected in real time.

Thus, in an SOA each subsystem is autonomous and the process flow management depends on the availability of services and equipment addition or removal when necessary. However, the synchronism of this management is not trivial. MENDES et al., 2008, foresees the challenge when describing the processes that govern the system's behavior, synchronize and coordinate the execution of the services offered by distributed entities in order to achieve the desired behavior.

3 Materials and methods

3.1 Communication protocol

One of the most important aspects for the integration of a wide range of devices in a network is the standardization of the communication. In industry, as well as on the Web, there is a huge variety of communication protocols, each one designed to meet the specific requirements of the application to which it intends to serve.

In internet oriented projects, the IP protocol has been the most used one for addressing machines. However, it is necessary to define an application-level protocol to be used on the addressing layer. Among various existing application protocols is the Hypertext Transfer Protocol (HTTP), the CoAP (Constrained Application Protocol), the XMPP (Extensible Messaging and Presence Protocol) and the MQTT (Message Queue Telemetry Transport), as protocols that meet the specific application needs on the internet.

HTTP is the most widely used application protocol on the internet. It has a wide list of components and a powerful authentication system, thus enabling a huge amount of applications. All this apparatus provides the needed standardization to manage the huge flow of information that travels currently through the internet. However, its basic structure sometimes is too extensive for applications involving low-capacity devices.

CoAP is a transfer protocol specialized in internet applications that uses limited capacity devices and networks. The mentioned devices have often eight-bit microcontrollers with small amounts of ROM (Read Only Memory) and RAM (Random Access Memory), while the nets are performed in low-power devices, with high rates of error and a very limited throughput. In this context, the CoAP is designed for applications that use M2M communication, such as smart energy and building automation. This Protocol provides a request/response interaction model between the system's endpoints, supports service discovery and includes key Web concepts, such as URIs (Uniform Resource Identifiers) and some types of media transfer. The CoAP is designed to easily interact with the internet through the HTTP protocol, while keeping important attributes such as multicast support, low overhead and simplicity for constrained environments (SHELBY, 2016).

The XMPP was originally developed as a protocol for messages to connect people through text messages. This protocol uses XML packages to text format, and Its main feature is to use a structure of type "name@domain.com" as addressing scheme, thus offering an easy way to identify devices on an IoT network.

The MQTT was born as a project within the American company IBM (HIVEMQ, 2017), and was designed for telemetry applications - remote monitoring. Typically, this Protocol is used to connect large networks of small devices that will be monitored or controlled by a cloud service (Bauer et al. 2010). Figure 3.1 shows a diagram that represents the dynamics of messaging in a MQTT environment.

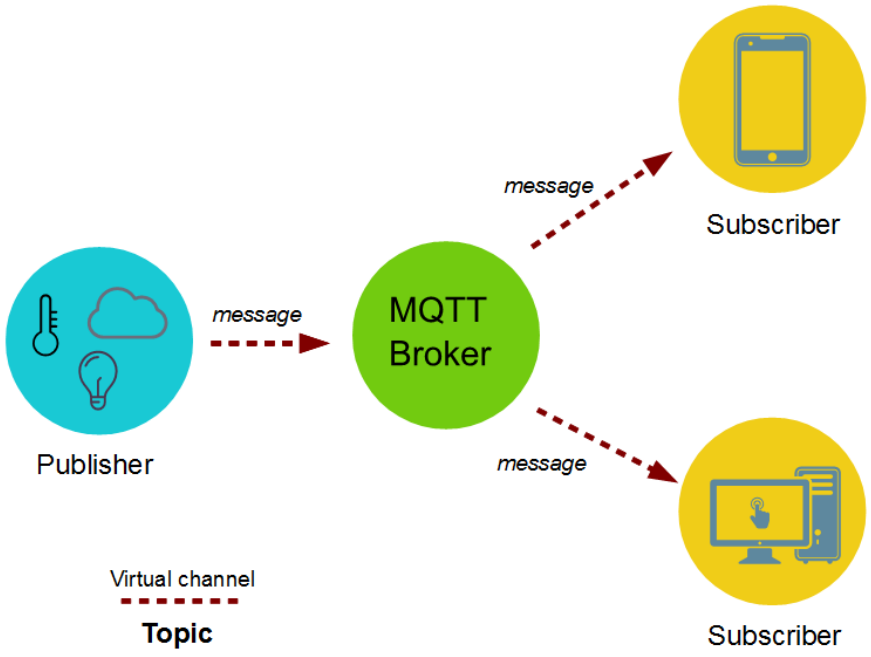

Figure 3.1 – Operating diagram of MQTT – source: SWA, 2017.

As can be seen in Figure 3.1, those interested in sending some information (Publishers) must post it in a topic – that serves as the address of the message. Those interested in receiving any information (Subscribers) should subscribe to the topic in which the same message shall be published. The entity that performs the routing of the messages is the server (Broker), through which travels the entire data stream.

Since data integrity is an essential point for telemetry systems, the protocol operates on TCP, which provides a simple and reliable mechanism for message exchange. Within the context of that MQTT, Publishers are customers of a TCP connection with the Broker, which also maintains the same kind of connections to the Subscribers.

Through an attributes comparison of the considered protocols in this section, it was decided to opt for the use of the HTTP and MQTT protocols, whose union properly supplies the scope of this project.

3.2 Node.js

Node is an event-driven asynchronous development environment, which was designed to create network applications with a large flow of information. This system is run by the operating system, and presents a cycle of events at runtime. The project is open source, and Its architecture is similar to systems such as Event Machine of Ruby language (EVENT MACHINE, 2017) and the Twisted of Python language (TWISTED, 2017), implementing a virtual machine that provides the environment for running applications, in the case of Node in JavaScript language.

In other systems, there is always a blocking call to start the event loop, and normally Its behavior is controlled through call-backs. In the Node environment there is a call to start the event loop. The applications simply enter in the event loop after running the script, and exits the loop when there are no more call-backs to run (NODE, 2017).

The logic found in Node contrasts with the ones traditionally found in operating systems. Rather than creating a new thread for each connection (and allocate memory attached to it), for each connection raises an event that is executed on the Node processes. In this type of behavior, the event loop is hidden from the user.

The library chosen to develop the MQTT server is called Mosca. This project is open source, was developed by Matteo Collina, and can be found on his website (COLLINA, 2017). Mosca library implements a MQTT broker, that is, provides communication services for any device connected to it, and allows programmers to develop a structure to manage the messages flow through event monitoring.

In addition, library Express will be used (EXPRESS, 2017), which is also open-sourced, and implements a server for Web pages (HTTP requests).

To store the data, it will be used MongoDb (MONGODB, 2017), which is an open source software that implements a document-oriented database. In addition to this structure it will be used the Mongoose library (MONGOOSE, 2017), which works as a driver between MongoDb and Node applications.

3.3 Web programming

The HTML (Hypertext Markup Language) is a markup language used to build the layout of Web pages.

JavaScript is an interpreted programming language, that is used to execute scripts on the client's machine, perform asynchronous communication, and change the content of the HTML document displayed in the browser of the users.

Although there are other programming languages serving the same purpose, together HTML and JavaScript standards constitute the vast majority of Web content that runs on today's browsers. By presenting a favorable environment for internet applications, this pair of languages will be used for the development of some of the software pieces that integrates the platform proposed in this work.

JSON (JavaScript Object Notation) is the notation for objects from the JavaScript language. A JSON structure consists of a set of properties, each of which is represented in such a way that on the left is its name and on the right its contents. This content can be a simple variable, of integer or string type, for example; an array of variables (delimited by "[]" and separated by ","); or another object (enclosed in "{}" and separated by ","). Figure 3.2 shows an example of a JSON structure.

```
person = {
            firstName: "john",
            lastName: "richard",
            age: 27,
            favoriteFruits: [ "watermelon" , "mango" ],
            friends: [
                        {
                            firstName: "nick",
                        },
                        {
                            firstName: "jeff",
                        },
                        ...
                    ]
        };
```

Figure 3.2 – Object notation in JavaScript language.

As can be seen in Figure 3.2, the "person" object contains five properties, the first two (firstName and lastName) has type string; the third (age) has type integer; the fourth (favoriteFruits) is an array of strings; and the fifth (friends) is an array of other objects.

The JSON structures will be used in this paper to represent the virtual entities of the agents.

To give the browser the ability to communicate with other agents in the system, it will be used MQTT.js library (MQTT, 2017), which is an open source project that implements the client-side environment for MQTT communication.

3.4 Embedded system

Electronic devices sometimes contain microcontrollers which allow the execution of tasks through a programmable algorithm, which is often referred to as embedded systems. This term is often used to refer to systems with low computational capacity, dedicated to specific tasks.

To represent the interaction of embedded systems with the platform developed in this work, will be used an Arduino UNO prototyping board. This board is composed by, among other components, an Atmel microcontroller model Atmega328p of eight bits, a gateway to power, a USB port where the desired algorithm is inserted through a computer, in addition to inputs and outputs (digital and analog) to perform measurements and perform low-power electric commands. These components make Arduino UNO a simple but complete system, suitable for electronic systems prototyping. More information can be found on the project's official website (ARDUINO, 2017).

For the purpose of this work, a firmware for the Arduino UNO will be developed, making it capable of integrating the system in the form of a production agent.

3.5 Supervisory software

Supervisory software is widely used as a means to control and monitor equipment and processes in industrial plants. In many cases this is done through PLC (Programmable Logic Controllers), which are robust electronic devices that, among other functions, collect electric signals (later translated into physical quantities) and execute programmable tasks through electrical commands. Supervisory software, therefore, provide an interface so that operators can control and monitor equipment and other processes through PLC in an industrial environment.

The choice made for this work is the supervisory E3 from the company Elipse Software, which is a product developed in Brazil, more precisely in Porto Alegre, and used all over the world. Elipse E3 stands out as a powerful tool in the industrial environment, being able to interact with numerous equipment, such as robots, machining stations and especially with PLC. Communication with such equipment takes place thanks to an extensive list of drivers, enabling interaction through various communication protocols aimed at the industrial area such as Modbus and OPC standard (Object Linking and Embedding for Process Control). More information about the E3 can be found on the company's website (ELIPSE, 2017).

In addition to the supervisory software, it is used a Dexter PLC model µDx100, and a didactic board containing, among other components, a temperature sensor and an arrangement of lamps connected to PLC ports.

An application in VBScript language in Elipse E3 environment, which controls some of the features of Dexter PLC, and implements the logic of a production agent, will be developed representing the possibility of interaction between industrial equipment of a real manufacturing system and the platform proposed in this work.

