

POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master of Science Thesis

**Extending MES Software: Enhancing Data
Integration and Usability in Siemens Opcenter
and Open Data Opera MES**



Supervisor

Prof. Ernesto Sánchez

Co-supervisor:

Prof.ssa Sara Vinco

Candidate

Diego Gazmuri L.

ACADEMIC YEAR 2022/2023

Acknowledgements

En primer lugar, quiero agradecer a mis padres, Sandra y Rodrigo, por su apoyo incondicional. Han sido un soporte inquebrantable durante toda mi vida y quiero que sepan todo lo que lo aprecio. Gracias por todas las oportunidades que me han brindado a nivel personal y educativo, especialmente por la experiencia de vida irrepetible que me regalaron al permitirme ir a Italia. Esta tesis no existiría sin ustedes.

Asimismo, quiero mencionar a las dos compañeras que la vida me ha regalado, Josefina y Emilia. Hermanas queridas, me encantó haber compartido mi infancia con ustedes y espero que la vida nos depare experiencias tan lindas como las que ya hemos vivido juntos.

También quiero destacar a una persona que ha sido fundamental durante mi vida y que me acompaña desde el día en que nací. Panchita, eres mi segunda madre y te estaré eternamente agradecido por el inmenso amor que me has dado desde que tengo memoria. Me has enseñado muchísimo y no sería la misma persona si no fuera por ti.

Los quiero mucho a todos.

Infine, desidero esprimere il mio sincero ringraziamento ad aizoOn per la fiducia che ha riposto in me nello sviluppo dei progetti inclusi in questa tesi. Vorrei ringraziare in particolare Simona, che mi ha accompagnato sin dal primo giorno, e anche i miei colleghi Laura ed Umberto, che mi hanno guidato e sostenuto all'inizio del mio percorso professionale. Grazie di cuore.

Abstract

In the era of Industry 4.0, manufacturing execution systems (MES) software plays a crucial role in supporting production planning, tracking and control. This thesis focuses on extending two MES software programs for two distinct companies, Lincotek and Robopac, in response to the evolving needs of managers, operators and modern manufacturing in general.

The first phase of the project regards Lincotek's need for efficient data exchange with a certain type of machine involved in their manufacturing process. A novel architecture is implemented, supported by a dedicated database and a new workflow in the Opera MES software. The solution automatises the data transfer process, thus eliminating the need for manual data copying and ensuring data accuracy.

The second part focuses on reimagining and redesigning the default work order diagram of operations in Siemens Opcenter, thereby enhancing the understanding of work order status. A visually appealing and informative new diagram is developed using a JavaScript library. It is then placed in a convenient panel within the interface that operators interact with the most, thus enabling them to swiftly access and interpret production information.

These projects illustrate the versatility and flexibility of MES software, which allows customisation in order to be tailored towards specific needs. Through continuous improvement of MES software, companies can enhance productivity and efficiency, thereby adapting to the complexities that frequently arise in the manufacturing world.

This thesis also contributes to the advancement of smart manufacturing practices by demonstrating the importance of MES software in the process of transformation from a traditional production plant into an agile smart factory. The insights and conclusions shared throughout this work can serve as principles for companies seeking to begin the journey of digitalisation and automation in their manufacturing processes.

Ultimately, and beyond any specific implementations, this thesis carries a broader message: the importance of continuous improvement and process optimisation in the era of Industry 4.0. This is a crucial lesson that manufacturing companies would be wise to learn. Additionally, and going even further, it highlights the significance of embracing a mindset of constant betterment not just in the manufacturing industry but in all aspects of professional and personal life.

Contents

1	Introduction	1
1.1	Context	1
1.2	Objectives	1
2	Background	3
2.1	Industry 4.0	3
2.2	MES Software	7
2.3	Companies Involved in the Thesis	10
2.3.1	aizoOn	10
2.3.2	Lincotek	11
2.3.3	Robopac	12
3	Lincotek and Opera MES	13
3.1	Opera MES	13
3.1.1	Overview	13
3.1.2	Stored Procedures and Variables	18
3.1.3	Example of the Advancement of a Phase of a Work Order	20
3.2	Airflow Machines	32
3.3	Novel Contribution: Automatic Data Collection and Storage for Enhanced Quality Assurance	35
3.3.1	Objectives	35
3.3.2	Providing Data to the Airflow Machines	36
3.3.3	Reading Data from Airflow Machines and Inserting it into Opera MES	43
3.3.4	Additional Changes to Workflow 397	47
3.3.5	Final Comments	51
4	Robopac and Opcenter	53
4.1	Opcenter	53
4.2	Robopac's use of Opcenter	58
4.3	Novel Contribution: Enhanced Work Order Visual Representation	60
4.3.1	Objectives	60
4.3.2	Implementing the Interface	61
4.3.3	Generating the New Diagram	65
4.3.4	Final Comments	74

5 Discussion	75
5.1 Added Value	75
5.1.1 Lincotek and Opera MES	75
5.1.2 Robopac and Opcenter	77
5.2 Comparing Opera MES and Opcenter	78
6 Conclusions	81
Bibliography	85
List of Figures	89

Introduction

1.1 Context

Nowadays, the world is going through a new Industrial Revolution, commonly referred to as Industry 4.0. This is a new paradigm that seeks to alter forever the way in which humans interact with machines in manufacturing and production environments [1]. Industry 4.0 aims to revolutionise manufacturing to achieve a new level of industrial automation, thus improving work conditions, productivity and quality [2].

One of the key factors of Industry 4.0 is that it is IT-driven, since its design principles heavily favour interconnection and information transparency, leading to better and more informed decision making [3, 4]. In other words, Industry 4.0 is all about using new technologies to collect more and better data, leading to a better decision-making process that improves efficiency and minimises manual work [5]. In this context, manufacturing execution systems (MES) software is emerging as a critical tool to support production planning and control [6] and will therefore play a crucial role in enabling the objectives of Industry 4.0 [3].

MES is an integrated information system and real-time compliant software that is used to plan, track and document the entire manufacturing process, controlling and monitoring the transformation of raw materials into finished products [7, 8]. The usage of MES allows decision makers to make important decisions with more and better information or detect potential issues as soon as possible, leading to an increase in production efficiency [9].

Considering all of the above, implementing MES is very advantageous for a company, especially considering the worldwide manufacturing revolution being thrust forward by Industry 4.0. This means that modern companies should be pushing to incorporate this software and be looking to improve its functionalities regularly, always aiming to optimise production.

1.2 Objectives

This thesis has been done alongside aizoOn, a company that specialises in technology consulting. Two of its clients, Lincotek and Robopac, are companies that implement

MES software to manufacture their products. Since the requirements these two companies have for their MES software vary with time, with new ones frequently arising, they have hired aizoOn to implement additional functionalities when new necessities emerge.

The aim of this thesis is to implement a number of functionalities for each company. Each of these firms uses a different MES software, each coded in a different programming language, meaning that the skills needed for each part of the project will vary. Furthermore, the ability to add new functionalities shown in the rest of the thesis will hopefully demonstrate the versatility and flexibility of MES software, which allows managers to quickly implement changes that are required by workers in order to improve the manufacturing process.

Background

This chapter delves deeper into the topics introduced by the Introduction. The two initial sections continue to discuss the relevance of Industry 4.0 and MES software in today's world, exploring their origins, features and how experts believe they will continue to evolve and gain prevalence.

The final section is focused on the three companies involved in the thesis: aizoOn, Lincotek and Robopac. With respect to the first, a deeper overview is given as well as an explanation of what it usually does. Regarding the final two, their manufacturing process and its relationship to MES software is presented. This gives a more clear image of how MES software is employed and how companies benefit from its usage, as well as deepening the understanding of what these firms want from it.

2.1 Industry 4.0

Throughout history, there have been a total of four Industrial Revolutions. Every one of them has been the result of greater knowledge and scientific progress, leading to important technological advances that have revolutionised the way we see industry and manufacture products. These four paradigm-changing shifts in production have brought drastic changes to all aspects of life, from the efficiency of our production processes to the quality of life of workers around the world.

The first Industrial Revolution deeply changed the way in which goods were produced, as humanity transitioned from mostly employing skilled artisans to employing relatively unskilled workers that used machinery powered by water or steam [10].

The second Industrial Revolution is usually dated between 1870 and the beginning of the 20th century. It involved the improvement of current technologies as well as the usage of electricity instead of water and steam, thus driving rapid technical changes. Additionally, machines were arranged to favour the smooth transition from one operation to the next, thus giving birth to the assembly line. This, alongside the increased usage of interchangeable parts, marked the beginning of mass production [10, 11, 12].

The third Industrial Revolution began on the 1970s and consisted of the spread of automation and digitisation as computers and the internet started to arise. Their introduction to the manufacturing process allowed reaching unprecedented levels of

precision and accuracy [10, 13]. Due to the third Industrial Revolution, we now have affordable digital manufacturing tools that are connected to the Internet [14].

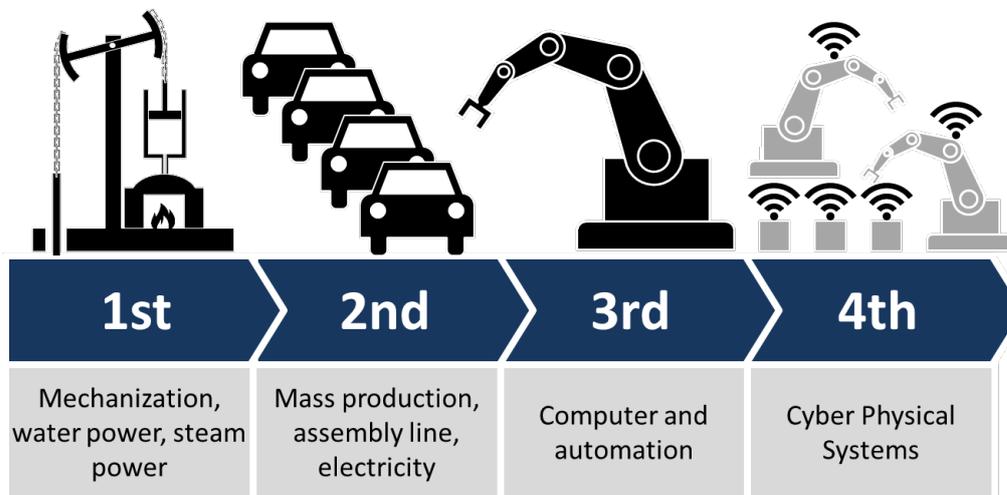


Figure 2.1: The Four Industrial Revolutions [15] (by Christoph Roser at AllAboutLean.com)

Lastly, there is the aforementioned fourth Industrial Revolution, also known as Industry 4.0. The last decades have seen technological innovation develop at an unprecedented rate, leading to changes in all aspects of life. Technologies like artificial intelligence (AI), machine learning, robotics and the Internet of Things (IoT) have revolutionised the way we live our daily lives, as well as our economic and financial sectors [16, 17].

Industry 4.0 has created many opportunities and is likely to create many more in a variety of economic areas [18]. For instance, regarding sustainability, Corfe [19] has listed multiple ways in which it might bring up opportunities to clean the environment and decarbonise our industries, thus curbing air pollution.

In the field of manufacturing, the term Industry 4.0 refers to the effect all the aforementioned technologies have on productivity, data collection, automation and product quality, among others [20]. These new technologies are allowing production plants to reach unprecedented levels of automation and connectivity. Herman et al. and Yao et al. [21, 22] consider the following technologies as key components of Industry 4.0:

- **Big Data** is “structured, unstructured and raw data stored in multiple disparate formats” [23]. Access to more data provides many benefits to companies, such as identifying failures and their causes in real time. Over the past decades, the amount of data has seen an unprecedented increase. Furthermore, it is only predicted to continue growing. Figure 2.2 shows Taylor’s [24] predictions regarding this subject.

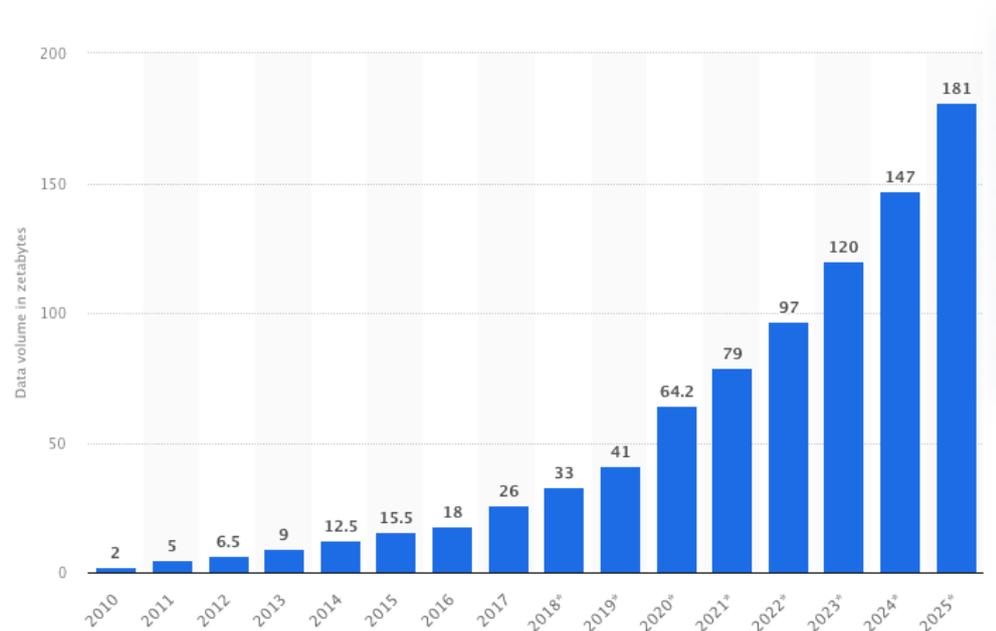


Figure 2.2: Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (by Petroc Taylor at statista.com)

- **Cloud Manufacturing (CM)** is the industrial version of cloud computing [23]. It works by encapsulating manufacturing resources into manufacturing services that are then available in the cloud as cloud services [25]. These cloud services are managed centrally and can be offered by various suppliers.
- The **Internet of Things (IoT)** is the infrastructure that allows a variety of objects, devices and sensors to connect, communicate and interact with each other through the Internet [26]. This interconnection between all items that are associated with the manufacturing process allows the generation of data that can be used throughout the supply chain as well as connecting machinery and people in unprecedented ways [27]. Additionally, IoT gives companies the ability to rapidly respond to malfunctions, since operators are notified instantly due to the effective collaboration network [28].
- **Cyber-Physical Systems (CPS)** consist of the integration between the physical and virtual worlds, usually supported by the aforementioned technologies [29]. Humans can interact with CPSs in many new ways that older systems simply cannot support. Cyber-physical systems allow computers to monitor and control physical processes, meanwhile the results of these physical processes affect future computations [30].
- **Smart Factories** is a term that has gained prevalence in recent years, as it is increasingly being used in industry. Despite this, it has no consistent definition throughout the scientific world [31]. However, the literature has proposed plenty of definitions. After analysing several definitions, Radziwon et al. [32] have defined it as follows:

“A Smart Factory is a manufacturing solution that provides such flexible and adaptive production processes that will solve problems arising on a production facility with dynamic and rapidly changing boundary conditions in a world of increasing complexity. This special solution could on the one hand be related to automation, understood as a combination of software, hardware and/or mechanics, which should lead to optimisation of manufacturing resulting in reduction of unnecessary labour and waste of resource. On the other hand, it could be seen in a perspective of collaboration between different industrial and nonindustrial partners, where the smartness comes from forming a dynamic organisation.”

From this definition and others found in the literature, we can confidently conclude that the technological advances of the last decades have made possible the creation of smart factories. The integration of some of the aforementioned technologies, such as the IoT and CPS, have given new capabilities to modern manufacturing plants, turning them into smart factories [23]. These new type of factories use information technology and data in new ways that improve the management and control of manufacturing resources, thus pushing the limits of the manufacturing processes of the past [33].

