POLITECNICO DI TORINO

Master's Degree course in Computer Engineering

Master's Degree Thesis

Design and implementation of a multi-agent system in an industrial environment for monitoring plants status and providing intelligent, predictive information

Supervisors

Giovanni Squillero

Luca Capano

Candidate

Salvatore Incaviglia

Academic Year 2023-2024

To my grandparents,
Salvatore and Alberto,
wherever you are.

Contents

1 Introduction 5

1.1 Research Objectives 5

1.2 Structure of the Thesis 6

2 SPEA: Company Overview 7

2.1 History of SPEA 7

2.2 Automatic Test Equipment & Industry 4.0 in SPEA 10

2.2.1 Technological aspects 10

2.2.2 Industry 4.0 in SPEA 11

2.3 SPEA business units 12

2.3.1 Principal Sub-Units of SPEA 12

2.4 Archimede: SPEA software for proactive and predictive monitoring 14

3 Monitoring and Proactive Solutions for Industrial Plant Management 17

3.1 Industrial Monitoring Systems 17

3.1.1 Features of modern Monitoring Systems 18

3.1.2 Business challenges and the need for monitoring systems . . 20

3.2 IoT Solutions in Plant Management 21

3.2.1 About Internet of Things 21

3.2.2 Artificial Intelligence & Internet of Things 22

3.3 Multi-Agent Systems (MAS): A different Approach 29

3.3.1 Key Features of MAS 30

3.3.2 Multi Agent System for Intelligent & Proactive Management 40

4 Ideation and Design of the Multi-Agent System 42
4.1 System Requirements Analysis and Objectives 42
4.1.1 Identification of business needs and system requirements . . 42
4.1.2 Monitoring and predictive analysis objectives 43
4.2 Multi-Agent Model Design 45
4.2.1 Multi-agent architecture: concepts and chosen approach . . 45
4.2.2 Definition of agents and their respective roles 47
4.2.3 Communication and coordination model among agents . . . 50
4.3 Predictive and Monitoring Features in the Web Application 51
4.3.1 Overview of the fundamental idea behind the web app's design and purpose 51
4.3.2 Visual representation of the conceptual architecture and key components of the web application 52

5 Implementation of the Multi-Agent System & Web App 55
5.1 Predictive Modules Integration 55
5.1.1 Agent 001 integration 55
5.1.2 Agent 002 deployment 56
5.1.3 Machine Agent deployment 58
5.1.4 Statistical Agent deployment 60
5.1.5 Optimizations inside Multi Agent System 64
5.2 Development of the Front-end for Data Visualization 67
5.2.1 Choice of frameworks and reference languages 67
5.2.2 Implementation of the user interface for the visualization of machine states 70
5.3 Implementation of the Back-End Architecture 82
5.3.1 Configuration and integration of Node.js 82
5.3.2 Inside the Back-End implementation 82
5.3.3 Connection between Back-End & Front-End 87
5.4 Demo in SPEA 89

6 Conclusions 93

Bibliography 95

Chapter 1

Introduction

1.1 Research Objectives

The core field of this thesis is the design and implementation of a Multi-Agent System (MAS) tailored for industrial environments, specifically to monitor the status of production plants and provide predictive, intelligent predictions. In modern industrial settings, managing and maintaining equipment demands more sophisticated approaches than traditional monitoring methods. This work seeks to explore how MAS can deliver a real-time, flexible solution that enhances decision-making processes and boosts overall efficiency in plant operations. The agent system aims to implement a real-time system, capable of continuously collecting data and using them to ensure an efficient response in the shortest possible time. The thesis focuses particularly on the collection of data relating to machines, highlighting how, through the historical analysis of production data, it is possible to identify patterns and behaviors that allow a proactive and systematic analysis. This allows to minimize production errors and to intervene promptly to resolve any failures. The agents, integrated within the system, operate actively and continuously, allowing a massive collection of data to prevent problems and monitor the status of the machines during the production process. Furthermore, the scalability of the system is highlighted which, unlike centralized systems, is not limited to a single production area, but is able to provide statistics on different sections of the company plant, offering an overall view of the operating conditions and improving the overall robustness of the system. The paper also describes the development of the multi-agent

system within the Archimede project, a monitoring system adopted by SPEA of Volpiano (TO). Archimede has been designed to monitor production machines in real time, collecting fundamental data on the status of the equipment. This system is not limited to the control of a single area of the plant, but offers a complete view of the operating conditions in different sections of the plant, improving the overall robustness and reliability of production operations.

1.2 Structure of the Thesis

The thesis is structured as follows: in the first part, the reference business context, i.e. SPEA, is analyzed, describing the current activities of the company and its positioning in the semiconductor industry, with particular attention to innovative testing machinery. It continues with a description of the company plant, carrying out an analysis of the internal operating units of SPEA and a mention of the research and development department, which has allowed the company to assume a leading role in technological innovation in the last thirty years. The second part addresses the context of the multi-agent system, examining in depth the characteristics of this artificial intelligence system, which has assumed a fundamental importance in the machinery monitoring industry in recent years. The technical analysis is accompanied by a comparison with past technologies, highlighting the future potential of the system. Reference is made to Archimede, the project carried out by SPEA to integrate predictive and proactive analysis in the production process. In the following chapters, the solutions adopted for the thesis are described, with a mention of the decision-making processes undertaken by the candidate and his collaborators, which led to the adoption of different implementation strategies. The last chapter presents a briefing that illustrates the implementation solutions, previously formulated as ideas and drafts, and describes the results obtained during the final testing phases.

Chapter 2

SPEA: Company Overview

2.1 History of SPEA

The 70s constitute a very important turning point in the Italian and international industrial panorama with regard to the development of the electronics industries. Industries that face the national or international markets need to produce electronic raw materials that have a high degree of quality and to test everything in an initial phase they rely on do-it-yourself operations, which are unsafe and often prone to large-scale errors. Thus the need arises for new industrial establishments to have a guarantee in terms of production that is as fast and efficient as possible. It is within this historical and industrial context that Luciano Bonaria, then a test engineer at General Electric, comes up with the idea of creating a company capable of producing testing machines for electronic devices and circuit board manufacturers. SPEA is thus established, with its headquarters in Volpiano (TO) [1].

Figure 2.1. SPEA headquarter in Volpiano(TO) Source: [2]

The 1980s see the company significantly expand its market in what was then Europe's industrial powerhouse, Germany. This is made possible by the production of new, even more modern machinery, which remains a milestone in SPEA's production for future projects: Unitest 500, the first model with multi-functional architecture, and Digitest, the first automatic digital ICT board tester. The following years open new development opportunities for the company. SPEA thus achieves the historic milestone of being the fourth company in the world in the production of testing machinery for electronic boards (1996). This result is made possible by a strong investment in Research and Development and the consequent approach to the semiconductor industry, leading to the creation of cutting-edge equipment for electronic testing, such as the 4040 mobile probe tester [1].

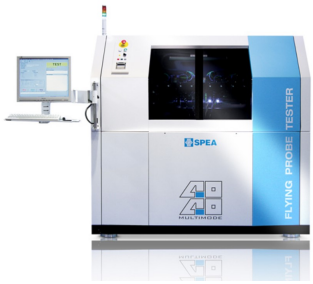

Figure 2.2. 4040 mobile probe tester Source: [3]

This tester ensures effective testing of electronic boards, providing a significant improvement in speed and mechanical precision, and representing a turning point compared to the competition of the time. The years straddling the late 1990s and the early 2000s mark an important crossroads in the world of electronics, witnessing a significant increase in production and a growing need for machines capable of taking on the role of quality control for increasingly complex electronic boards. It is in this context that SPEA emerges as a global leader in developing innovative solutions, which in turn allows the company to acquire numerous clients in the electronics manufacturing sector worldwide. From 2003 onwards, notable achievements include the multi-function in-circuit test platform 3030 and the test cell for MEMS accelerometers. Following the global crisis of 2008, SPEA, thanks to previous investments, successfully overcomes this major challenge, reaffirming its position as a leader in the electronics and semiconductor sectors. This leads to a significant

increase in the production of testers, which now incorporate additional stimuli such as pressure, humidity, UV rays, microphones, and magnetic fields. Over this period, 65% of global ATE (automatic test equipment) is dedicated to MEMS testing, and by 2011, SPEA holds the position as the world's leading company in the testing of inertial MEMS (gyroscopes, accelerometers). The growing process culminates in 2016 with the invention of the 8-axis moving probe tester 4080, combining precision in detection with high speed during testing [1].

Figure 2.3. 4080 probe tester Source: [4]

Today, SPEA is no longer a reality confined to a single continent or limited to America and Canada as in the past (Singapore, China, Germany,Korea), but now has locations in various parts of the world and in addition in terms of turnover it recorded in 2023 a turnover of approximately 190 million euros per year with estimates that see a notable increase for the future. The company now has a workforce of over 1,000 employees, with more than 70% being specialized labor. It is also equipped with the most modern technologies for testing electronic components in autonomous vehicles, Smart Objects, IoT, and systems connected to renewable energy. Automation has also been extended to testers for electronic boards and modules, allowing them to operate autonomously during production. Additionally, optomechatronic testers have been adopted, capable of performing electronic, optical, and mechanical tests on a single system. 2023 marks a year of great growth for

the company, strengthening its international partnerships and gaining new market shares. In that same year, Luciano Bonaria's work also receives significant recognition with an honorary degree in Mechatronic Engineering *'for his exceptional contributions to the field of mechatronic engineering as a designer of innovative testing equipment for electronic boards and circuits, as well as the founder, CEO, and president of SPEA, leading the company over the years to a position of global technological leadership in the sector of testing machines for electronic circuits and systems'* [5].

2.2 Automatic Test Equipment & Industry 4.0 in SPEA

This paragraph focus on the functional characteristics of SPEA's production sector. Through a description of the features of an Automated Test Equipment and an overview of SPEA's Industry 4.0 plan, the intention is to further explore the operational aspects that establish SPEA as a leading company worldwide.

2.2.1 Technological aspects

Automated Test Equipment(ATE) is a crucial apparatus in the production context, capable of performing tests that evaluate production performance at any moment. It includes systems of varying complexity, the Controllers,that execute sophisticated analyses on real or simulated data, thus providing a clear view of performance over time. One of the key principles behind the operation of ATE is Engineering to Order [2], an approach that involves designing and manufacturing the product only after receiving an order. This production management philosophy focuses on the customer, enabling the creation of highly specific and customized products based on the requirements of the client. This method fosters innovation due to the close collaboration between the customer and the company. Additionally, one of the main features of Engineering To Order is the reduced risk of storage, which leads to lower inventory management costs compared to standard solutions. Another advantage is the ability to penetrate niche markets, offering performance levels that traditional companies cannot achieve, resulting in increased profits. On the negative side, the increased complexity due to the high level of product customization

leads to greater management challenges. There is also a higher risk at the production level. Additionally, specialized labor is required, resulting in higher costs for companies that must manage these demands. SPEA, in particular, uses its handlers to mechanically position the electronic components of chips or electronic boards, performing all necessary tests and saving the test data in an integrated database. This approach allows for a reduction in the time needed to identify any faults in the machinery of the production plant and in a reduction of human errors on test phase. SPEA develops this technology to analyze electronic boards and MEMS(Micro Electro-Mechanical Systems)devices that consist of a silicon unit in which mechanical structures and electronic circuits are integrated.
To date, the most important companies operating in the automated test equipment market are Cohu Inc., , Teradyne Inc., Seica S.p.a., Acculogic Inc., and then also SPEA S.p.a. . These companies are internationally renowned giants and their annual turnover has increased significantly in recent years with estimates that are expected to be exponential in terms of revenue.

2.2.2 Industry 4.0 in SPEA

After reviewing the technological aspects related to various Automatic Test Equipment, it is now essential to closely examine SPEA's Industry 4.0 plan, which aims to improve and integrate an intelligent monitoring system within the company's production process, with the ultimate goal of addressing the negative aspects (previously analyzed) of traditional ATE. The goal of the new smart factory, according to SPEA's guidelines, is to integrate intelligent testers connected with the factory's production system with a prescriptive and intelligent maintenance system, aiming to improve the overall quality of production. SPEA's Industry 4.0 plan outlines several clear features:

Highly productive machines: the objective is to generate a large amount of data while simultaneously incorporating intelligent sensors.

Prescriptive maintenance:this involves utilizing IoT (Internet of Things) to develop precise and effective real-time monitoring through sensors, recording data on

temperature, humidity, pressure, and energy waste.

Archimede: SPEA's software that combines sensor data at a higher level to anticipate production risk factors based on the machine's available data.

Integration with MES, SECS/GEM, and TEMS systems: It is based on the targeted and intelligent exchange of selected information between testing systems and various machines in the plant, with an open interface that can suggest possible improvements in the production process. It also involves the exchange of information with products yet to be tested and the company's mobile networks.

2.3 SPEA business units

On this section, the object is to describe the business areas of SPEA in order to better understand how production is organized and how the various departments collaborate with each other to achieve good results.

2.3.1 Principal Sub-Units of SPEA

SPEA can be divided into three main Business Units:

First Unit: represented in Figure 2.4, it manages the design and production of ATE for MEMS semiconductors. In this work unit, products with heterogeneous characteristics are generated. The variety of product characteristics means that the testers used are equipped with specific conditions in terms of temperature and pressure in order to achieve a high level of efficiency for the sensors mounted on the various devices. The handler's purpose is to transport devices for testing. To save time, it can use two buffers, allowing it to move devices simultaneously while tests are being conducted. Within a chamber housing the test jig, specific conditions are created, enabling the device to undergo testing with the support of the tester.
Second Unit: it focuses on special systems that cannot be handled by the other work units due to the high volume of orders and the complexity of the machinery required to meet customer needs. Initially, at the launch of this unit (shortly after 2008), the customers were few, but today their number is steadily increasing in

response to the changing market demands from various companies.

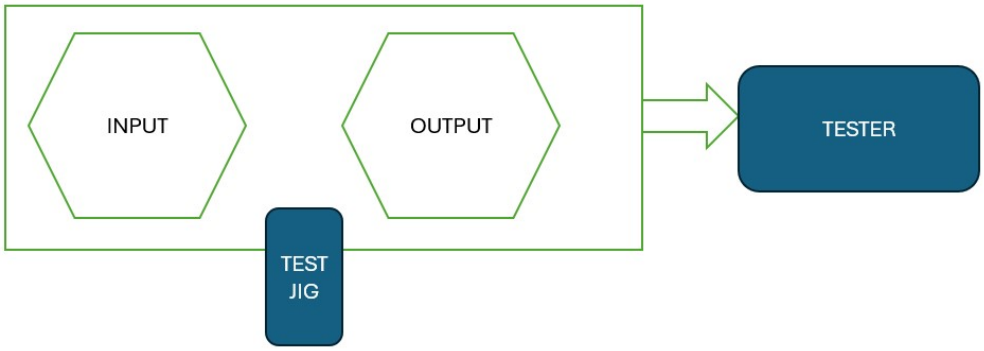

Figure 2.4. First Unit

Third Unit: it deals with machinery for testing electronic boards (bed of nails and flying probe). Bed of nails(figure 2.5) are boards positioned on a testing fixture designed so that the test points can be verified simply by placing the board on top of it. Flying probe devices feature non-fixed test points, where the machinery is equipped with mechanical arms that position the probes at the most challenging access points. This complexity arises from the nature of the electronic equipment, which can make thorough study and testing difficult. The Business Units certainly

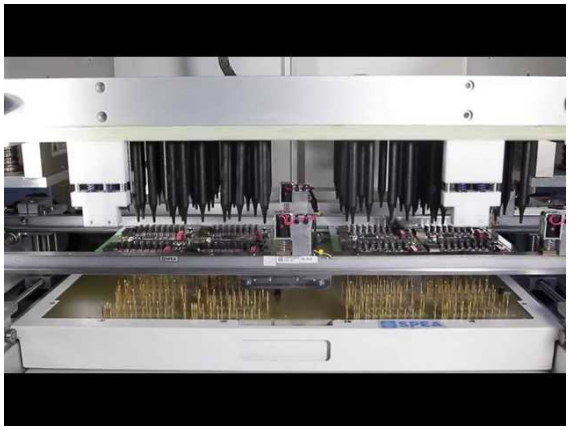

Figure 2.5. Bed of Nails Test Machine 3030

differ from each other in the number of individuals employed in each of them. In fact, it can be said that the majority of SPEA personnel (about 90%) are employed

in the Business Units related to the production branch, while the others are linked to aspects concerning special systems, which are currently having a great importance in production as they interface with prestigious customers and with them the aim is to create specific and highly innovative products. It must also be said that each Business Unit is equipped with one or more purchasing offices, legal offices, and also a Resources and Development office that manages particular, non-standard and highly customized orders.

2.4 Archimede: SPEA software for proactive and predictive monitoring

Archimede is SPEA's flagship software, designed to manage the real-time monitoring of the machinery present in the various industrial plants of the Volpiano company. Its functions are therefore linked to the monitoring of the health status of the machines present within the different Plants, scattered around the world, in addition to the control of the productivity of individual machines and the entire industrial system. Archimede is also based on the need for notifications to be sent to those working on the machines, as well as to the various designers who work in different parts of the world; the main purpose is to represent a degradation at the production level of the machinery which constitutes an important problem to study, especially during the production peak, as it can be costly to have to solve a problem at the various Plants scattered around the world and at the individual machines with too great a delay. The consequences can be disastrous at both an economic and industrial level for the company. From a structural point of view, as

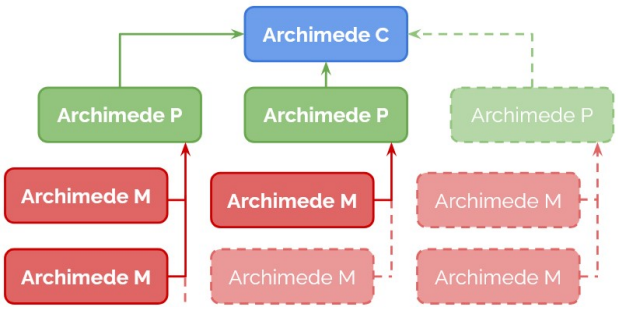

Figure 2.6. Archimede Structure

shown in figure 2.6, Archimede has a 3-level hierarchical structure which represents, from the bottom up, the Machine, Plant and finally Corporate levels.

The Machine level performs its actions in SPEA's automated test equipment producing raw data and time series. It is mainly suitable for small companies; especially for those that own few machines it is possible to use only the Archimede Machine to perform all the computations of monitoring data and machine actions.

The Archimede Plant level instead aggregates all the machines present in the company and allows access to a machine individually. It is already more oriented to companies that own a large and medium or large-sized machine park.