3.6 Organizational philosophy

Adopting a philosophy or organizational methodology is an attitude that can significantly facilitate the implementation of a project. In addition to that fact, it is possible to save time and use the available resources more effectively during its execution. When dealing with software projects, one of the approaches that has been gaining more prominence is called Agile. The Agile methodology presents itself as an alternative comparing to traditionally used methods in project management, and will be used as a way to organize the development process of the software pieces that will compose the platform proposed in this work.

In the Agile perspective, the project is divided into components with low complexity, that are implemented in development-focused cycles called sprints. This way the project can be planned, executed and monitored from the sprints, making its management come down to the control of fewer variables. However, this is only one aspect of the Agile method, and more information can be found on the initiative's website (AGILE, 2017).

4 Platform development

4.1 Architecture

The platform proposed in this work is composed by a set of independent components that together form an environment that runs and manages the production demands and other operational tasks in a simulated manufacturing system.

The integration of the components mentioned in the preceding paragraph is carried out through an architecture that is designed to meet the specific objectives of this work. The components that make up the system are described below:

- Cloud computing service: responsible for receiving, storing and providing the operation data;

- Communication Agent: responsible for providing the MQTT communication infrastructure for other agents in the system;

- Sales Agent: responsible for generating and monitoring the production demands;

- Supervisory Agent: responsible for managing the operation of the equipment and the operators on the factory floor environment;

- Production Agents: responsible for performing the production tasks in a self-organized way.

The Figure 4.1 shows a diagram of the architecture designed for the platform.

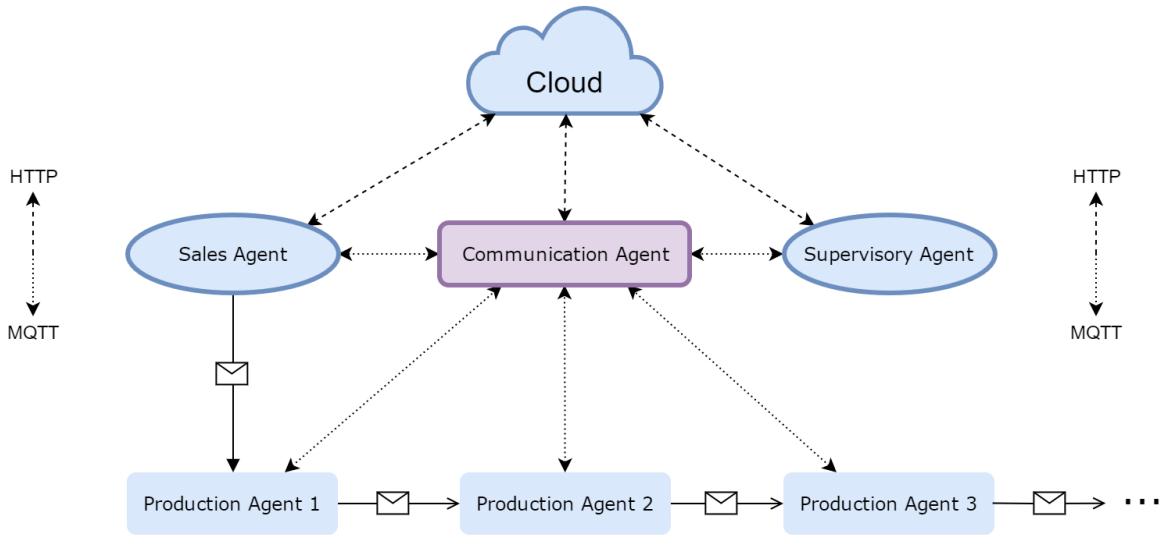

Figure 4.1 – Designed architecture representation.

Through the HTTP protocol the Sales Agent, the Communication Agent and the Supervisory Agent can interact with the Cloud computing system (whose connection is symbolized by the dashed line in Figure 4.1), publishing new data or looking for the previously stored data.

Once an agent connects to the Communication Agent through MQTT protocol (symbolized by the dotted line in Figure 4.1), it acquires the ability to send messages to other agents in the system through specific topics. Since a Production Agent has the ability to communicate, these agents exchange messages between each other, forming coalitions and thus perform the production orders entered into the system autonomously.

Another important role of the Communication Agent is to receive the operating data (the change of a services' state or a sensor value) from the production and forward it to the cloud computing service, to then be stored.

On the Supervisory Agent's interface, it is possible to monitor the list of connected production agents. One may also access the data history reported by these agents, such as service's state changes and the sensor values, as well as productivity analysis for each service of each Production Agent.

On the Sales Agent's interface, it is possible to access the product portfolio, create new product structures, carry out production orders, and access the manufacturing history of each product in each production order.

A production order's flow starts in the Sales Agent, which sends it to the more suitable Production Agent. This agent, at the end of its task, forwards the production order (symbolized by the envelope icon in Figure 4.1) to the next production agent, and so on, until all the manufacturing jobs on that order are finalized.

4.2 Virtualization

As already discussed in the previous chapters, virtualization is an important process to obtain the benefits offered by the concept of Industry 4.0. In this work, it will be performed the virtualization of a generic production system through persistent data structures, i.e., each entity to be virtualized will be represented by a JSON that identifies and represents its current state. This data is stored in a database in the cloud computing service, and updated on every change of a service's state. It is important note that the structures were designed to contain only the strictly necessary data for the realization of simple production flows.

A similar approach to this is used by Azure, which is the Microsoft's cloud computing service, which offers a specialized platform for the Internet of things called IoT Hub. In the nomenclature of Azure, the virtual copy of devices attached to the system is called device twin and has JSON format. The implementation details of the Azure IoT Hub can be found at the official documentation of the project (AZURE, 2017).

The three data structures that make up the production system virtualization in this work are: product; production agent; and production order. Figure 4.2 shows the data structure that represents a production agent.

```
agent = {
            agentId: Number
            agentName: String,
            Serivces: [
                        {
                            serviceId: Number,
                            serviceName: String
                            serviceStatus: Number
                        },
                        ...
                        {
                            sensorId: Number,
                            sensorValue: Number
                        },
                        ...
                      ]
        };
```

Figure 4.2 – Data structure that represents a production agent.

As can be seen in Figure 4.2, each production agent is represented by two identification properties (agentId and agentName), and a set of services and sensors, which have two identification properties (serviceId or sensorId and serviceName in the case of a service), in addition to a property that represents the current state of the service or the value read on the sensor (serviceStatus or sensorValue).

Figure 4.3 shows the data structure that represents a product.

```
product = {
            productId: Number,
            productName: String,
            Services: [
                        {
                           serviceId: String,
                           serviceName: String
                        },
                        {

                        },
                        ...
                      ]
          };
```

Figure 4.3 – Data structure that represents a product.

As can be seen in Figure 4.3, each product is represented by two identification properties (productId and productName), and a set of services, which are represented each by two identification properties of the service (serviceId and serviceName).

Finally, Figure 4.4 shows the data structure that represents a production order.

```
order = {
          orderId: Number,
          orderStatus: String,
          customerId: String,
          customerName: String,
          orderItems: [
                        {
                            productId: Number,
                            productName: String,
                            serviceOrderId: Number,
                            serviceOrderStatus: Number,
                            Services: [
                                        {
                                            serviceId: Number,
                                            serviceName: String
                                        },
                                        {
                                            serviceId: Number,
                                            serviceName: String
                                        },
                                        ...
                                      ]
                        },
                        ...
                      ]
        };
```

Figure 4.4 — Data structure that represents a production order.

As can be seen in Figure 4.4, each production order is represented by two order identification properties (orderId and orderStatus), two custumer identification properties (customerId and customerName) and a set of ordered services. These are represented by two product identification properties (productId and productName), two service order identification properties (serviceOrderId and serviceOrderStatus) and a set of services represented each by two service identification properties (serviceId and ServiceName).

4.3 Cloud computing service

The cloud computing service was developed in the programming environment Node using Express library, which implements a Web server (HTTP requests). In addition, this service uses the software MongoDb for storing the data, which is manipulated through the Mongoose library.

This structure is performed on a computer that is used strictly for this purpose, and the data provided by this server can be accessed from any internet-connected browser. This format of implementation is now-a-days commonly found in the backend architecture of Web systems.

The function of the cloud computing service is hosting the Web pages that contains the logic of the Supervisory Agent, Sales Agent, and the Production Agent developed for operators. Another task of the cloud computing service is to receive, store and update virtual entities of the production agents that are connect to the platform, as well as the new product structures created by users and transmitted through the Sales Agent. Finally, this service offers all information collected from the operation that has been stored for any agent capable of performing HTTP requests, providing, for example, the list of previously saved products in the system, the historical content of the services and sensors of a Production Agent.

4.4 Communication Agent

The communication was developed in Node's environment and uses Mosca library, which implements a MQTT broker. Its function is to provide communication services to the other agents in the system. In addition, this entity is responsible for managing the Yellow Pages service, which is the list of available devices.

After its initiation, the Communication Agent monitors new connections to register production agents. Once connected, the agent's services turn available in the Yellow Pages. This procedure stands as a check-in of the equipment and the operators in the platform.

Once a production agent is registered, the Communication Agent creates a virtual entity of that agent, which is updated with each change of state and reported to the cloud computing service, where the data is stored. Every production agent may offer one or more services, which must be in one of the four state categories: "available", "not available", "working" and "waiting".

The Yellow Pages service registers in a list, the production agents that are available to perform the manufacturing tasks. When an agent needs to forward a production order, it

checks to see what the next manufacturing task (service) to be performed is and sends the serviceId of this service to the Yellow Pages service. This service returns the request with a message containing the agentId of the best suited production agent to perform such service. The criteria adopted to determine the most appropriate agent to execute a service is the available period of a service. That is, the most appropriate agent will be the one whose service has the status "available" registered for the longest period of time in the Yellow Pages. This algorithm is designed to prevent production agents to remain idle for long periods, but it is a simple method and a bit far-fetched. However, is not within the scope of this work to develop advanced scheduling algorithms, although it is a subject of the highest importance in the context in which it is placed. A highly rich and in-depth analysis about task scheduling theories in industrial production environments can be found in the work of PINEDO et al., 1995.

As mentioned earlier, the Communication agent does not command the operation, but only provides support for communication. So, the production agents are those who create the production flows autonomously, generating thus a self-organized network.

4.5 Production Agents

The production agents are the components that perform manufacturing services, i.e., represent the equipment and the operators of a production environment.