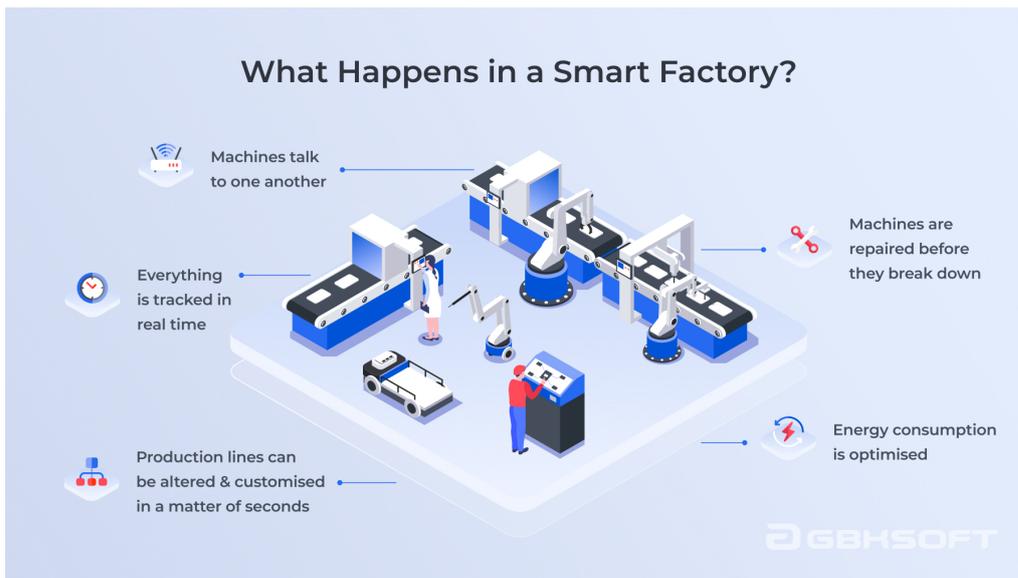


Figure 2.3: Smart Factories [34] (by Olha Didenko at altamira.ai)

The impact Industry 4.0 has had so far and will continue to have in the next decades has been widely discussed in the literature. It certainly will have deep effects on several domains beyond the industrial sector, bringing about paradigm-shifting changes [35].

Among the affected sectors, Industry will be the most impacted one [35]. The manufacturing paradigm will be deeply changed by all the aforementioned technologies, leading to a digitised production that will be made up of elements capable of communicating with each other, making decisions, responding to the environment and autonomously controlling themselves [36]. Furthermore, Industry 4.0 will have a

deep influence on industrial processes, manufacturing systems and supply chains [35].

Every industrial revolution has had at its core an increase in productivity. However, the fourth industrial revolution is expected to go even further by affecting the entire supply chain, from product development and engineering processes to outbound logistics [35].

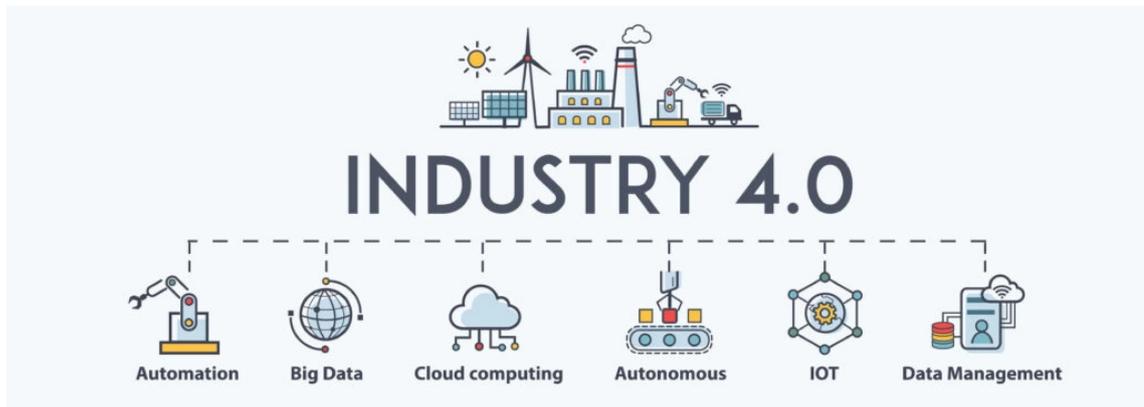


Figure 2.4: Industry 4.0 [37]

2.2 MES Software

From the previous section, we can conclude that factories that want to get on the Industry 4.0 train need to fulfil a series of criteria. Mainly, these include several technological features that will become inseparable from the factories of the coming years, since their benefits are simply too great and companies that do not take advantage of them will be at a serious, perhaps even insurmountable, disadvantage.

Due to this, companies will be likely to start developing smart factories with cyber-physical systems supporting them. In production plants like these, effective coordination and communication between all elements is essential to achieve automation and make the most out of the smart factory. These elements are connected through the IoT and CPS. In this context, a comprehensive and robust tool is needed to properly manage and control the manufacturing process, watching over all its intricate parts.

Manufacturing execution systems (MES) are an integrated information system and real-time compliant software that are used to plan, track and document the entire manufacturing process, controlling and monitoring the transformation of raw materials into finished products [7, 8]. MES software performs the important function of acquiring, saving, managing and forwarding all the information being generated, thus aiding the decision-making process and optimising the manufacturing process [38]. Therefore, usage of MES provides decision-makers with more data of higher

quality, meaning the firm is better prepared when making important decisions. This leads to an increase in production efficiency [9].

MESA (Manufacturing Enterprise Solutions Association) International is a global nonprofit that provides education and shares good practices regarding Industry 4.0 and Smart Manufacturing. MESA “was formed in 1992 as a trade association representing developers and vendors of MES software” [39]. Back in 2001, when MES was a lot less developed than it is today, McClellan [40] stated some of the benefits MES offers to its users. These benefits were a compilation done by MESA during those times and they included:

1. Reduces manufacturing cycle time.
2. Reduces work-in-process inventory.
3. Reduces paperwork between shifts.
4. Eliminates lost paperwork/blueprints.
5. Improves customer service.
6. Reduces or eliminates data entry time.
7. Reduces lead times.
8. Improves product quality.
9. Empowers plant operations people.
10. Responds to unanticipated events.

Out of these benefits, we can see that most of them (Benefits 1, 3, 6, 7, 8, 9 and 10) provide even more value to a company nowadays due to Industry 4.0 and the technologies that comprise it. It is also pretty evident that this list is somewhat outdated, since it does not really convey just how useful and powerful MES software is for a company in the Industry 4.0 context.

Tzedef [41] identifies four main areas in which MES software provides great added value:

1. **Manufacturing Indicators:** the company has access to more information regarding the efficiency and availability of the resources needed for the manufacturing process. Additionally, the system also realises when a certain resource is being wasted.
2. **Quality:** full management of all quality related stages in the process. Whenever a product does not meet quality standards, the information reaches the competent persons practically immediately, allowing fast responses and adjustments. Furthermore, due to the large amount of data being stored, it is easier to identify a certain faulty machine that is more prone to producing items that do not meet quality standards.
3. **Traceability:** due to the connection between all elements involved in production, which is possible due to the IoT, administrators have full visibility of every product that is being manufactured and every product manufactured in the past. Thus, the company will always know what process any given product went through and what resources were used to create it.
4. **Planning:** MES facilitates the planning of production due to the additional knowledge of production floor constraints and limitations it provides. It also allows for easy comparisons between what was planned and what was the

actual output of the plant. Furthermore, involving MES in scheduling ensures the optimal usability of resources [42].

The first list of benefits, put together by MESA, is reason enough for a manufacturing company to incorporate MES software into their production processes, since it provides it with many desirable and even necessary improvements. When this list was compiled over twenty years ago, the benefits were already very attractive and compelling, but with the passage of the years, the power of MES has only increased, as has its added value. During these years, MESA's list of benefits has evolved into the much broader one proposed by Tzedef. Nowadays, the benefits of properly implementing MES software go beyond just the manufacturing process, also affecting the entire supply chain.

Before Industry 4.0 was conceived and understood as it is today, MES already increased the efficiency and effectiveness of the entire manufacturing process by allowing better control of resources, including raw materials, equipment, personnel and facilities [43]. Nowadays, in the midst of the fourth industrial revolution, the literature agrees that it has become a crucial tool to enable smart factories. Mantravadi and Møller [3] state that MES software is and will continue to be crucial to enable smart factories, thus playing a key role in the manufacturing systems pushed forward by Industry 4.0. Additionally, they hypothesise manufacturing enterprises will continue to invest in MES to serve their future factories. However, the authors have also claimed that the full impact of MES is yet to be quantified in the literature, meaning that it could turn out to be even more important than is currently believed.

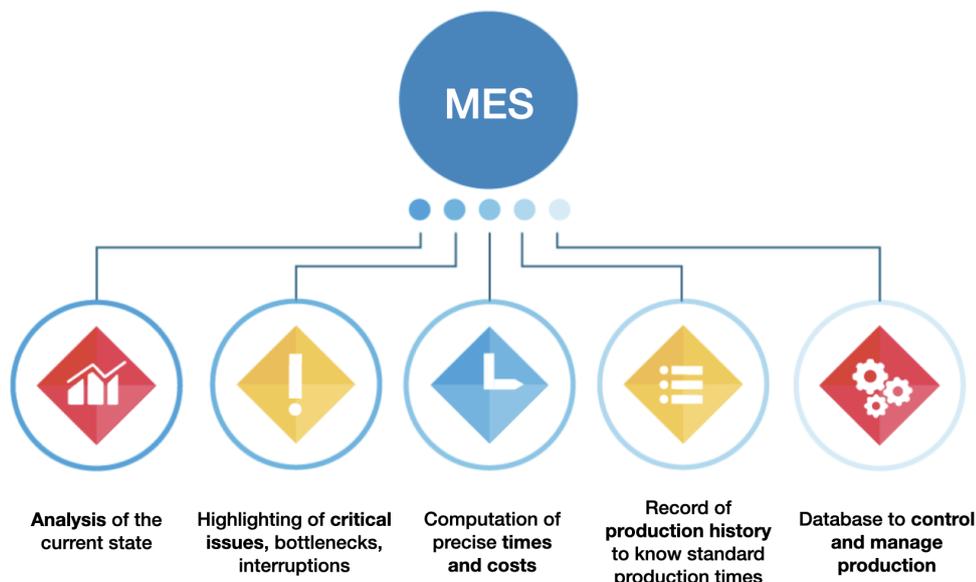


Figure 2.5: Features of Manufacturing Execution Systems [44]

Due to all of the above, manufacturing companies should begin implementing MES software as soon as possible. This is what companies like Robopac and Lincotek have done in order to work towards having their production plants evolve into smart factories. Each of these two companies has bought a license for a different MES software and have hired aizoOn to implement all modifications and additional functionalities they might require. By doing so, they have begun to access the benefits MES software provides and are on their way to automatise further their production plants, hoping to reap the rewards Industry 4.0 promises to deliver.

2.3 Companies Involved in the Thesis

The final section of this chapter delves deeper into each of the three companies involved in the projects undertaken in this thesis. It offers a general introduction to each company, enabling the reader to gain a better understanding of their objectives and roles in the implemented projects.

2.3.1 aizoOn

aizoOn Technology Consulting is a company that was founded in Turin, Italy. Since then, it has expanded to multiple cities around Italy, as well as into the United Kingdom, the United States and Australia. The firm defines itself as a “global technology consulting company focused on innovation” [45]. aizoOn is structured in eight different market areas, each of which offers variety of technological services. These areas are:

- Aerospace and Defence
- Energy
- Finance
- Consumer Goods & Services
- Transportation
- Government
- Health & Life Sciences
- Industrial Goods & Communication

Among the offered technological services is the maintenance and development of MES software. Lincotek and Robopac are two companies that have hired this service and are currently working with aizoOn to improve their respective MES software. Each firm uses a different software, as will be seen in the following subsections, meaning that aizoOn’s MES team must adapt to different software and programming languages.

In the development of this thesis, the implementation of functionalities for both of these firms was carried out through a collaboration with aizoOn. The nature of the features was very different and therefore required different skills to implement. The entire project was estimated to last four months, meaning two months were dedicated to each company.

2.3.2 Lincotek

Lincotek is an Italian manufacturing company that defines itself as follows:

“[Lincotek] is a global contract manufacturer for services in markets including Industrial Gas Turbines, Aviation and Medical Device applications, as well as a leading manufacturer of industrial coating equipment and one of the most respected producers in the Additive Manufacturing field.” [46]

The firm has its headquarters in Italy but has expanded into other European countries, as well as Asia and North America. Within Italy, it has many production plants, among which is one located in Rubbiano. This plant organises its manufacturing process with Opera MES, a MES software developed by Italian company Open Data, part of the Zucchetti group. aizoOn has been employed by Lincotek to maintain the software and implement any requested changes.

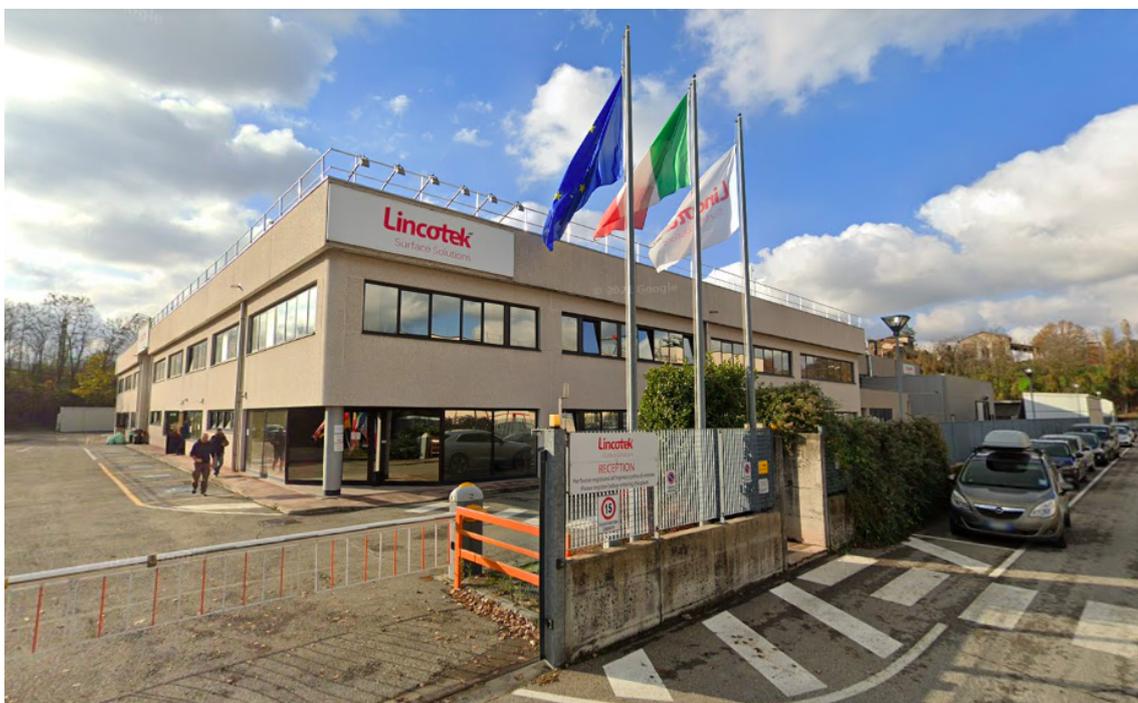


Figure 2.6: Rubbiano Manufacturing Plant

One of Lincotek’s many partners is an American company named FlowSystems that designs, manufactures and supports a wide range of flow measurement products for gasses and liquids [47]. This firm produces a special machine that Lincotek uses in its manufacturing process called an Airflow machine. This important piece of equipment is at the centre of all modifications that were done in this thesis.

In the following chapters, a deeper dive into Opera MES and how it works internally will be done, as well as explaining in detail how workers interact with it and how modifications are implemented. Additionally, the characteristics of Airflow machines and their role in Lincotek’s manufacturing process will be illustrated.

2.3.3 Robopac

Robopac is another Italian company that was founded in 1935. It “offers a wide range of solutions and services for the end-of-line packaging industry: from semi automatic wrapping machines to automatic machines, shrinkwrappers and case packers” [48]. In the years since its founding, Robopac has expanded across Europe and also into Asia and the Americas.