The Corporate level instead aggregates all the company plants present throughout the world and allows individual access to the data of a single plant and to the machines of that plant considered. The cognitive horizon in terms of the amount of data processed is suitable for big companies. From an architectural point of view,

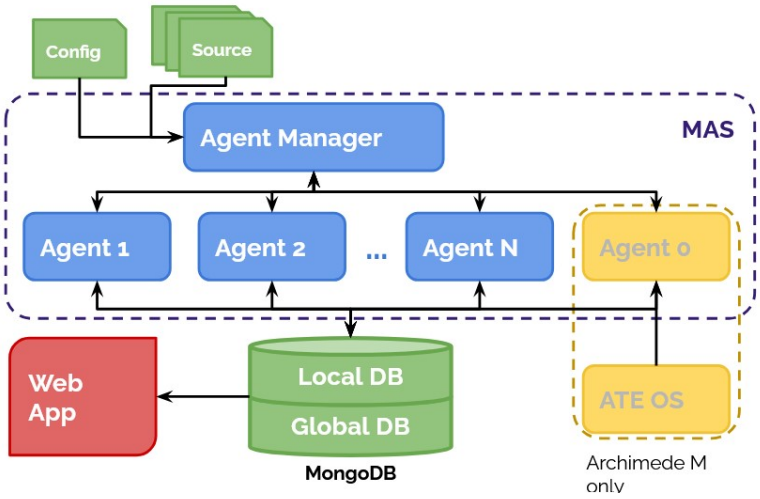

Figure 2.7. Proposed Architecture

the agents that act at the levels seen above, as shown in Figure 2.7, are seen as simple and autonomous agents contained within a container for Archimede M, P, C. The structure that is formed is extensible by nature as the number of agents can vary from time to time based on the different needs that develop at the various

levels for computational needs. The system is also scalable as the agents allow not having a single point of failure and therefore the actions can be divided in a co-operative manner between the various agents. The data produced is inserted into a MongoDB database which has a hierarchical structure and which contains the data to then send it to the reference Web Application, with the aim of displaying real-time monitoring. The structure of the MongoDB database in particular has

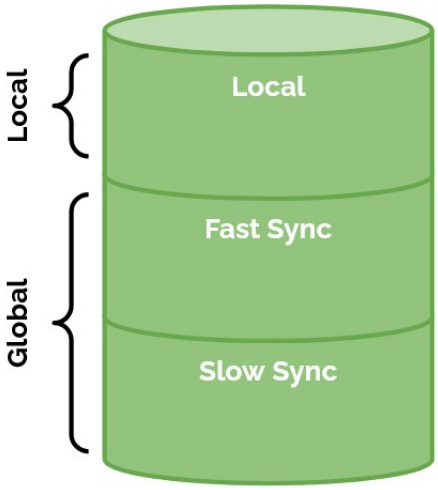

Figure 2.8. MongoDB levels

an innovative form that makes it excellent for computations of different temporal levels. As shown in figure 2.8, to a local part designed to manage data and sensor measurements and to be able to create persistent statistics, a Global DB is also added which in turn has two levels of synchronization of the fast type for critical data and slow for daily data that are produced by the machines. Having made these architectural premises, the aim of the thesis work was to work on this architecture to make it operational and develop the first agents capable of studying the state of the machines present at the SPEA Plant in Volpiano.

Chapter 3

Monitoring and Proactive Solutions for Industrial Plant Management

3.1 Industrial Monitoring Systems

Monitoring systems are used in all industrial sectors to display in real time the status of the equipment involved in the production process. The main objective is to identify any anomalies and promptly report them to operators, allowing them to quickly take the necessary actions to prevent production from being compromised and find efficient solutions. The fundamental principle underlying all monitoring systems currently available on the market, regardless of the technology used for troubleshooting, consists in the continuous collection of data from the machines in production (for example, data relating to temperature, voltage, timestamps, various measurements, etc.) and in using this information to prevent potential failures in the equipment of the company plant. The ability of a monitoring system to prevent failures is based on the risk assessment, which is estimated using the available data. To obtain a rough but effective estimate of the risk, alarm thresholds are often used, whose purpose is to alert operators and allow timely intervention in the event of a failure at an early stage, thus avoiding damage during critical moments of production. This allows to avoid not only production interruptions, but also economic losses and damage to the company's image.

3.1.1 Features of modern Monitoring Systems

The monitoring systems currently used in various companies are capable of handling large amounts of data that are automatically and periodically generated by the machines themselves. These data are stored and managed over months or years, creating a useful database for managing potential anomalies in future processes requiring less and less operator intervention, intelligently identifying problems with the use of predictive analysis [6] [7]. They are particularly effective at detecting problems in their early stages, allowing for the prediction of how the machine will behave in the near future. For the identification of instantaneous or sudden faults, protection systems [8] are used, which employ alarms to immediately signal errors. A fundamental role in the information detection process is played by sensors, which are installed on the machinery and, through a direct connection with the operational center, allow the sharing of relevant information about the machine parameters. The sensors used in monitoring are many and each has a specific role. They can be divided into two main categories: active sensors and passive sensors. Active sensors are those that emit signals, such as light or sound waves, and then measure the response they receive, allowing them to detect any changes in the parameters. On the other hand, passive sensors do not produce signals; instead, they simply detect changes in the environmental energy, such as temperature or pressure, without actively interacting. Both types of sensors are essential for effective monitoring, thus helping to improve performance and safety in production processes. The data are then evaluated by a centralized analysis software, which provides guidance on how to improve the parameters and bring them back within an acceptable range for production.

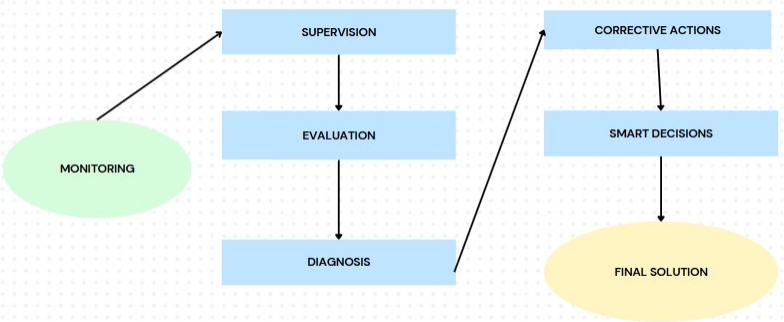

Figure 3.1. Monitoring Flow Chart

Figure 3.1 synthetically illustrates the fundamental phases of the monitoring process and the subsequent search for solutions to problems in the industrial plant. The first phase is Monitoring, during which the changes in state of the machines over time are recorded, within specific Databases. This process is possible thanks to the sensors installed in various points of the machines [9]. Monitoring can adopt different methodologies, including Digital Recorders, which allow tracking the behavior of a variable over time, and Threshold Sensors, which, through comparison with certain limits, allow activating or not a process in the machine.
The monitoring phase is followed by the supervision phase, which is composed of several sub-phases aimed at analyzing the data coming from the monitoring system and providing additional feedback to the structure. First, a thorough analysis of the information collected by the monitoring system is performed. Next, a diagnosis of the variations in the collected data is carried out, assessing whether these changes have caused a failure and, if so, identifying its location. If failures are identified, they must be removed to preserve the integrity of the system that supports the entire production process. The next phase is Evaluation, during which the evolution of the behavior of a parameter is analyzed in real time compared to the past, in order to identify any anomalies or problems. The current state of the parameters under examination is examined and compared with the previous and current behaviors [6] [7], using the real-time data as a reference. The diagnosis phase is used to understand what the situation is inside the controlled machine or device and compares the current situation with specific parameters or thresholds to ensure that there is the possibility of understanding the state of the machine at this precise moment and then, if necessary, implement corrective maneuvers or not. Once the problem has been diagnosed, it is then time to try to remedy the problem itself and therefore we try to use methodologies or techniques capable of correcting various anomalies in the parameters. Among these, the expert systems or ES certainly stand out, which have as their modus-operandi that of using shared whiteboards in order to collaborate with other ES and ensure that using personal knowledge we can arrive at an acceptable solution through the exchange of information, or we use neural networks that evaluate the results of the expert systems thanks to interconnected processing units. This last phase is the one that precedes the final phase of solving the problem that we are trying to mitigate. To date, the solutions that we are moving towards are those that concern real-time predictive

maintenance. This allows to identify problems in an industrial environment without stopping production, in real time and with a real-time monitoring capacity.

3.1.2 Business challenges and the need for monitoring systems

Companies, as already mentioned in the previous paragraph, believe that monitoring systems at production level are essential to ensure that a balance is reached at production level and above all that there is a greater awareness of how production is proceeding over time. At data level, approximately 93% of European companies do not consider their monitoring processes to be unsuitable while approximately 55% of companies are using predictive monitoring and not to analyze production data and try to fix any problems [10]. These are important data that make it clear that the phenomenon has not been taken lightly and that it now plays a leading role within the production fabric, also leading to a hypothetical future profit [10]. The numbers could be even higher, but due to poor infrastructure and a lack of specialized manpower in the process of managing a large amount of data, it can be said that the process is growing but not in an exponential manner [10]. Among the various countries that have focused on predictive labor in Europe, we have Italy, which is one of the nations most attentive to the phenomenon of real-time monitoring, with a percentage of 52% (a unique figure and an absolute record in Europe) of companies that today have these systems in their company [10]. An example that supports this data is given by the transport company Trenitalia, which is using predictive monitoring techniques integrated with IoT systems with reduced maintenance costs, according to statistical estimates, of around 8%/10% [10]. However, the most important danger that companies consider is the problem of downtime that is not monitored correctly and the maintenance to be done on a machine that has had a sudden breakdown [10]. These are multi-purpose issues that can be found in every sector, from transport to manufacturing, and constitute the first obstacle that companies encounter when it comes to making the production sector of which they are an integral part function [10]. Precisely for this reason there is a continuous search for solutions that can resolve everything. In conclusion, monitoring has become a central focus in today global economy. Both at an industrial and economic level, it is essential to safeguard and continually enhance this process to

ensure that everyday life continues to improve.

3.2 IoT Solutions in Plant Management

3.2.1 About Internet of Things

Internet of Things is a computing paradigm that has developed in recent years in parallel with the industrial revolution known as Industry 4.0. The term IoT was first coined by Kevin Ashton in 1999 [11], and subsequently, throughout the early 2000s, it saw significant recognition in the field of Telecommunications, which ultimately led to the official recognition of the term Internet of Things in 2011. If a theoretical and philosophical definition of IoT is to be given, it can be described as an infrastructure capable of connecting people and things everywhere in a constant and precise manner [11]. IoT foundation lies in the interconnection of physical objects through wireless and wired networks, utilizing the same Internet Protocol (IP) [12]. At the core of IoT is the use of sensors and actuators such as RFID (Radio-frequency identification) [13], which are embedded in physical objects, allowing them to communicate with one another. The tracking of individual object behaviors and the improvement of situational awareness have driven the development of IoT over the past decade, making it a crucial tool, sometimes indispensable, in a wide range of contexts from industrial applications to everyday household environments (e.g., Google devices, Alexa, etc.).

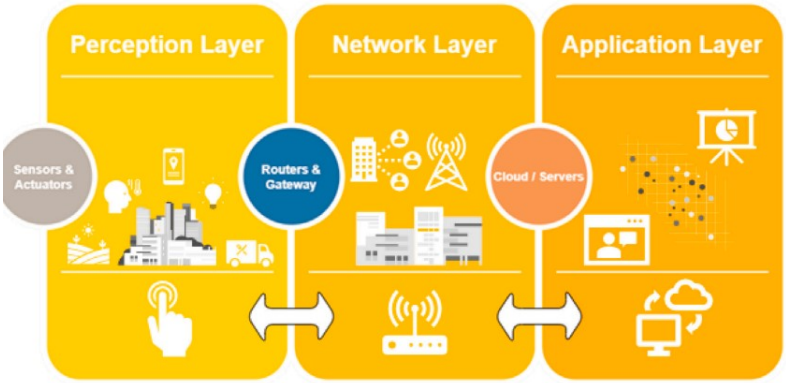

Figure 3.2. IoT system structure Source: [14]

As shown in Figure 3.2 [14], the structure of a system that leverages the operational power of the Internet of Things can be outlined in three main areas or layers [15]. The first is the perception layer, whose purpose is to collect data from the external environment that any Internet-enabled device interacts with, using sensors and actuators. In the second layer, the network layer, the data is transmitted over the network to other devices connected to the smart factory that makes up the IoT system. The third and final layer is the application layer, where an intelligent IoT environment is created for users (such as companies, etc.). In this layer, real-time visualization and the development of intelligent ideas based on the data provided in the previous phases by the devices are utilized. As previously mentioned, IoT technologies rely on communication between devices connected to the same network. To achieve this, various types of connections are necessary, allowing the devices within the system to interact and exchange data with each other. The most well-known protocols in the IoT world today are Wi-Fi, Bluetooth, and Zigbee for wireless communications, while for wired connections, CAN and Modbus are noteworthy [16]. However, the protocol that plays a central role is GSM [16], which enables communication between IoT devices and mobile networks, allowing monitoring in areas where Wi-Fi or Bluetooth is unavailable. Currently, the 5G protocol is also in use, offering low latency and very short response times, alongside the LoRa model [16], which allows data exchange over long distances with significant energy savings.

3.2.2 Artificial Intelligence & Internet of Things

Predictive maintenance using IoT sensors and AI

Predictive maintenance is an advanced approach within the management system of industrial plants during the production process. Today, it combines IoT solutions, based on sensors and actuators that are installed on machines to collect data, in order to collect as much information as possible regarding the operating conditions of the machines. The data collected refers to multiple measurements such as temperature, pressure, vibrations of the machinery and the purpose of IoT systems is to also identify the passage from one state to another by a parameter within the production itself. IoT technologies in the last period have seen a great increase in terms of collaboration with the emerging development of Artificial Intelligence and

data analysis. In fact, there are several algorithms that are used today by industrial plants to make predictions and these are part of the large macro area of Machine Learning that allows machines to automatically learn a series of parameters and process them accordingly.
Among the AI algorithms, we can mention, in terms of importance, Logistic Regression and Random Forest:

Logistic Regression: used for the binary classification of events (**0/1, TRUE/-FALSE**). It can be used to understand whether a machine is broken or not within a production system. The method is based on the probability modeling that a given event may or may not occur, obviously given a set of predictive variables, and transforms everything with a linear combination of variables with results that in a probabilistic way go between 0 and 1 [17]. The formula that describes logistic regression is represented here:

$$P(y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}$$

Where:

- $P(y = 1|X)$ represents the probability of a failure occurring,
- $X_1, X_2, \ldots, X_n$ are the predictive variables,
- $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients estimated through model training.

This way, it is possible to understand which factors have the greatest impact on the risk of failure, through the analysis of the individual parameters.
Logistic regression can be represented by a well-known curve called a Sigmoid(Figure 3.3). It maps any real value to a range between 0 and 1. This means that regardless of the input values, the output will always be limited to this range, allowing for the interpretation of the probability that a machine is in a state of imminent failure. For example, if the output of the logistic regression is 0.85, we can say that there is an 85% probability that the machine will fail soon. As a result, maintenance can be scheduled before the failure occurs.

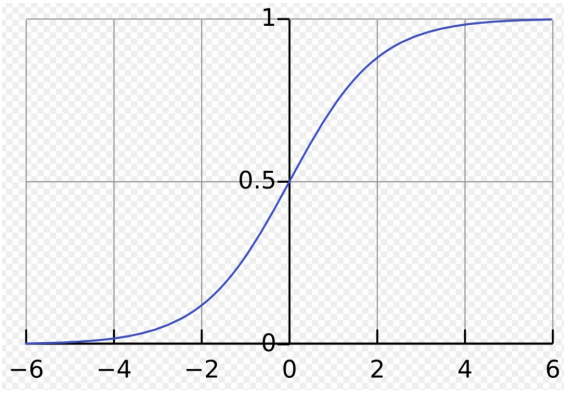

Figure 3.3. Sigmoid Function Source: [18]

Random Forest: Random Forest algorithm is a Machine Learning algorithm that combines the predictions of multiple decision trees with the aim of improving the accuracy of a given model. It is widely used in the IoT field because it allows the analysis of a large amount of data collected by the sensors of IoT machines. As shown in Figure 3.4, Random Forest is an algorithm that builds several trees on sub-samples of random datasets and this is a great advantage because it allows to avoid the problem of over-adaptation that occurs when a model loses generality and cannot be used for purposes other than those related to the context (current dataset) in which it is used [19]. In the case of the Random Forest each tree contributes in its own way to the prediction and the final result is equal to the majority of votes expressed by the individual trees as follows:

$$H(X) = \text{argmax}_c \sum_{i=1}^{n} I(h_i(X) = c)$$

Where:

- $H(X)$ is the predicted class, for example, "failure" or "no failure",
- $h_i(X)$ is the prediction of the individual tree,
- I is an indicator function that checks if the tree assigns the class c.

The approach using the Random Forest Algorithm allows to combine data from different IoT sources such as temperature, pressure, vibration and thus provides us

with a more versatile and precise model [20]. Another feature that makes Random Forest very useful is that it is robust even if there are missing values or noise [19].

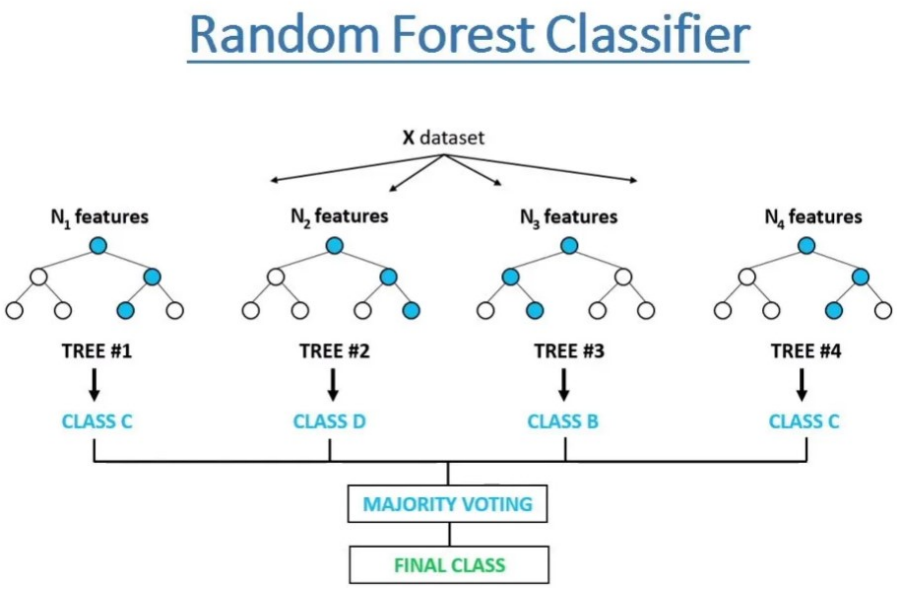

Figure 3.4. Random Forest Trees Source: [21]

Real-time monitoring with AIoT

The union between artificial intelligence and the Internet of Things is opening the doors to an ever-evolving future. This powerful synergy allows systems to learn from data collected by connected devices, to identify patterns and to make autonomous decisions [22]. Predictive maintenance is just one of the many examples of how this technology is transforming production processes. AIoT has permeated the entire scientific world and so these technologies are becoming increasingly crucial in sectors such as healthcare, where they allow real-time monitoring of patients' health status and to personalize therapies, agriculture, where they allow to optimize the use of resources and increase productivity, and smart cities, where they facilitate traffic management, public lighting and waste collection. The development of AIoT, in addition to placing an emphasis on great development, also leads to addressing another issue, namely privacy, security and ethics, especially for the huge amount of data that is exchanged every day between the devices that make up the various company plants but also homes and everyday life. All this to ensure

sustainable and responsible development of these technologies.
Intelligent systems can anticipate failures and malfunctions using simple historical data and thus optimize maintenance management, thus reducing machine downtime during peak production [22]. To do this common AI algorithms that systems use to real-time monitoring are:

K-Means Clustering: clustering algorithm that is based on the partition of data into K clusters(Figure 3.5). Each cluster is represented by a centroid that is the average of the data points belonging to that cluster. In this algorithm, first of all there is a random selection of the K initial centroids, then each given point is assigned to the cluster whose centroid is closest and finally the centroids are recalculated as the average of the points assigned to each cluster. The whole process is repeated until the centroid value remains fixed. The following formula permits to obtain the distance between a point and a centroid:

$$D(x_i, c_k) = \sqrt{\sum_{j=1}^{n}(x_{ij} - c_{kj})^2}$$

where n corresponds to the number of dimensions of the data [23].