To organize the communication between agents with the publish/subscribe model of MQTT Protocol, it was created a standard for the topics where messages travel through. In this pattern each agent sends and receives messages by topics created specifically for each task to be performed. In this way, the address of the message (topic) contains itself the nature of the task, and the content of the message carries the parameters of this task. Figure 4.5 shows the list of addresses (topics) of the messages that are sent and received by an agent of production during its life cycle.

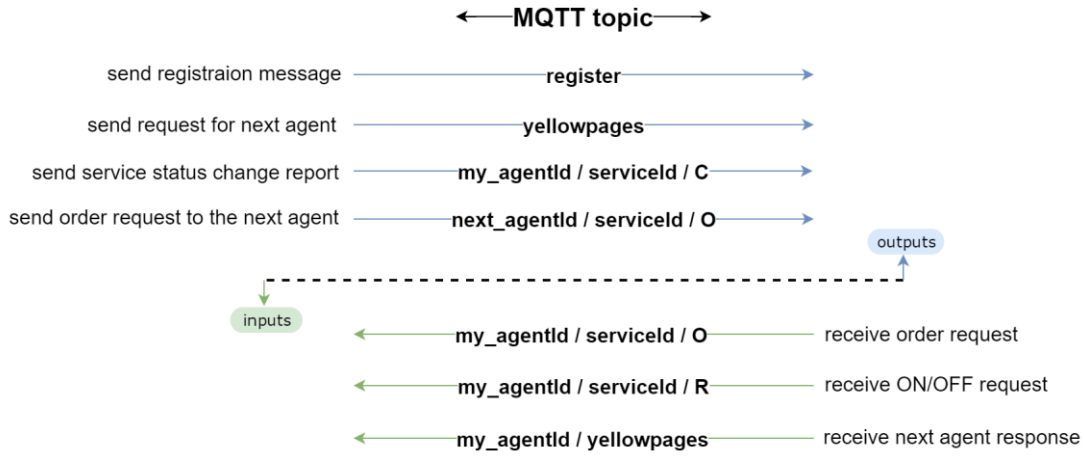

Figure 4.5 – Addressing of messages in a production agent.

The life cycle of a production agent begins when it registers in the system by publishing a message on topic "register" with its identification information, similar to the format shown in Figure 4.2. After the registration, the agent starts to monitor the topics represented by green arrows in Figure 4.5, and sends messages through the topics represented by blue arrows in Figure 4.5.

To illustrate the pattern created to manage message exchange via topics, consider a production agent whose agentId is "machine1", which provides a service with serviceId equals to "service1", and has a sensor with serviceId "sensor1".

After registered, the production agent starts to monitor the topic "machine1/service1/", which will eventually receive production orders in a similar format to that shown in Figure 4.4. Similarly, the production agent monitors the topic "machine1/service1/R" which will eventually receive requests to enable or disable one service. Each change of State of the "service1" or "sensor1" is reported to the Communication Agent through the topic "machine1/service1/C" or "machine1/sensor1/C", which will be forwarded to be stored in the cloud computing service.

To illustrate the message exchange dynamics during a product manufacture cycle, consider now a production order containing only one product, and that this product requires only two services to be manufactured, which are identified by productId "service1" and "service2", and that must be performed in the order in which they were shown. Also consider another production agent whose agentId is "machine2", which provides a service with serviceId "service2".

At the moment "machine1" receives a production order, it performs its manufacturing task (service1) during a certain time, and then searches in the structure of the order the serviceId of the next service that must be run on the product, in the case of example "service2". Once obtained this information, "machine1" sends it to Yellow Pages service through the topic "yellowpages". After receiving the request, the Yellow Pages service sends a message to "machine1" through the topic "machine1/yellowpages" containing the agentId of the best suitable production agent, in our example the content of this message would be "machine2". Finally, "machine1" sends the production order's structure to "machine2" through the topic "machine2/service2/", continuing the production flow.

The logic presented in this section was applied to three distinct software entities, which are described below:

- Arduino UNO: implementation made with C and C++ languages, representing embedded systems;

- Elipse E3: implementation made with VBScript language, representing industrial equipment and supervisory software controlled PLC systems;

- Webpage: implementation made with HTML and JavaScript languages, representing the interface for operators.

On running the software described above, it is created an environment that enables different types of equipment and several operators to interact in the form of production agents in a manufacturing system.

4.6 Sales Agent

The implementation of the Sales Agent was made on a Webpage, among other reasons, by the ease of interaction with users, with cloud computing service and with other agents in the system. This agent was developed with the goal of creating and managing production demands, and has three essential functions: consulting the product portfolio and adding new product structures; consulting the history of production orders; and adding production orders in the system.

As mentioned previously, is the Sales Agent that triggers the manufacture cycle of the products. When a new production order is carried out by a user (customer), a structure is created with the manufacturing instructions of each product, similar to that illustrated in Figure 4.4. To forward that order, as the production agents, the Sales Agent requests for Yellow Pages service the address of the best suited production agent to execute the first service of that product, and then, after the Yellow Pages' response, it forwards the production order to the corresponding agent.

4.7 Supervisory Agent

The supervisory Agent was also made on a Webpage, being developed with the purpose of monitoring the production operation. This agent has three essential features: to show a list of the agents that are currently connected to the system – online agents – as well as the states of its services and sensors; to query and plot the historic values of the sensors; and to query and plot a basic productivity analysis of the services.

After connected to the system, the Supervisory Agent conducts an HTTP request to the cloud computing service, whose response is a data structure containing the virtual entities of the online production agents. With this information the Supervisory Agent has access to a historic view and the current status of the agents' network. Shortly thereafter it starts the monitor the corresponding topics related to each service of each online production agent. In this way, the Supervisory Agent keeps its interface up-to-date, enabling real-time visualization of the system's current state as well as the operation historic data.

When a user requests a query to the history of a certain service, the Supervisory agent performs again a HTTP request to the cloud computing service, which response is a data structure containing the desired information. Therefore, one can print the chart of the historic values of the sensors as well as analyze productivity and print it on its *interface*.

5 Results

The platform described in the previous chapter was developed as planned, generating the necessary infrastructure to carry out the partial monitoring of a simulated manufacturing system. In this chapter the interfaces of the agents that make up the system will be illustrated, and the final comments about the features acquired, its limitations and the results obtained by means of a test carried out with the platform.

The first component to be illustrated represents embedded devices. The code developed to implement a production agent's logic on the Arduino UNO uses C and C++ programming languages in addition to specific libraries for this purpose. The internet connection of this agent was performed through a Wi-Fi Shield.

The hardware uses a set of batteries as energy source, and a led connected to a digital port was installed to represent the four possible states of a hypothetical service, which behaves as follows: led off to represent the state "not available"; led on to represent the State "available"; led flashing with high frequency to represent the state "working"; and led flashing with low frequency to represent the state "waiting". Figure 5.1 shows a picture of the production agent developed in a prototyping platform Arduino UNO.

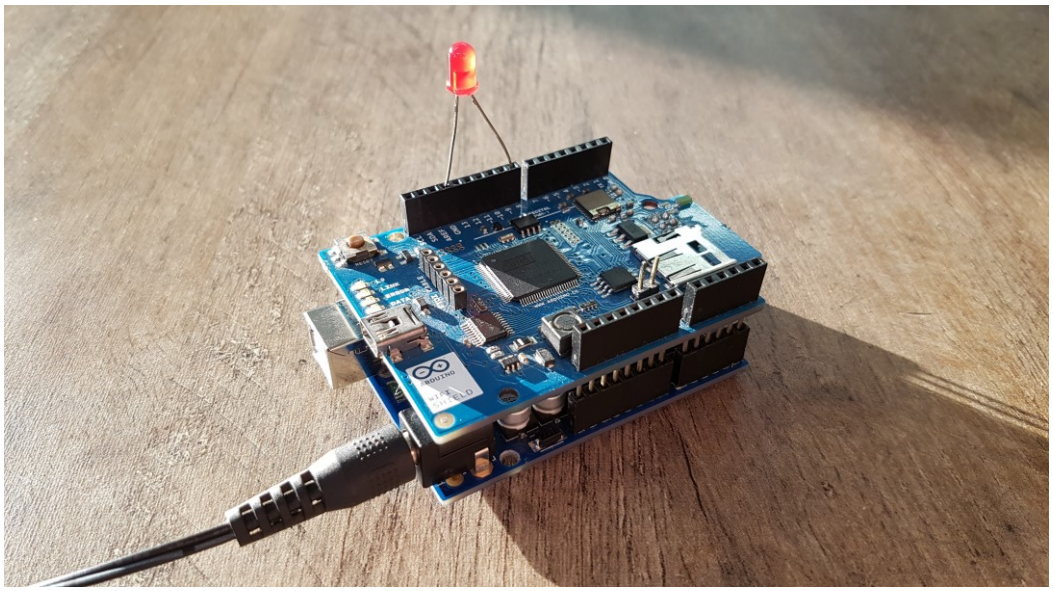

Figure 5.1 – Production agent developed in Arduino UNO.

Just after received a production order, the production agent changes the state of its service to "working", and the led starts to flash. After a programmed period, the production order is forwarded to the next production agent and the service's state goes back to "available". If no agent is available, the service's state changes to "waiting" and the production agent starts periodically sending a request to the Yellow Pages service, until there is a production agent available to run the next service.

The implementation of the production agent on the supervisory software was made in order to offer: a variable which is interconnected with one of lamp of the PLC's board representing a service; and another variable which is interconnected to a temperature sensor of the PLC's board. Figure 5.2 shows an image of the production agent's interface developed with the Elipse E3 and the Dexter PLC attached with a board of components.