Robopac divides itself into six different business units:

- Robopac Machinery
- Robopac Systems
- Robopac Packers
- Sotemapack
- Robopac Brasil
- Toptier

The business unit that concerns aizoOn and therefore this thesis is Robopac Machinery. It was established in 1982 and has its headquarters in San Marino. It currently is the world leader in wrapping technology with stretch film [49].



Figure 2.7: Robopac Machinery Manufacturing Plant in San Marino

Robopac Machinery uses a MES software called Siemens Opcenter to organise its manufacturing process. Just as with Opera MES and Lincotek, aizoOn has been hired by Robopac to add functionalities to this MES software. Subsequent chapters will delve deeper into the way Siemens Opcenter works and how aizoOn modifies it.

Lincotek and Opera MES

This chapter is dedicated to the first phase of the project, regarding Lincotek, which spanned a duration of just over two months. The initial section explains in detail how Opera MES works and what are its essential components, along with any other important details regarding the project. Subsequent sections delve into the specific customisations requested and their implementation, in addition to a more comprehensive exploration of Airflow machines.

3.1 Opera MES

3.1.1 Overview

Opera is a Manufacturing Execution System platform developed by Open Data, a software company part of the Zucchetti group. They claim Opera is a “factory IT system that guides, governs, controls and optimises the entire manufacturing process, from launch of order to finished goods, connecting people, machines and functional processes” [50].

The Opera MES manual [51] states that the software allows the user to “understand and improve plant production performances, managing and monitoring the whole factory by providing real-time information”. Opera is designed to be part of a company’s IT infrastructure in a central location, between the Plant and enterprise resource planning (ERP), thus making its analysis tools more powerful in supporting strategic decision making.



Figure 3.1: IT architecture where Opera works best

Opera manages and tracks information in the following way:

Operator X started Activity Y on Resource Z

The key entities involved are:

- **Operators (X)**
- **Work Orders (WO)**, which are composed of a series of required activities and all the necessary data, such as quantities needed, delivery dates, etc. (Y)
- **Machines**, which can either be the machine itself or any resource where operators perform their activities. (Z)

An activity is any action taken in the factory floor. Whenever an action is performed, a new record is generated in the Opera MES software. This record contains all necessary information to answer the question *who did what and where?* Due to this, every step of the manufacturing process is fully tracked, meaning decision makers have an abundance of information and always know what worker did what activity and what resources were needed to do so.

The main components, including hardware and software, involved in Opera's factory layout configuration are illustrated in Figure 3.2. Without these components, the functioning of the Opera MES software is not possible.

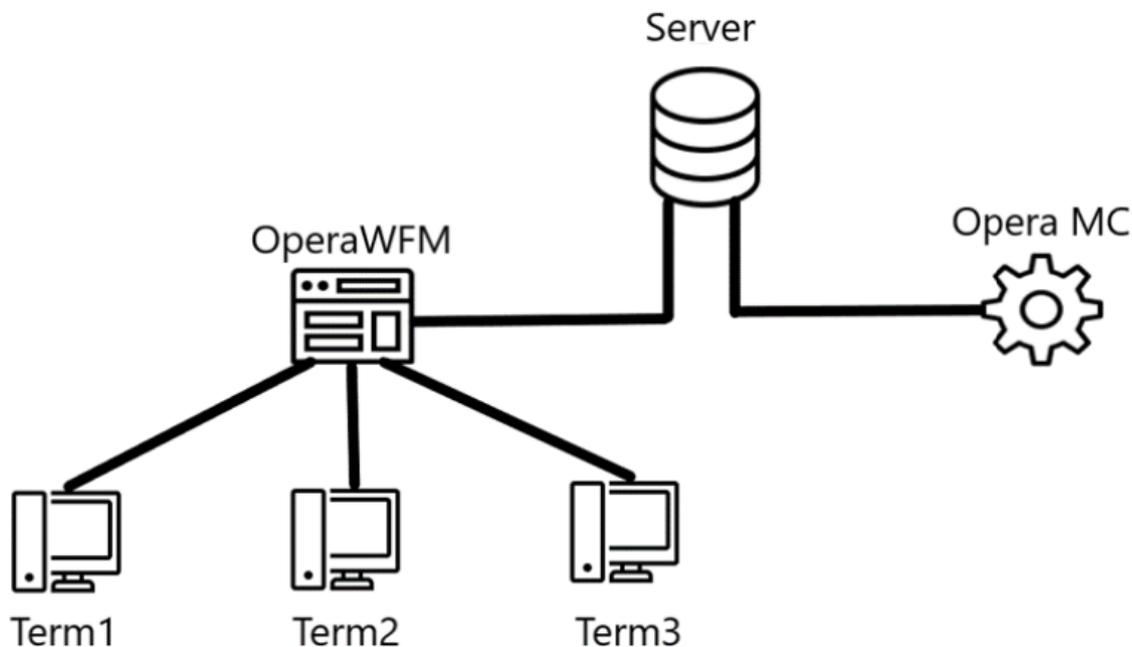


Figure 3.2: Opera's Factory Layout Configuration

The **server** is a computer where the SQL server and **Opera WFM** are installed. Naturally, all requests are managed by the server. Opera WFM enables the execution of Workflows (more on them later) to an authenticated user. In order to use Opera WFM, it must be configured to a Monitor, which then must be linked to **terminals** (Term in Figure 3.2). These are devices that can interact with the Monitor and therefore with Opera. By setting up all of this, users can properly use Opera MES. Finally, there is **Opera MC**, which represents the management side of Opera. This is what decisions makers use to access all the saved information regarding the manufacturing process.

As previously mentioned, work orders are composed of a series of activities. In Opera, these activities are called phases. Whenever an operator wants to work on a phase, they must record the progress on the MES software, which they do by accessing it through terminals in the factory floor. In order to advance a work order in the software, **workflows** must be executed.

Workflows are the main entity Opera WFM works with. A Workflow is a process that assigns values to a structured set of variables [51]. There are four elements that characterise a workflow:

- The set of variables.
- The order in which the variables are shown.
- The order in which events are propagated between variables.
- The actions to be taken when major events occur. Such events include the Workflow creation, the alteration of a variable's value and the Workflow's finalisation.

Figure 3.3 shows the interface an operator interacts with whenever they want to register something in Opera MES. This is the main window where the operator selects a Workflow to execute.

Each coloured rectangle corresponds to a different Workflow. Every Workflow is identified by its ID and is also given a name, as seen in each rectangle in Figure 3.3. By hovering over a rectangle, the Workflow's ID is visible, as illustrated in Figure 3.4. Since the manufacturing plant is located in Italy, everything is in Italian.



Figure 3.3: Opera's main window



Figure 3.4: Opera - Hovering over a Workflow

In this case, we can see that the ID of the Workflow with name *Inizio Piazzamento* is 3. Naturally, clicking on a coloured rectangle redirects the operator to the associated Workflow's interface. For example, clicking on *Riepilogo Commessa* presents information regarding all work orders. Through this particular Workflow, the operator can know the details of every work order and every phase that comprises it. Figure 3.5 shows what the operator sees when they access this workflow.

The screenshot displays the 'Riepilogo Commessa' (Order Summary) interface. It features two main tables:

- Ordine di Lavoro (Work Order Table):** This table lists various work orders. The selected row (highlighted in blue) is:

Ordine di lavoro	Descrizione	Articolo	Cliente	Ordine Cliente
000000016171	GT13E2MXL2 Vane 3 ENG READY HTC25200	002202/10	GENERAL ELECTRIC (SWITZERLAND) GMBH	4101469092
000000016174	GT13E2MXL2 Vane 3 ENG READY HTC25200	002202/10	GENERAL ELECTRIC (SWITZERLAND) GMBH	4101378272
020011000018	GT13E2MXL2 BLADE 2	003214	GENERAL ELECTRIC (SWITZERLAND) GMBH	4101322514
020011000160	Rear Hook	000285/03	SIEMENS INDUSTRIAL TURBOM. AB	7000759742
020011000200	GT13E2MXL2 Vane 3 ENG READY HTC25200	002202/10	GENERAL ELECTRIC (SWITZERLAND) GMBH	4101469672
020011000221	AE94.3A EVO2 Plate6 SI2464+TBC spessa	001306/41	ANSALDO ENERGIA S.P.A.	4500233263
020011000224	AE94.3A EVO2 Plate7 SI2464+TBC spessa	001307/41	ANSALDO ENERGIA S.P.A.	4500233263
020011000225	GT13E2MXL2 Vane 3 ENG READY HTC25200	002202/10	GENERAL ELECTRIC (SWITZERLAND) GMBH	4101378665
- Fase di Lavoro (Phase of Work Order Table):** This table shows the phases for the selected work order. The selected row (highlighted in blue) is:

Ordine di lavoro	Operazione	Bolla di lavoro	Descrizione	Mac.Priv	Mac.in uso	Q.tà Previsti	Q.tà Versata	Da rifavoran	Rilavorati	Scarti	NC aperte	NC totali	Operatore
020033000348	0010	0001689601	Ispezione preliminare	INC		1	0	0	0	0	0	0	
020033000348	0030	0001689603	FINITURA PEZZO	FIN		1	0	0	0	0	0	0	
020033000348	0040	0001689604	T.T. IN ARGA	HA0011;HA0014;		1	0	0	0	0	0	0	
020033000348	0050	0001689605	F.P.I.	F.P.I.		1	0	0	0	0	0	0	
020033000348	0060	0001689606	Ispezione finale	INC		1	0	0	0	0	0	0	
020033000348	0070	0001689607	COLLAUDO, IMBALLIO, T1, Cdc	CC		1	0	0	0	0	0	0	

Figure 3.5: Opera - *Riepilogo Commessa* Workflow

In the row labelled as *Ordine di Lavoro* (work order) there is a table containing every work order in the server and some information about it, such as the order ID and the client that requested it. By clicking on a row in the table, additional information about the selected work order is displayed in the row labelled as *Fase di Lavoro* (Phase of the work order), as shown by Figure 3.6.

The screenshot displays the 'Riepilogo Commessa' (Order Summary) interface with a work order selected. The 'Fase di Lavoro' (Phase of Work Order) table is expanded to show detailed phase information for work order 020033000348. The selected row (highlighted in blue) is:

Ordine di lavoro	Operazione	Bolla di lavoro	Descrizione	Mac.Priv	Mac.in uso	Q.tà Previsti	Q.tà Versata	Da rifavoran	Rilavorati	Scarti	NC aperte	NC totali	Operatore
020033000348	0010	0001689601	Ispezione preliminare	INC		1	0	0	0	0	0	0	
020033000348	0030	0001689603	FINITURA PEZZO	FIN		1	0	0	0	0	0	0	
020033000348	0040	0001689604	T.T. IN ARGA	HA0011;HA0014;		1	0	0	0	0	0	0	
020033000348	0050	0001689605	F.P.I.	F.P.I.		1	0	0	0	0	0	0	
020033000348	0060	0001689606	Ispezione finale	INC		1	0	0	0	0	0	0	
020033000348	0070	0001689607	COLLAUDO, IMBALLIO, T1, Cdc	CC		1	0	0	0	0	0	0	

Figure 3.6: Opera - *Riepilogo Commessa* Workflow When a Work Order is Selected

In this case, work order number 020033000348 has been selected. The second table displays a breakdown of this order, where each row is a different phase. For the purpose of this thesis, the most important columns are:

- *Operazione*: code of the phase
- *Descrizione*: description of what the phase does
- *Mac. Prv*: machine(s) that is(are) capable of executing the operation

- *Q.tà Prevista*: amount of items that are expected to pass through this phase
- *Q.tà Versata*: amount of items that have passed through the phase
- *Operatore*: operator that is executing the phase

By analysing the table, we can conclude that this work order has not yet been initiated. This is because 0 items have passed through in the first phase, as shown by the 0 under the *Q.tà Versata* column. Furthermore, the *Operatore* field is also empty, meaning no operator has been assigned to this phase.

When an operator decides to carry out this work order, they can obtain all necessary information from the table. By quickly looking at the table, the operator will know that the machine with code INC is necessary to complete the first phase of the work order.

To exemplify the typical use of Workflows to advance a work order, the final subsection of the Opera MES section will show the completion of the first phase of work order number 020033000348.

3.1.2 Stored Procedures and Variables

Opera MES is written in three different programming languages: JavaScript, HTML5 and Structured Query Language (SQL). The first two are mainly utilised to design the front-end, i.e. the interface the operator interacts with in the monitor, whilst SQL is used for the back-end. This means that the functionalities of all Workflows are defined and determined by SQL code. The Lincotek project was centred around implementing changes to Workflows, so all changes were done in SQL. Due to this, this thesis will not delve into the parts of Opera MES written in HTML5 and JavaScript.

All Opera MES Workflows work by calling stored procedures. A stored procedure is a set of SQL statements that can be saved, so they can be used over and over again. In other words, it is an SQL block of code that can be called by other pieces of code, thus allowing it to be reused often.

Every Workflow is composed four essential stored procedures:

- ***Commit***: called on Workflow finalisation
- ***OnChange Value***: called when a variable's value is updated
- ***OnWfCreate***: called when a Workflow is created (accessed by operator)
- ***ViewFillGrid***: called to populate the data for a variable

Together, they make all necessary changes to the database to register everything that is done during production. Figure 3.7 shows that these four stored procedures are the ones that make up Workflow 385.

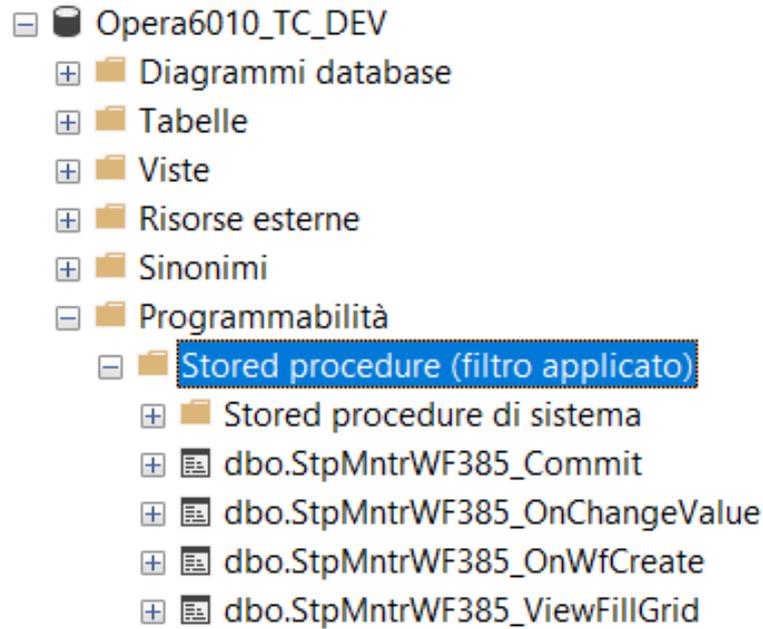


Figure 3.7: The Four Stored Procedures

While defining Workflows in the previous subsection, it was mentioned that these entities assign values to a set of variables which are displayed in a determined order. Once all necessary variables of a given Workflow have been set, the Workflow can be committed, thus running the *Commit* stored procedure. If one or more variables have not been properly set, Opera MES will raise an error.

All variables are stored in the database in a table called *MntrVars* that contains information about all of them. Figure 3.8 shows the first 10 entries of this table in the DEV environment. If a Workflow wants to use a variable, it must link to it through another table called *MntrWV*. This table not only establishes the connection, it also determines the behaviour of the variable in said Workflow through a series of parameters and flags. Additionally, the order of the variables is also detailed in *MntrWV*.

```
1 SELECT * FROM MntrVars;
2
```

100 %

Risultati Messaggi

	mv_strId	mv_strDescr	mv_strType	mv_nSize	mv_nNotVoid
1	ab_filtro	Abilitazione Filtro	string	50	0
2	Actions	Azioni	String	20	0
3	airflow_data	Airflow Data	string	50	0
4	airflow_test	Airflow Test	string	50	0
5	ar_codice	Malfunzionamento	string	50	0
6	ar_filtro	Filtro Articolo	String	50	0
7	att_program	Programma	String	50	0
8	att_programOK	Prog.Inserito	String	50	0
9	Attivita	Attività	String	20	0
10	Atz	Attrezzature	string	0	0

Figure 3.8: First Ten Entries of MntrVars Table

To create a variable and link it to a Workflow, only two insert queries are required, one to each table. In the second table, the correct order of appearance for that variable must also be set. This will all be done in detail later in the project when implementing the customisations.