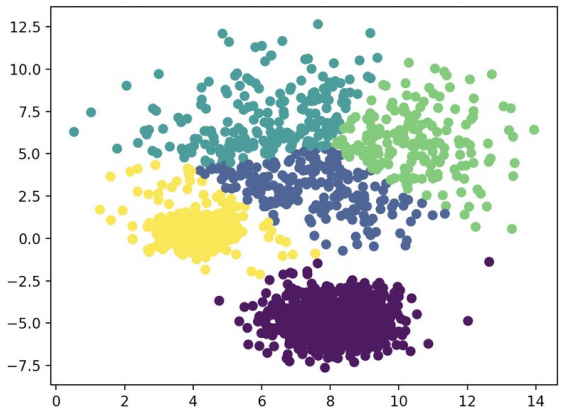

Figure 3.5. K-Means with 5 clusters Source: [24]

Auto-encoder: Auto-encoders(Figure 3.6) are a neural networks that are made up of two parts, the encoder and the decoder. The first compresses the input and encodes it while the second tries to reconstruct the original input from the encoding. A loss function such as the mean squared error is used to measure the output with

the input and ultimately understand how far away it has gone from the original representation. The mean squared error can be defined as:

$$L(x, \hat{x}) = ||x - \hat{x}||^2$$

where x is the original input $\hat{x}$ is the final output.

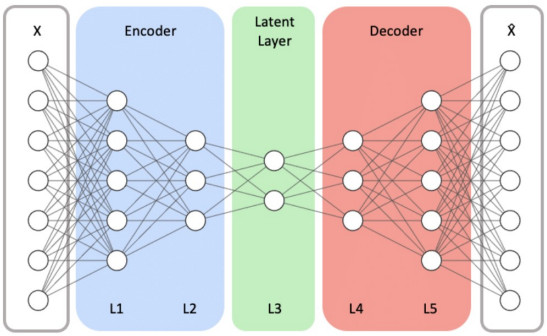

Figure 3.6. Auto-encoder architecture Source [25]

These two approaches, although different, are often processed using **Edge Computing** [26], which allows data processing to be done close to the source rather than transferring it to a decentralized server, leading to a significant performance boost. A good example of this is a plant using IoT to monitor vibrations and other parameters in machines. By analyzing data in real-time with algorithms like K-Means, various anomalies in the machine's behavior can often be identified, allowing immediate action to address any issues in production.

Digital Twins for plant monitoring

Digital Twins(Figure 3.7) is a new and emerging technology that allows the replication of a physical process or system in a fully digital form. The birth of the term Digital Twins occurs around the early 2000s when NASA used modern virtual systems to simulate the space conditions that their spacecraft would have to encounter during space travel [27]. The system thus allowed them to collect data in real time and therefore to predict possible failures that could compromise the quality of the future mission [27]. The Digital Twins bases its operation on sensors,

advanced models, real-time systems that have the aim of capturing in real-time the operating conditions on which the system operates [28]. The data collected is continuously compared with the digital model through simulation and machine learning techniques to faithfully replicate the physical reference model [29]. Simulation is essential as it allows for testing future and hypothetical scenarios, quickly and accurately capturing changes in operational parameters. However, the Digital Twin goes beyond this, enabling modifications to the real system through actuators, generating a feedback loop between the digital and physical worlds and vice versa [30].

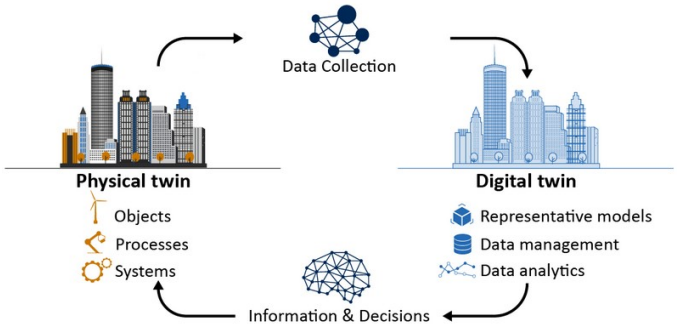

Figure 3.7. Modern Digital Twin Chain Source: [31]

In the context of monitoring and predictive maintenance, the Digital Twin plays a fundamental role. This approach allows to continuously monitor the state of a machine or a plant and to identify signs of deterioration or malfunctions before they turn into actual failures. For example, a Digital Twin used in a production plant can collect data related to vibrations, temperature, pressure and other critical parameters of a machine, processing this data in real time to identify anomalies. Through machine learning techniques such as time series analysis or predictive analytics, the system can anticipate failures and suggest preventive maintenance interventions [32].

The advantage of the Digital Twin is that it allows to test different maintenance strategies without interrupting the physical operations of the plant. For example, a predictive model can use algorithms such as K-means clustering to classify different operating conditions of a machine and identify when it is entering a critical state. At the same time, the algorithm can predict the time remaining before a failure occurs, allowing operators to plan maintenance activities in advance [32].

This process reduces the costs associated with unexpected failures and increases the operational efficiency of the plant. Another relevant aspect of the Digital Twin in this context is the ability to simulate the effects of maintenance interventions. This allows engineers to see in advance what the results of a certain action will be and choose the most effective maintenance strategy [33]. In this way, operations are optimized and machine downtime is minimized.
Today many company have invested the most in the digital twin phenomenon, in particular from the already mentioned NASA and smaller aerospace-related companies but in other companies such as Siemens and NVIDIA invested a lot of money in programs that include digital twins in production. Siemens and NVIDIA for example, have formed a partnership in recent years with the goal of integrating their monitoring, advanced computing, and industrial simulation systems to create a product capable of providing excellent predictive maintenance and safe production optimization [34]. Omniverse is the flagship product created by NVIDIA in this field, allowing the creation of realistic virtual environments in real time [35]. Xcelerator, on the other hand, is a portfolio of digitalization software, enabling companies to develop simulated industrial plants, thanks to the power of AI and the powerful GPUs made available [36]. These two products have been combined to create dynamic digital twins, with continuously updated models and strong capabilities in testing, monitoring, and optimizing the industrial systems where the final solution is applied.

3.3 Multi-Agent Systems (MAS): A different Approach

This section aims to illustrate the operational paradigm of the Multi-Agent System (MAS), in order to deepen the central technical aspects of this thesis. The study of the MAS starts from the definition of what is meant by Agent in terms of programming and functions within a distributed system and not to then move on to the description of the fundamental characteristics that make up a Multi Agent System. The chapter continues with a literary overview that sees the Multi Agent System as its central theme and then finally focuses on the applications of this paradigm in today's monitoring and predictive analysis industry [37].

3.3.1 Key Features of MAS

Multi Agent System is a system in which many agents act that have a common goal and try to cooperate and share their knowledge to ensure that by joining forces they reach the goal. Each agent is therefore part of a mechanism that tries to divide the tasks to produce a final output. The following subsection, starting from the definition of Simple Agent and then moving the focus to Multi Agent systems and their composition, intends to delve into the technical aspects that make agents and multi agent systems unique solutions in their kind with regard to their application also in the various sectors of industry and technology.

About Agent

An Agent is an entity that, thanks to the use of sensors and actuators, can receive and use in turn the signals captured from the surrounding environment [37]. This is obviously a partial definition that refers to the application of Agents in a purely industrial context, especially with regard to the analysis of the production systems of a normal company. That said, what mainly characterizes a normal Agent that lives within a more or less complex system are some properties, which make it unique in its kind, mainly for its utility [37]. Among these, we certainly include the ability of an agent to be independent in the choice of actions to carry out. In fact, each agent is an individual who acts regardless of the actions of others in the same system. It is not disturbed by what is outside but is stable in the computations that it carries out in a given environment [37]. Agents are also able to take and critically evaluate the changes that occur in the system in which they operate, that is, they take and analyze the data of the environment on which they are performing analyses to deduce a certain behavior. Agents, however, act on the environment on which they are mounted and can influence or not the activities carried out [37]. The Agent is then real-time [37], that is, it must analyze the data that is provided to it at the moment and immediately after provide accurate analyses that can influence the operational life of the system in which it operates. Obviously, it should not be forgotten that the agent must not only interact with the environment but it must also be able to adapt to changes in order to continue with its objectives as well as cooperate with other agents so as not to disturb each other on the analyses and to be able to collect higher level information to reach the goal in the shortest possible

time [37].

Focus on MAS

As already mentioned in the introduction of the main paragraph, Multi Agent Systems are a distributed solution for more or less complex problems and they are used to solve multiple tasks in parallel that then have a common root, which is to reach a final result for the system in which the MAS are adopted. However, what makes Multi Agent Systems have taken hold as solutions to complex problems are some characteristics that make them unique and useful [38]:

Scalability, as each agent is responsible for its own sub-area and can be removed or added based on the computational needs of the system.
Speed, as each agent usually performs computations limited to a particular area and does not cover areas that are too large that would take a long time.
No single point of failure, as the agents are part of a distributed system that ensures that the structure is not centralized, but decentralized and with computations that occur with a high degree of parallelism.

These aspects, although positive, are accompanied by other factors that can still be a weak point or a point to pay attention to in the structuring of a multi-agent system. Indeed, although an agent's actions are limited to a specific part of the environment in which the MAS operates, the decisions made by an agent in a dynamic system can also affect those of other agents. Therefore, it is crucial to recognize that the parallelism between agents' actions might not always be complete, potentially leading to situations where one agent's choices negatively impact the dynamics of another [38]. This issue is compounded by the partial view each agent has of the environment, which, while advantageous in some respects, can also create a bottleneck in the results, as they are based on an incomplete understanding of the system's state. Sharing information among agents could resolve this, but it becomes a significant challenge when agents are in competition, as they may act erroneously on the same portion of the system state [37]. The issues just analyzed therefore raise a question regarding the possible organization of the agents. This aspect has a particular focus and the management of the various levels in which

the agents can develop constitutes an element of great interest for understanding the communicative aspects and those more closely linked to possible interferences in the results produced [37].
The first structure in which a MAS can be found is the hierarchical one [39]. This structure means that the agents are placed in levels and that each level ensures that the agent of a higher or lower level can communicate with another agent at a subsequent level as if there were a tree structure, typical of the most common computer science problems.

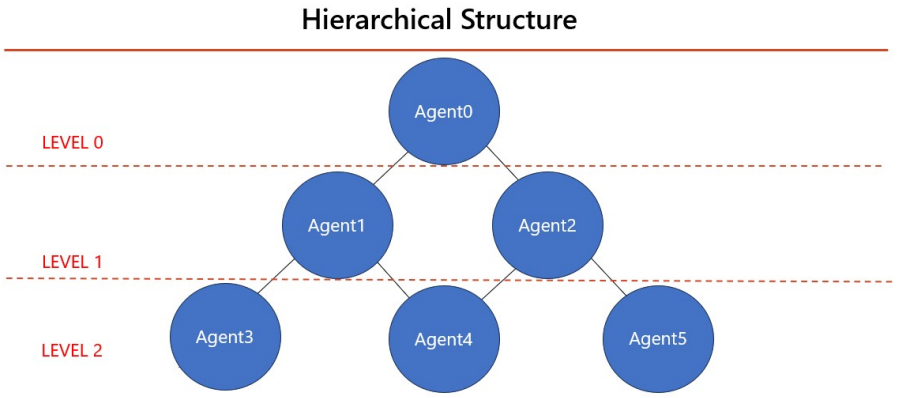

Figure 3.8. Hierarchical Structure

As shown in Figure 3.8 the hierarchical structure is composed of levels and these levels are one the consequence of another in terms of importance of a certain task or in terms of outputs that can be preparatory to the action of another agent. Figure 3.8 above shows an exemplary abstraction of the hierarchical structure that in reality can be traced back to two typologies of hierarchical structures which are the simple one [40] and the uniform one [37]. The two typologies have different characteristics and the most important difference is given by the different management capacity, as the simple hierarchy is more vulnerable to failures because it depends on a single agent, while the uniform hierarchy is more robust thanks to the distribution of decision-making power and also has a gradual degradation of the system in order to avoid collapse in case of multiple failures in a system and decisions are taken by agents with the necessary information and are sent to higher levels only in case of conflict between agents in different parts of the hierarchy.

The two structures just described have been an important starting point in the development of more efficient and stable multi-agent systems in terms of performance over time. As a result, more complex structures have been created, adding an increasing degree of complexity to the resolution of different problems and improving the management of dynamics that previously could have been a significant bottleneck in the creation of intelligent multi-agent systems.

A first example of a more complex structure that has been studied for the development of new multi-agent systems is the *Holon structures* [41]. These structures (Figure 3.9) allow the multi-agent system to be seen as a large pool of agents, with a single main agent on the outside that contains many sub-agents inside. These sub-agents exchange information with each other in order to achieve a suitable result. The sub-agents, in turn, group together into subgroups, each of which has a Head Agent whose role is to be the "spokesperson" for the results obtained by the group they lead. The Head Agent is usually chosen based on criteria that range from complete randomness or according to a principle that considers the composition of the general architecture of the subgroups and the resource availability of the agent in question [37]. Although this model may seem like a simple evolution of the hierarchical structure, it stands out for the high level of autonomy granted to the agents [42]. However, this also presents a disadvantage in terms of knowledge of the architecture, which remains often variable and unpredictable, even in decision-making [43].

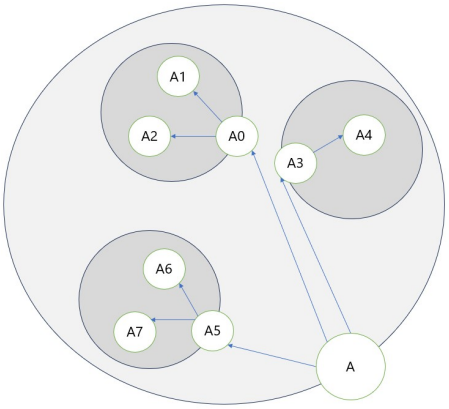

Figure 3.9. Holon Structure

Another architecture, which takes inspiration from certain characteristics of the Holonic architecture, is the *Coalition Architecture*. This architecture has features that make it dynamic but also difficult to manage at the same time. In fact, the ability for groups of agents to aggregate leads to the creation of temporary coalitions, either hierarchical or flat, [37] which may have leader agents representing entire coalitions or may result in overlaps where agents temporarily exchange information to achieve the goal and then separate again. Therefore, the key characteristics of this type of multi-agent system are the dynamism in the creation of coalitions and the high reorganization capacity of agent groups, making this type of architecture a good alternative in the development of high-capacity multi-agent systems [44].

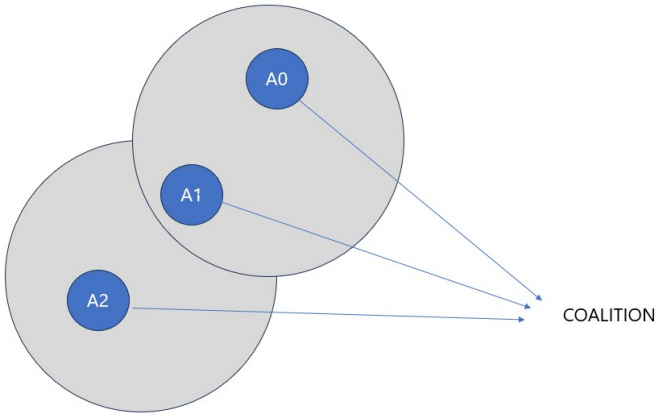

Figure 3.10. Coalition Structure

Figure 3.10 above shows a clear example of a coalition structure in which we have the presence of an overlap by an agent, which will act as a glue in terms of knowledge between one group of agents and another. This is the great advantage of the coalition structure which, yes, implies a greater cost in the separation of common agents, but ensures that most of the information is exchanged between adjacent groups to reach the goal of the system in the shortest possible time.
A final structure that can be considered as a further evolution of the coalition structure is the *Team structure*. In this architecture [45], agents collaborate within a group to improve the overall performance of the group, with the goal of enhancing both efficiency and speed over time. The issue of subgroup size in this architecture is crucial, as small groups may lead to sub-optimality, while large groups may face

challenges in managing the environment [46]. Finding the right balance between team size and system efficiency is essential to optimize performance.
After seeing this overview of the various MAS structures we can say that the exploration of multi-agent architectures, from coalitions to teams, reveals a continuous evolution of strategies aimed at optimizing collaboration and performance. Each system balances autonomy and coordination differently, adapting to the complexity of dynamic environments. The true strength of these architectures lies in their ability to flexibly reorganize, providing intelligent solutions to ever-changing challenges.

Agents Communication Strategies

Communication is one of the most significant aspects of multi-agent systems, as effective communication leads to reduced costs and instability within these systems. It can be local in nature, relying on direct information transfer between one communication system and another, or it can utilize a blackboard system for information exchange [37]. Regarding local communication (Figure 3.11), it can be stated that it is based on a distributed architecture specifically designed to mitigate failures of central agents [47].

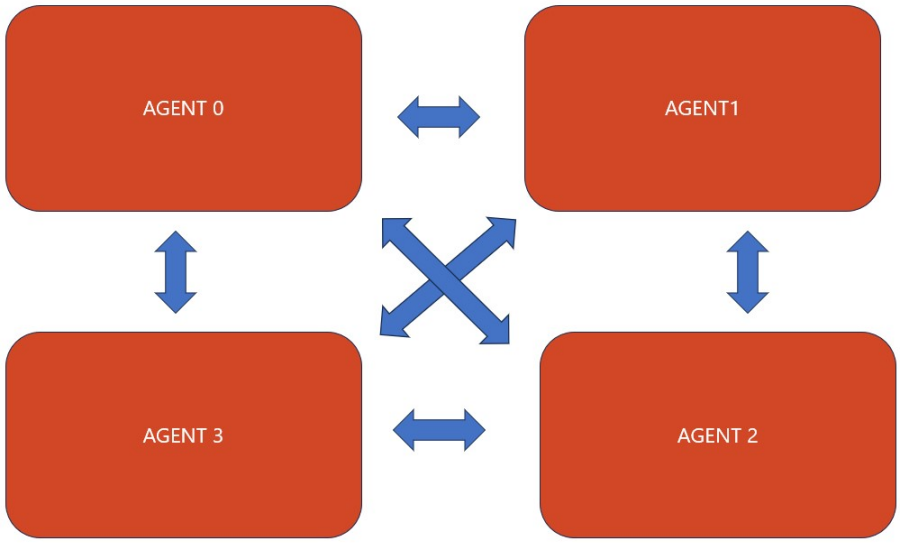

Figure 3.11. Local Comunication

Figure 3.12 instead focuses on the structure with blackboard communication. The blackboard is a repository where there is continuous storage and retrieval of data for the design and control of the agent system. There is a control shell that manages access to the blackboard by the agents and allows for data sharing [48]. This mechanism, although dependent on the blackboard, is widely used as it can mitigate failures through continuous data redundancy, which, however, raises an additional question regarding the large amount of redundant data provided by the various agents.