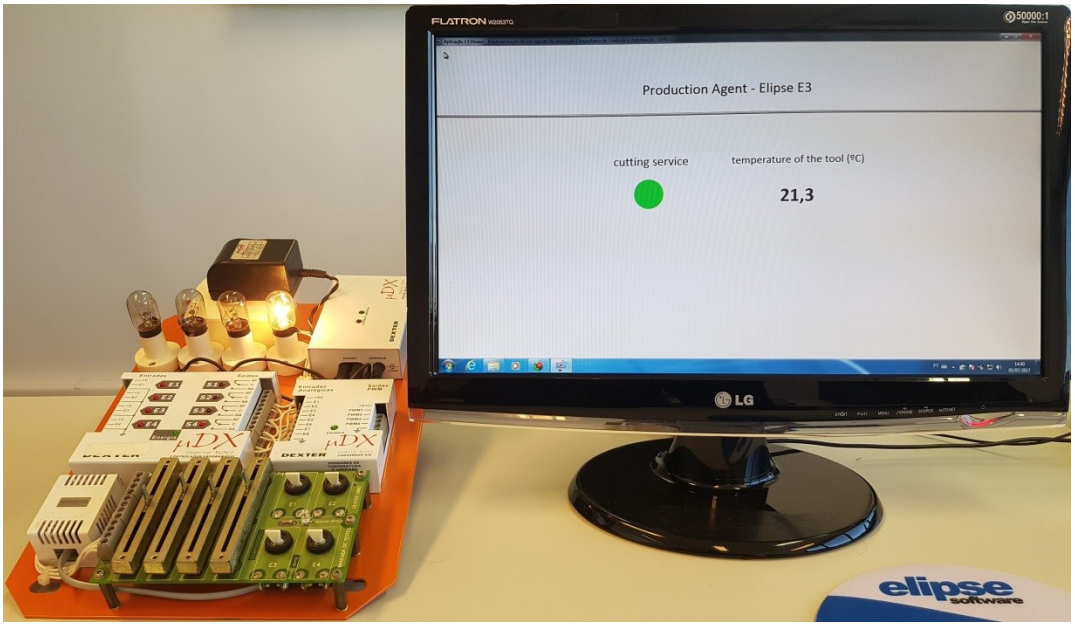

Figure 5.2 – Production agent developed with Elipse E3.

On this interface, the sensor temperature value is shown shortly after being collected, and the service is represented by a circle, as can be seen in Figure 5.2. The circle functions as an activation button, the red color represents that the service is not available (state "not available") and the lamp is switched off. By clicking in the circle it turns green, and the lamp lights up, showing that the service is now available (state "available Just after received a production order, the service state automatically changes to "working", and the light behaves similarly to led's standard on Arduino UNO implementation. Once started, this application connects to the Communication Agent, and monitors its respective topics (as illustrated in Figure 4.5) in order to receive requests, as well as to report service status changes and the sensor values.

Similarly, though written with Web programming languages, the operator oriented production agent was developed, which can be accessed by any browser connected to the internet (as well as the sales Agent and Supervisory Agent shown below). Figure 5.3 shows an image of the operator oriented production agent.

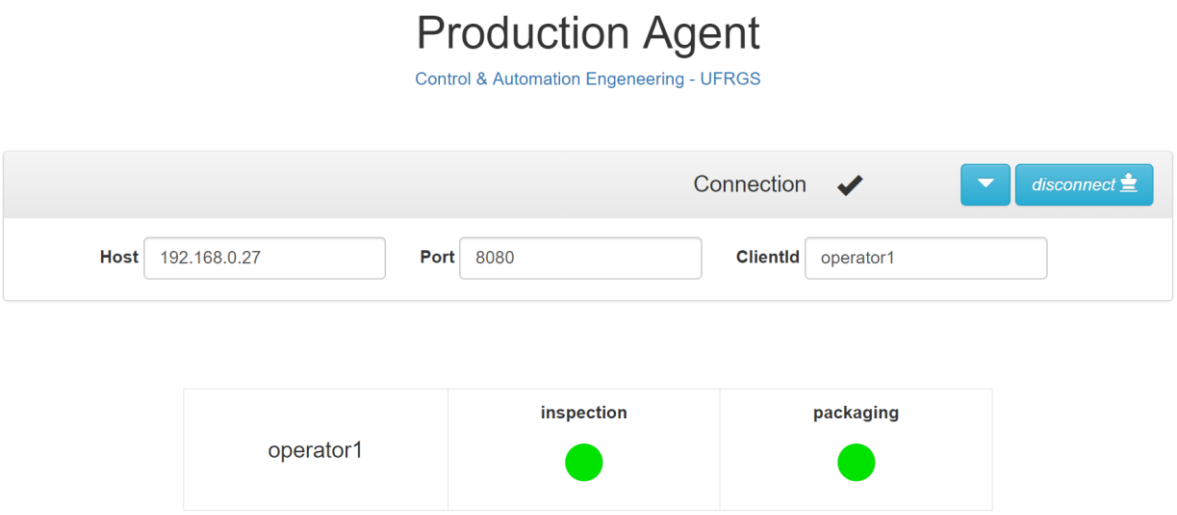

Figure 5.3 - Production Agent's Interface oriented to operators.

This implementation features a connection bar that requires the operators to report their identification code to connect to the system. This same functionality was added to the Sales Agent and the Supervisory Agent. However, in the following illustrations, the area containing the connection information is hidden - what can be done by clicking on the button containing an arrow on the connection bar. As can be seen in Figure 5.3, the production agent shown represents an operator identified as "operator1", that provides two services: "inspection" and "packaging", both showing the state "available". One can also observe that there is a connection icon indicating that the program is connected to the system, that is, ready to receive production orders.

Upon receiving an order, a small window is opened in the production agent's interface. Figure 5.4 shows the two variations of the service routine.

Figure 5.4 – Service routine windows.

As soon as the production order comes to an operator through the production agent, the service routine window presents a button so that the operator can confirm the start of the operation, as shown to left in the picture in Figure 5.4. By clicking "start" the service changes its state to "working", and the routine service window changes its appearance: the warning message is replaced by a time counter; and the start button has its title changed to "stop", as shown to the right in the picture in Figure 5.4. By clicking the "stop" button the service routine window closes. Then the production agent forwards the order to the next production agent and the state of the service returns to "available".

The next component to be illustrated is the Sales Agent, which was designed to add and monitor the production demands, in addition to managing product structures. Figure 5.5 shows an image of the Sales Agent.

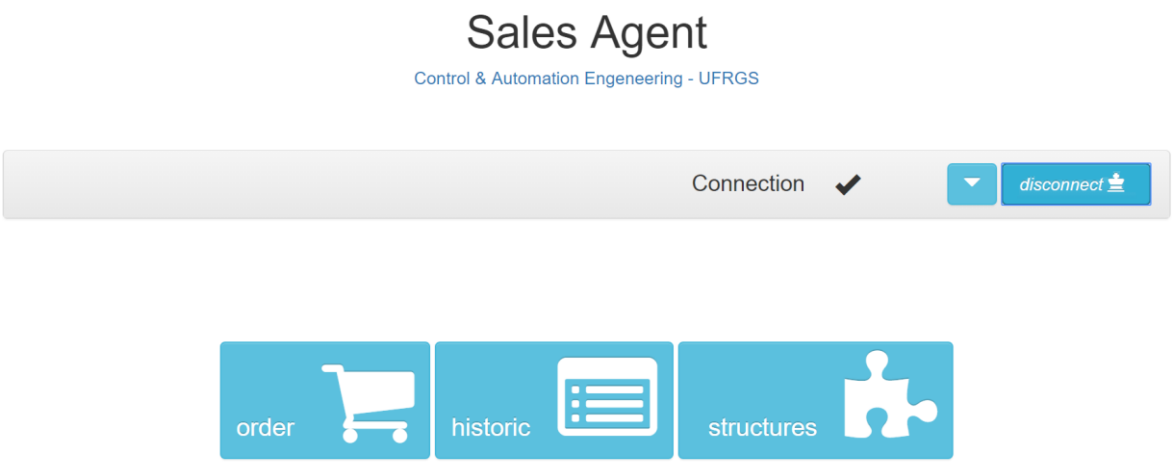

Figure 5.5 – Sales Agent's Interface.

As can be seen in Figure 5.5, the Sales Agent is connected to the system, and has three buttons to perform its functionalities. By clicking on "structures" it opens the product structures window. Figure 5.6 shows an image of the Sales Agent's product structure window.

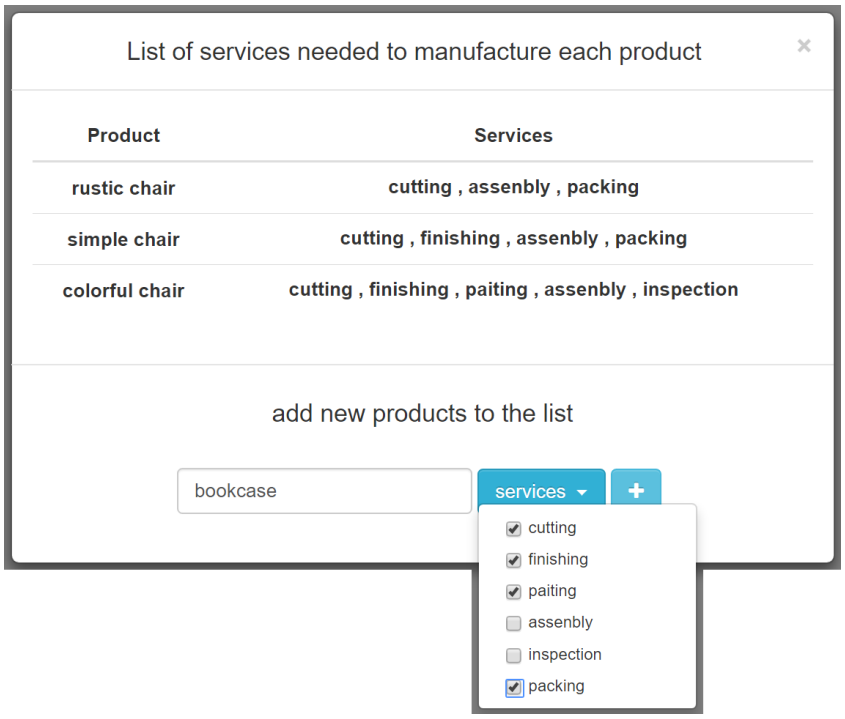

Figure 5.6 – Product structures window.

On the product structures window one can see the list of structures previously stored, as well as add new product structures. As can be seen in Figure 5.6, the system has three product structures already registered: "rustic chair", "simple chair" and "colorful chair".

The lower area of the product structures window is dedicated to adding new structures. In the case illustrated in Figure 5.6, by clicking on "+" a new product called "bookcase" will be added, which requires the following services: “cutting”, "finishing", "painting" and "packing". After that, a dialog box opens to confirm the storage of the new structure in the cloud computing service, and so the product structures window is also updated with the new structure that has just been added.

By clicking on "order" in the Sales Agent it opens the order adding window, in which one can add production orders to the system. Figure 5.7 shows an image of the order adding window from sales agent.