Whenever a Workflow is being executed, variables manifest themselves as tabs in that Workflow. Each tab in the Workflow's display screen corresponds to a variable, and the order of the tabs follows the order determined in the MnrWV table. The next subsection will illustrate this.

3.1.3 Example of the Advancement of a Phase of a Work Order

The first step to beginning a phase of a work order is preparing and readying the machine, which is done with the first Workflow of the grid, as shown in Figure 3.3. This is Workflow 3 - *Inizio Piazzamento*. Figure 3.9 shows what the operator sees when accessing this Workflow.

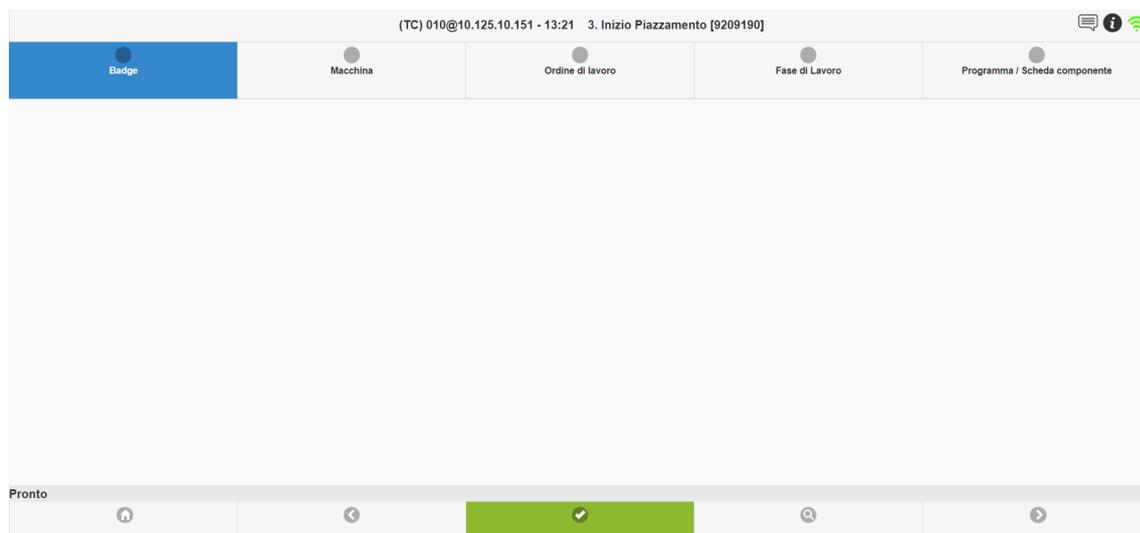


Figure 3.9: Opera MES Example - *Inizio Piazzamento*

Here we can see there are five different tabs, one corresponding to each variable associated to this Workflow, as mentioned in the previous subsection. The order of the tabs is also the aforementioned order of the variables. In the case of Workflow 3 - *Inizio Piazzamento*, there are five variables: *Badge*, *Macchina*, *Ordine di Lavoro*, *Fase di Lavoro* and *Programma/Scheda componente*. In order to properly execute the Workflow and thus correctly start the first phase of work order number 020033000348, the operator must go through each tab, inputting the correct information.

The first step is completing the tab *Badge*. For this, the operator simply needs to scan their badge in the scanner next to the monitor where he is accessing the Workflow. Once they do this, their information will appear on screen. For this

example, a random worker from the database was used. His badge number and name are shown in Figure 3.10.

Whenever the operator selects something in a tab, they are essentially giving a value to the associated variable. Therefore, by scanning their badge and selecting themselves, the operator has saved their badge ID in the variable. Whenever this is done, the tab is updated and starts showing the current saved value. This can be appreciated by comparing Figures 3.9 and 3.10. In the former, in the top left corner, one can see the selected tab is *Badge*, but there is no information in the button to open the tab, shown in blue. In the latter figure, once the operator has scanned their badge, their information is shown in that same button, meaning the variable has been set. The same occurs for each tab.

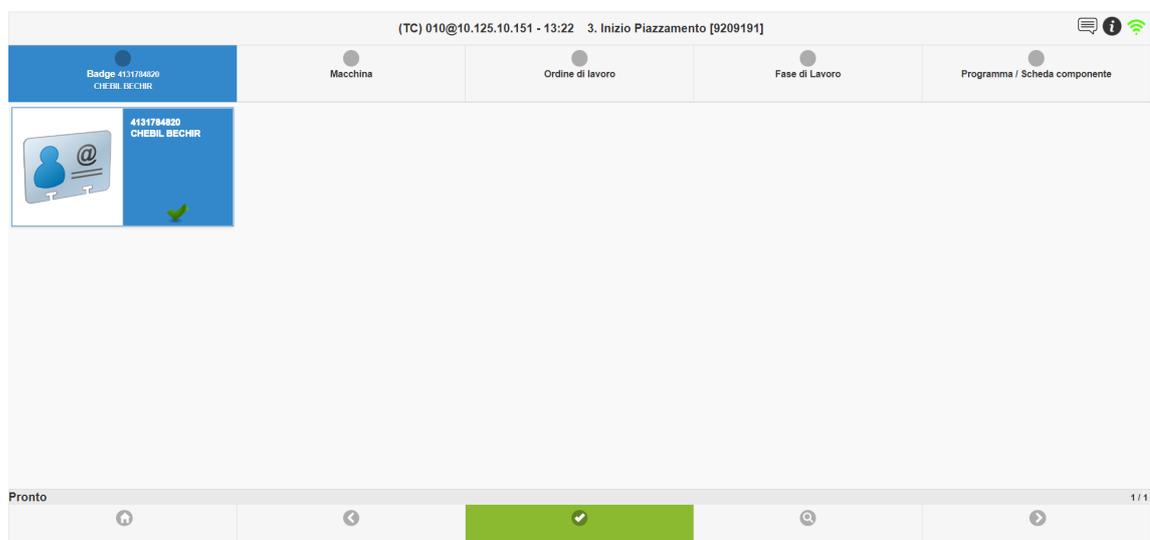
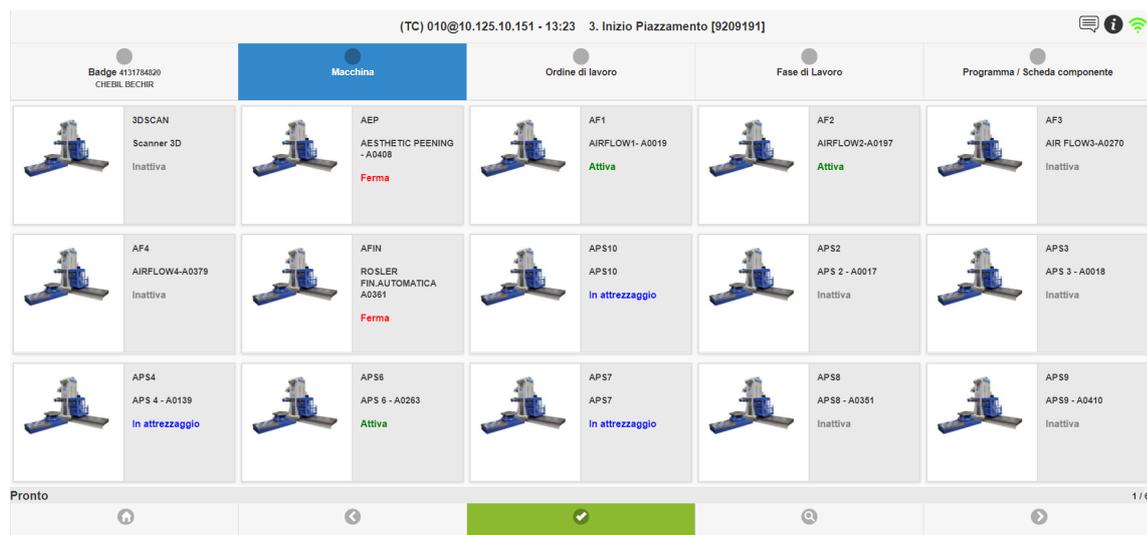
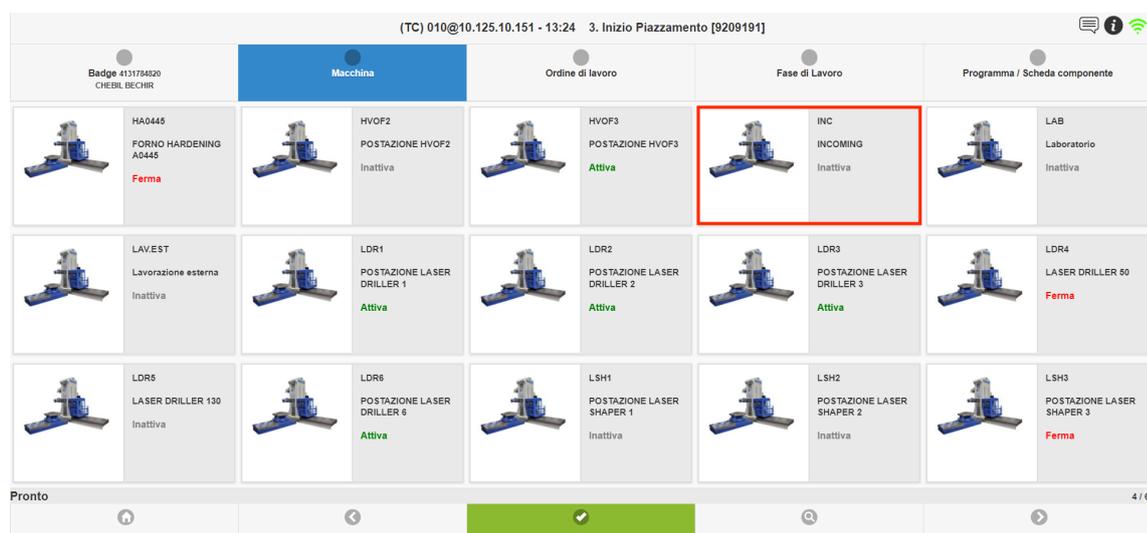


Figure 3.10: Opera MES Example - *Inizio Piazzamento*: Badge Selected

Once the badge has been scanned, the program knows what worker is going to execute the Workflow. This is essential to the proper functioning of any MES software, since traceability is a crucial part of it. The next step is selecting the machine that will be used. As mentioned above, this phase requires the use of machine INC, so the operator must select it. When first opening the *Macchina* tab, the operator is shown all machines, as shown in Figure 3.11. The operator must then select the correct machine either by scrolling with the arrow to the bottom right or by looking up the machine code after clicking on the magnifying glass, located to the left of the arrow. Figure 3.12 shows this same tab after the operator has scrolled and found the correct machine, INC.

Figure 3.11: Opera MES Example - *Inizio Piazzamento: Macchina* TabFigure 3.12: Opera MES Example - *Inizio Piazzamento: Macchina* Tab After Scrolling

In Figure 3.12, the correct machine is shown by the red rectangle. After selecting it, the operator must then select the correct work order from the *Ordine di Lavoro* tab. When opening this tab, Opera shows all work orders that can be executed with the selected machine. This is illustrated in Figure 3.13.

Every item in this list of work orders displays some information to recognise the desired one. In this case, the item corresponding to work order number 020033000348 is the one in the middle. Figure 3.14 zooms in into this item to display all the information it possesses, where the first row corresponds to the work order number. After finding the correct item, the operator must click on it to save it into the variable.

(TC) 010@10.125.10.151 - 13:24 3. Inizio Piazzamento [9209191]

Badge 413178420 CHEBL BECHIR	Macchina INC	Ordine di lavoro	Fase di Lavoro	Programma / Scheda componente
020033000508 ODL : 020033000508 Articolo : ORW012/Q4R3 - SGT800A GUIDE VANE 2 ITEM1004 Q4R3	020033000507 ODL : 020033000507 Articolo : ORW012/Q7R3 - SGT800A GUIDE VANE 2 ITEM1007 Q7R3	020033000444 ODL : 020033000444 Articolo : ORW012/Q1R2 - SGT800A GUIDE VANE 2 ITEM1001 Q1R2	020033000401 ODL : 020033000401 Articolo : ORW012/Q4R2 - SGT800A GUIDE VANE 2 ITEM1001 Q4 R2	020033000389 ODL : 020033000389 Articolo : ORW012/Q1R2 - SGT800A GUIDE VANE 2 ITEM1001 Q1R2
020033000387 ODL : 020033000387 Articolo : ORW012/Q1R2 - SGT800A GUIDE VANE 2 ITEM1001 Q1R2	020033000385 ODL : 020033000385 Articolo : ORW012/Q4R2 - SGT800A GUIDE VANE 2 ITEM1001 Q4 R2	020033000348 ODL : 020033000348 Articolo : ORW012/Q9R0 - SGT800A GUIDE VANE 2 ITEM1009 Q9R0	020033000347 ODL : 020033000347 Articolo : ORW012/Q9R0 - SGT800A GUIDE VANE 2 ITEM1008 Q9R0	020033000346 ODL : 020033000346 Articolo : ORW012/Q4R0 - SGT800B GUIDE VANE 2 ITEM 1004/Q4R0
020033000332 ODL : 020033000332 Articolo : ORW020/Q1R0 - SGT700B BLADE 2 ITEM1001 Q1R0	020033000266 ODL : 020033000266 Articolo : ORW012/Q4R0 - SGT800B GUIDE VANE 2 ITEM 1004/Q4R0	020033000265 ODL : 020033000265 Articolo : ORW012/Q7R0 - SGT800A GUIDE VANE 2 ITEM1007 Q7R0	020033000264 ODL : 020033000264 Articolo : ORW012/Q4R0 - SGT800A GUIDE VANE 2 ITEM1004 Q4R0	020033000215 ODL : 020033000215 Articolo : 003800/00 - MS91E S1B Stripping

Pronto 1/2

Figure 3.13: Opera MES Example - *Inizio Piazzamento*: *Ordine di Lavoro* Tab



Figure 3.14: Opera MES Example - *Inizio Piazzamento*: Item Containing Work Order Information

As mentioned earlier, a work order consists of many phases. Now that the operator has identified the work order, they must tell Opera what phase of this work order is to be executed. This is what the tab *Fase di Lavoro* is for. Figure 3.15 shows what the software displays to the operator. There are two available options, corresponding to the two phases of the selected work order that can be executed in the selected machine.

The operator must determine what the correct phase is. This is done by looking at the third row of the information each item displays. This row contains the work order number with the *Operazione* number appended to its end after a period. The operator can know what the *Operazione* number is by looking into the table in Figure 3.6, located in the *Riepilogo Commessa* Workflow. Figure 3.16 zooms in into the items displayed in Figure 3.15.