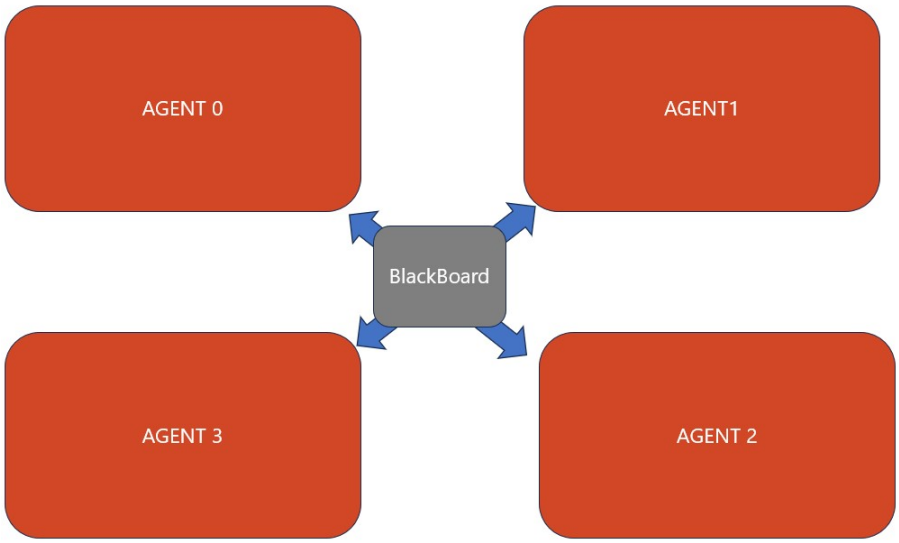

Figure 3.12. Black Board Comunication

After having seen the maxims in terms of communication between agents that are part of a multi-agent system, it is necessary to make a digression regarding the coordination of communications in multi-agent systems. In fact, since multi-agent systems are composed of different agents that act most of the time in a completely independent way with respect to other agents on the same level, it is normal and necessary to understand which are the solutions that have been adopted to allow a connection between agents in terms of communication, which therefore does not create a bottleneck and allows the smooth flow of the data flow that make up multi-agent systems. Certainly a prominent place in these terms is occupied by the *Graph Representation* [49], which allows complex problems to be broken down into more manageable parts. Using coordination graphs, each problem is divided

into smaller sub-problems, thus facilitating the calculation of optimal joint actions. This approach, based on the assumption that the results can be expressed as linear combinations of local solutions, proves to be particularly useful in critical scenarios. By adopting techniques such as variable elimination, agents can coordinate more efficiently, reducing the risk of communication congestion and optimizing their overall performance.
Then we have the representation with *Beliefe Models* [37] that is based on assumptions that see time as a fundamental parameter. This makes sense if we are talking about agents that must be extremely co-ordinated in order to exchange information and above all to perform operations that may be more or less onerous, and therefore we must adopt a model that is able to recognize the environment in which it operates dynamically and quickly. This can be done by following different strategies that may vary from the adoption of evolutionary methods combined with the use of neural networks [50] or methodologies based on simple heuristics [51].

Machine Learning Approaches for Multi-Agent Systems

Machine Learning (ML) is a field of artificial intelligence that allows systems to learn from data and enhance their performance over time without the need for explicit programming. Over time, it has grown from simple methods of recognizing patterns to more sophisticated models capable of tackling complex problems. This progress has been fueled by the increasing amounts of available data and improvements in computational capabilities. The most common Machine Learning techniques use probability calculation as their main reference point, which has its own specific foundation on the data provided by a given reference system that is to be analyzed. It is also for this reason that among the numerous techniques advanced with regard to the development of Intelligent Multi-Agent Systems, the calculation of probabilities concerning the occurrence of certain system conditions has played a central role. The calculation of probabilities is based on the assumptions related to *Bayes Theorem* in which the posterior probability is determined by using the prior probability of an event and the relative *likelihood*, which is an estimate based on the observations of the samples of the reference system [37]. According to Bayes' Theorem, the posterior probability $P(\theta|X)$ is given by:

$$P(\theta|X) = \frac{P(X|\theta) \cdot P(\theta)}{P(X)}$$

where $P(\theta)$ represents the prior probability, $P(X|\theta)$ is the likelihood, and $P(X)$ is the marginal likelihood or evidence, which can be computed as:

$$P(X) = \int P(X|\theta)P(\theta)\, d\theta$$

This framework allows the updating of probabilities as more data becomes available, refining the estimates of the model parameters.
The definitions above, however, makes the actual application in a MAS by a single agent problematic since the actions of multiple agents influence the continuum system and this implies that the estimates that one agent performs at one point in time are immediately changed given the action that was performed by another agent even at the same time.
In order to have estimates that can be less dependent on the reference environment, even though in reality there is always a degree of dependence, one can switch to a Multi Agent System representation that sees agents connected to one another like synapses in a representation similar to that of normal Neural Networks [37]. Here the learning is given by the adjustment of the weights between the layers (back-propagation) that make up the normal neural network so that one layer can affect the estimates of the environment on which the neural network acts in a more or less marked manner. Among the various examples that can be given in this regard are those based on MAS using feed forward neural networks [52], which are a particular type of artificial neural network in which information flows in one direction only, from input neurons to output neurons, through one or more hidden layers.

An other Machine Learning technique that differs from those seen so far in its operation is that dictated by the model called Reinforcement Learning (RL). In this approach, the goodness of a choice is characterized by a reward, which provides feedback on the quality of an action taken, irrespective of whether the final goal has been achieved or not. In essence, the system learns which actions are most effective in achieving the goal based on the feedback received. It may be described

as a sequential decision-making process in which an agent interacts with the environment in a series of states S_t, and at each step t, the agent selects an action A_t based on a policy π, which attempts to map states and actions. After performing the action, the agent receives a reward R_t and transits to a new state S_{t+1}. One of the most widely used mathematical models for reinforcement learning and which also has its basis in actions in a Multi Agent System is the *Markov Decision Process* [53], which is based on sequential models and in which the decisions to be taken depend only on the current state and not on past history. A Markov decision process is described with these parameters(S, A, P, R):

S: set of possible states.
A: set of the actions.
$P(s'|s, a)$: transition function,to define probability to pass from the state s to the state s' with the action a.
$R(s, a)$: Reward Function, that gives us reward for the action a to the state s.

Here, the agent's goal is to maximize the expected value of the future return, which is the sum of the rewards discounted. The function that needs attention is the action-value function (Q-value), which is defined as:

$$Q^{\pi}(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)|S_0 = s, A_0 = a, \pi\right]$$

The function above can be maximized using recursive methods that aim to explore the solution space associated with a given action, applying dynamic programming techniques that continue the exploration until an optimal solution to the problem is identified. This leads to determining the best policy π for the specific system state, identifying the solution that best represents the entire process [37].

3.3.2 Multi Agent System for Intelligent & Proactive Management

As highlighted, multi-agent systems have been widely used in various industrial and operational contexts to optimize the management of production plants and support peak production periods. During these phases, companies must manage large volumes of data, which are essential to keep production smooth and optimized. In addition, continuous monitoring of anomalies allows real-time control of the status of the machines, allowing rapid reactions to any unexpected issues. This approach helps reduce downtime, thus increasing the operational efficiency of the plant. However, the operational power of multi agent systems has also been used in operational sectors not necessarily related to industrial production, strictly understood as the production of objects or goods to be marketed. In fact, for example, over the years, experiments have been seen regarding the adoption of MAS to improve the allocation of resources and the management of shipments in automated warehouses with the aim of improving the positioning of products or the flow of materials, or MAS have been used to manage urban transport networks, with the aim of coordinating the flow of vehicles and optimizing travel times.

Real-World Applications of Multi-Agent Systems in Industry and Beyond

After reading the overview of industrial and non-industrial applications of the multi-agent system, we can now take a look at the case studies of various research and scientific journals that over the years have studied the operational capacity of multi-agent systems in applications that concern the most varied contexts.

A first example is the one related to the decentralized demand management systems highlighted in the reference [54]. Here each agent is programmed to respond to demand signals and price fluctuations, making it possible to have an automated response that balances the system in real time. In this way, the Multi Agent Systems are able to optimize the use of energy with consequent stabilization of the electrical network and reduction of load peaks.

Another case study is the demand response architectures for smart grids, which

can be based on MAS to coordinate distributed energy resources including renewable energy. Multi Agent Systems in this case improve the stability of smart grids through a careful management of energy resources, and they adjust in real time the loads and generation capacities according to the grid specifications (Parhizkar et al., 2014) [55].

In the logistics sector, one of the most innovative applications of Multi Agent Systems is the one related to Physical Internets, where MAS are used to manage logistics flows and freight transport networks with a global interconnected network. The agents employed in this context operate independently from each other and manage resources or coordinate various loading and unloading operations (Montreuil et al., 2016) [56]. The agents in this context are represented as logistics nodes such as a warehouse or a transport hub, and communicate with other agents to minimize delays and improve the robustness of the network connections.

Multi Agent Systems have also seen application in urban transport networks for traffic management and optimization. In this context, each agent acts operationally on a segment of the road network and through communication with other agents that study another road section, they are able to predict congestion and improve travel times as well as manage unexpected events such as accidents or various slowdowns (Farahani et al., 2013) [57]. City contexts related not only to traffic management but also to the creation of smart cities and smart buildings represent another emerging area where Multi Agent Systems have been applied for energy optimization. Here, agents within MAS coordinate their tasks autonomously to reduce energy consumption and also improve efficiency in complex urban environments with dynamic resource management [58].

The examples just described demonstrate how Multi-Agent Systems have gained significant attention over the years and continue to be a crucial element in the development of intelligent systems both in industrial contexts and beyond.

Chapter 4

Ideation and Design of the Multi-Agent System

4.1 System Requirements Analysis and Objectives

The main objective of this section is to delve into the design and conceptual phases that guided the subsequent solutions realized during the course of the thesis work. The intention is therefore to examine in detail the decision-making process that guided the choice of the final architecture for the Multi-Agent system, attempting to offer a complete overview of the alternatives considered and the motivations that led to the choice of specific technologies . In particular, the functional and non-functional requirements that influenced the approach adopted will be discussed, as well as the challenges that emerged and the innovative solutions deployed to address them, with particular emphasis on the key ideas that inspired the system configuration, highlighting the added value of each choice for the application context in which the multi-agent architecture was implemented.

4.1.1 Identification of business needs and system requirements

The thesis project began with an initial phase of gaining an in-depth understanding of the operational structure in which we would later be working. To achieve this,

initial meetings were held to align the team on the primary objectives to pursue. During these sessions, the programmatic points were presented, highlighting from the outset the need to develop an operational system capable of representing the operational units of the machinery within the Volpiano Plant, which serves as the main reference point for this work.

During the first meeting with the company, the Archimede software was presented in detail, and a guided tour of the production department allowed us to directly observe SPEA's operational units and machinery. This analysis allowed us to clearly define the main goals of the research work, which are:

- Developing a system that could serve as a foundation for the future creation of a large-scale commercial software for real-time monitoring.
- Designing a system that, in the short term, could be used by SPEA to monitor the real-time operational performance of its machinery.

In addition, the guided tour gave an insight in the field into the importance of real-time monitoring for employees working in close contact with machinery. A rapid analysis of possible malfunctions allows them to be highlighted immediately in the eyes of the operator, who in the specific case of SPEA is in charge of assembling electronic boards on a machine, thus facilitating the detection of possible faults in the testing of the same. This aspect is crucial, since placing a machine on the market with faults that are not detected in time can lead not only to economic damage for the manufacturing company, but also to serious damage to its image. In fact, ineffective monitoring software, if poorly designed, would not be able to alert the designer and operators of machine-related problems.

4.1.2 Monitoring and predictive analysis objectives

The second phase of the conceptual study focused mainly on analyzing the agent scheduler provided by SPEA, in order to understand how to co-ordinate real-time monitoring and thus establish the current objectives to be achieved in order to obtain an adequate analysis of the company's plant. This required access to the data format, still at a conceptual level, which could be extracted from the machines in production. At this stage, agents had not yet been introduced, as before even their

conceptual development was defined, it was essential to understand the monitoring objectives and conduct a thorough study of the available data.

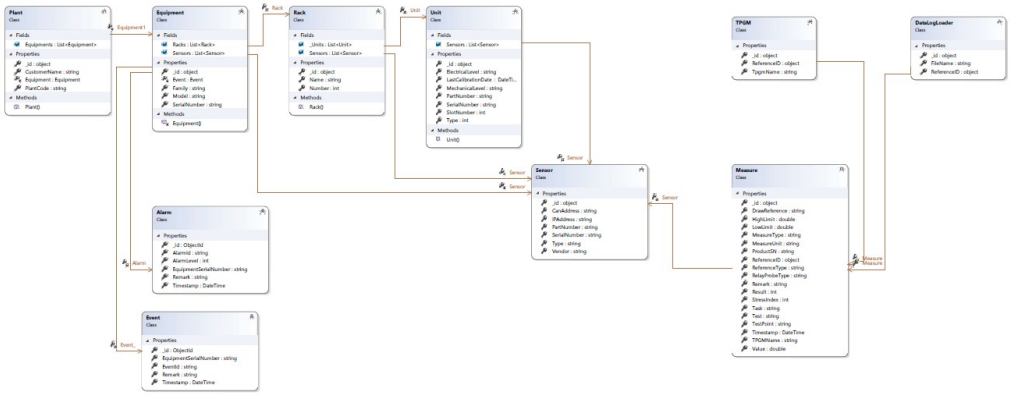

Figure 4.1. Diagram Figure of elements in production

The figure above shows the conceptual scheme, based on the format of the data extracted from the machines via the sensors. This schema was useful in subsequent SPEA meetings to identify which data were really relevant for a multi-agent systems-based analysis. In particular, it emerged that the scheme follows a precise guideline: sensor data can be formally divided into four basic macro-sets, which contain all the information necessary for an accurate machine analysis.

These conceptual ensembles are:

- **Events**: Data on events recorded by the machines over time, useful for monitoring and understanding what events occur in the system.
- **Measures**: Measurement data, which, although large in volume, contain key parameters for predicting potential anomalies and system behaviors.
- **Equipment**: Data related to the equipment, which clearly specify which system is being referenced, helping to maintain focus on the monitored system, even in a multi-agent context.
- **Production Data**: Data on the machine's status, allowing the understanding of its last operational state and the various states that occur during the production process of the machine or machines in the plant. This set is not

specifically represented in the diagram, but was deduced from the combination of the measurement data collected by the sensors and the information on the equipment and events recorded in the machinery under investigation. These sets were highlighted in order to determine the direction for the monitoring process. Specifically, they played a crucial role in the early developmental stages, helping to identify which elements should be analyzed in greater detail and which ones could be given less attention. It became clear that the most relevant data are those related to the sensor measurements, as they provide technical information about the machines, which could be of significant importance for the subsequent development of the agents.

4.2 Multi-Agent Model Design

4.2.1 Multi-agent architecture: concepts and chosen approach

The definition of structured data sets, the basis for subsequent analyses, was the first step on which the software development team focused in the initial weeks of work. Subsequent meetings facilitated the conceptual elaboration of the ideas needed to create the first agents, which were charged with operating on the available data structures. However, one of the first issues that emerged concerned which agent could extract data from the machine and how it could operate within the scheduling architecture, which SPEA had previously developed with the company's research and development team.As illustrated in the Figure 4.2, the agent scheduler devised by the SPEA team is conceptually a priority queue, in which agents are scheduled according to priority, determined primarily by synchronization times. This means that an agent with FAST synchronization can be scheduled at the top of the queue, overtaking one with SLOW synchronization.
The two levels of synchronization are fundamental as they allow the tasks of the agents to be divided according to their functions and the type of data they have to process. For example, for real-time statistics, an agent with FAST synchronization is required, whereas for daily statistics, which are not urgent for the system in the short term, it is preferable to use an agent with SLOW synchronization. In this way, tasks are organized according to time priority and urgency.

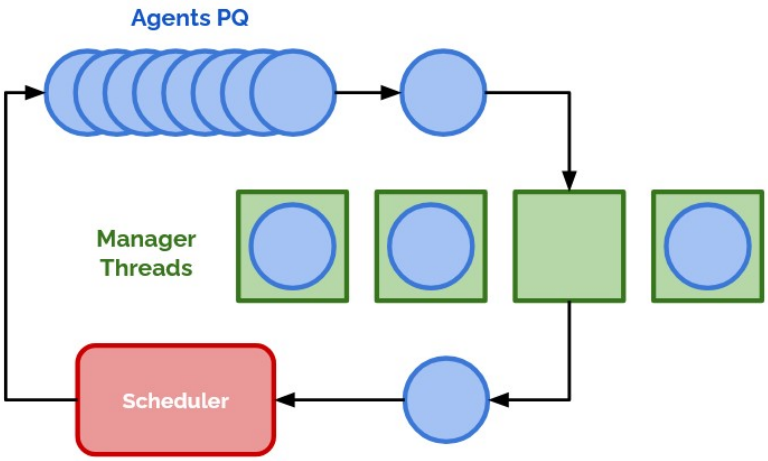

Figure 4.2. Multi Agent System Scheduler

With regard to the agent responsible for extracting the raw data from the machines, the first step in the next steps, ideas were developed from the first meetings involving the core team of the thesis and other figures within SPEA with specific expertise in database and software scheduling.
Analyzing the scheme of the current scheduler, a number of relevant hypotheses emerged:

1. Use an agent with the highest priority within the scheduler, continuously running on the SPEA machines to collect data without interruption.

2. Use an agent outside the agent system, working alongside the internal agents in the scheduler to facilitate data sharing. This external agent, independent of the scheduler's structure, allows the system to continue operating seamlessly even if this agent encounters issues, ensuring continuity with previously acquired data.

The second idea received strong support from the team, as it enables the creation of a dedicated agent, referred to as Agent 001, for each machine in SPEA. This agent can extract raw data and make it available for processing, offering a clear advantage over an architecture in which the extraction agent is part of the main agent system. In the latter case, a delay in data extraction could result in the extraction agent being blacklisted by the scheduler, making it permanently inaccessible. This risk

can arise from various issues, such as IP/TCP connection problems or delays due to computations on complex data, like Measures, which are extensive. Agent 001, as designed by the team, now aims to extract data, put it into a local machine DB and then this data can be used by agents within the agent system.