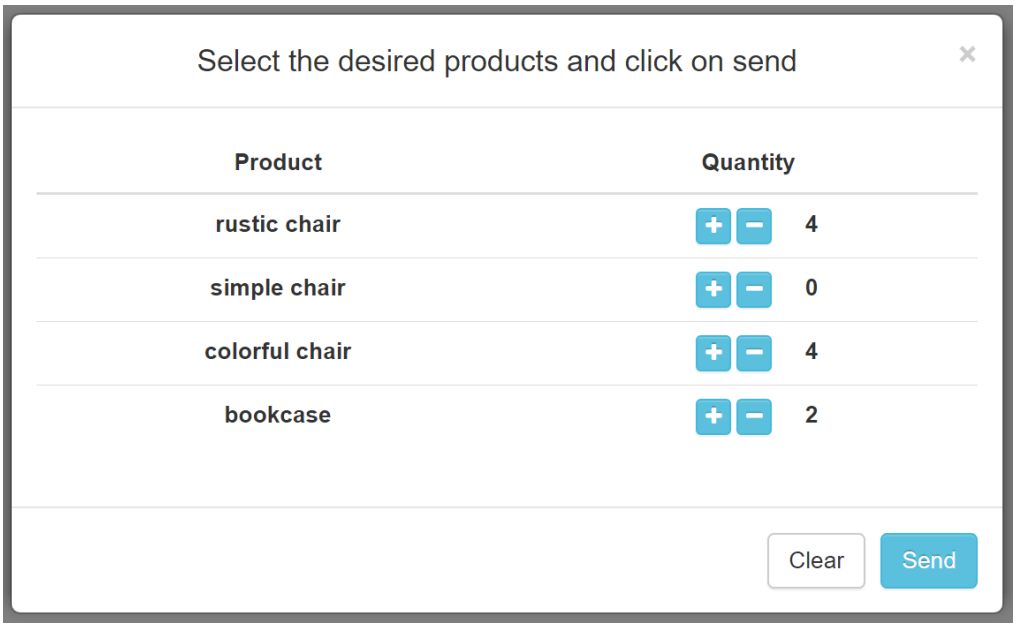

Figure 5.7 – Order adding window.

As can be seen in Figure 5.7, the orders adding window contains a list of product options that may be requested. By clicking on "+" in the desired product corresponding line, a new product its added to the production order. To remove products from the order, one should simply click on "-" at the target product corresponding line. One can also clean up all the products in the order by clicking on "clear". As soon as the order is in accordance with the desired, the user must click on "send" in the lower right corner of the window. After that, the Sales Agent reports the new production order to the cloud computing service and searches for production agents available to carry out the first service of the order.

By clicking on "historic" in the Sales Agent it opens the orders historic window, where one can view details of the operation. Figure 5.8 shows a picture of the orders historic window from Sales Agent.

Figure 5.8 – Orders historic window.

As can be seen in Figure 5.8, in this window there is a bar dedicated to each order. On the right side of the bars it is possible to see the status of the order – in the case illustrated in Figure 5.8 all orders present the status "done" indicating that the orders have been completed successfully.

By clicking on any bar, it opens a detailed report of the corresponding production order, as can be seen in Figure 5.8. In this report, each row represents a work order, and in its first three columns it is possible to see its respective "serviceOrderId", the name of the product and the corresponding manufacturing status. Each colored bar represents the contributions of some production agent in the manufacturing of one product. Each color represents a distinct production agent, which may have contributed to the manufacture of one or more products in the same production order. When positioning the mouse on one of the colored bars it appears on the screen a caption containing the ID of the corresponding production agent and the operation's start and end time of the corresponding services.

The last interface to be illustrated is the Supervisory agent, which follows the same front-end design of the other interfaces, and features similar connection characteristics as the other agents. Figure 5.9 shows the Supervisory Agent's interface.

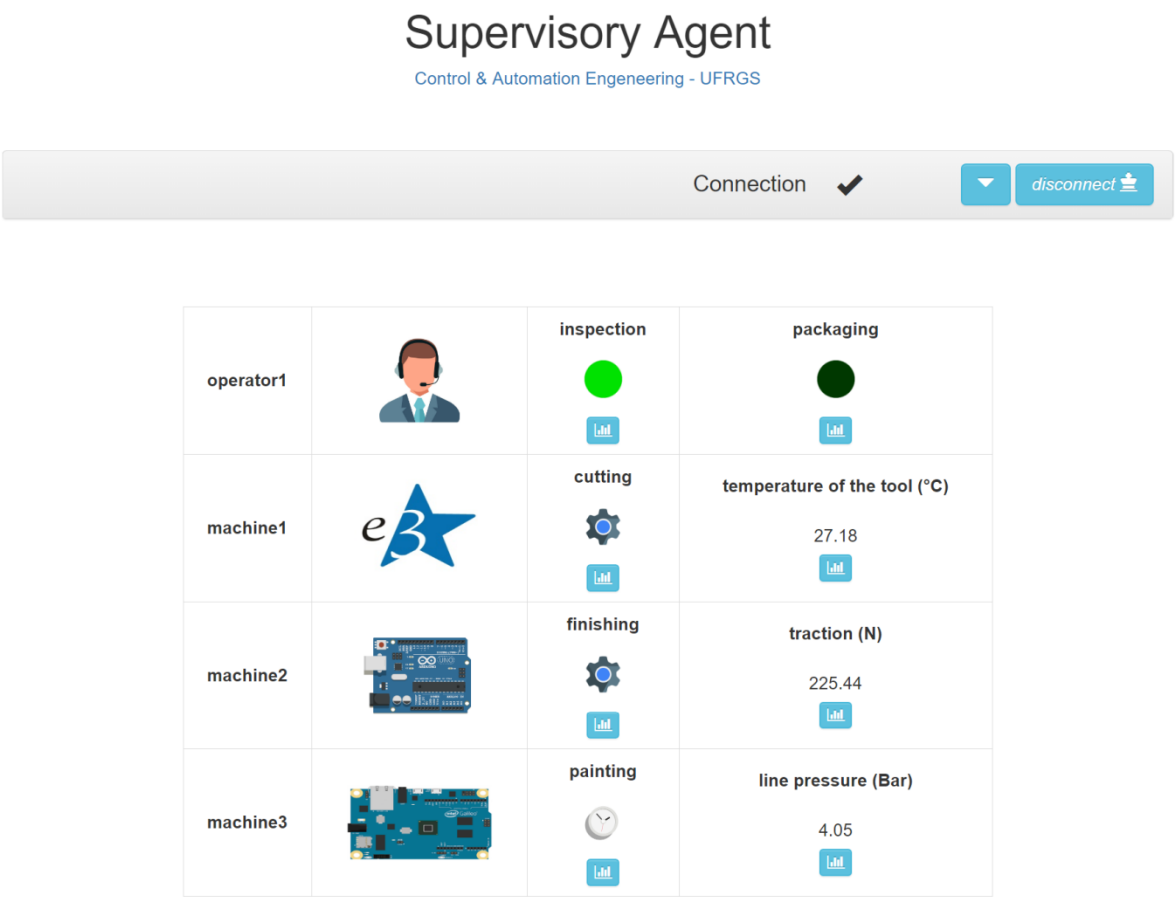

Figure 5.9 – Supervisory Agent's Interface.

The interface of the Supervisory Agent offers in its main panel, the list of production agents connected to the system, as well as the state of their services and the values of their sensors. As in the case illustrated in Figure 5.9, the current state of the system features four agents of production: "operator1", offering two services; "machine1", providing a service with a "working" status (represented by the gear icon), and a sensor; "machine2", with the same features as the previous agent; and, finally, "machine3", providing the service of "painting" in state "waiting" (represented by the clock icon).

By clicking on the button containing the graphic icon, set just below a sensor value in the Supervisory Agents' interface, it opens the sensor historic window. Figure 5.10 shows an image of a sensor historic window.

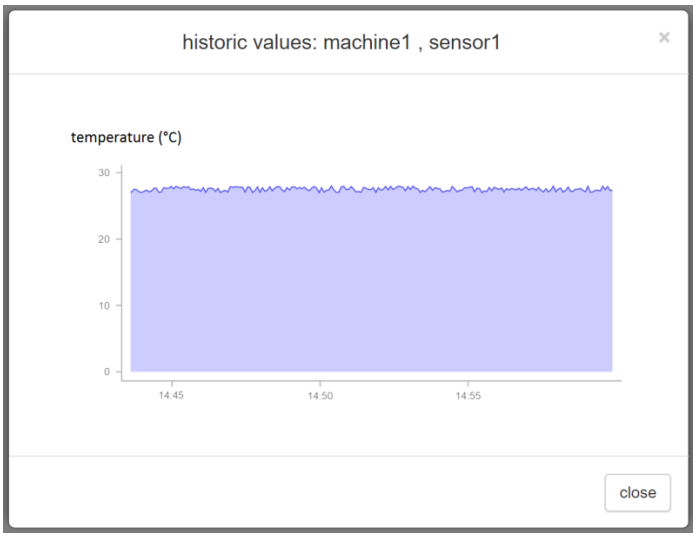

Figure 5.10 – Sensor historic window.

As can be seen in Figure 5.10, in this window it is possible to see the sensor values' chart during the operation period.

By clicking on the button containing the graphic icon, set just below the service icon in the Supervisory Agents' interface, opens the service productivity analysis window. Figure 5.11 shows an image of a service productivity analysis window.

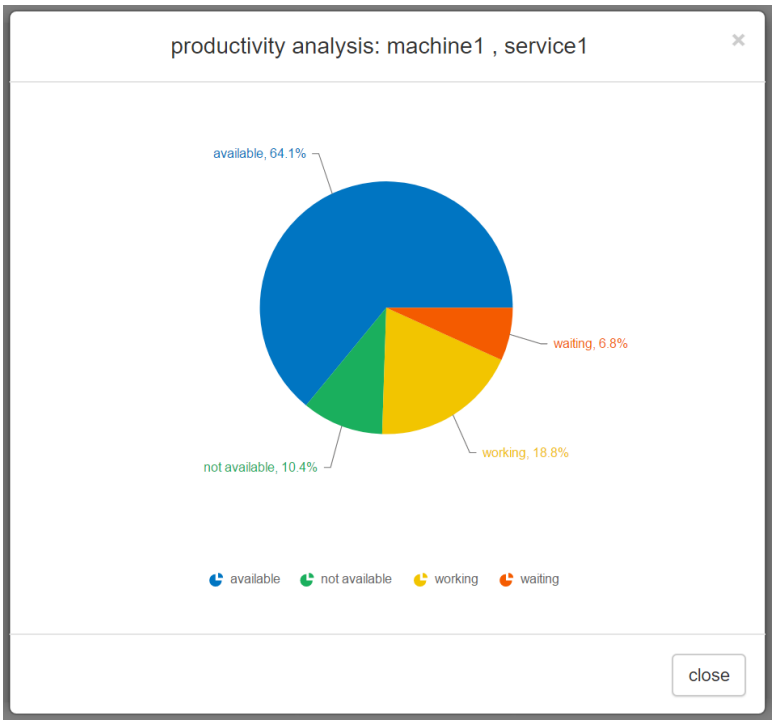

Figure 5.11 – Service productivity analysis window.