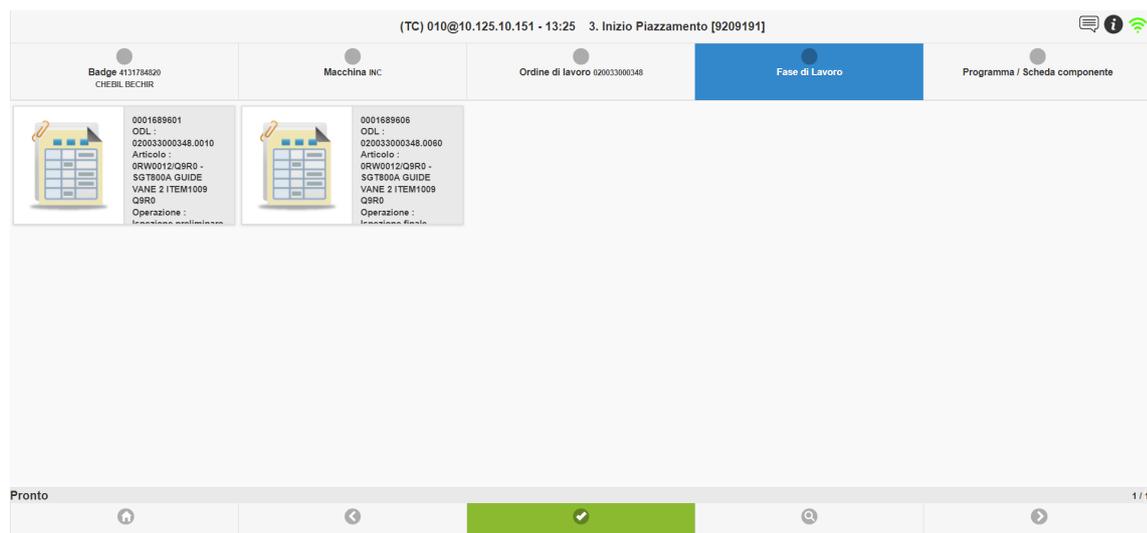


Figure 3.15: Opera MES Example - *Inizio Piazzamento*: *Fase di Lavoro* Tab

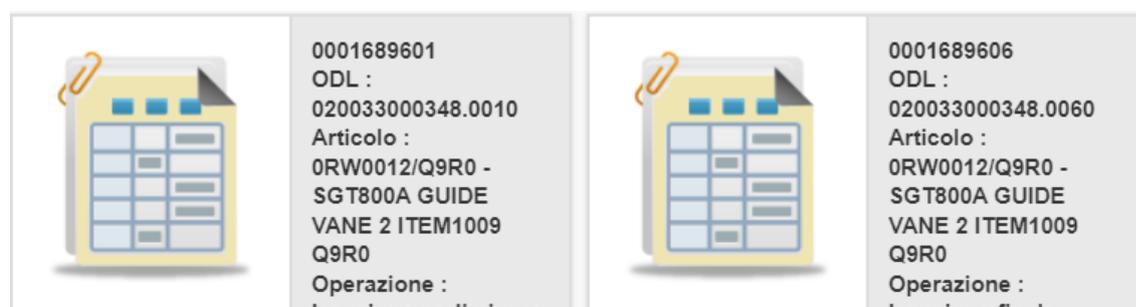


Figure 3.16: Opera MES Example - *Inizio Piazzamento*: Items in *Fase di Lavoro* Tab

After analysing the table in Figure 3.6, the operator should conclude that the first phase has an *Operazione* number equal to 0010 and should thus select the first item from Figure 3.16.

After doing so, the Workflow goes into its final tab: *Programma/Scheda componente*. In this particular phase of this work order, nothing needs to be selected in this tab, as illustrated in Figure 3.17. This is because this phase does not have options in this category. Therefore, the operator can just commit the Workflow by clicking on the green check mark located in the bottom of the screen. This tells Opera that all the variables have been set and that the Workflow is ready to be executed. If there have been errors in the setting of variables, Opera will let the operator know through an error message.

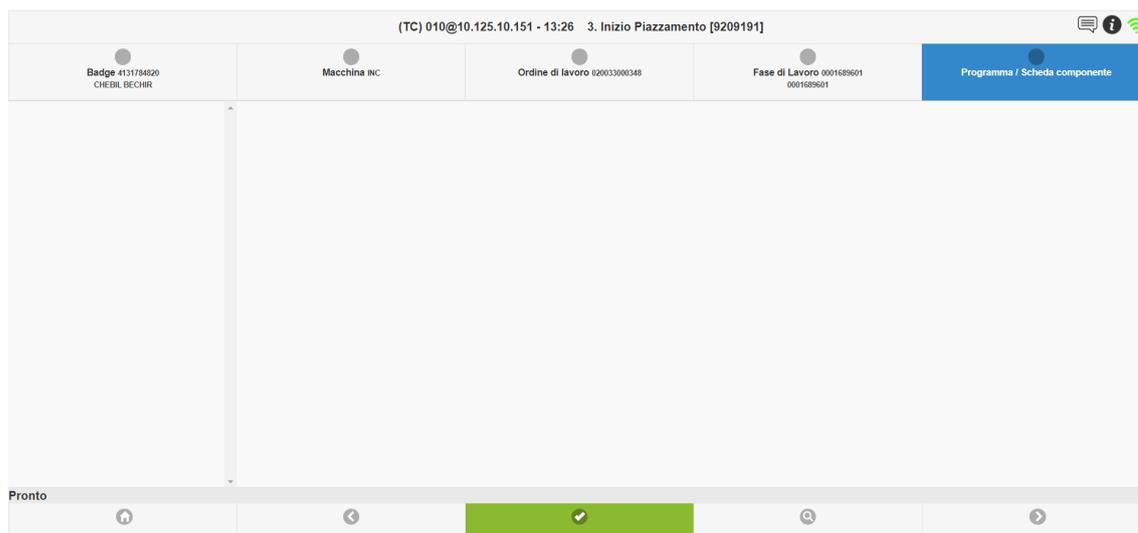


Figure 3.17: Opera MES Example - *Inizio Piazzamento*: *Programma/Scheda componente* Tab

If everything has been done correctly, Opera will be correctly notified that the setup for phase 1 has been concluded. This can be verified by going into the *Riepilogo Commessa* Workflow. Now, when selecting the work order, the table in the row labelled as *Fase di Lavoro* has been slightly updated, as shown by Figure 3.18. Now, in the first phase's row, the *Operatore* column has been updated to the name of the operator who executed the *Inizio Piazzamento* Workflow. This means that this phase is in process of being completed and that the operator that is executing it is Chebil Bechir. It is worth noting that the value of *Q.tà Versata* is still 0, meaning that the phase has not been completed yet.

Ordine di lavoro	Operazione	Bolla di lavoro	Descrizione	Mac.Priv	Mac.in uso	Q.tà Previsti	Q.tà Versata	Da rilavoran	Rilavorati	Scarti	NC aperte	NC totali	Operatore
020033000348	0010	0001689601	Ispezione preliminare	INC	INC	1	0	0	0	0	0	0	CHEBIL BECHIR
020033000348	0030	0001689603	FINITURA PEZZO	FIN		1	0	0	0	0	0	0	
020033000348	0040	0001689604	T.T. IN ARIA	HAD011;HAD014;I		1	0	0	0	0	0	0	
020033000348	0050	0001689605	F.P.I.	FPI		1	0	0	0	0	0	0	
020033000348	0060	0001689606	Ispezione finale	INC		1	0	0	0	0	0	0	
020033000348	0070	0001689607	COLLAUDO, IMBALLO, T1, COC	CC		1	0	0	0	0	0	0	

Figure 3.18: Opera MES Example - *Riepilogo Commessa* After Committing the *Inizio Piazzamento* Workflow

The next step is executing Workflow 4 - *Fine Piazzamento*, which tells Opera that the setup process has been completed. When opening this Workflow, located just to the right of *Inizio Piazzamento*, the operator faces the screen showed in Figure 3.19. This is very similar to the previous Workflow and works in exactly the same way. By following the same procedure as with *Inizio Piazzamento*, the Workflow advances the current phase and lets Opera know that the set up is ready and that the activity can be properly started.

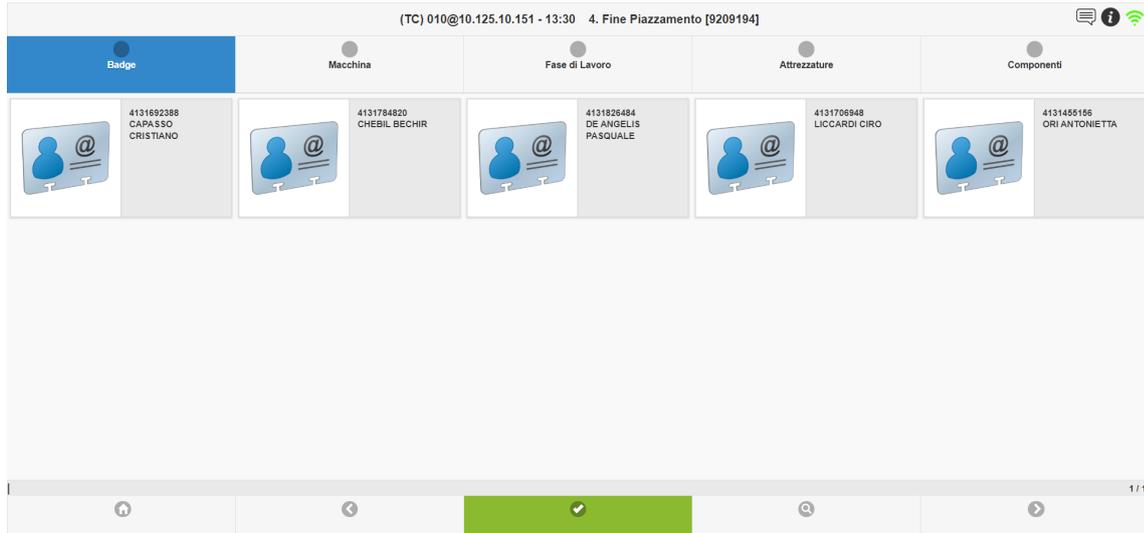


Figure 3.19: Opera MES Example - *Fine Piazzamento* Workflow

Now, after committing Workflow 4, the operator can begin the activity itself. This is done by executing Workflow 1 - *Inizio Attività*, located to the right of *Fine Piazzamento*. As evidenced by Figure 3.20, this Workflow has six variables and therefore six tabs. The first four are exactly the same as *Inizio Piazzamento* and work in the same way. The final two tabs are presented in Figures 3.21 and 3.22, and they are called *Attrezzature* and *Componenti*, meaning equipment and components respectively. For this particular work order, they are both empty since there is no additional information that needs to be set, just like the final tab in the *Inizio Piazzamento* Workflow. In other work orders, it is possible for options to appear here.

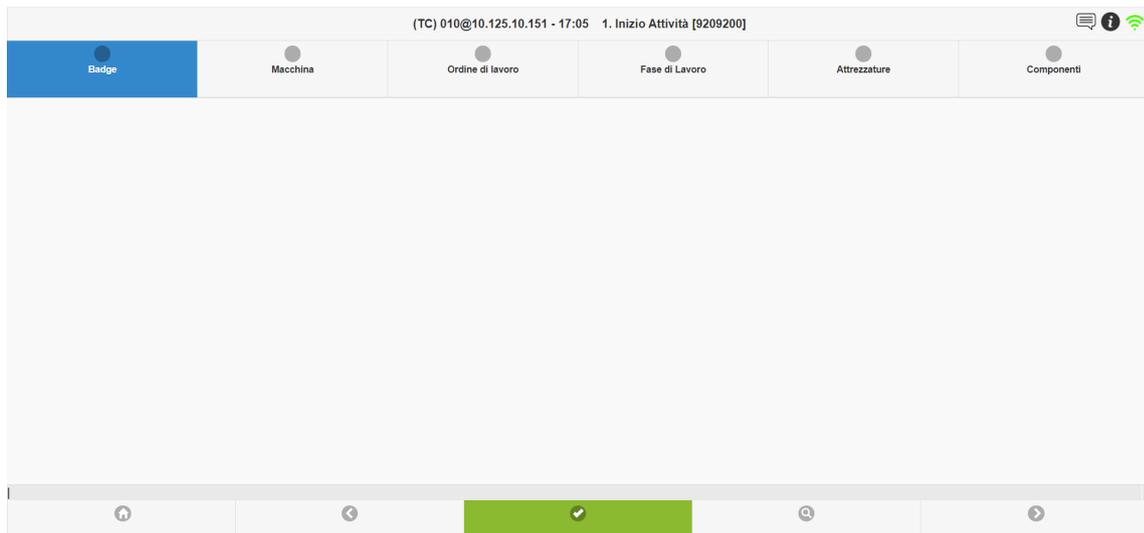
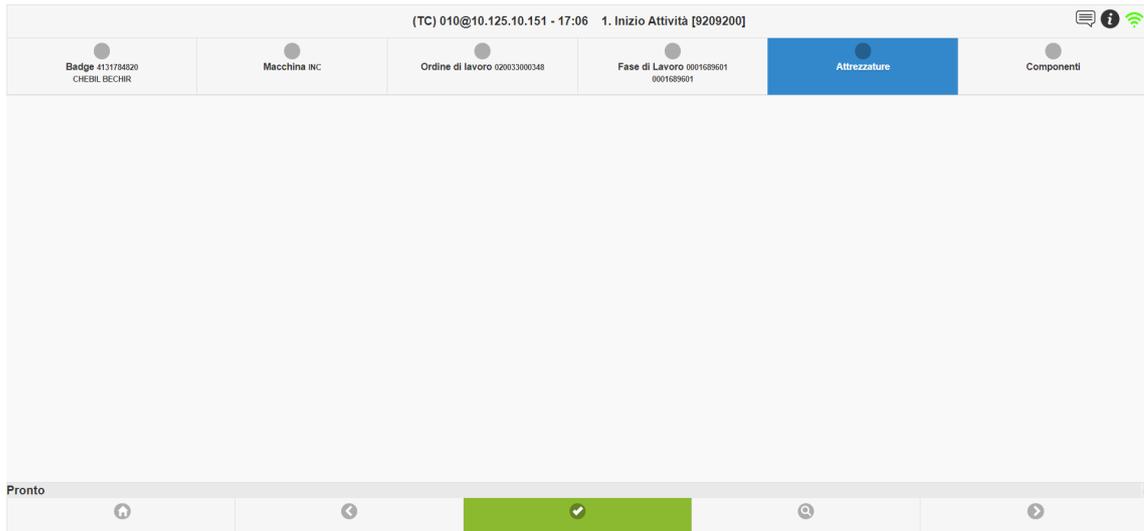
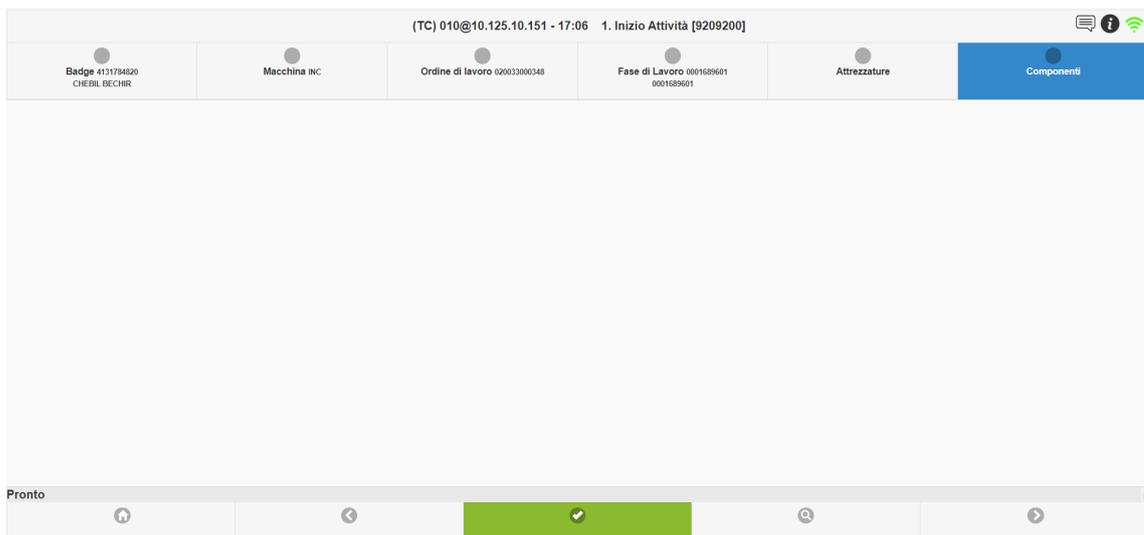


Figure 3.20: Opera MES Example - *Inizio Attività* Workflow

Figure 3.21: Opera MES Example - *Inizio Attività: Attrezzature* TabFigure 3.22: Opera MES Example - *Inizio Attività: Componenti* Tab

When this Workflow is committed, Opera is being told that the activity has been executed. This means that some quality control tests can be run and that is exactly what Opera expects the operator to do. Therefore, after committing the operator is taken directly to a new Workflow, not back to the Home page. This is Workflow 385 - *Controlli*, meaning checks. Figure 3.23 shows what the operator sees after being redirected here. There are just four tabs here and the first one, *Badge*, is completed automatically when this Workflow is accessed via another Workflow, since Opera assumes that the same operator that was executing an activity is the one that is going to run the quality control tests for it. Workflow 385 can also be accessed through the Home page, as evidenced by Figure 3.3.