4.2.2 Definition of agents and their respective roles

The definition of the agents followed the study of the structure and the design of Agent 001, which provides baseline data to other agents, enabling them to process it and perform useful operations, particularly for machine monitoring. The team gave me the opportunity to initially familiarize myself with the data made available in the previous phases, which were first processed by Agent 001, and to develop agents aimed at managing statistics and gaining further familiarity with the multi-agent system. This allowed me to examine reference collections such as Events, Measures, ProductionData, and Equipment, and to develop some ideas through specific agents, including:

- **Agent Classify Events**: an agent capable of classifying the machine responses over time using fixed thresholds, allowing us to determine whether an event is more or less critical for the machine or if it should simply be analyzed at a later stage.
- **Agent Daily Failures**: an agent that identifies the critical states experienced during a machine's operational day, providing timestamps and details on when these critical states occurred.
- **Agent Lifetime**: an agent that indicates how long the machine was operational within a single day.
- **Agent Percentage of Events**: an agent that gathers the total percentage of all events recorded across all operational days of the machines.
- **Agent Status Percentage Day**: an agent that performs the same functions as the previous agent, but represents the data as daily percentages rather than total percentages for each day.
- **Agent Time in a Status**: an agent that provides the daily time spent by the machine in each of its various states.

These agents represent the first developed ideas, focusing on machine states related to the equipment status in the collections processed by Agent 001. The preliminary concepts implemented with these agents were analyzed by the development team at SPEA and received positive feedback, as a main approach was identified for real-time data analysis: machine state analysis. This analysis is essential to understand which states recur throughout the working day on the machines, which are critical, which need monitoring, and which should be preserved. Consequently, we moved on to defining the actual operational agents within the multi-agent system, aiming to manage tasks more accurately and distribute them across the Machine and Plant levels present in the core Archimede architecture.

Machine Agent was the first step in developing agents within the agent system itself. It operates at the machine level and is present in each machine to develop computations within the machine itself. The initial idea was to have an agent at a lower level that could simply provide data to the higher Plant level. The idea developed is based on the fact that the Machine Agent receives data from the Measures, Events, Equipment, ProductionData collections. The data is sorted temporally to have a reference base. After sorting, the agent calculates the difference between consecutive timestamps to measure the time intervals between events. For each timestamp, the Equipment-status from the ProductionData data is associated or, in case of lack, the Equipment-status temporally closest in the past is used. On the calculated temporal differences, the agent performs an average and a standard deviation (Mean & Sigma). The average can be calculated as a moving average on the last ten elements for a constant update. The agent establishes whether the machine is connected or not. The machine is disconnected (or OFF) if the interval between the most recent timestamp and the current time exceeds a predefined threshold, given by the sum of the mean and 10 times the standard deviation (taken experimentally by observing the data and the distribution later in the non-ideative but applicative phases). Using a custom configuration file, the agent associates a color status (Color Status) to each event to facilitate its visual representation in the app. Finally, the agent generates and returns everything in a final Status collection, composed of the fields Timestamp, Equipment-status, Machine-status, Color Status, Mean, and Sigma, providing an updated and synthetic picture of the

machine's status.

The second step in the development of the agents actually operating within the agent system is given by the **Statistical Agent**, which is the heart of the Plant-level processing. It was designed as an agent that acts both at the P (Plant) level and at the M (Machine) level. The agent in the original idea takes data from the Status collection, which collects relevant information about the status of the machines. Using the timestamp present in the data, the agent can calculate the difference between the current time and the last timestamp recorded for each machine present at the P level, thus verifying how much time has passed since the last response provided by the machine. Subsequently, based on this time difference, the agent calculates the ghosting percentage of the color associated with the machine, that is, how much the color is changing or lightening over time. Ghosting refers to the process in which the color associated with the machine tends to gradually lighten. To obtain this percentage, the agent uses two thresholds defined as *t1 = mean + sigma* and *t2 = mean + 10*sigma*. If the difference between the current time and the last timestamp recorded by the machine exceeds the threshold *t1*, and falls between *t1* and *t2*, the color of the machine begins to gradually lighten. If, however, the difference exceeds the threshold *t2*, the color lightens completely, reaching a shade of gray. The choice of *t1* and *t2* was very important, particularly because it is based on the analysis of the temporal averages of the past response times of the machines. Even if a machine, in the case of *t1*, responds with a time interval slightly higher than the average response times, it is considered a small variation due to the standard deviation, which creates a kind of acceptable tolerance range to indicate that the machine has acceptable response times. For the decision of the *t2* threshold, an acceptable time interval was used that could deviate from *t1* for a certain amount of time, representing the ghosting of the machine's color and indicating the machine's inactivity in the short time.

Agent 002 is another agent, separate from the core agent system, alongside Agent 001. It performs conceptual functions similar to those of Agent 001. However, it was developed later, as it became apparent that an external agent was needed that would be less computationally demanding than Agent 001, specifically to track the

"pulses" of the machines. This agent collects data and inserts it into the HeartBeat collection, also located in ArchimedeLocal, specific to each machine. Agent 002 only provides Timestamp data, which represents the communication between the machine and the reference software that extracts data from the machines in proprietary string format, known as RunPack (the same software used in the same way by Agent 001).

4.2.3 Communication and coordination model among agents

The following paragraph focuses on the design phases of the communication schemas between the agents previously described. From a design perspective, it was important to determine which architecture would best facilitate direct and effective communication between the agents involved. A model similar to that of blackboards, conceptually discussed in Chapter 3, was considered. In this model, a synchronization system between agents is present, allowing data to flow from the machine level to the plant level, and, most importantly, enabling the aggregation of data from all machines into a simple final collection at the upper level.

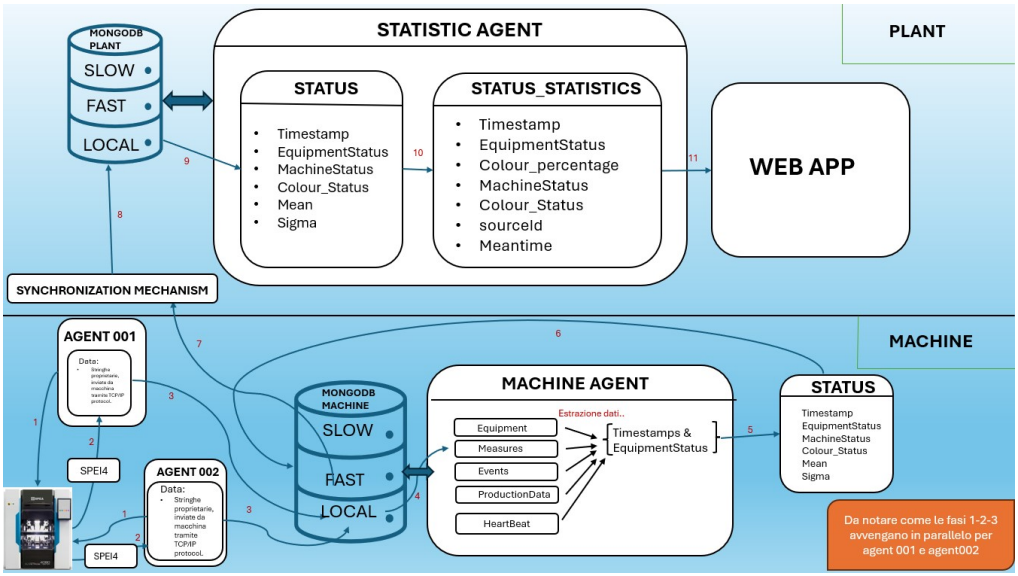

Figure 4.3. Conceptual Schema of Agents System

Figure 4.2 above shows the conceptual and technical diagram that was progressively developed during the project phases and completed in the final stages of the

project described in the thesis. It illustrates the flow of data, starting from the machines and being sent to the Machine Agent at the lower level. From there, the data is synchronized through a specific synchronization system at the plant level. This system enables fast synchronization with the local database at the plant level. Finally, the processing carried out by the statistical agent produces data that is sent to the web app/graphical interface, which serves as the final point of connection for the entire project.

4.3 Predictive and Monitoring Features in the Web Application

4.3.1 Overview of the fundamental idea behind the web app's design and purpose

The final design phase involved the creation of a Graphic User Interface in the form of a Web App to be used on SPEA machinery for real-time monitoring of machine states, as highlighted in previous sections with the agent-based system. In this context, we will not delve into the technical details of the web app, which will be addressed in the next chapter along with the implementation specifics of the Multi-Agent System. Instead, we will focus on the design phases that laid the foundation for the subsequent development of the web app. The initial development phases included the selection of the technologies used for the web app, followed by the conceptualization of the user interface to be represented. Particularly, feedback from SPEA highlighted the necessity of an interface that, starting from the Plant level, could represent the entire Volpiano plant and subsequently provide, for each individual machine within the Plant, all the details resulting from real-time machine state analysis.
The meetings held thereafter clarified two main objectives:

- To create a homepage capable of representing the Volpiano plant.
- To develop a secondary page accessible from the homepage, capable of displaying and detailing the analysis results for each individual machine.

4.3.2 Visual representation of the conceptual architecture and key components of the web application

In terms of graphical representation, the development phase of the Web App led to the formation of ideas that gradually took shape as manual graphical sketches, highlighting the key components of the Web App itself. The initial concepts were all sketched informally on paper, with the aim of visually capturing the fundamental design of the pages that would later make up the graphical interface.

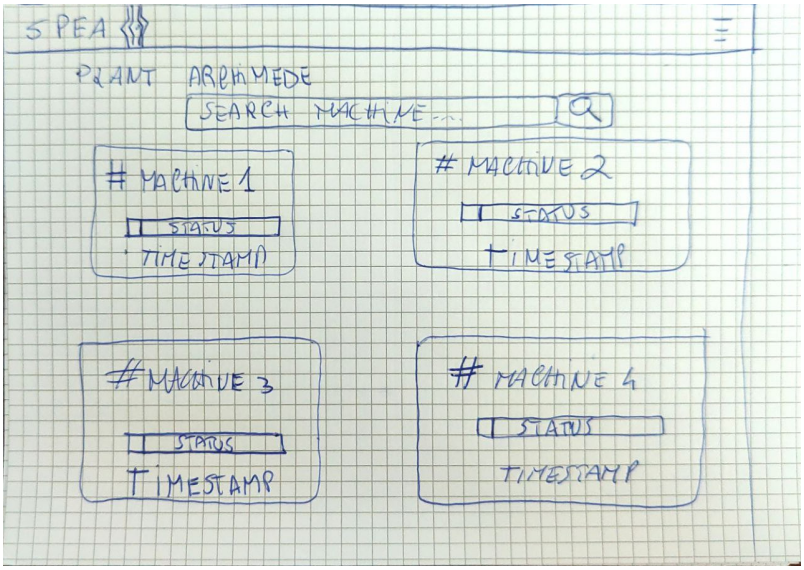

Figure 4.4. Homepage Paper Schema

The Figure 4.3 shows the sketch on paper of the webpage related to the Homepage. This sketch represents a very rough draft, which, in the project phases with SPEA, was used as a fundamental starting point to identify the main features for the web app that would later be developed. This way, the key points of the Homepage were identified as follows:

- Specific name of each machine
- Search bar to filter machines by name
- Status bar with associated color
- Timestamp showing the last defined signal from each machine

Everything is represented within a very simple frame that includes a small navigation bar with the SPEA logo on the left and a drop-down menu icon on the right. This menu, when opened, allows the user to reload the homepage if necessary.

In a second phase, the paper-based design for the page specific to each individual machine was defined.

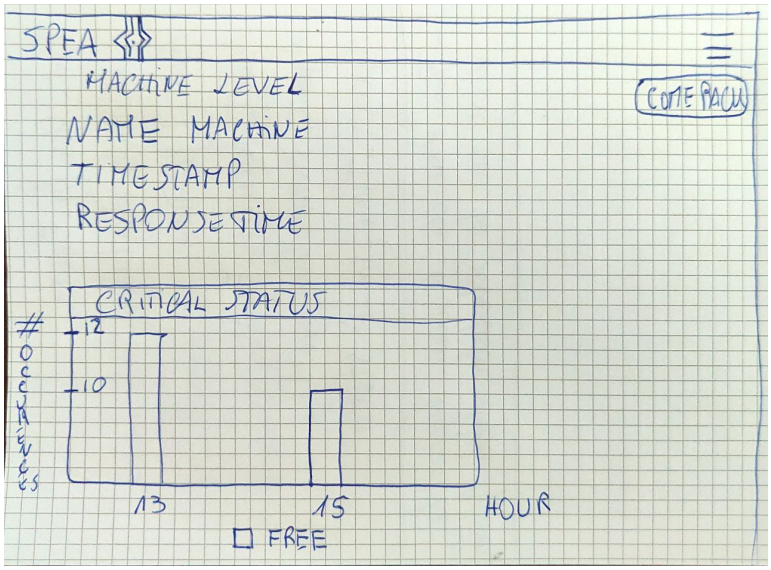

Figure 4.5. Machine-page Paper Schema

In this page, shown in Figure 4.4 and accessible only when a user clicks on an individual machine on the Home page, the user can view specific data related to that single machine. Additionally, a chart displays the data regarding the sequence of critical states occurring throughout the day. This chart is a simple histogram where the x-axis represents the hours of the day, and the y-axis indicates the number of occurrences of critical states within the same day. Key points identifiable on this page include:

- Name of Machine
- Response Time of the current Machine

- Button on the right place to came back home if necessary
- Timestamp showing the last defined signal from each machine

Having identified the foundational elements behind the agent-based system and the web application, we can now delve into the implementation details. The following chapter will focus on the specific tools, technologies, and programming languages employed to bring the thesis project to fruition. This next section will provide an in-depth look at the practical steps taken to turn the conceptual models into a functional system, covering the software choices, technical solutions, and the integration process essential for realizing the project's objectives.

Chapter 5

Implementation of the Multi-Agent System & Web App

5.1 Predictive Modules Integration

In this section of Chapter 5, we present the technical and organizational aspects of the multi-agent system, detailing the technologies and algorithmic steps considered for the implementation of the various agents. A clear description is provided of the implementation process for all agents currently included in the system developed during this thesis work.

5.1.1 Agent 001 integration

As specified in the previous chapter, Agent 001 constitutes a cornerstone in the development of the multi-agent system underlying Archimedes in SPEA. In particular, through the activity of agent 001, it is possible to extract data from the machines that enable the use of predictive information that can then be analyzed and visualized by the agents and the web app. From a technical point of view, although for reasons of privacy and internal SPEA policies it was not possible to work directly on agent 001, it is possible to represent certain characteristics that make it unique and fundamental within the developed architecture. In particular, Agent 001 exploits the analysis carried out by SPEA's RunPack compartment, which is a software that, through C# programming, is able to extract information as log files in proprietary strings. Figure 5.1 shows in a schematic but at the same time precise

way, the steps that lead to the transcription of data by Agent 001 on the MongoDB database used by the individual machines. In the first step on the left in blue, the use by Agent 001 of a SPEAi40CommunicationLibrary is underlined, which is a library that serves to transcribe data in a proprietary string format and to subsequently carry them via TCP/IP protocol to port 50000 of a local server. The second phase is the one that leads to the management of the data contained in the local server 50000 and then transcribe the same into MongoDB collections capable of representing in a readable and well-structured way, everything that is processed by the machine, on which Agent 001 operates. At this stage, the server also switches to local port 50000 to write the data into the MongoDB collections on local-host port 27017. The process is the same for each machine in the SPEA Plant and therefore different connections are created to the local-host 27017 servers of each individual machine, which in turn allow the communication of the data present in the individual machines. The reference collections created by Agent 001 are Measures, Events,

Figure 5.1. Agent 001 Communication Schema

Equipment, and ProductionData. Among these collections, only specific data will be used, with the most important being the *Timestamp* field from each collection and the *EquipmentStatus* field from the ProductionData collection, which play a fundamental role in the subsequent computations developed by the higher-level agents.

5.1.2 Agent 002 deployment

Agent 002 serves as the 'backup' agent, integrated into the external architecture that communicates with the scheduler of the internal agents within the multi-agent system to address computational shortcomings of Agent 001. Specifically, this agent is capable of extracting a very large volume of data, which can result in highly demanding record processing. While this is manageable in MongoDB, it

still presents significant challenges. This led to the creation of Agent 002, tasked with integrating a new collection into the MongoDB database mentioned earlier. It operates in parallel, in a manner similar to Agent 001. The integrated collection is named 'HeartBeat,' and it manages the Timestamp field, which indicates communication between the machine and the proprietary RunPack software responsible for extracting any data processed by the machine over time. For this Agent, an implementation in C# code has been introduced, which is equally viewable as follows and it shows the implementation of Agent 002, which sends "Heart-Beat" signals at regular intervals, configured through a value read from a configuration file. It uses a timer that periodically executes an action, logging the interval and the service status. At each interval, the agent creates a BSON document containing the current timestamp, which is inserted into a MongoDB database, the same one locally used by Agent 001, to track communications. In this way, the system ensures continuous and automated monitoring of the system's heartbeat. Finally, when the service is stopped, the timer halts, and the operation is logged

```
private readonly Logger _logger = LogManager.GetCurrentClassLogger();
private MongoClient _client;
private Timer _timer;

public Agent002()
{
  int.TryParse(ConfigurationManager.AppSettings["HeartBeatPeriod"],
  out var period);
  period = Math.Max(period, MIN_PERIOD);

  _logger.Info($"Heart-Beat Period is {period} ms");
  _logger.Info($"Period: {period} ms");

  var timer = new Timer { Interval = period };
  timer.Elapsed += new ElapsedEventHandler(OnTimer);
  timer.Start();
  _timer = new Timer { Interval = period, AutoReset = true };
  _timer.Elapsed += new ElapsedEventHandler(OnTimer);
```

```
      _timer.Start();

      _logger.Info("Agent002 Heart-Beat Service Started");
    }

      var document = new BsonDocument
      {
        { "Timestamp", DateTime.UtcNow }
        { "Timestamp", $"{DateTime.Now:yyyy-MM-ddTHH:mm:ss.fffffffK}" }
      };
      _logger.Trace($"Inserting {document} . . .");

    protected override void OnStop()
    {
      _logger.Info("Agent002 Heart-Beat Service Stpping");
      _timer.Stop();
    }
  }
}
```

5.1.3 Machine Agent deployment

As outlined in Chapter 4, where the implementation ideas were introduced theoretically, the agents in the multi-agent system operate at a different level compared to the two previous agents, which acted as autonomous entities independent of the computations managed by the scheduler's priority queue. In this context, the focus is on the machine-level agent, which operates individually for each machine and is scheduled with FAST and immediate priority by the system's scheduler. From this section onward, the reference language is C++, chosen for designing the scheduler's architecture and the agents. This language ensures flexibility, speed in computations, and process management, while also supporting specific and efficient libraries for MongoDB queries, which are crucial for the agents' actions. The Machine Agent, as well as the other previously developed agents, calls the constructor of a class named Agent, passing the parameters id (the agent's identifier)

and schedule (which indicates the agent's scheduling or priority). It initializes a member called cluster using the value of the string uri (to connect to a database or external system localhost). Additionally, the uri class member is initialized with the value passed to the constructor.
The *run* function is the core of the Machine Agent implementation. It begins by establishing a connection to the MongoDB cluster and creating a client connection using the specified URI. Next, several variables are defined referring to the *SpeaArchimedeData* database, which is an alias for *ArchimedeLocal*. These variables specifically reference two collections: *ProductionData* and *HeartBeat*. The processed data will then be stored in *ArchimedeFast*, which contains the *Status* collection where the results generated by the Machine Agent are inserted.