As can be seen in Figure 5.11, this window allows one to visualize the productivity analysis of a service, contains the percentage of time that the service presented in each of the four status categories ("available", "not available", "working" and "waiting") during its operation period.

Once that all interfaces have been presented, the performance attributes obtained with the platform developed in this work will be discussed.

The code containing the cloud computing service logic algorithm was run on a server computer located in the United States, provided for a cloud computing company. The designed algorithm proved to be quite adequate for the application in terms of data storage, and HTTP requests processing, performing both tasks with consistence and agility. The offered services by the Communication Agent also proved to be quite appropriate for the project, and there was no critical limitation observed in relation to the proposed features.

In order to evaluate the functioning of the agents' self-organization logic, to determine population capacity limits, and, finally, to validate some concepts, an essay was carried out with the platform developed in this work. In the essay, 200 agents were simultaneously connected to the production system: one of them running on an Arduino UNO; four other agents being executed with the Elipse E3 application version running on different computers connected with Dexter µDx100 PLC; and the rest of them running on Web pages containing the implementation of operator oriented production agent. To make the test easier, it was developed an automatic version of the operator oriented production agent, that is, a version that does not require confirmation of both start and end of the service operation – which will start automatically as soon as the orders get coming, remaining in the state "working" for a fixed interval of five seconds. The Web pages containing the production agents were opened on five different computers and all agents present in the assay were connected to a local wireless network. It is worth mentioning that the choice for a local network is common in industrial environments, since such systems cannot entirely depend on the internet for the course of their production routine.

During the test several production orders were made, one of which containing 500 products, which were manufactured (virtually) during a period of approximately eight minutes. Figure 5.12 shows an image of some of the equipment used in the validation of the platform.

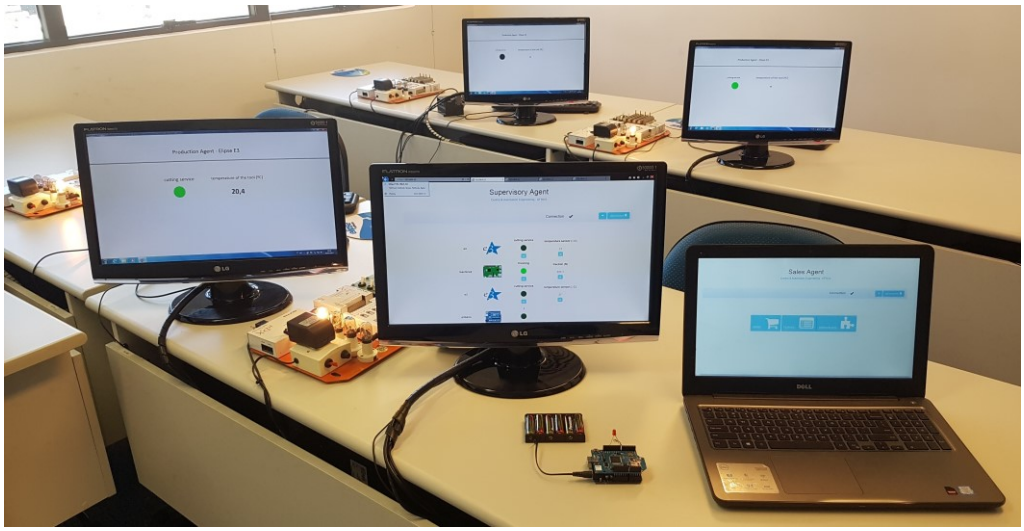

Figure 5.12 – Equipment used on the validation of the platform.

The test was performed in the city of Porto Alegre, more precisely at Ellipse Software training center, that kindly provided one of its laboratories for the purposes of this project. Its execution was performed with apparent success, and it could be shown, at first sight, that the Communication Agent supports a fairly large number of simultaneous connections without showing any anomaly. It was also possible to infer that the array of production agents was able to self-organize, and so establish production flows efficiently and autonomously: efficiently, because during the simulation the production agents did not present significant idle time intervals - even though it was possible to observe some gaps between sequenced tasks during the manufacture of a product from the same order due to message transport delays; autonomously, because it was not necessary, at no time, to indicate which agent should work on the desired products, or any other information to preset the route by which the products should follow in order to be manufactured, even when it came from production orders with a large number of products. One of the system's limitations, observed during the test was the Sales Agent's orders historic window, which loses its clarity when presenting reports with a large number of production orders. A strategic layout improvement at this point would surely be interesting for future work on the subject.

Observing its characteristics, it is possible to infer that the platform developed in this work: presented interoperability on connecting different types of devices; implemented a virtualization for entities abstraction; turned out to be decentralized on leaving the agents in charge of defining the production flows; showed real time ability to display the list of connected agents, as well as other production information; has a Service Oriented Architecture; and, finally, showed modularity by allowing the rotation of agents to form coalitions, even with production agents going in and out during the production cycle.

As explained so far, one can say that this work has succeeded on applying the general concepts of the Industry 4.0 perspective in a simulated manufacturing system. With the application of these concepts, the system acquired certain attributes, as discussed in Chapter 2, that are considered highly promising for organizations that want to achieve better results in efficiency, agility and flexibility in their production systems. On acquiring these attributes means that enterprises shall enjoy important competitive advantages in the current product consumption conjecture, and gain the ability to explore new business models, being some of them quite disruptive compared to what is being currently used in the industry.

However, it is important to highlight that there is much more to explore on several issues related to the management platform proposed in this work, and one of the most important points is the development of algorithms for production flow optimization, in order to: reduce the lead time of the products; decrease the interval between products in the same order, reducing the risk of intermediate stocks; and, finally, reduce the idle time of equipment and operators during the work shift. These improvements could be parameterized though indicators like OEE (Overall Equipment Effectiveness), which is commonly used in industry to measure the local efficiency of a given resource. However, OEE is just one of a large set of tools found in TPM (Total Productive Maintenance) that is characterized as an industrial management methodology (CHIARADIA, 2004), which could be applied in the platform developed in this work to increase production efficiency.

Another limitation worth to be highlighted is the absence of some important aspects of a real manufacturing system, such as the control of inputs for machines and operators, the control of finished product inventory, and the integration with the company's engineering sector, which are key to the effective management of a system with this nature. Control systems like those mentioned are used by all current industrial organizations, which are extremely important for the well-being of its operations and that sometimes presents gigantic virtual structures. With that being said, the task of integrating the platform developed in this work with ERP, MRP and MES systems currently installed in companies, can be considered as a major challenge.

It was also possible to infer the need of treating some products as a set of other products or by-products, which can also be marketed. Thus, a product would be composed by several sub products, which can be locally manufactured or provided by suppliers. This approach is widely used in the industry, and could even make the integration of factories and its supply chain in the platform developed in this work possible.

Another important topic to be explored is known as data-driven analytics, which aims to take strategic information in real time from the big pile of data that is continuously collected during the operation in a production system. These analyses have great potential to assist managers, from planning to important decision-making which must be taken during the work routine on the factory floor. Currently there are companies specializing in data-driven analytics for industrial purposes, of which Mitek Analytics located in Palo Alto, California, at Silicon Valley is worth highlighting. More information about Mitek can be found on the official website of the company (MITEK, 2017). According to Dr. Dimitry Gorinevsky, Stanford University engineering professor and CEO of Mitek Analytics, the Industrial Internet of Things will create value for organizations through the information obtained from data-driven analytics, not only in the field of manufacturing systems, but in general production operations (GORINEVSKY, 2016).

6 Conclusions and future works

The changes in the dynamics of consumption impose to production organizations a constant process improvement so that they competitively continue to meet the needs of the consumers. The number of new technologies being developed, designed for the most diverse applications, has been very large, and while it is relatively delicate to apply them in certain conservative environments, there is no doubt that several of these can bring real benefits to production chains.

The platform developed in this work reached the goals of managing a certain number of parameters in a simulated production system. Its structure consists in a set of software pieces designed for different categories of agents present in the system. Through its interface it is possible to view some reports, among other information that is updated in real time, as the production progresses goes on. In addition to that, the system features a self-organized arrangement of machines and operators, which perform the manufacturing of products autonomously, once received basic instructions.

The system was able to integrate, in a simple and transparent way, different equipment, operators and production demands, in a flexible manufacturing simulated environment. According to some tests, the network implemented proved to be robust enough to support a fairly large amount of agents connected simultaneously.

Despite providing important information about the production progresses, there is a need for more information and better analysis tools to effectively control a real manufacturing system. However, the present work shows, in an illustrative way, the application example of the Industry 4.0 concepts, and points out some very attractive features that organizations are able to get with such concepts.

The major difficulties encountered during the development of this system, with no doubt, were related to the architecture modeling. However, the observation of some models used for computer companies was of great value, with emphasis on the Microsoft Azure project, which recently began offering Internet of things specialized services, showing up to be at the forefront of this area, whose virtualization methodology served as reference for this project.

In future works it would be interesting to add to the platform a few other aspects found in the value chain of production systems, such as the control of inputs for the machines and operators, the management of the finished products stock, integration with the engineering sector, among others. In this context, it is also necessary to align with some approaches typically used in industrial environments, and explore the concepts of data-driven analytics, in order to obtain strategic production data in real time. These improvements would help in the maturation of this project, making it closer to becoming a viable tool for adding real value to industrial organizations.

References

ARDUINO; Projeto Arduino. Available at https://www.arduino.cc/. Accessed on April 2017.

AGILE; Agile Methodology. Available at http://agilemethodology.org/. Accessed on March 2017.

AZURE; Serviços de computação em nuvem da Microsoft. Available at https://docs.microsoft.com/en-us/azure/iot-hub. Accessed on July 2017.

ATZORI, L.; IERA, A.; MORABITO, G.; The internet of things: A survey. Computer networks, Elsevier. 2010.

BAUER, E.; WILEY, J.; Design for Reliability: Information and Computer-Based Systems. 2010.

BASSI, A.; BAUER, M.; FIEDLER, M.; THORSTEN, K.; VAN KRANENBURG, R.; LANGE, S.; MEISSNER, S.; Enabling Things to Talk. 2013.

BRAGA, N.; Indústria 4.0 – O que é isso? 2014. Available at: http://www.newtoncbraga.com.br/index.php/eletronica/52-artigos-diversos/7571-industria-4-0-o-que-e-isso-art1350. Accessed on April 2017.