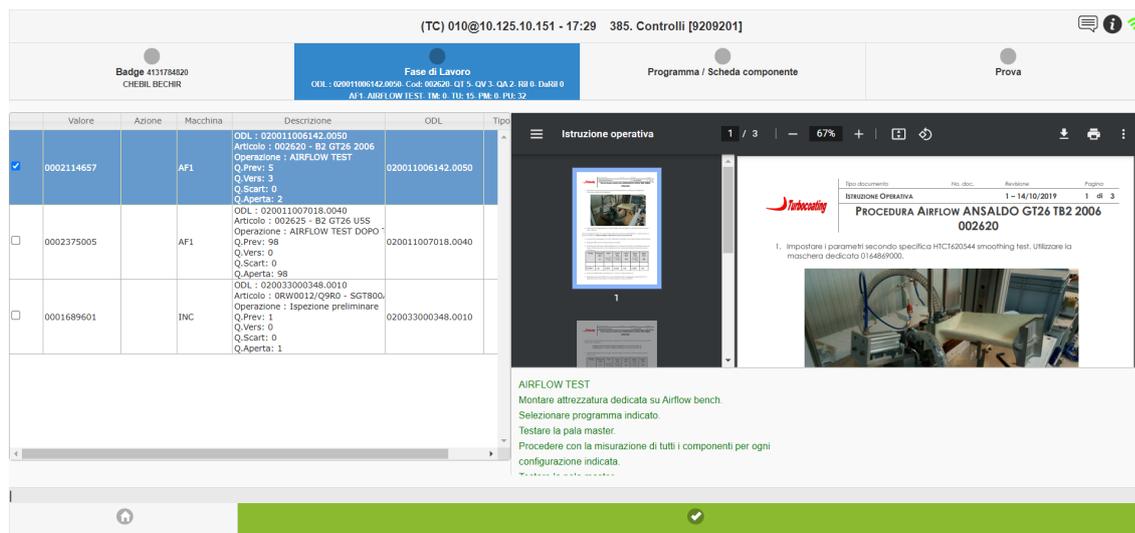


Figure 3.23: Opera MES Example - *Controlli*

In this Workflow, the operator must once again select the correct work order that the checks are being run for. Figure 3.24 zooms in into the table and shows the correct selection for work order number 020033000348, as well as the correct phase.

Valore	Azione	Macchina	Descrizione	ODL	Tipo
<input type="checkbox"/>	0002114657	AF1	ODL : 020011006142.0050 Articolo : 002620 - B2 GT26 2006 Operazione : AIRFLOW TEST Q.Prev: 5 Q.Vers: 3 Q.Scart: 0 Q.Aperta: 2	020011006142.0050	
<input type="checkbox"/>	0002375005	AF1	ODL : 020011007018.0040 Articolo : 002625 - B2 GT26 USS Operazione : AIRFLOW TEST DOPO Q.Prev: 98 Q.Vers: 0 Q.Scart: 0 Q.Aperta: 98	020011007018.0040	
<input checked="" type="checkbox"/>	0001689601	INC	ODL : 020033000348.0010 Articolo : 0RW0012/Q9R0 - SGT800 Operazione : Ispezione preliminare Q.Prev: 1 Q.Vers: 0 Q.Scart: 0 Q.Aperta: 1	020033000348.0010	

Figure 3.24: Opera MES Example - *Controlli*: Correct *Fase di Lavoro* Selection

Once the correct work order and phase have been selected, the Workflow moves into the *Programma/Scheda componente* tab. This tab was also present in the *Inizio Piazzamento* Workflow and did not show anything to the operator, so the same should occur here. This is the case, as illustrated by Figure 3.25.

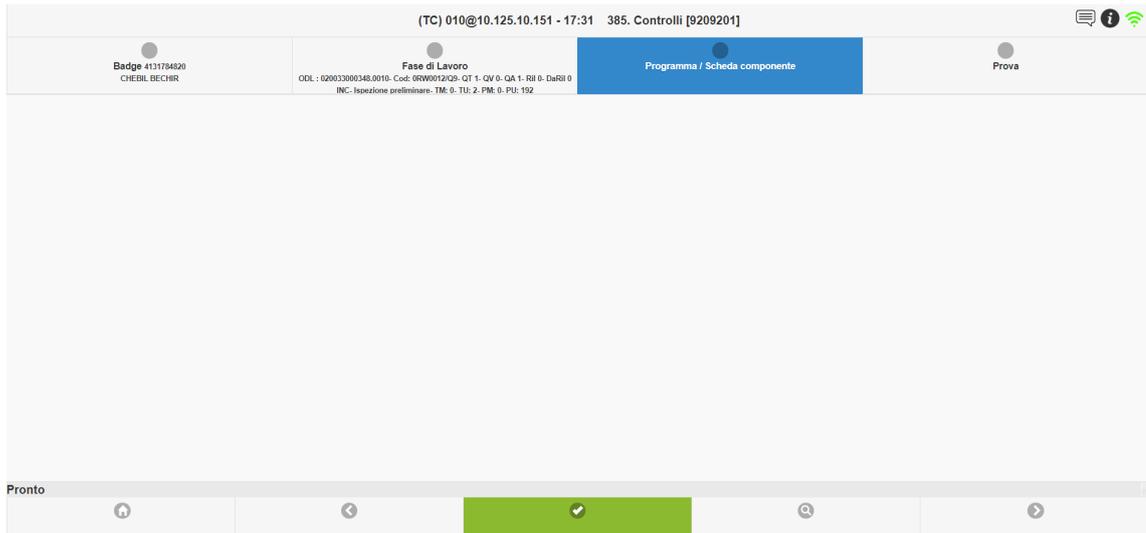


Figure 3.25: Opera MES Example - *Controlli: Programma/Scheda componente* Tab

The final tab is called *Prova*, which means test, and it is where the operator selects the test to be run. Some activities have many tests while others only have one. In this case, there is only one test that needs to be run. Due to this, only one item is present in this tab, as shown by Figure 3.26.

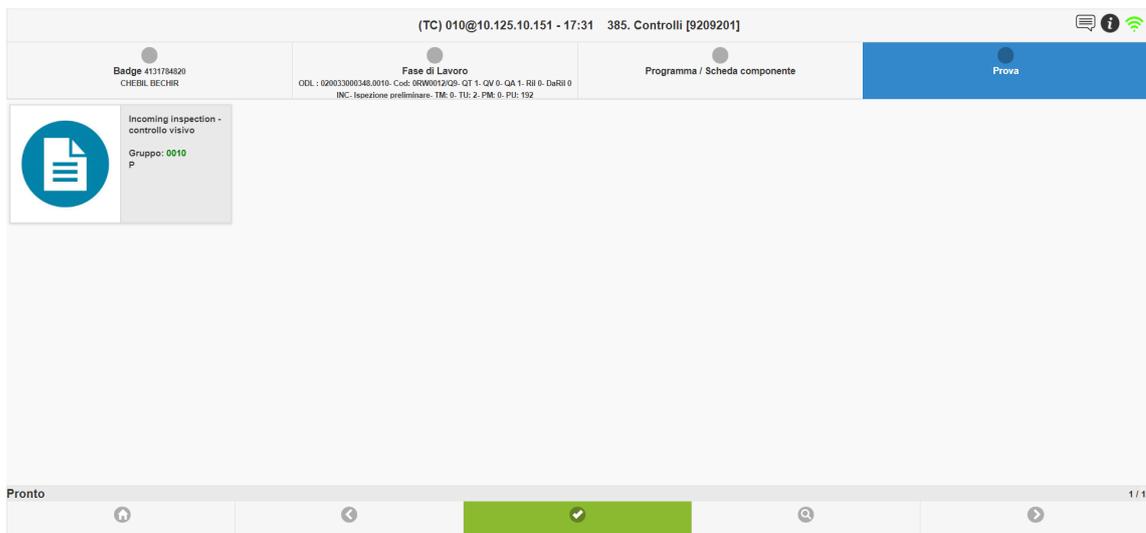


Figure 3.26: Opera MES Example - *Controlli: Prova* Tab

After selecting the test and clicking on the green check mark to commit, Opera once again redirects the operator to a new Workflow, either 380 - *Controlli per Prova* or 381 - *Controlli per Matricola*. The destination depends mainly on what the test is and what the chosen parameters were in Workflow 385 - *Controlli*. In this case, the operator is redirected to Workflow 380. This is shown in Figure 3.27. Just like in the previous case of redirection from one Workflow to another, some variables are automatically assigned based on what was selected in the Workflow that was committed.

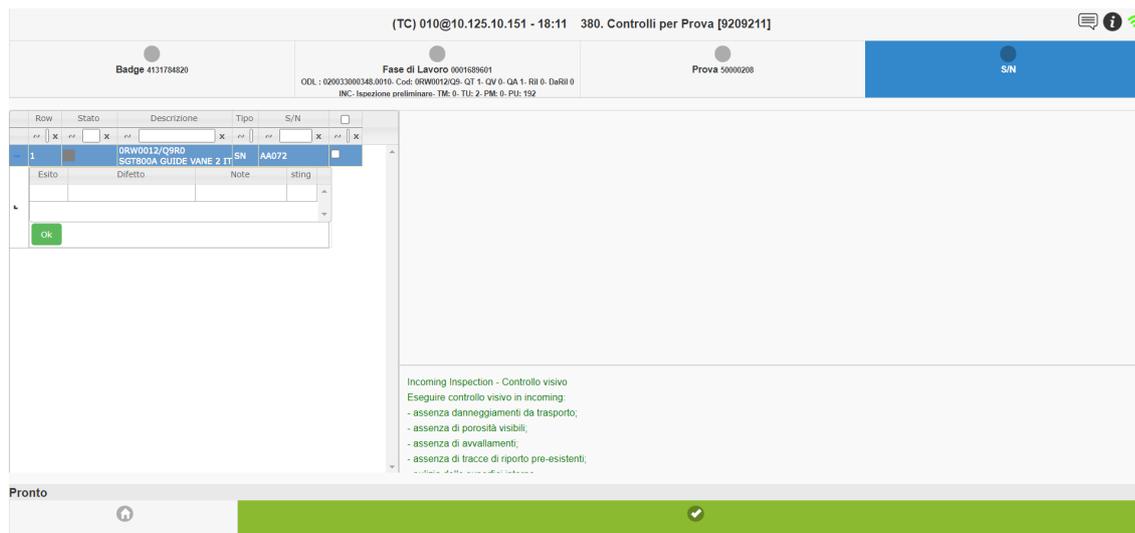


Figure 3.27: Opera MES Example - *Controlli per Prova* Workflow

As evidenced by Figure 3.27, the first three tabs of Workflow 380 (*Badge*, *Fase di Lavoro* and *Prova*) have been completed automatically. The only thing the operator must do is complete the information in the table present in the last tab. If the test was passed, the operator selects *Positiva* under the *Esito* column. If the test was not passed, then the *Negativa* option must be selected, as well as a reason under the *Difetto* column. For this example, it is assumed that the test was passed. Once the operator has completed the table, they can commit the Workflow with the green check mark. If something was not completed correctly, Opera will throw an error and the Workflow will not be committed.

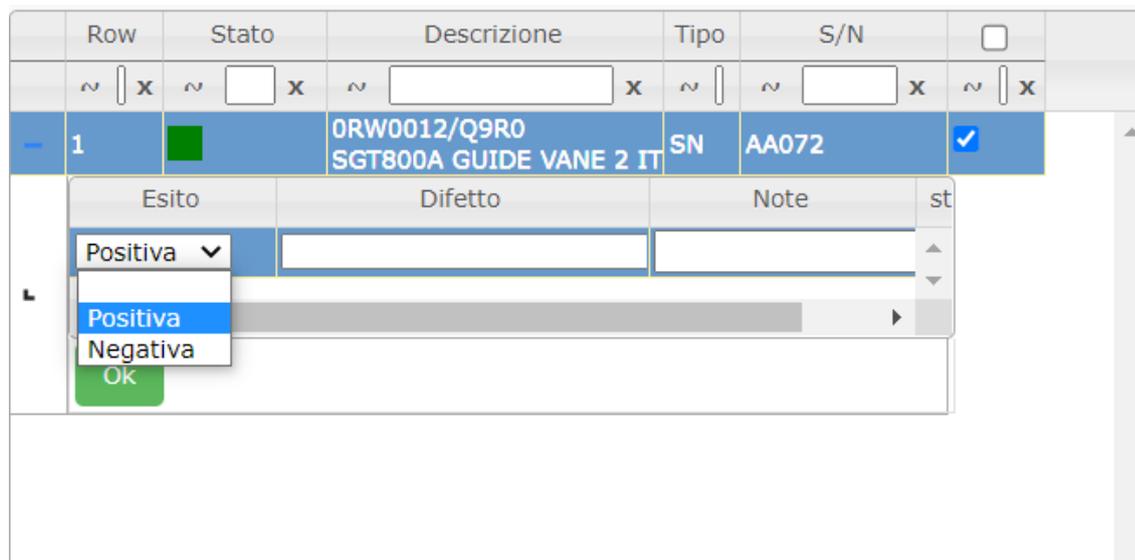


Figure 3.28: Opera MES Example - *Controlli per Prova*: Test Successfully Passed

After executing the activity with the *Inizio Attività* Workflow and informing the system that all tests were run successfully with the *Controlli* and *Controlli per Prova* Workflows, only the final step of the phase remains. This consists on running

Workflow 2 - *Fine Attività*, which tells Opera that the machine is now free to use again. Figure 3.29 shows this Workflow and its three tabs. After completing all tabs with the correct data and committing, the phase will be complete and will have been correctly executed. Figure 3.30 shows the updated table in *Riepilogo Commessa*. In it, we can see that *Q.tà Versata* is now equal to *Q.tà Prevista*, meaning that the phase has been completed. Furthermore, the *Operatore* column is now empty since no operator is working on that phase now that it is over.

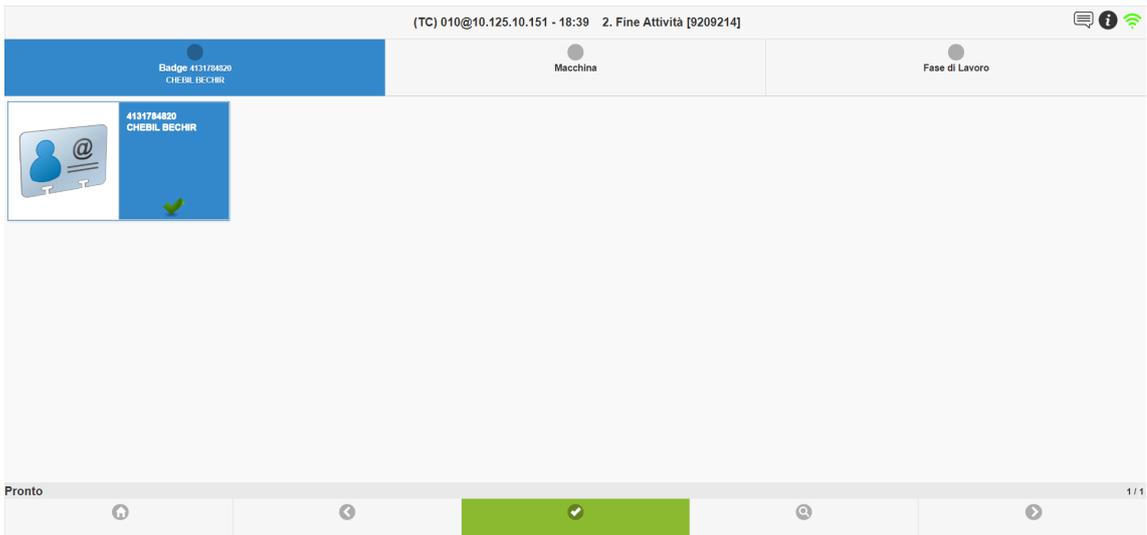


Figure 3.29: Opera MES Example - *Fine Attività* Workflow

Ordine di lavoro	Operazione	Bolla di lavoro	Descrizione	Mac.Priv	Mac.in uso	Q.tà Previst:	Q.tà Versata	Da rilavorari	Rilavorati	Scarti	NC aperte	NC totali	Operatore
020033000348	0010	0001689601	Ispezione preliminare	INC		1	1	0	0	0	0	0	
020033000348	0030	0001689603	FINITURA PEZZO	FIN		1	0	0	0	0	0	0	
020033000348	0040	0001689604	T.T. IN ARIA	HA0011;HA0014;I		1	0	0	0	0	0	0	
020033000348	0050	0001689605	F.P.I.	FPI		1	0	0	0	0	0	0	
020033000348	0060	0001689606	Ispezione finale	INC		1	0	0	0	0	0	0	
020033000348	0070	0001689607	COLLAUDO, IMBALLO, T1, C/C	CC		1	0	0	0	0	0	0	

Figure 3.30: Opera MES Example - *Riepilogo Commessa* After Phase is Complete

Now that the first phase is over, the second can be executed. It is not necessary that the same operator executes all phases, so the second one can be started by any worker.