The entire process starts with extracting data from two MongoDB collections: *HeartBeat* and *ProductionData*. For the *HeartBeat* collection, a function retrieves the 10 most recent records, sorting them by timestamp. Each timestamp is cleaned of unnecessary details, and the records are initially marked with an UNDEFINED state. Similarly, for the ProductionData collection, another function collects both the records and their associated EquipmentStatus, also standardizing the timestamps.
The data from both collections is then combined into a single list called *all-timestamp-records*, which is sorted chronologically to ensure records are processed in the correct order. The processing loop goes through the ordered records, and during this step, records with an UNDEFINED state are replaced with the previous non-UNDEFINED state, if available.
If no valid state is found in previous records, the status is set to OFF. Next, the time differences between consecutive timestamps are calculated. The mean and standard deviation of these time differences are computed using a moving window of 10 records. If there are fewer than 10 records, the calculations are made using all available records; otherwise, only the last 10 records are considered. These calculations help determine whether the machine is operating normally or if there are delays in the signals. Additionally, based on the time difference between the last record and the current time, the *MachineStatus* is determined: if the delay exceeds a defined threshold, the machine is considered *Non Operative*; if the delay is shorter, it is classified as *Signal Delay* or *Critical Signal Delay*. The code then checks the

latest record in the Status collection and compares it with the new record. If the last timestamp is earlier than the new one, a new document is created with the updated data, including Timestamp, EquipmentStatus, the new *MachineStatus*, the color associated with the machine's status (determined through a function that reads an XML file produced by SPEA operators), and the previously calculated time statistics. This document is then inserted into the *Status* collection (an example is showed in the Figure 5.2 below). Finally, after the data is inserted, the connection to the MongoDB cluster is closed, completing the process.

Figure 5.2. Status collection Representation

5.1.4 Statistical Agent deployment

The Statistical Agent serves as the link between all representations and implementations of the Multi-Agent System developed during this thesis work. It is, in fact, the Agent that, operating at the Plant level, collects data from the machines synchronized across the entire Plant and performs the final computations before providing a collection named *StatusStatistics*. This collection is used by the Web

App to display the computations of the agents and the Multi-Agent System itself. In terms of development within the agent system architecture, the Statistical Agent is similar in definition to the Machine Agent. In fact, it has a constructor equal and identical to the Machine Agent and it also has a run method from which to then develop the computations necessary for the results. Proceeding step by step, the first step is the initialization of a MongoDB client through the *mongocxx::client* object, which establishes the connection to the database using the URI passed as a parameter (uri). Once the connection is established, the program accesses the specified database (*SpeaArchimedeData*) and the *Status* collection, from which a cursor is extracted that allows iterating over all the documents contained in the collection:

```
mongocxx::client client{ mongocxx::uri{uri.c_str()} };
auto db = client["SpeaArchimedeData"];
auto collection = db["Status"];
auto cursor = collection.find({});
```

Next, a map called *total-timestamps-map* is created, which associates each sourceId with a vector of tuples containing information about the various timestamps recorded for each sourceId. The key of the map is a *std::tuple<std::string>*, which represents the sourceId, while the values are vectors of tuples containing the following data:

- timestamp
- equipmentstatus
- mean
- sigma
- colourstatus
- machinestatus

The goal of this map is to organize the data so that it can be analyzed later, grouping the records by sourceId. For each document extracted by the cursor,

the code extracts the values of the sourceId, Timestamp, EquipmentStatus, Mean, Sigma, ColourStatus, and MachineStatus fields. If the sourceId field is of type document, the associated value is extracted, otherwise the default value is "SPEA-MACHINE". The timestamp is then converted from string to type *std::tm*:

```
std::tm tm = {};
std::istringstream ss(timestamp);
ss >> std::get_time(&tm, "%Y-%m-%dT%H:%M:%S");
```

After successfully processing the timestamp, it is converted to a *std::chrono::system-clock::time-point*, which allows for more precise time operations. The record is then inserted into the *total-timestamps-map*, which collects all data related to that sourceId:

```
auto doc_time = std::chrono::system_clock::from_time_t(std::mktime(&tm));
total_timestamps_map[std::make_tuple(sourceId)]
.push_back(std::tuple(doc_time, equipment_status, mean,
sigma, colour_status, machinestatus));
```

After collecting all the data in the map, the program proceeds to process each sourceId. For each sourceId, the associated data is extracted and sorted in ascending order of timestamp:

```
std::sort(sorted_timestamps.begin(), sorted_timestamps.end(),
    [](const auto& a, const auto& b) {
        return std::get<0>(a) < std::get<0>(b);
    });
```

The mean, sigma, and machinestatus value for the first record are also extracted. Once the records are sorted by each sourceId, the code examines the last record, i.e. the one with the most recent timestamp, and calculates the interval time from the current moment:

```
interval = std::chrono::duration_cast<std::chrono::seconds>
(std::chrono::system_clock::now() - std::get<0>(sorted_timestamps[i])).count();
```

This interval is used to determine whether the machine is operational, has a signal delay, or is out of service. Threshold limits are used to define these states:

- If the range is greater than *mean + 10 * sigma*, the machine is considered **Non-Operational**.
- If the range is between *mean + sigma* and *mean + 10 * sigma*, the status is updated to **Critical Signal Delay**.
- If the range is between *mean + 0.75 * sigma* and *mean + sigma*, the machine is classified as **Continuing Signal Delay by Machine**.
- If the range is between *mean* and *mean + 0.75 * sigma*, the status changes to **Signal Delay**.
- If the range is less than *mean*, the machine is **Operational**

The values of sigma, mean, and interval are then used to calculate the *colourPercentage* parameter through a specific function:

```
    std::string calculateShadingPercentage(double diff,
    double mean, double sigma) {
    auto first_threshold = mean + sigma;
    auto second_threshold = mean + 10 * sigma;
    if (diff >= second_threshold) {
        return "100.0%";
    }
    if (diff <= first_threshold) {
        return "0.0%";
    }
    double normalized_diff = (diff - first_threshold) / second_threshold;
    if (normalized_diff < 0) normalized_diff = 0;
    if (normalized_diff > 1) normalized_diff = 1;

    double shading_percentage = normalized_diff * 100.0;
    return std::to_string(shading_percentage) + "%";
}
```

The function above calculates the percentage of shading based on the time difference (diff), using a normalized range between two thresholds (mean + sigma and mean + 10 * sigma). It ensures the result is capped within the interval [0%, 100%], corresponding to the transition from no shading to complete shading. Finally the results are put on the state vector which provides the vector that is inserted into the MongoDB StatusStatistics collection whose parameters are now:

- **sourceId**: reference string regarding the source of the data
- **timestampStr**: string representing the time instant at which the machine communicates with the agent system
- **machineStatus**: string representing the life state of the machine based on the calculations seen above
- **colourPercentage**: string representing the percentage of colour brightening according to time instants and the function seen above.
- **equipmentStatus**: string representing the state of the machine at the time of communication, identifies the nature of the signal produced
- **colourStatus**: string representing the status associated with the colour, i.e. it is used to identify later in the web app which colour is represented by the equipment status
- **interval**: double number representing the time difference between the last current instant and the last instant at which the machine sent a signal

With the final construction of the **StatusStatitics** collection, the work of the designed Multi Agent System ends and now the next phase, equally important, is that of building a GUI capable of showing in a simple but effective way the results produced by the agent system with regards to the real time monitoring of SPEA machinery.

5.1.5 Optimizations inside Multi Agent System

One of the aspects that was considered during the tests carried out to verify the functioning of the agent system was first of all to have a functioning system and

able to give valid results, and then another aspect of no small importance was certainly the management of the speed of the agents used within the agent system. In fact, although the code in its writing was not touched, it was sometimes necessary to modify the position of some lines of code to allow an agent to proceed with its computations more quickly. The distributed programming paradigm connected to the Multi Agent System was also used to divide the tasks between the agents so that none of these agents were overloaded, in addition of course to the analysis of the Databases and collections made available for the thesis project.
In particular, the agent that was initially overloaded the most was the Machine Agent, which in itself develops a large number of computations that make it expensive from an implementation point of view. As shown in the graph in figure 5.3, the

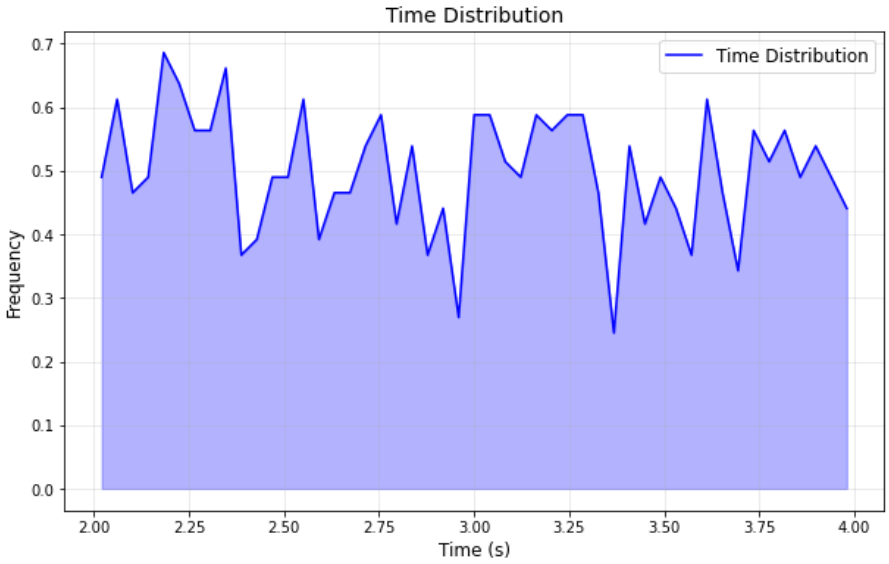

Figure 5.3. Time Distribution from Machine Agent

first trend of the computations relating to the Machine agent immediately showed a trend in times that oscillated between 2 and 4 seconds in the computations carried out. In this case, the computations initially used all the values of the Measures collection and the workload on the machine agent also included the reproduction of the percentage of color fading, which in the final version is instead the responsibility of the statistical agent at plant level. In particular, however, the amount of data processed by the Measures collection put the system in crisis, practically blocking the machine agent for a period of time too long to represent the real-time

monitoring of the machines. It is at this stage of the project that we thought of reducing the workload of the Machine Agent through some simple mechanisms:

- moving the management of the percentage of color fading in the statistical agent at plant level
- loading only the last 10 records of the Measures collection
- introducing the HeartBeat collection by inserting the last 100 data produced, the most recent in chronological order

In this way, it was possible to notice, as shown in figure 5.4, a great improvement in the performances related to the Machine Agent.In particular, the computational

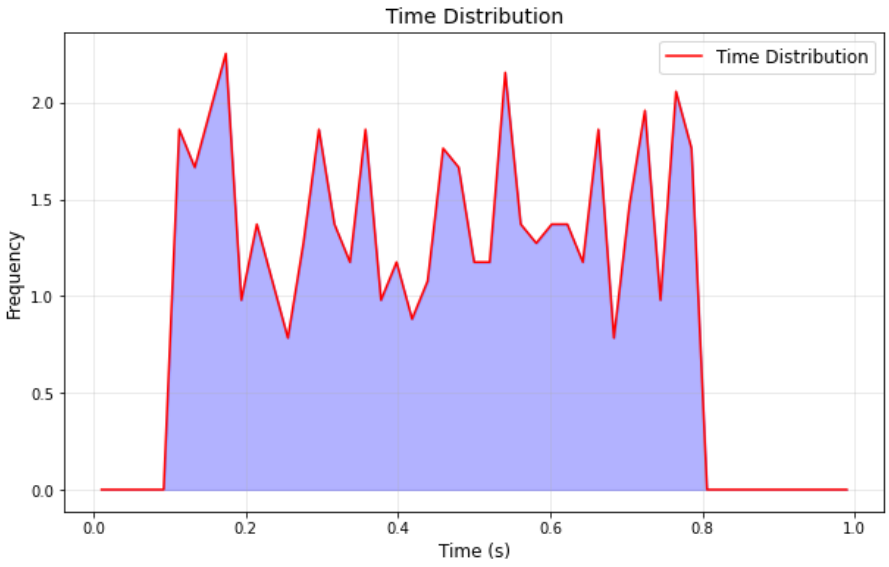

Figure 5.4. Optimized Time Distribution from Machine Agent

time went from an average of 3 seconds previously recorded to an average of 0.4 seconds, significantly improving system performance.

5.2 Development of the Front-end for Data Visualization

5.2.1 Choice of frameworks and reference languages

The development of a graphical user interface capable of being dynamic and representative of the computations performed by the agent system led the entire team to search for a programming strategy capable of creating a Web App capable of acting as a Graphic User Interface useful for monitoring the machine state in real time. At first, a technology was proposed by SPEA based on the Dart language, a programming language developed and presented by Google in 2011 and used for Web programming instead of the Javascript language. Dart can be used in conjunction with a framework by the name of Flutter, which came into being in 2014 and enables the development of applications for iOS, Android, Linux, macOS and Windows. However, this was problematic because the Dart application development team in SPEA could not actively participate in the development phases of the entire thesis project, and so after a few clarifying meetings, a different, yet effective solution was arrived at. In fact, the past academic experiences of some of the members of the working group, both on the SPEA side and on the polytechnic side (myself), were utilized to converge towards a solution based on JavaScript and one of the most popular JavaScript language frameworks in the field of Web programming of recent times, namely ReactJS.

JavaScript & React choice

JavaScript [59] is a programming language born in 1995 from the idea of Brendan Eich. It is an event-oriented programming language and is used for both client-side and server-side web programming with the appropriate use of Node.js (we will see this section later in the paragraph dedicated to it). This language is used for the development of web applications that must have a certain dynamism and interactivity, due to the events triggered by a user who carries out actions on the web page itself [59]. JavaScript has features that make it an excellent solution in the case of web programming because from the client's point of view it allows the execution of the code directly on the client and not on the server, with the consequent advantage that the web server is never overloaded precisely because all the work

is performed on the client side. Problems can arise when you have to access data stored in a database because there is a need for an additional language, capable of processing the data and putting it into real JavaScript variables [59]. The analysis of these language dynamics led us to consider the fact that in the case of the thesis project in question the WEB pages to be developed for real-time monitoring were not excessively onerous from the point of view of the data to be considered since the agents proposed in the multi agent system were designed with the intent of limiting the number of data to be transmitted to the application itself. Then a further motivation that led us to consider the use of JavaScript is that precisely thanks to the possibility of developing an app independent from any server-side data and above all split in speed from the data from any server loads allowed us to do preventive tests with only the front-end and then subsequently consider client and server together at a later time.
Having made these premises on JavaScript and some of the reasons that led us to use it for the implementation of the Web App, we cannot help but mention the key motivation for using JavaScript in Web programming, namely its library, famous in the world of Web programmers, namely ReactJS.

Figure 5.5. ReactJS Logo Source: [60]

ReactJS [60], represented in Figure 5.3 with his logo, is one of the most widely used libraries for front-end web application development, alongside other notable tools such as Angular, Vue, Svelte, and other frameworks. React stands out for its growing popularity in the front-end development industry due to its speed and flexibility during the development process. One of its key features is the ability to integrate and test new components in real-time: developers can simply refresh the project's web page to immediately verify any changes, thereby optimizing the

development cycle [60]. This capability proved particularly valuable in the context of this thesis work. During the project, new requests or changes in requirements were easily accommodated by quickly adding front-end components or updating existing ones, improving their design or functionality without disrupting the workflow. Another strategic advantage of ReactJS, which made it an ideal choice for this project, is the ability to reuse the same code base for applications targeting both Android and iOS devices [60]. This feature resulted in significant time and resource savings, which was crucial for meeting company deadlines. Moreover, React is a constantly evolving library, with regular updates introducing new features, ensuring a robust platform suitable for long-term use. Unsurprisingly, leading companies such as Facebook adopt React for creating stable and efficient interfaces, both for web and mobile applications. From a technical perspective, React is built on the Virtual DOM, a virtual model developed by Facebook to optimize the user experience [60]. This system allows for seamless modifications to the graphical interface without compromising performance or the user's interaction with the application [60]. Lastly, React is an extremely practical tool for developers, providing a straightforward and effective testing environment that facilitates the rapid optimization of applications through intuitive and precise actions.

Bootstrap

Figure 5.6. Bootstrap Logo Source: [61]

Bootstrap [61] is a set of integrated tools for HTML and CSS, useful for producing templates for WEB sites that include navigation buttons, graphical interfaces, various modules. Bootstrap is used together with React to create WEB pages with pre-set components that allow for easy reproduction of objects within a WEB page [61]. The library was born from the idea of Mark Otto and Jacob Thornton at Twitter with the intention of creating a library that would unify the existing libraries for modules that had problems due to inconsistency in the representation of interface modules and above all, on the company side, a large maintenance cost. Starting in 2011, Bootstrap became open source on the idea of Mark Otto himself, who invited programmers from all over the world to improve its capabilities and develop an increasingly dynamic and advanced model to produce integrated modules for the web [61]. Several versions of Bootstrap have been released, including the latest one was released in 2021 [61].
The choice of Bootstrap was the consequence of all the premises made above, as it is an open source library, easy to manage and dynamic in development. All this allowed during the creation phases of the web app to be able to refer to modules, to icons pre-set in appearance that facilitated the production work on the front-end side. The choice of Bootstrap also saw the favorable opinion of the SPEA company that saw in the library, supported by the use of React and JavaScript, a good development opportunity also for the future.

5.2.2 Implementation of the user interface for the visualization of machine states

In terms of development, the application was created using a master folder, called **SPEA APPLICATION**, to which two basic sub folders were added:

- spea-app
- spea-app-backend

To create the front-end part, the **spea-app** folder was used, inside which modules and libraries useful for development were installed, thanks to the use of NodeJS commands (description in the next paragraph) that allowed us to start the app and verify its validity.
The first step towards creating the front-end was installing the **node modules**

folder. This folder allows you to install all the dependencies needed to create the project and to load the script on which the project will later run. Starting from this first step, we can immediately notice the creation of two basic files in JSON format:

- **package.json**
- **package-lock.json**

The first contains crucial information such as the name of the project, the version associated with it, the author and the script used. It also allows you to know the required project dependencies and development dependencies, making sure that everyone can know which package is being referred to for the development of the project itself.
The second instead serves to block the exact versions of the installed dependencies, ensuring that the developers and environments involved have the same version of the project and then also improves the stability of the application.

The next part of the front-end project consists of creating the various components that are the basis of the entire project. To do this, each fundamental component will now be described to highlight the significant implementation aspects that characterize it.