CHIARADIA, A.; Utilização do indicador de eficiência global de equipamentos na gestão e melhoria contínua dos equipamentos: um estudo de caso da indústria automobilística. 2004.

COLLINA, M.; Biblioteca Mosca. Available at http://www.mosca.io. Accessed on July 2017.

DAIS, S.; Industrie 4.0 – Anstoß, Vision, Vorgehen. In: Bauernhansl. 2014.

DAVIES, R.; Industry 4.0 Digitalisation for productivity and growth. 2015.

ELIPSE**;** Elipse Software. Available at https://www.elipse.com.br. Accessed on July 2017.

EVANS, D.; The Internet of Things: How the Next Evolution of the Internet is Changing Everything. 2011.

EVENT MACHINE; Ambiente de desenvolvimento Ruby. Available at https://github.com/eventmachine. Accessed on June 2017.

EXPRESS; Biblioteca Express. Available at http://expressjs.com. Accessed on May 2017.

FERBER, J.; Multi-Agent System: An Introduction to Distributed Artificial Intelligence**.** 1999.

GARTNER; Gartner Identifies the Top 10 Strategic Technology Trends for 2013. Available at http://www.gartner.com/newsroom/id/2209615. Accessed on May 2017.

GORINEVSKY, D.; Analytics for Industrial Internet of Things. 2016.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M.; Internet of Things (IoT): Avision, architectural elements, and future directions. Elsevior. 2013.

HERMANN, M.; PENTEK, T.; OTTO, B.; Design Principles for Industrie 4.0 Scenarios: A Literature Review. 2015.

HIVEMQ; MQTT Essential: Part 1 – Introducing MQTT. Available at http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt. Accessed on July 2017.

IVACE; Informe sobre el Estado del Arte de la Industria 4.0. 2016.

JADE; Ambiente para desenvolvimento de Sistemas Multiagentes. Available at http://jade.tilab.com/. Accessed on June 2017.

KAGERMANN, H., WAHLSTER, W.; HELBIG J.; Recommendations for implementing the strategic initiative Industrie 4.0: Final report of the Industrie 4.0 Working Group. 2013.

MENDES, J.; LEITÃO, P.; COLOMBO, A.; Service-Oriented Control Architecture for Reconfigurable Production Systems. 2008.

MITEK; Advanced Analytics for the Industrial Internet of Things. Available at http://www.mitekan.com/. Accessed on June 2017.

MONGODB; Software de banco de dados. Available at https://www.mongodb.com. Accessed on June 2017.

MONGOOSE; Driver de comunicação. Available at http://mongoosejs.com. Accessed on June 2017.

MQTT; Biblioteca MQTT.j. Available at https://github.com/mqttjs/MQTT.js. Accessed on June 2017.

NODE; Ambiente de desenvolvimento JavaScript. Available at https://nodejs.org/en/about. Accessed on May 2017.

ONORI, M.; SEMERE, D.; LINDBERG, B.; Evolvable systems: an approach to self-X production. 2011.

PECHOUCEK, M.; MARÍK, V.; Autonomous Agents and Multi-Agent Systems. 2008.

PEIXOTO, J.; Sistema Minimamente Invasivo Baseado em Agentes Aplicado em Controladores Lógicos Programáveis. 2016.

PINEDO, M.; Scheduling: Theory, Algorithms, and Systems. 1995.

RUSSEL, S.; NORVIG, P.; Artificial Intelligence: A Modern Approach. 2010.

SANCHEZ, O.; Antecedentes da Adoção da Computação em Nuvem: Efeitos da Infraestrutura, Investimento e Porte. 2012. Available at http://gvpesquisa.fgv.br/publicacoes/gvp/computacao-em-nuvem. Accessed on May 2017.

SANTOS, B.; SILVA, L.; CELES, C.; BORGES, J.; PERES, B.; VIEIRA, A.; VIEIRA, L.; GOUSSEVSKAIA, O.; LOUREIRO, A.; Internet das Coisas: da Teoria à Prática. 2015.

SCOVINE, C.; Arquitetura Orientada a Serviços. 2006. Available at https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/soa?lang=en. Accessed on April 2017.

SWA; MQTT Protocol Tutorial: Step by step guide. Available at https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html. Accessed on June 2017.

SHELBY, Z.; HARTKE, K.; BORMANN, C.; The Constrained Application Protocol (CoAP). 2014.

TWISTED; Ambiente de desenvolvimento Phyton. Available at https://twistedmatrix.com/trac/wiki. Accessed on June 2017.

VERTEN, J.; ThyssenKrupp launches MAX: Maximum efficiency in cities with IoT technologies from Microsoft. 2015. Available at https://news.microsoft.com/de-de/thyssenkrupp-startet-max-maximale-effizienz-in-staedten-mit-iot-technologien-von-microsoft/#sm.001977ezp17tve3msbd2as8oaqyy6#hQE1kdgO1brCRtZh.97. Accessed on April 2017.

ZHU, Q.; WANG, R.; CHEN, Q.; LIU, Y.; QIN, W.; IoT gateway: Bridgingwireless sensor networks into internet of things. In: IEEE. Embedded and Ubiquitous Computing (EUC), IEEE/IFIP 8th International Conference on. 2010.

Attachments

Annex A – State-of-the-Art

Currently there are already some tools focused on applying the concepts of Industry 4.0. However, it is interesting to note that there's a history of large manufacturing companies that have developed tools to improve their own production systems. These tools, before being commercialized, were implemented in some or all stages of the production process of the companies that have developed it, indicating that the needs for these kind of tools are real. In the work IVACE, 2016, as already mentioned, there are examples of the implementation of the Industry 4.0 concepts, which some of them are described below.

A.1 General Electric

The American General Electric was one of the first companies to use the philosophy of Industry 4.0 on its plants. Recently, the company created the Predix platform, which is based on cloud computing and is designed for industrial-scale analyses. This tool allows the management of assets and the optimization of operations, by providing a standard way to connect people, data, and machines. In Figure A.1 is possible to see a representation of Predix platform architecture.

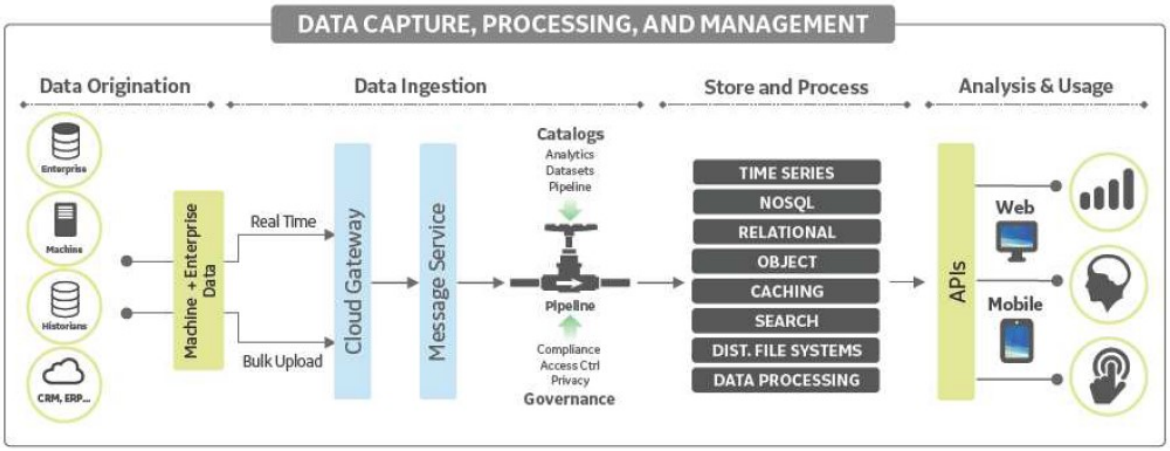

Figure A.1 – Predix platform architecture – source: IVACE, 2016.

The following is a brief description of the four layers that make up the Predix platform architecture.

- Origination Date – is the process of data capture, whose components are directly linked to the production processes on the shop floor, or to the strategic and operational tools such as ERP, CRM and MES systems.

- Data Ingestion – is the treatment of the input data, which can be done in real time or based on historical data.

- Store and Process – is the storage process and preprocessing of the input data, which uses different storage and analytical tools.

- Analysis & Usage – it's the process of exposure, to users, the results of the analyses carried out with the input data.

A.2 ThyssenKrupp

The German company ThyssenKrupp recently presented the first predictive maintenance system for elevators in the industry. The project named MAX has as main objectives to increase the availability of elevators and reduce the resolution time of problems through real-time diagnostics. The system infers failure events before they occur. This avoids unscheduled shutdowns of lifts, reducing the need for replacement of some components before the end of its life cycle. The reasons that led to the development of MAX are the major differences between traditional lifts maintenance procedures and the daily needs of the modern urban environment. Remote monitoring of elevators emerged at the end of the 1980's, but even if these systems alert the company when the lifts stop working, it does not reduce the number of stops. And this is precisely the problem that MAX platform is designed to solve, using cloud computing with Microsoft Azure support. Figure A.2 presents an image of the ThyssenKrupp's MAX platform brochure.

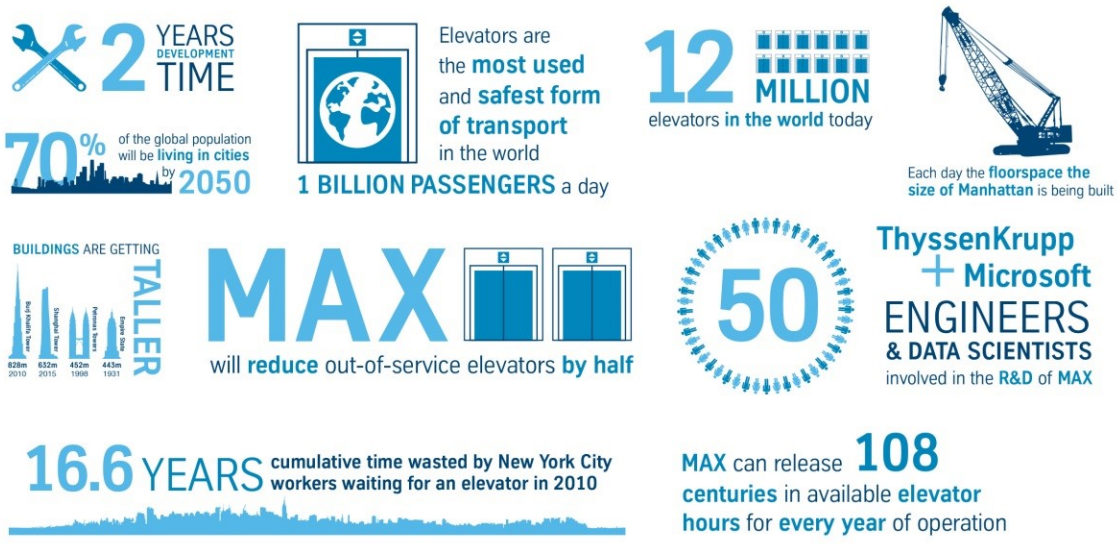

Figure A.2 – MAX platform brochure – source: VERTEN, 2015.