3.2 Airflow Machines

In its manufacturing processes, Lincotek uses several machines of diverse nature. Opera MES registers which machine has been used for what activity, as well as what machines are able to execute which tasks. The database contains all machines in a table called *Macchine*, which literally means machines. This table has 85 entries, meaning Opera MES is monitoring the usage of 85 different machines in the manufacturing process. Some columns of the first fifteen entries of this table can be seen in Figure 3.31. Additionally, Figures 3.11 and 3.12 also displayed some of the available machines.

	ma_codice	cc_codice	cl_codice	tm_codice	tfm_codice	ma_codalt
1	3DSCAN	3DSCAN	3DSCAN	M	01	MAC.
2	AEP	20SHOTAV	AEP	M	01	MAC.
3	AF1	20AIR1	AF	M	01	MAC.
4	AF2	20AIR2	AF	M	01	MAC.
5	AF3	20AIR3	AF	M	01	MAC.
6	AF4	20AIR4	AF	M	01	MAC.
7	AFIN	20FINA	AFIN	M	01	MAC.
8	APS10	20APS9	APS	M	01	MAC.
9	APS2	20APS2	APS	M	01	MAC.
10	APS3	20APS3	APS	M	01	MAC.
11	APS4	20APS4	APS	M	01	MAC.
12	APS5	20APS5	APS	M	01	MAC.
13	APS6	20APS6	APS6	M	01	MAC.
14	APS7	APS7	APS7	M	01	MAC.
15	APS8	20APS8	APS8	M	01	MAC.

Figure 3.31: First Fifteen Entries of the *Macchine* Table

Four of these machines, visible in Figure 3.31, are **Airflow** machines. Their *ma_codice* (machine code) is AFX , where X is the number of the machine, ranging from 1 to 4. Airflow machines are produced by Flow Systems, an American company that designs, manufactures and supports a wide range of flow measurement products for gasses and liquids [47].

Flow Systems defines an Airflow machine as a “compact air flow measurement system specially designed for highly accurate and repeatable measurement” and state that it “meets a wide range of component flow range air testing needs” [52].

These Airflow machines also provide an interface with which the user can interact. It is worth noting that this interface is completely independent from Opera MES. Figures 3.32, 3.33 and 3.34 show three screens of this display.

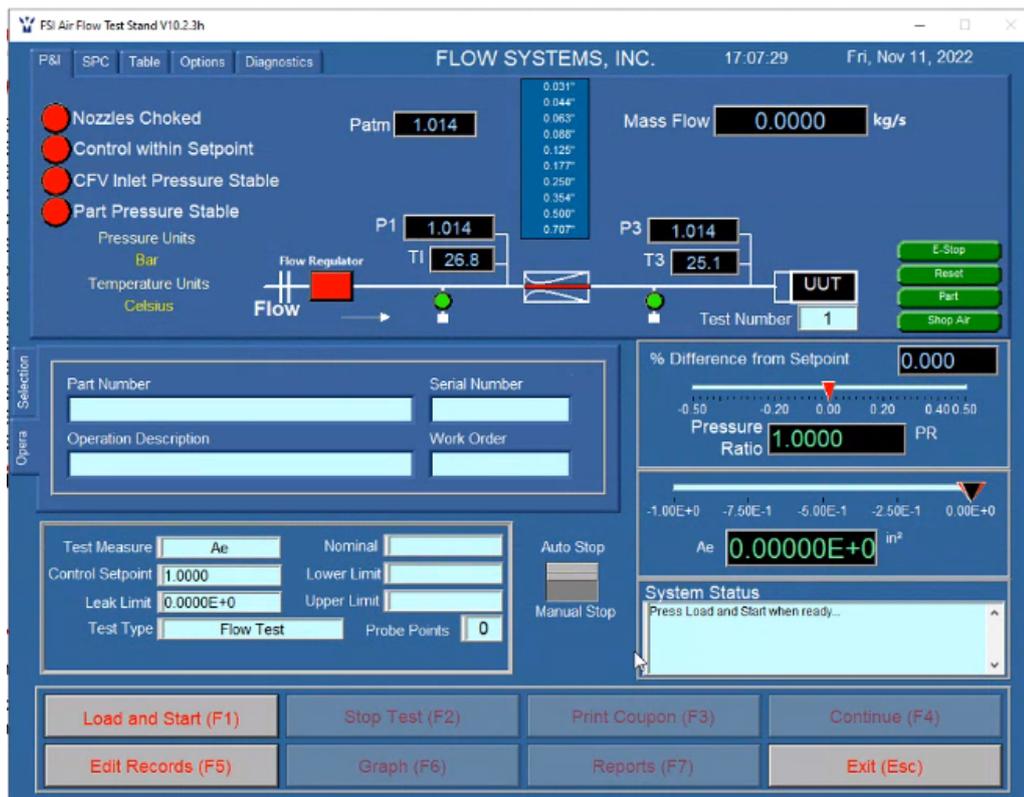


Figure 3.32: Airflow Machine Interface

Figures 3.32 and 3.33 show a screen where measured data is being displayed to the user. The former also shows the setting of parameters for the test just above the “Load and Start (F1)” button located in the bottom left corner. The latter figure displays important data in the bottom left corner rectangle with a silver edge. These data regard the individual items that are worked on during a phase of a particular work order. In fact, the work order number can also be seen there.

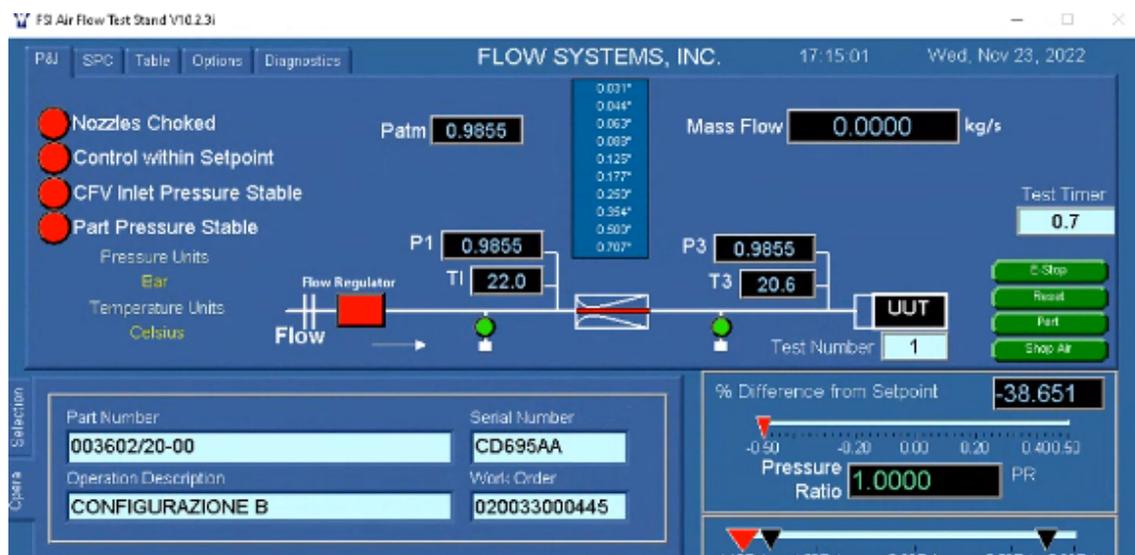


Figure 3.33: Airflow Machine Interface - Operation Data Inserted

Figure 3.34: Airflow Machine Interface - Operation Description Menu

Figure 3.34 displays a series of test parameters, as well as the test's results, that the Airflow machine collects every time it is run. Before the thesis project started, Lincotek's operators had to manually insert these data into Opera MES, since no direct and automatised connection existed between the machines and the software.

3.3 Novel Contribution: Automatic Data Collection and Storage for Enhanced Quality Assurance

3.3.1 Objectives

Every time it is run, an Airflow machine collects valuable data regarding the item it is working on, which can then be used to run quality control tests and learn more about the machine itself. After doing this, a document called Certificate of Conformity (COC) is generated and sent to Lincotek's clients as evidence that the work done has complied with certain quality criteria. This document is created automatically by reading data stored in Opera MES. The problem was that the collected data from Airflow machines were not being inserted into the system and were just saved locally in the computer connected to the Airflow machine, which meant that operators had to copy and save them into Opera MES by hand after the machine had run.

Obviously, collecting the data by hand can potentially lead to the information being stored wrongly due to human error. It is also inefficient and a suboptimal use of a worker's time, especially in the context of a factory that aims to become a smart factory over the next few years. Due to this, the decision was made to modify the Opera MES software so that it would automatically collect, store and process the data generated by the four Airflow machines. Thus, the following two objectives were defined for the project:

1. Integration of the data regarding Airflow machines stored in the Opera MES software.
2. Integration of the measurements of flow tests done by Airflow machines into the Opera MES software.

By achieving these two objectives, the Airflow data would be correctly stored into Opera MES, thus eliminating the manual copy-paste step and facilitating the generation of the COC document, whilst guaranteeing that the data used to create it is correct. In order to attain both objectives, two fundamental steps were identified:

1. Provide Airflow machines with the data they require.
2. Read measurements from Airflow machines and insert them into Opera MES.

Once these two steps are completed, the Workflow in charge of interacting with the operator must be edited and adjusted to allow everything to work properly.

The following subsections will delve deeper into what each step actually means and implies, as well as all modifications that were done to complete it. Additionally, they will explain why a new Workflow had to be created and how it was edited in order to adjust to the needs of Airflow machines and operators.

3.3.2 Providing Data to the Airflow Machines

Before the Airflow machine can actually operate and run any of its tests, it needs to gather some information about how it is registered in the database. Therefore, before the Airflow machine starts to run, all the necessary information must be gathered and provided to it to ensure it runs properly. If this is not done, then the information Lincotek desired to gather would not be available.

To achieve this, a stored procedure and four tables, one for each Airflow machine, were created in the database. The former is used to insert the information into the tables, where it will be stored and read by the machine. The tables are called OperaMESData and were not created in the main Opera MES database where everything else is located but rather in a new dedicated database created solely to communicate with the four Airflow machines. This is because this information will only be read by these machines and is not necessary for any other entities, meaning that it would introduce an unnecessary extra complexity to the already intricate system that consists of hundreds of tables and entities. Furthermore, by doing this one limits what the Airflow machine can access. Figure 3.35 shows where the created table is located in the case of Airflow machine 1.

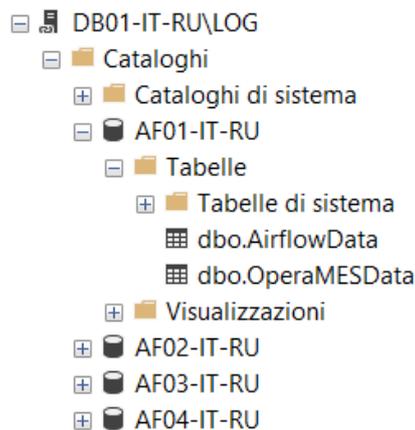
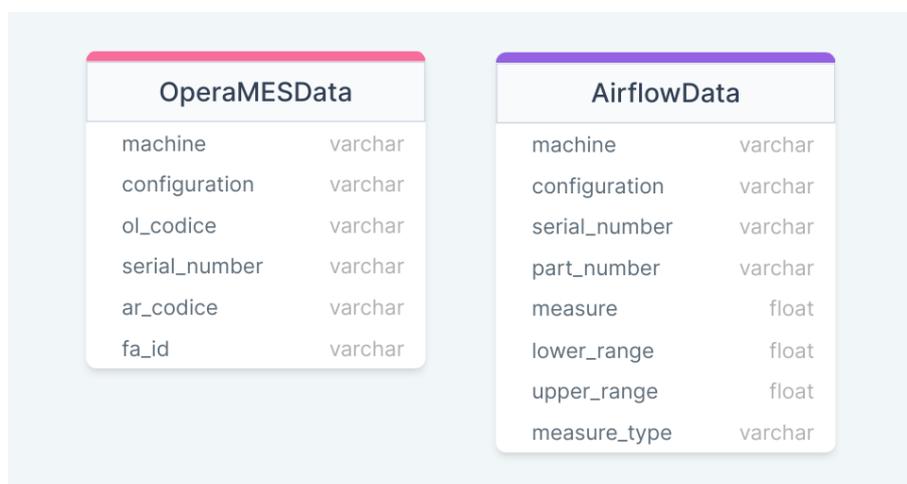


Figure 3.35: External Airflow Machines Database

As highlighted in Figure 3.35, the external database is called DB01-IT-RU\Log. In there, each Airflow machine has its own section called AF0X-IT-RU, where X is the number of the machine. Within this section there are two tables, the aforementioned OperaMESData and AirflowData, which was created later on and will be explained in the following subsection. The OperaMESData table consists of the following fields:

- **machine:** The code of the machine (AF1, AF2, AF3 or AF4).
- **configuration:** Configuration the machine has been set to.
- **ol.codice:** Code number of the work order that is going to be executed.
- **serial_number:** Code that identifies each individual item that is worked on during a phase.

- **ar_codice**: Code of the type of items that are being produced. All items belong to a category of item that has a general *ar_codice*.
- **fa_id**: Code number of the phase of the work order that is going to be executed.



OperaMESData	
machine	varchar
configuration	varchar
ol_codice	varchar
serial_number	varchar
ar_codice	varchar
fa_id	varchar

AirflowData	
machine	varchar
configuration	varchar
serial_number	varchar
part_number	varchar
measure	float
lower_range	float
upper_range	float
measure_type	varchar

Figure 3.36: Tables Created in the External Database

Most of these pieces of information were mentioned during the first subsection of this chapter, although not always citing how they are called within the system. To better understand what each of them is, a brief summary of their relationship will be provided.

Each work order is identified with a number, internally saved as *ol_codice*. Work orders consist of many steps that must be followed sequentially to complete them. These steps are called phases and they also have a number that identifies them, internally referred to as *fa_id*.

Every work order has a certain number of items it aims to produce. For example, a certain order might expect to produce one hundred nails. These items are also identified via a number called the serial number (*serial_number*). In Opera MES, every phase of the work order keeps track of the amount of items that complete it, and the operator can access this information through Workflow 370 - *Riepilogo Commessa*, as evidenced in Figures 3.5, 3.6, 3.18 and 3.30. The amount of items that are expected to pass through the phase are displayed under the column *Q.tà Prevista* and the amount that have passed already are under the column *Q.tà Versata*.

Now, better understanding what some of these fields are, it is clearer why the Airflow machine wants this information before running, since it provides it with a lot of data about the operation it is about to perform.

As mentioned earlier, a stored procedure was created to insert the data in the OperaMESData tables. This stored procedure is called *InviaDatiAirflow*, which means send Airflow data. Naturally, this stored procedure is saved in the main database system and not in the other one created for Airflow machines. This is because it

needs to access all the information in order to send it. The stored procedure has three steps:

1. Adjust machine code string.
2. Delete all current records from the OperaMESData table.
3. Insert data from the operation about to be executed.