Home Component

The Home component was developed using a JavaScript file and a related CSS file that take care of the implementation and stylistic aspects of the components and how they are placed within the startup screen of the monitoring App. Inside the *Home.js* file, the first step is to define the application configuration to use state and navigation management tools, predefined style components, loading animations, API functionality and custom styles, as shown in the following code:

```
import { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { Circles } from 'react-loader-spinner';
import { Form, Container } from 'react-bootstrap';
import * as api from "../api/api";
```

```
import 'bootstrap-icons/font/bootstrap-icons.css';
import './Home.css';
```

The imports represented are fundamental as they allow the definition of the actions to be executed dynamically in the application code and they have important characteristics. **useState** and **useEffect** are two React hooks that allow the first to create reactive state variables while the second allows you to execute side effects such as APIs and DOM updates following responses to changes in the state or properties of a component. **useNavigate** is a hook used to manage navigation within the application in React and this allows you to move between pages dynamically and without the need to reload the entire application. The other important components are Circles, which allows you to display a circular page loading component when you have to load a lot of data from the reference database, the Form and Container components, which are predefined components of react bootstrap and simplify the creation of responsive layouts and interfaces. The functions from the *api.js* file are also imported which are used to manage subsequent API requests to the backend and the CSS file of the bootstrap icons, which is a library of vector icons to be used to add visual symbols such as arrows, buttons for example. Finally, the *Home.css* file is also imported, which contains the custom style rules for that part of the application.

The implementation follows with the construction of the **loadMachines** module:

```
const loadMachines = async () => {
  try {
    setLoading(false);
    setHasLoadedOnce(true);
    const data = await api.getStatistics();
    const groupedMachines = data.reduce((acc, machine) => {
      if (!acc[machine.sourceId]) {
        acc[machine.sourceId] = [];
      }
      acc[machine.sourceId].push(machine);
      return acc;
    }, {});
```

```
        setMachines(groupedMachines);
      } catch (error) {
        console.error("Errore nel caricamento delle macchine:", error);
        setLoading(false);
      }
};
```

The loadMachines function is an asynchronous call that loads data about machines from an API contained in the api.js file; then the data is grouped by their sourceId identifier and the application state is updated. Before making the call, it is set that the loading is finished and that the data has been loaded at least once so that the data obtained is organized in an object where each key represents a sourceId and the value is an array of associated machines. Finally, the state is updated with the grouped data and, in case of error, it is logged in the console and the loading indicator is deactivated.

The loadMachines function is called inside the Home component, the real cornerstone of the implementation of the Homepage of the Monitoring App at Plant level. The component just mentioned displays a list of machines loaded from an API that are filtered through a search bar and it allows you to navigate to a detailed page for each machine that is selected after a click by the operator. The Home component is mainly based on states; the first state is machines, which is an object that stores the machines grouped by sourceId; then there is the sourceTerm field which is a string that essentially represents the current value of the search bar; then there is the loading component which indicates whether the data is still being loaded or not and finally through the hasLoadedOnce field the cases in which the data has been loaded at least once in the current page are reported. Another aspect to take into account inside the Home component is the useEffect that calls the loadMachines function to start the data and sets an interval of 10 seconds to automatically update the data by calling loadMachines:

```
const Home = () => {
  const [machines, setMachines] = useState({});
  const [searchTerm, setSearchTerm] = useState('');
  const [loading, setLoading] = useState(true);
```

```
const [hasLoadedOnce, setHasLoadedOnce] = useState(false);
const navigate = useNavigate();

const loadMachines = async () => {
  try {
    setLoading(false);
    setHasLoadedOnce(true);
    const data = await api.getStatistics();
    const groupedMachines = data.reduce((acc, machine) => {
      if (!acc[machine.sourceId]) {
        acc[machine.sourceId] = [];
      }
      acc[machine.sourceId].push(machine);
      return acc;
    }, {});
    setMachines(groupedMachines);
  } catch (error) {
    console.error("Errore nel caricamento delle macchine:", error);
    setLoading(false);
  }
};


useEffect(() => {
  loadMachines();
  const backendInterval = setInterval(loadMachines,10000);
  return () => clearInterval(backendInterval);
}, []);

const handleSearchChange = (event) => {
  setSearchTerm(event.target.value);
};

const filteredMachines = Object.keys(machines).filter(sourceId =>
```

```
        sourceId.toLowerCase().includes(searchTerm.toLowerCase())
      );

      const handleRowClick = (sourceId) => {
        navigate(`/machine/${sourceId}`);
      };
      return(...)
  }
```

The highlighted code is a significant part of the component to which a subsequent management of the search bar is added so that the user can update the value every time he types something in the status bar and thus also the machines are filtered based on what is written by the user while the app is being processed. The handleRowClick function uses the navigate hook instead to direct the user to a page dedicated to the selected machine, with the URL that is specific to each given machine based on the sourceId. As for the machines in the return function that is not specified on this case, they are then redirected each with a specific icon along with their name, the bar that represents the status and the recorded timestamp.

Nav Bar Component

The NavBar component is used to build the navigation bar present in all pages of the application and from an implementation point of view it is composed of a NavBar function:

```
const Navbar = () => {
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const history = useNavigate();

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };

  const handleNavigation = (path) => {
    history(path);
    setIsMenuOpen(false);
```

```
  };

  return (
    <nav className="navbar">
      <div className="navbar-symbol">
        <img src={SpeaLogo} alt="Spea Logo" className="navbar-logo"
        onClick={() => handleNavigation('/')}/>
      </div>
      <button className="menu-button" onClick={toggleMenu}>
        <i className="bi bi-list"></i>
      </button>
      <div className={`side-menu ${isMenuOpen ? 'open' : ''}`}>
        <button className="close-button" onClick={toggleMenu}>
        Ã</button>
        <ul>
          <li><a href="#home" onClick={() => handleNavigation('/')}>
          Home</a></li>

        </ul>
      </div>
    </nav>
  );
};

export default Navbar;
```

The function allows you to set a simple but effective search bar that opens only when the user clicks on the appropriate icon and it allows you to return to the application Home by clicking either on the SPEA logo on the left of the bar or by clicking on the drop-down menu that opens and provides the possibility of clicking on Home to return to the company Plant screen.

Machine Component

The Machine component is the second section of the Web Application created for real-time monitoring. Its structure is similar to that of the Home component as we initially have an import of the various React and Bootstrap libraries.

```
import { Container, Paper, Typography, Box,
IconButton, CircularProgress, useMediaQuery, useTheme}
    from '@mui/material';
import { PieChart, Pie, Cell, Tooltip, Legend,
ResponsiveContainer, BarChart, Bar, XAxis, YAxis, CartesianGrid }
    from 'recharts';
import GaugeChart from 'react-gauge-chart';
```

In the code above, some of the most significant imports have been reported in addition to those already seen in the Home component and also proposed here for the changes of state and effect. Among these, those relating to the constitution of the graphs certainly stand out, which in the Machine section are important for representing real-time data regarding the states assumed by the machine in the short term. The main function, then exported to the App.js file, is the machine function. It is composed of various internal modules and particular sections; however, in the following paper only the most significant ones for understanding the code and its general operation will be represented.

```
const fetchData = async () => {
  try {
    setLoading(false);
    const stats = await api.
    getStatisticsSpecificReferenceID(sourceId);
    const len=stats.length;
    setStatistics(stats);
    setMachineDetails(stats[0]);
    aggregateDataByDate(stats, selectedDate);
    aggregateCriticalDataByDate(stats, selectedDate);
    calculateMeantimeRange(stats);
  } catch (error) {
    console.error('Errore nel recuperare i dati:', error);}};
```

The above code defines an asynchronous function fetchData that retrieves and manages data for a specific machine using an API that fetches data for that machine. It starts by setting a loading indicator, then calls the API to get statistics based on a sourceId, and then the retrieved data is stored in local states, including machine details and aggregate statistics for a selected date. Finally, the aggregate critical data and meantime range are also calculated, and any errors during the operation are handled and logged to the console.

```
useEffect(() => {
  fetchData();
  const interval = setInterval(fetchData, 17000);
  return () => clearInterval(interval);
}, [sourceId, selectedDate]);
```

Another key section of code is the next definition, depicted in the code above, of the useEffect to execute the fetchData function when the component is mounted and whenever the sourceId or selectDate changes. An interval is set to retrieve fetchData every 17 seconds, keeping the data updated in real time and when the component is disassembled or dependencies are changed, the interval is cleared to avoid duplication; the component is very useful for periodically updating the data of a specific machine.

We also continue with the definition of a blendWithGray function that mixes a base color with gray based on a percentage (alpha). It converts the base color from hexadecimal to RGB components, calculating the values mixed with gray (RGB: 211, 211, 211) and using a percentage factor, and finally returning the result as an rgb() string. The base color is retrieved from a COLORSSTATUS state map based on the previously defined currentData.colourstatus. Finally, the faded color is calculated using the function with the specified percentage (percentageValue). All this is showed in the code below.

```
const COLORS_STATUS  = {
  Ok: "#008000",          // Verde
  Not: "#FF0000",        // Rosso
  Other: "#0000FF",      // Blu
  Warning: "#FFFF00"     // Giallo
};
  const blendWithGray = (baseColor, alpha) => {
```

```
    const color = baseColor.startsWith('#')
    ? baseColor.slice(1) : baseColor;
    const r = parseInt(color.slice(0, 2), 16);
    const g = parseInt(color.slice(2, 4), 16);
    const b = parseInt(color.slice(4, 6), 16);
    const gray = [211, 211, 211];
    const blendedR = Math.round((1 - alpha / 100)
    * r + (alpha / 100) * gray[0]);
    const blendedG = Math.round((1 - alpha / 100)
    * g + (alpha / 100) * gray[1]);
    const blendedB = Math.round((1 - alpha / 100)
    * b + (alpha / 100) * gray[2]);
    return `rgb(${blendedR}, ${blendedG}, ${blendedB})`;
};
const baseColor = COLORS_STATUS[currentData.colour_status]
|| COLORS_STATUS.Other;
const shadedColor = blendWithGray(baseColor, percentageValue);
```

Now follows the description of the sections represented on the web page and returned by the function. In addition to the definition of a button to return to the Home page if necessary, there is a first section that represents the characteristics of the selected machine including the name and the last recorded timestamp. This is followed by a section in which a graph is represented that represents the behavior of a machine following the response given in real time. A further section has the purpose of illustrating the latest states that occurred during the day and subsequently there is a representation of a histogram graph that describes the states that occurred at a critical level during the day considered and the number of critical states based on the time of day considered.

App Component

The App component takes on the role of the main file of the entire application. In fact, it defines the slicing, structure and routing of the app, which are essential for identifying the execution tree of the entire app. From an implementation point of view, the first part is always dedicated to imports:

```
import './App.css';
```

```
import 'bootstrap/dist/css/bootstrap.min.css';
import { library } from '@fortawesome/fontawesome-svg-core';
import { fab } from '@fortawesome/free-brands-svg-icons';
import { fas } from '@fortawesome/free-solid-svg-icons';
import { far } from '@fortawesome/free-regular-svg-icons';
import NavBar from './components/Navbar';
import { BrowserRouter, Navigate, Outlet, Route, Routes }
from "react-router-dom";
import { Container } from 'react-bootstrap';
import Home from './components/Home';
import './App.css';
import Machine from './components/Machine';
```

As we can see from the code above, style files are included: a custom one, called App.css, to define the specific look of the application, and Bootstrap CSS files, which provide ready-to-use responsive styling. Next, FontAwesome libraries are imported, which allow you to use vector icons within the app, including brand icons (fab), solid icons (fas), and regular icons (far). These libraries are then registered globally within the application, making them available wherever you need them. Moving on to React components, custom components are called such as NavBar, which represents the navigation bar, and two other main components of the app: Home, for the main page, and Machine, for displaying machine details. Finally, React Router is imported, which allows you to manage navigation between different sections of the app without reloading the page. Key elements are included such as BrowserRouter to enable routing and Routes to define the main paths. To complete the interface structure, the Container component of React Bootstrap is imported, useful for organizing and styling the contents in a responsive way.
This is followed by the implementation of the App function which is the heart of this section:

```
  function App() {
return (
  <BrowserRouter>
    <Routes>
      <Route element={
```

```
          <>
            <NavBar className="fixed-navbar"/>
            <Container className="content-container">
              <Outlet/>
            </Container>
          </>}>
          <Route index
            element={<Navigate replace to="/home" />} />
          <Route path="/home"
            element={<Home/>} />
            <Route path="/machine/:sourceId"
            element={<Machine/>} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

export default App;
library.add(fab, fas, far);
```

The structure is also based on Routes which contains the various paths defined by Route. The layout is made up of a navigation bar NavBar which is common between all the pages of the app and a central container Container which hosts the contents of the various sections. A fundamental role is played by the Outlet component which acts as a space dedicated to the child components, allowing the common container to dynamically adapt to the content of the sub-pages. The routes are based on a main navigation towards the home page when the app is initialized and then subsequently the navigations can pass to the various pages dedicated to the elements of each single machine. Finally, thanks to the integration with React Router, the application behaves like a Single Page Application (SPA): it is not necessary to reload the entire page during navigation, ensuring a smooth and interactive experience for the user.

5.3 Implementation of the Back-End Architecture

5.3.1 Configuration and integration of Node.js

The Back-End configuration was the part developed in parallel with the construction of the front-end architecture of the monitoring web app. The entire project, as briefly mentioned in the previous paragraphs, was carried out not only thanks to the React and Bootstrap libraries for the front-end part but also thanks to the use of an additional JavaScript environment that was able to allow the connection between the front-end and the back-end in a safe and easy to manage way, in addition to obviously starting the application for the tests to be carried out in the project phase. **NodeJS** in particular was a fundamental environment for both the implementation of the front-end but above all for the back-end. It is an environment that allows you to run JavaScript outside of a browser, facilitating the management of requests and responses from a server that is created preliminarily. Furthermore, NodeJS allows the use of JavaScript for both the front-end and the back-end and this was important for the development of the Web App in the thesis project. Node can be compared to a back-end brain capable of communicating with the application's front-end. During the project, the use of Node was important as it allowed the development of the front-end part in parallel with the back-end, facilitating and speeding up operations for all those involved in the Web App. Version 18.4.2 LTS was installed for Node in order to support a version that can always be updated with changes in the components present in the application and the deprecation of some functions.

5.3.2 Inside the Back-End implementation

The implementation of the backend takes place in the spea-app-backend folder, as specified in the previous paragraph regarding the implementation of the front-end part of the web app. From an organizational point of view, the backend modules are composed of 2 main files which are:

- **index.js**: file containing the configuration of the libraries to be associated with the Node library in order to create a secure connection between client

and server.

- **StatusStatistics.js**: file where the data schema to be taken from the reference MongoDB collection is reproduced.

Starting from the description of the index file implementation, this file is organized as follows: first there is the import of the dependencies

```
const express = require('express');
const morgan = require("morgan");
const { check, validationResult } = require("express-validator");
const cors = require("cors");
const mongoose = require('mongoose');
const StatusStatistics = require('./StatusStatistics');
const compression = require('compression');
```

As we can see, there is the import of the fundamental libraries that intervene for a reasonable back-end development. Among these, the **Express** library certainly stands out, which allows you to create the routes with which you can then access the data from the database in the file. It also uses the so-called middleware to process requests and responses, simplifies the creation of RESTful APIs for Create, Read, Update and Delete operations that are commonly used in web apps and is highly performing with compression tools (used in this web app) that allow you to improve the management of requests. Then the use of **Cors** also stands out, which is used to manage a browser for requests that come from different domains of the client and server of the web app. Also important is the use and import of the **mongoose** library that allows you to well manage requests that come from MongoDB type databases as in the case in question in the thesis project.

We continue with the implementation of Express and the local server port 3001 that must host the backend data. Following this, the middleware is formally defined, including the compression one, the data format one which is the Json format in particular. We continue with the definition of morgan for debugging purposes and then we move on to the definition of the Cors features, which as anticipated earlier, allow the connection and sending of requests between different domains in the browser in a secure manner. The section then ends with the management of the

connection to MongoDB which is developed through the mongoose library to which IPV4 requests are associated in order to access the MongoDB database contained in the local port 27018. The following code shows what has just been described.

```
const app = express();
const port = 3001;

app.use(compression());
app.use(express.json());
app.use(morgan("dev"));  // Per il debug
const corsOptions = {
    origin: 'http://localhost:3000',
    optionsSuccessStatus: 200,
    credentials: true
};
app.use(cors(corsOptions));

mongoose.connect('mongodb://127.0.0.1:27018/SpeaArchimedeData', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    family: 4  // Forza l'uso di IPv4
})
.then(() => console.log('Connesso a MongoDB!'))
.catch(err => console.error('Errore di connessione a MongoDB:', err));
```

Looking at the code above wee can see how the reference ports are:

- Back-End: http://localhost:3001
- Front-End: http://localhost:3000

These requests are handled by Cors library and when the frontend tries to make a request to the backend, without configuring Cors, the browser blocks the communication. CORS also adds HTTP Headers to the server response to tell the browser that the request from a different origin is allowed. This is followed by the final part where the specific routes are written when the client needs to send a GET request.

```
app.get('/api/status-statistics', async (req, res) => {
try {

  const machines = await StatusStatistics.aggregate([
    { $sort: { timestamp: -1, meantime: -1 } },
    { $group: { _id: "$sourceId",
    latestStatus:{$first:"$$ROOT"}}},
    { $sort: { _id: 1 } }
  ]);
  res.json(machines.map(machine => machine.latestStatus));
} catch (error) {
  console.error("Errore nel recupero delle macchine:", error);
  res.status(500).send("Errore nel recupero delle macchine");
}
});
```

In the first case, represented by the code above, asynchronous requests are sent from the server with the aim of obtaining data from the back-end relating to the part concerning the Home of the web app. The instances are found in the Mongo DB and ordered by timestamp and decreasing mean time; subsequently, there is a grouping by sourceId and the first document of each group is taken. The results are ordered by decreasing sourceId. Finally, a document in Json format is returned that refers to the latest state of each machine. In this implementation, the aggregation function plays a fundamental role, which is optimized for MongoDB and allows you to use only the most recent filtered data, without overloading the system.

```
  app.get('/api/status-statistics/:sourceId',
  async (req, res) => {
try {
    const sourceId = req.params.sourceId;
    let statistics = await StatusStatistics
    .find({ sourceId: sourceId })
        .sort({ timestamp: -1,meantime: -1 })
```

```
            .limit(3000);
        if (!statistics.length) {
            return res.status(404).send("Nessuna statistica trovata
            per il sourceId specificato");
        }
        res.json(statistics);
    } catch (error) {
        console.error("Errore nel recupero dei dati:", error);
        res.status(500).send("Errore nel recupero dei dati");
    }
});
```

The second case, developed in the code at the beginning of the page, is instead represented by the asynchronous request carried out on behalf of a specific machine and this occurs when you access with a click in the front-end the specific page for each machine that is present in the company plant. The operations are similar to those before only that this time the GET used has the specificity of taking the data available for each machine, taking them all but not all, that is, in fact the last 3000 records that allow you to obtain real-time information on the latest states assumed by the machine in the shortest possible time.
Finally the server is activated on port 3001 as reported in the final code.

```
    app.listen(port, () => {
    console.log(‘Server listening at http://localhost:${port}‘);
});
```

Regarding the StatusStatistics file, the code is represented here:

```
    const mongoose = require(’mongoose’);
    const Schema = mongoose.Schema;
    const statusStatisticsSchema = new Schema({

        sourceId:String,
        timestamp: String,
        machinestatus: String,
        colour_percentage: String,
```

```
    equipment_status:String,
    colour_status:String,
    meantime: Number
}, {
    collection: 'StatusStatistics'
});
const StatusStatistics=mongoose.model('StatusStatistics',
statusStatisticsSchema);
module.exports = StatusStatistics;
```

As we can see from the code, we have the schema definition through the mongoose library, followed by the basic schema with the data of each record that must be represented in the server along with the reference collection from which to take the data. Then the model is created that is based on the schema and finally it is exported to make it usable in the index.js file whose implementation we have just finished discussing.