This system is based on reports generated from data collected by the remote monitoring of lifts. These data is sent to the MAX platform and analyzed by machine learning algorithms. The idea is to analyze and estimate the length of the elevator's components so that the operators can predict failures, and thus reduce the number of breakdowns, enabling a strategic scheduling for the maintenance of the elevators.

Appendices

Appendix A – Illustration of Cloud Computing's code structure.

```
/*******************************************************
                    Cloud service

            author: João Ricardo Cardoso
 *******************************************************/

////////////////////////////////////////////////////////
// Express requests and iniciation
////////////_____________________________________________
var express = require('express')…
////////////////////////////////////////////////////////
// MongoDb requests and iniciation
////////////////////////////////////////////////////////
var mongoose = require('mongoose')…
////////////////////////////////////////////////////////
// API infrastructure
////////////////////////////////////////////////////////

// http server configuration
app.use(express.static('www'),bodyParser.json());
// provides the historical data of agents
app.get('/graphics', function (req, res) {…
// provides the products list
app.get('/products', function (req, res) {…
// provides the service orders list
app.get('/orders', function (req, res) {…
// provides the production orders list
app.get('/order', function (req, res) {…
// http server inicialization
server.listen(80, function (err) {…
////////////////////////////////////////////////////////
// data base operactions
////////////////////////////////////////////////////////

// MongoDb connection through mongoose
mongoose.connect('mongodb://localhost/devices', function(err, res) {…
// modelling of a production agent
var deviceModel = mongoose.model('devices', deviceSchema);

// modelling of a service
var serviceModel = mongoose.model('service', serviceSchema);

// modelling of a product
var productsModel = mongoose.model('product', productsSchema);

// modelling of a production order
var orderModel = mongoose.model('order', orderSchema);

// modelling of a production order item
var orderItemModel = mongoose.model('orderItem', orderItemSchema);
```

Appendix B – Illustration of Communication Agent's code structure.

```
1   /*******************************************************
2                   Communication Agent
3
4               author: João Ricardo Cardoso
5    *******************************************************/
6   ////////////////////////////////////////////////////////
7   // Mosca requests and iniciation
8   ////////////////////////////////////////////////////////
9   var mosca = require('mosca')···
15  ////////////////////////////////////////////////////////
16  // Global variables
17  ////////////////////////////////////////////////////////
18  var online_agents = {}···
23  ////////////////////////////////////////////////////////
24  // Mosca events
25  ////////////////////////////////////////////////////////
26  // fired when the broker is ready and running
27  broker.on('ready', function ( err , res ) {···
35  // fired when a agent connects on the broker
36  broker.on('clientConnected', function(client) {···
38  // fired when a agent disconnects from the broker
39  broker.on('clientDisconnected', function (client) {···
44  // fired when some agent subscribes on a topic
45  broker.on('subscribed', function ( topic , client ) {···
50  // fired when some agent unsubscribes on a topic
51  broker.on('unsubscribed', function( topic , client ) {···
56  // fired when a message is published on the broker
57  broker.on('published', function(packet, client) {···
80  // fired when a message acknowledge is received for the broker
81  broker.on('delivered', function(packet, client) {···
87  ////////////////////////////////////////////////////////
88  // communication infrastructure
89  ////////////////////////////////////////////////////////
90  // fired when some arrives on broker
91  function broker_processMessage( deviceId , topic , message ) {···
140 // fired when an agent send its properties on topic "register"
141 function broker_registerAgent( device ) {···
185 // fired when an agent disconencts from the broker
186 function broker_unregisterAgent( device ) {···
205 // fired when some agent reports a status change
206 function broker_switchReport( deviceId , tagId , value , timeStamp ) {···
303 // fired when some new production order arrives from the ERP Agent
304 function broker_processOrder( order ) {···
322 // used to forward an order to some agent when it comes from ERP agent
323 function broker_forwardOrder ( nOrder , orderItem , fromLine ) {···
382 // fired when some agent reposts an order's status change
383 function broker_updateOrder ( deviceId , orderInfo ) {···
425 // fired when some agent post a new product structure
426 function broker_addProduct( product ) {···
435 // used to publish agents activities on the broker
436 function broker_logPublish ( method , deviceId , tagId , value ) {···
459 // used to generate broker's time stamp
460 function broker_TimeStamp() {···
466 ////////////////////////////////////////////////////////
467 // yellow pages infrastructure
468 ////////////////////////////////////////////////////////
469 // fired when some service is turned on
470 function yellowpages_Add( nService , deviceId ) {···
477 // fired when some service is turned off
482 function yellowpages_Remove( nService , deviceId ){···
494 // used to find a service on yellow pages, returns an agentId
495 function yellowpages_Find( nService ) {···
503 // fired when some agent requests a yellow pages service
504 function yellowpages_Report( deviceId , nService ){···
```

Appendix C – Illustration of operator oriented Production Agent's code structure.

```
1   /*******************************************************
2                       Production Agent
3
4                author: João Ricardo Cardoso
5    *******************************************************/
6
7   ////////////////////////////////////////////////////////
8   // Global variables
9   ////////////////////////////////////////////////////////
10  var mqtt_client···
22  ////////////////////////////////////////////////////////
23  // Multiagent algorithm through MQTT
24  ////////////////////////////////////////////////////////
25
26  // used when the agent connects on MQTT broker
27  function agent_Connect() {···
235 // used when the agent disconnects from MQTT broker
236 function agent_Disconnect() {···
247 // used to send the registration message to MQTT broker
248 function agent_Register() {···
266 // used to process incoming messages
267 function agent_processMessage( topic , payload ) {···
335 // used when the agent receives a service ON/OFF request from another agent
336 function agent_switchCommand( serviceId , value ) {···
355 // used when agent receives a order service request from another agent
356 function agent_serviceRequest( serviceId , sOrder ) {···
396 // used to send a request of the next agent for the yellow pages
397 function agent_getNextAgent() {···
408 // used to forward the order to the next agent
409 function agent_orderForward( agentId ) {···
427 // used to publish on MQTT broker the status change of a service
428 function agent_publishStatus( serviceId , value ) {···
507 // used to publish on MQTT broker random data to simulate sensor values
508 function agent_generateRandomData () {···
532 ////////////////////////////////////////////////////////
533 // screen operations
534 ////////////////////////////////////////////////////////
535
536 // used to create a row on the srcreen to ilustrate the agent's services
537 function screen_createRow( tags ) {···
600 // used to process ON/OFF commands sent by users through screen buttons
601 function screen_switchCommand( serviceId , value ) {···
649 // used to animate the connection status icon on connection bar
650 function screen_connectionAnimation( online )  {···
```

Appendix D – Illustration of Sales Agent's code structure.

```
1   /*********************************************************
2                         Sales Agent
3
4                author: João Ricardo Cardoso
5    *********************************************************/
6
7   ////////////////////////////////////////////////////////////
8   // Global variables
9   ////////////////////////////////////////////////////////////
10  var mqtt_client ···
99  ////////////////////////////////////////////////////////////
100 // Multiagent algorithm through MQTT
101 ////////////////////////////////////////////////////////////
102
103 // fired when the agent connects to the MQTT broker
104 function agent_Connect() {···
164 // fired when the agent disconnects from the MQTT broker
165 function agent_Disconnect() {···
178 // used to send an order to production agent
179 function agent_sendOrder() {···
197 ////////////////////////////////////////////////////////////
198 // Multiagent algorithm through http
199 ////////////////////////////////////////////////////////////
200
201 // used to get the product list from the cloud service
202 function agent_getProducts() {···
290 // used to get the order list from the cloud service
291 function agent_getOrders() {···
374 ////////////////////////////////////////////////////////////
375 // screen Operactions
376 ////////////////////////////////////////////////////////////
377
378 // used to send a new product structure to the cloud service
379 function agent_addProduct() {···
424 // used to add a product to the order
425 function srceen_addProduct ( nProduct ) {···
464 // used to remove a product from the order
465 function screen_removeProduct ( nProduct ) {···
491 // used to clear the order
492 function screen_clearOrder() {···
499 // used to animate the connection status icon on connection bar
500 function screen_popConnectionStatus()  {···
513 // used to plot the gantt graphic of an order
514 function screen_ganttPlot ( nOrder , div ) {···
```

Appendix E – Illustration of Supervisory Agent's code structure.

```
1   /*********************************************************
2                       Supervisory Agent
3
4                author: João Ricardo Cardoso
5    *********************************************************/
6
7   ////////////////////////////////////////////////////////////
8   // Global variables
9   ////////////////////////////////////////////////////////////
10  var mqtt_client···
14  ////////////////////////////////////////////////////////////
15  // MQTT Operactions
16  ////////////////////////////////////////////////////////////
17
18  // fired when the agent connects to the MQTT broker
19  function agent_Connect() {···
74  // fired when the agent disconnects from the MQTT broker
75  function agent_Disconnect() {···
87  // fired when some agent connects on the MQTT broker
88  function agent_Register ( device ) {···
94  // fired when some agent disconnects on the MQTT broker
95  function agent_Unregister ( deviceId ) {···
102 // fired when some agent reports a status change
103 function agent_switchReport( deviceId , tagId , value ) {···
149 // fired when some agent starts a working routine
150 function agent_workingRoutine ( img ) {···
157 ////////////////////////////////////////////////////////////
158 // screen operations
159 ////////////////////////////////////////////////////////////
160
161 // used to create a row on the srcreen to ilustrate the agent's services
162 function screen_createRow( deviceId , tags ) {···
304 // used to revome a row of an agent from the table
305 function screen_removeRow(row) {···
310 // used to process ON/OFF commands sent by users through screen buttons
311 function screen_switchCommand( deviceId , tagId ) {···
321 // used to plot a graphic of the historical activity of a sensor
322 function screen_plotHistoric( deviceId , tagId ) {···
362 // used to plot a graphic of the production analisys of a service
363 function screen_plotAnalisys( deviceId , tagId ) {···
527 // used to animates the connection status icon on connection bar
528 function screen_popConnectionstatus()  {···
```