5.3.3 Connection between Back-End & Front-End

The connection between the client and server side is a topic that has been partly covered in the previous paragraph regarding the creation of reference ports for the client and the server and also the definition of the connection channel between client and server via Cors. However, an important aspect has been omitted that can be found in the **api.js** file, contained in the reference folder of the spea-app front-end. This file is used to fetch data on the front-end side following the initialization of the back-end. It is made up of two asynchronous functions in particular, used to fetch data for the Home of the application and the page relating to the individual machines present in the Plant. These functions allow the front-end side to retrieve updated or historical information, ensuring effective communication with the REST API.

The first one is designed to get a general overview of the current status of the monitored machines. Through an HTTP GET request to the /api/status-statistics endpoint, the function queries the server to receive the latest recorded status of each machine in the system and once the response from the server is received if

the request was successful, the data is decoded from JSON format and returned as JavaScript objects, ready to be used by the user interface, as shown in the code below.

```
const SERVER_URL = "http://localhost:3001/api";

export const getStatistics = async () => {
    try {
        const response = await fetch(SERVER_URL+
        '/status-statistics');
        if (response.ok) {
            const statistics = await response.json();
            return statistics;
        } else {
            const message = await response.text();
            throw new Error(response.statusText + " " +
            message);
        }
    } catch (error) {
        throw new Error(error.message, { cause: error });
    }
};
```

The second one was developed to answer a more specific need, that is, the detailed analysis of the status history of a single machine. To do this, the function accepts a parameter, sourceId, which uniquely identifies the machine of interest. It then sends an HTTP GET request to the /api/status-statistics/:sourceId endpoint, where the value of the parameter is dynamically included in the URL, as shown in the code below.

```
export const getStatisticsSpecificReferenceID=async(sourceId)=>{
    try {
        const response = await fetch(`${SERVER_URL}/status-
        statistics
        /${sourceId}`);
        if (response.ok) {
```

```
            const statistics = await response.json();
            return statistics;
        } else {
            const message = await response.text();
            throw new Error(response.statusText + " " + message);
        }
    } catch (error) {
        throw new Error(error.message, { cause: error });
    }
};
```

The two functions have been used in the front-end architecture and have allowed the connection with the back-end in an effective way. Now that the picture is complete, we can analyze the Demos through which the graphical interface has been tried and tested together with the SPEA Research & Development department.

5.4 Demo in SPEA

The final phase of the thesis work was characterized by the development of a demo at the SPEA plant in Volpiano, specifically in the Production department. In this context, it was possible to test the validity of the implementations described earlier. The Multi-Agent System was initially installed on five production machines: two 4080 tester machines and three 4060 tester machines. Furthermore, the validity of the Web Application, which monitors the machine status in real-time, was also tested. The demo specifically involved the designers who acted as SPEA's representatives in the development of the thesis project, with my support as a thesis student. My role was primarily focused on assisting in the validation of the results produced by the created agents and in the installation and operation of the Web Application on the designated PCs for the SPEA machines. Alessandro Bolatto, head of the Research and Development department at SPEA, supervised the operation of the designed system. He provided valuable feedback and approved the further development of both the multi-agent system created during this phase and the associated web application. Figure 5.7 shows a screenshot of the homepage of the web application during the demo process. The boxes representing the machines,

Figure 5.7. Home Page of Web Application

each displaying their respective names, are immediately visible, followed by the last timestamp processed in real-time by the machines. Additionally, each box includes a status bar that, along with a blinking dot, the color associated with the current state, and the state name, provides a visual indication of the machine's status during real-time monitoring. A search bar is also present, which during certain phases of the demo allowed users to select a specific machine and review the information processed by the multi-agent system concerning its status. The minimalist design of the homepage was appreciated by the stakeholders and Alessandro Bolatto, who regarded it as a solid starting point for future enhancements, including graphical improvements. Finally, the behavior of the system concerning the color fading based on the machine's response times was verified, yielding positive results.

The second phase of the demo involved testing the functionality of clicking on individual machines to verify whether the Web Application correctly displayed data for each machine. As shown in Figure 5.8, after clicking on one of the machines in

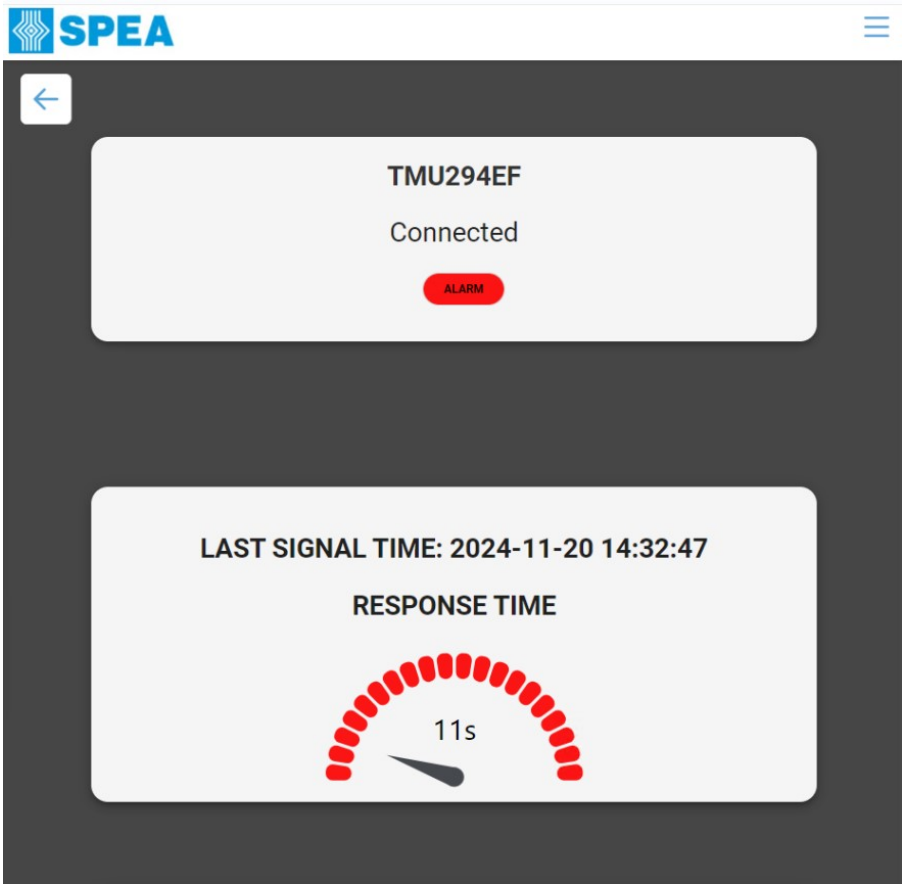

Figure 5.8. Machine Page of Web Application

the Plant system tested during the demo, a dedicated page for the specific machine opened. This page displays all the necessary information to enable a designer or operator working on the machines to monitor response times and, consequently, the machine's status in real time. The provided screenshot highlights the main aspect of the monitoring focus in this thesis work. The machine is described with its primary name, response times,machine status,equipment status and a graphical representation of the response time behavior relative to the current time. Specifically, using a representation similar to an odometer, it is possible to observe the real-time delay of the machines during their operation. The color gradually fades

as the pointer moves to the right, and within the response counter, the number of seconds the machine takes in real time to send data to the multi-agent system is displayed. This system processes the data and subsequently transmits it to the web application, which graphically represents it. To further verify the functionality, the machine's real-time activity was monitored. Specifically, the "Alarm" state was intentionally triggered by SPEA operators for this test, and it was correctly displayed by the web application. The entire process received very positive feedback from all users involved in the testing phase. Following the aforementioned demo, several potential improvements to the system were identified for future implementation. In particular, it became evident that minimizing delays in data representation is necessary. Currently, the system exhibits an average delay of approximately 5 seconds between data loading and visualization. However, this issue was met with optimism, as the possibility of reducing this delay is highly feasible for future system developments. The Web Application, with all its features, was received with enthusiasm for its clear and effective representation. The demo was a complete success, fully achieving the initial goal shared by the Politecnico di Torino and SPEA. The objective was to make the multi-agent system operational to initiate predictive analyses on SPEA machinery and to represent these predictive data through a functional graphical interface.

Chapter 6

Conclusions

The primary objective of this thesis was to design and test a multi-agent system for monitoring the operational status of industrial machinery and generating predictive insights, with the aim of improving the overall efficiency of business processes. This work is set within an evolving industrial context, where the ability to analyze and forecast machine behavior is a critical factor in optimizing production activities. In the presented case study, the adoption of a multi-agent approach proved particularly effective in managing the complexities associated with monitoring the status of industrial machines. Leveraging the distributed and cooperative capabilities of intelligent agents, the system was able to deliver real-time, detailed information about machine status while maintaining high adaptability to changes in operational conditions. Integration with predictive analysis techniques further enhanced the system, enabling the anticipation of critical events and fostering proactive decision-making processes. Notably, a constant focus was maintained on productivity, a fundamental cornerstone of business success both in terms of economic and operational outcomes. Among the key practical benefits observed during the development phase was the systemâs ability to reduce downtime and maintenance costs by identifying anomalies before they escalated into major issues. Specifically, the study of machine response times over defined time intervals allowed for the prediction of future behaviors, providing timely alerts to designers or operators. This proactive analysis of response times not only contributed to process reliability and safety but also facilitated a more strategic allocation of resources, enabling their reallocation to other critical areas or sections of the industrial plant.

The multi-agent system paradigm emerged as particularly well-suited for this operational framework. By designing cooperative agents operating on multiple levels, it was possible to consolidate and aggregate data and observations from individual agents into a centralized plant management system. This architecture enabled fast reaction and analysis times, with processes completed within seconds, optimizing operations for those responsible for monitoring and production activities. Experiments on internal agent optimization further highlighted that overburdening a single agent could negatively impact system speed and responsiveness. Conversely, evenly distributing tasks among agents created an efficient "production chain" within the plant, enabling real-time support and assistance. The development of the graphical user interface was another essential component, allowing the identification of critical information to be presented to operators during production. Key details, such as machine lifespan, were included, while less relevant elements were excluded to maintain focus on the specific operational context. The multi-agent system developed during this project was positively received by SPEA, the collaborating company, which validated the approach and highlighted areas for further refinement, such as improving agent optimization to ensure even greater precision in real-time representation. This project provided an opportunity to deepen understanding of the operational dynamics of an industrial environment, demonstrating the importance of real-time data processing in managing complex situations and improving decision-making efficiency.

In conclusion, this project demonstrated how a multi-agent system integrated with predictive analytics can serve as an innovative solution for monitoring and managing industrial plants. The work not only contributes to improving business processes but also lays the foundation for future developments, establishing these technologies as pivotal tools in modern industrial contexts.

Bibliography

[1] https://www.spea.comit/chi-siamo/.

[2] https://www.wepower.it/eto-che-cose-la-produzione-engineer-to-order/.

[3] EES, "Sistema di collaudo flying probe," 2024. Accessed: 2024-10-29.

[4] SPEA, "Flying probe testers," 2024. Accessed: 2024-10-29.

[5] https://www.polito.it/ateneo/comunicazione-e-ufficio-stampa/poliflash/a-luciano-bonaria-la-laurea-honoris-causa-in-ingegneria.

[6] R. Isermann, *Fault Diagnosis SystemsâAn Introduction from Fault Detection to Fault Tolerance*. Berlin, Heidelberg, New York: Springer, 2009.

[7] S. X. Ding, *Model-Based Fault Diagnosis Techniques*. Berlin, Heidelberg: Springer-Verlag, 2009.

[8] P. A. H. Jr, A. F. S. Levy, and A. T. Carvalho, "Estudo sobre a influencia dos acopladores capacitivos na sensibilidade da medicao de descargas parciais em maquinas eletricas rotativas," in *XX SNPTEE*, (Recife, Brasil), 2009.

[9] L. Selak, P. Butala, and A. Sluga, "Condition monitoring and fault diagnostics for hydropower plants," *Computers in Industry*, vol. 65, no. 6, pp. 924–936, 2014.

[10] CXP Group, "Digital industrial revolution with predictive maintenance," 2018. Accessed: October 29, 2024.

[11] M. H. P. Rizi and S. A. H. Seno, "A systematic review of technologies and solutions to improve security and privacy protection of citizens in the smart city," *Internet of Things*, 2022.

[12] "Visions of the future," in *Augmented Reality*, pp. 129–142, 2013.

[13] K. Ashton *et al.*, "That 'internet of things' thing," *RFID Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[14] S. A. Taj *et al.*, "Iot-based supply chain management: A systematic literature review," *Journal of Industrial Engineering and Management*, 2023.

[15] S. A. Al-Qaseemi *et al.*, "Iot architecture challenges and issues: Lack of standardization," in *2016 Future Technologies Conference, FTC*, pp. 731–738, IEEE, 2016.

[16] D. Witczak and S. Szymoniak, "Review of monitoring and control systems based on internet of things," *Journal of Future Technology*, 2024.

[17] Unknown, "Application of logistic regression in industrial maintenance management," *Journal-Economic Development Technological Chance and Growth*, June 2023.

[18] Wikipedia, "Sigmoid function," 2024. Accessed: 2024-10-29.

[19] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[20] A. Verikas *et al.*, "Mining data with random forests: A survey and results of new tests," *Pattern Recognition*, vol. 44, no. 2, pp. 330–349, 2011.

[21] Mr. Master, "Introduction: Random forest classification by example," 2024. Accessed: 2024-10-29.

[22] `https://www.hpe.com/emea_europe/en/what-is/ai-iot.html`.

[23] J. MacQueen, "Some methods for classification and analysis of multivariate observations," pp. 281–297, 1967.

[24] ML Science, "K-means clustering," 2024. Accessed: 2024-10-29.

[25] ResearchGate, "An example autoencoder model architecture with symmetrical encoder and decoder networks," 2024. Accessed: 2024-10-29.

[26] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[27] M. Grieves, *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. 2015.

[28] R. Rosen *et al.*, "About the importance of autonomy in digital twins for the optimization of production systems," *Journal of Artificial Intelligence and Applications*, 2015.

[29] E. Negri *et al.*, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, 2017.

[30] T. Gabor *et al.*, "A simulation-based architecture for smart cyber-physical systems," *Journal of Systems and Software*, 2016.

[31] U.S. Government Accountability Office, "Gao-23-106453 report," 2023. Accessed: 2024-10-29.

[32] J. Lee *et al.*, "Service innovation and smart analytics for industry 4.0 and big data environment," *Procedia CIRP*, 2014.

[33] E. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and u.s. air force vehicles," *AIAA Journal*, 2012.

[34] `https://www.nvidia.com/it-it/omniverse/digital-twins/siemens`.

[35] `https://www.nvidia.com/it-it/omniverse/`.

[36] `https://www.siemens.com/it/it/prodotti/xcelerator.html`.

[37] P. G. Balaji and D. Srinivasan, "An introduction to multi-agent systems," 2010. Department of Electrical and Computer Engineering National University of Singapore.

[38] N. Vlassis, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. 2007. Synthesis Lectures on Artificial Intelligence and Machine Learning, 1st edition.

[39] A. Damba and S. Watanabe, "Hierarchical control in a multiagent system," *International Journal of Innovative Computing, Information Control*, vol. 4, no. 2, pp. 3091–3100, 2008.

[40] P. G. Balaji, G. Sachdeva, D. Srinivasan, and C. K. Tham, "Multi-agent system based urban traffic management," in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 1740–1747, 2007.

[41] A. Koestler, *The Ghost in the Machine*. London: Hutchinson Publication Group, 1967.

[42] M. Schillo and K. Fischer, "A taxonomy of autonomy in multiagent organisation," in *Autonomy 2003, LNAI 2969*, pp. 68–82, Springer, 2004.

[43] L. Bongaerts, *Integration of Scheduling and Control in Holonic Manufacturing Systems*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1998.

[44] M. Van De Vijsel and J. Anderson, "Coalition formation in multi-agent systems under real-world conditions," in *AAAI Workshop â Technical Report*, vol. WS-04-06, pp. 54–60, 2004.

[45] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *Knowledge Engineering Review*, vol. 19, no. 4, pp. 281–316, 2004.

[46] A. K. Agogino and K. Tumer, "Team formation in partially observable multi-agent systems," tech. rep., NASA Ames Research Center, 2004. NTIS.

[47] M. C. Choy, D. Srinivasan, and R. L. Cheu, "Cooperative, hybrid agent architecture for real-time traffic signal control," *IEEE Transactions on Systems,*

Man, and Cybernetics - Part A: Systems and Humans, vol. 33, no. 5, pp. 597–607, 2003.

[48] S. E. Lander, "Issues in multiagent design systems," *IEEE Expert*, vol. 12, no. 2, pp. 18–26, 1997.

[49] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored mdps," in *Advances in Neural Information Processing Systems*, vol. 14, MIT Press, 2002.

[50] D. Srinivasan and M. C. Choy, "Neural networks for real-time traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 3, pp. 261–272, 2006.

[51] P. G. Balaji and D. Srinivasan, "Distributed multi-agent type-2 fuzzy architecture for urban traffic signal control," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 1624–1632, 2009.

[52] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuro evolution," in *Proceedings of the European Conference on Machine Learning (ECML)*, pp. 654–662, 2006.

[53] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[54] S. D. Ramchurn and et al., "Agent-based control for decentralised demand side management in the smart grid," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, vol. 5, pp. 5–12, 2012.

[55] M. Parhizkar and et al., "A multi-agent based demand response architecture for smart grids," *Renewable Energy*, vol. 69, pp. 266–275, 2014.

[56] B. Montreuil and et al., "Towards a physical internet: Meeting the global logistics sustainability grand challenge," *Logistics Research*, vol. 5, no. 2, pp. 71–87, 2016.

[57] S. Farahani and et al., "A multi-agent approach for traffic management in intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 730–740, 2013.

[58] *Multi-Agent Systems and Complex Networks*. MDPI, 2020.

[59] Wikipedia contributors, "Javascript — wikipedia, l'enciclopedia libera," 2024.

[60] Ranktracker, "Why react js is the most favored front-end technology for startups," 2024.

[61] W. contributors, “Bootstrap (framework),” 2024.