The code of the *InviaDatiAirflow* stored procedure is included on the next two pages. The first thing the stored procedure does is change the string that contains the code machine. This is because each Airflow machine has its own AirflowData table which is identified by its machine code, as mentioned above. However, the machine code in the Opera MES database is AFX, meanwhile the other database considers it as AF0X (where X is the number of the machine). Due to this, a 0 had to be added in the middle of the string. The new string is saved into *@ma_codice_new_format*.

The next step is fairly straightforward. It first checks if there are any records in the OperaMESData table. If there are, it deletes all of them. Regardless of what happens in the IF statement, the result is that the table is always empty after this step.

The final step gathers the required information and inserts it into the OperaMES-Data table that corresponds to the Airflow machine involved in this activity. As evidenced in the code, getting this information requires joining a lot of different tables.

It is worth noting that this stored procedure has three parameters, *@ma_codice*, *@pb_codice* and *@ce_despro*. These three must be provided by any stored procedure, function or piece of code that calls *InviaDatiAirflow*. Default values for these parameters have been set, as seen in lines 2, 3 and 4 of the code, for debugging purposes.

In these code blocks, comments are shown in green, SQL instructions or statements in blue, strings in red and regular text in black. In order to make the queries work for all four machines, they had to be written in an indirect way, first saving them as strings into a variable and then executing these strings. This is because the Airflow machine in use affects what part of the external database is being accessed, so the query must take the machine code into consideration. To do this, the variable *@ma_codice_new_format* must be appended to the query, which must be done in this method.

```
ALTER PROCEDURE [dbo].[CUST_InviaDatiAirflow]
@ma_codice TipoChiave = 'AF1',
@pb_codice TipoChiave = '0001904682',
@ce_despro TipoChiave = 'CONFIGURAZIONE A'
AS
BEGIN

DECLARE @ma_codice_new_format TipoChiave
IF @ma_codice = 'AF1' BEGIN
    SET @ma_codice_new_format = 'AF01'
END ELSE IF @ma_codice = 'AF2' BEGIN
    SET @ma_codice_new_format = 'AF02'
END ELSE IF @ma_codice = 'AF3' BEGIN
    SET @ma_codice_new_format = 'AF03'
END ELSE IF @ma_codice = 'AF4' BEGIN
    SET @ma_codice_new_format = 'AF04'
END

DECLARE @sql Nvarchar(500) = ''
DECLARE @sql2 Nvarchar(1500) = ''

-- STEP 1: DELETE EVERYTHING FROM TABLE
SET @sql = 'IF ((SELECT COUNT(*) FROM [DB01-IT-RU\ LOG].[ '
+ @ma_codice_new_format + '-IT-RU].[dbo].[OperaMESData]) > 0)
BEGIN
DELETE FROM [DB01-IT-RU\ LOG].[ ' + @ma_codice_new_format +
'-IT-RU].[dbo].[OperaMESData]
END'

EXEC sp_executesql @sql
```

```

-- STEP 2: INSERT
SET @sql2 = 'INSERT INTO [DB01-IT-RU\ LOG].[ ' + @ma_codice_new_format +
'-IT-RU].[dbo].[OperaMESData] (machine, configuration, ol_codice,
serial_number, fa_id, ar_codice)
SELECT A.ma_codice as [Macchina], CCE.ce_despro as [Configurazione],
L.ol_codice as [ODL], [dbo].[fn_SplitString](U.um_codice, ''|'', 5) AS
[Seriale], F.fa_id as [fa_id], OL.ar_codice as [ar_codice]
from Azioni as A (nolock)
    inner join Fasi as F (nolock) on F.fa_id = A.fa_id
    inner join Pianificati as PN (nolock) on PN.pi_id = F.pi_id
    inner join Lavori as L on L.la_id = PN.la_id
    inner join OrdiniDiLavoro OL (nolock) on OL.ol_codice = L.ol_codice
    inner join Articoli AR (nolock) on AR.ar_codice = OL.ar_codice
    inner join Dipendenti as D (nolock) on D.di_matrico = A.di_matrico
    inner join Lotti as LT (nolock) on LT.lt_codice = L.ol_codice
    inner join UDM as U (nolock) on U.lt_autoinc = LT.lt_autoinc
    inner join OrdiniDiCollaudo OC (nolock) on OC.um_autoinc =
        U.um_autoinc and OC.la_id = L.la_id
    inner join CicloColleEffettivo as CCE (nolock) on CCE.oc_codice =
        OC.oc_codice and CCE.ce_flags = 1
    inner join Prove as PR (nolock) on PR.pr_codice = CCE.pr_codice
where F.pb_codice = ''' + @pb_codice + ''' AND
    CCE.ce_despro = ''' + @ce_despro + '''

EXEC sp_executesql @sql2
END

```

It is worth noting that, in order to access the table, it is necessary to call it in the following way:

```
[DB01-IT-RU\LOG].[AF01-IT-RU].[dbo].[OperaMESData]
```

And not simply

```
[dbo].[OperaMESData]
```

This is because, as previously mentioned, the OperaMESData table is located on a different database, meaning that the name of this database must be included for the query to know where to search.

As seen in the example of subsection 3.1.3, advancing an activity and collecting data from it is done through three Workflows. The first step is always starting the activity through WF 1 - *Inizio Attività*, then choosing the quality control test to be run and setting its parameters through WF 385 - *Controlli* and finally running the test through either WF 380 - *Controlli per Prova* or WF 381 - *Controlli per Matricola*. Whether the final involved Workflow is WF 380 or WF 381 depends on the machines involved, the parameters set and the activity.

In the case of Airflow machines, before implementing any modifications, the flow was the following:

$$\text{WF 1} \rightarrow \text{WF 385} \rightarrow \text{WF 381}$$

Given that the Airflow machine needs to receive the data sent by *InviaDatiAirflow*, the first approach was to edit Workflow 381 and make it work differently when the chosen machine was of type Airflow. The problem with this approach was that Airflow machines need additional variables that other machines do not, which made the entire Workflow much more complex than it needed to be. Therefore, the decision was made to create a new Workflow dedicated just to Airflow machines: Workflow 397 - *Controlli per Airflow*. It was based off WF 381 but the four stored procedures were edited and adapted to the needs of Airflow machines. The ways in which each stored procedure was edited will be explained in the corresponding steps.

This means that whenever the operator is working with a phase that uses an Airflow machine, Workflow 385 will redirect to Workflow 397, not Workflow 381. So, the new flow for Airflow machines is:

$$\text{WF 1} \rightarrow \text{WF 385} \rightarrow \text{WF 397}$$

The first change that was implemented to WF 397 was the calling of *InviaDatiAirflow*. As previously mentioned, one of the necessary parameters to do this is the configuration, called *ce_despro*. Therefore, WF 397 would require a tab to set the desired configuration, which is shown in Figure 3.37. When the configuration is selected by the operator, the *OnChangeValue* stored procedure of WF 397 calls the *InviaDatiAirflow* stored procedure. The code is added into *OnChangeValue* because this is the stored procedure that is called every time a variable is set.



Figure 3.37: Workflow 397 - *Controlli per Airflow*: Configuration Tab

The code to call *InviaDatiAirflow* was added into the section corresponding to the *Configurazione* tab. The following is the code:

```
ELSE IF (@idEvVar = 'sn_config') BEGIN

    IF (@sn_config IS NOT NULL) BEGIN
        SELECT 'airflow_data' AS idVar,
            '' AS idValue,
            '' AS value,
            '0' AS bDefValue,
            '0' AS bEvPropagation,
            '4' AS nMode

        EXEC [dbo].[CUST_InviaDatiAirflow]
            @ma_codice = @ma_codice,
            @pb_codice = @pb_codice,
            @ce_despro = @sn_config

    END
END
```

The ELSE IF statement at the beginning is used to identify what tab the Workflow is in: *@idEvVar* contains the name of a variable (which, as previously explained, corresponds to a tab). In other words, *@idEvVar* saves the name of the variable that needs to be given a value. The variable that saves the configuration is called *@sn_config*, so *@idEvVar* will store that string when the *Configurazione* tab comes up.

When this is the case and a configuration is chosen, two things happen. Firstly, the variable *airflow_data* is selected as the current variable, meaning the Workflow moves to that tab. Secondly, the *InviaDatiAirflow* stored procedure is called. Its parameters are *@ma_codice* (the code of the machine), *@pb_codice* (the ID of the phase) and *@ce_despro* (the chosen configuration). By doing this, the OperaMES-Data table is updated and saves the data the Airflow machine needs, thus completing Step 1.

3.3.3 Reading Data from Airflow Machines and Inserting it into Opera MES

Once the machine has received the data it needs through *InviaDatiAirflow*, the Workflow continues to advance. Then, after the machine has executed its operation, the data Lincotek wants can be accessed. These data are saved into the aforementioned AirflowData tables. There are four AirflowData tables, one for each Airflow machine. Since the Airflow machines interact directly with these tables, it is also located in the external database, just like the OperaMESData tables. Therefore, to display the information in the Opera MES interface, the first task is to read and import it into the Opera MES database. The AirflowData table consists of the following fields:

- **machine**: The code of the machine (AF1, AF2, AF3 or AF4).
- **configuration**: Configuration the machine has been set to.
- **serial_number**: Code that identifies each individual item that is worked on during a phase.
- **part_number**: Code that identifies the manufactured item.
- **measure**: Measurement made by the Airflow machine.
- **lower_range**: Lowest value that *measure* can have for a valid test.
- **upper_range**: Highest value that *measure* can have for a valid test.
- **measure_type**: Type of test being run.

Once again, a stored procedure was created to do this specific duty: *LeggiDatiAirflow* (read Airflow data). Its function is to take all the data from AirflowData and insert them into a new table located in the Opera MES database called AirflowDataHistory. The former database is then emptied to avoid any confusion. The code of this stored procedure is included on the following page.

The stored procedure takes three parameters:

- *@ma_codice*: The code of the machine.
- *@ol_codice*: The code number of the work order.
- *@fa_id*: Code number of the phase.

```
ALTER procedure [dbo].[CUST_LeggiDatiAirflow]
@ma_codice TipoChiave = 'AF1',
@ol_codice TipoChiave = '020033000445',
@fa_id TipoChiave = '65995'
AS
BEGIN

DECLARE @ma_codice_new_format TipoChiave
IF @ma_codice = 'AF1' BEGIN
    SET @ma_codice_new_format = 'AF01'
END ELSE IF @ma_codice = 'AF2' BEGIN
    SET @ma_codice_new_format = 'AF02'
END ELSE IF @ma_codice = 'AF3' BEGIN
    SET @ma_codice_new_format = 'AF03'
END ELSE IF @ma_codice = 'AF4' BEGIN
    SET @ma_codice_new_format = 'AF04'
END

DECLARE @sql Nvarchar(500) = ''

SET @sql = 'INSERT INTO [dbo].[CUST_AirflowDataHistory] (
machine, configuration, serial_number, part_number, measure,
lower_range, upper_range, measure_type, ol_codice, fa_id, timestamp)

SELECT
T.machine, T.configuration, T.serial_number, T.part_number, T.measure,
T.lower_range, T.upper_range, T.measure_type, ' + @ol_codice +
' as [ol_codice], ' + @fa_id + ' as [fa_id], getdate() as [timestamp]
FROM [DB01-IT-RU\ LOG].
[' + @ma_codice_new_format + '-IT-RU].[dbo].[AirflowData] as T

WHERE T.machine = '''+ @ma_codice + ''''

EXEC sp_executesql @sql

-- Delete everything in AirflowData table
DECLARE @sql Nvarchar(500) = ''

SET @sql2 = 'DELETE FROM [DB01-IT-RU\ LOG].[' + @ma_codice_new_format +
'-IT-RU].[dbo].[AirflowData]'

EXEC sp_executesql @sql2

END
```

Just like in *InviaDatiAirflow*, default parameters have been set for debugging purposes. Also like that stored procedure, the first step in *LeggiDatiAirflow* is changing the string that contains the code machine. It is done in the same way and for the same reasons as before.

Then, there are two queries to be done, which are both fairly straightforward. The first inserts into *AirflowDataHistory*, in the main Opera MES database, the data from the *AirflowData* table. The second query deletes everything from the *AirflowData* table corresponding to the machine used. The queries were written in an indirect way, saving them as strings into variables and then executing these strings, just like in *InviaDatiAirflow*.

Once *LeggiDatiAirflow* was completed, a way to call it in Workflow 397 had to be added. To do this, a new variable called *airflow_data* was created. This same variable was mentioned in the previous section, since it is set as the current variable just before calling *InviaDatiAirflow*.

It was mentioned earlier in this chapter that creating variables required two insert queries, one into the *MntrVars* table and another into the *MntrWV* table. In the case of the *airflow_data* variable, the first insert was:

```
INSERT INTO MntrVars
(mv_strId, mv_strDescr, mv_strType, mv_nSize, mv_nNotVoid)

VALUES
('airflow_data', 'Airflow Data', 'string', 50, 0)
```

And the second was:

```
INSERT INTO MntrWV
(wf_nId, mv_strId, wv_IdSequence, wv_IdCtrlType, wv_bHide, wv_bReport,
wv_bNotVoid, wv_bEnableEvent, wv_bValidate, wv_idEvSeqNext,
wv_spOnVarChange, wv_spOnVarValidate, wv_spDataGridView,
wv_IdSequenceNested, wv_idParent, wv_dwCtrlCliFlags, wv_sCtrlParam,
wv_spOnQueryElements, wv_sIdImage, wv_spOnDocumentList)

VALUES
(397, 'airflow_data', 20, 14, 0, 2, 0, 1, 0, 21, '', 'sp_default',
'StpMntrWF397_ViewFillGrid', NULL, NULL, '69632', NULL, 'sp_default',
NULL, '')
```

The first query simply creates the variable, gives it a name and a maximum length. In this case, the variable holds a string of maximum length 50 characters. The second query has a lot of parameters, but the most important ones, for the purpose of this thesis, are *wf_nId*, *wv_IdSequence* and *wv_idEvSeqNxt*. The first one corresponds to the Id of the Workflow that this variable is being linked to. In this case, it is Workflow 397. The second one, *wv_IdSequence*, assigns a number to the variable within the Workflow. This number acts as the Id of the variable for sequencing purposes. Finally, *wv_idEvSeqNxt* says what variable comes after the current one. This is

indicated by inserting the *wv_IdSequence* of the next variable. For *airflow_data*, its *wv_IdSequence* is 20 and the next variable that will be called after its completion is the one with *wv_IdSequence* equal to 21. Figure 3.38 shows some columns of the row corresponding to *airflow_data* in the MntRWV table.

```
3 | select * from MntRWV where wf_nId = 397
```

wf_nId	mv_strId	wv_IdSequence	wv_IdCtrlType	wv_bHide	wv_bReport	wv_bNotVoid	wv_bEnableEvent	wv_bValidate	wv_idEvSeqNxt
397	airflow_data	20	14	0	2	0	1	0	21

Figure 3.38: MntRWV Table Showing *airflow_data* Variable

Now, after having created the variables, one has to make the corresponding tab appear when accessing Workflow 397. This is done by editing the *ViewFillGrid* (VFG) stored procedure by adding an IF statement. The VFG stored procedure is in charge of retrieving the information the user should visualise and sending it to the client. The VFG has a series of IF statements, where each one corresponds to a different variable. It has an internal variable, called *@idvar*, that saves the name of the variable corresponding to the tab that is currently open. Thus, a typical IF statement in a VFG stored procedure does the following:

```
IF @idvar = some_variable BEGIN
    Execute some code
    ...
END
```

In the case of the *airflow_data* variable, this is the code:

```
ELSE IF @idvar = 'airflow_data'
BEGIN
    SELECT 'Dati da Airflow' AS [idInternalVal],
        ' AS [idVal],
        '<b><h1 style=color:GREEN>Aggiorna Dati Macchina
        Airflow</h1></b>' AS [ShortDescr],
        '/images/pr_esitoOK.png' AS [ImageRef],
        ' AS [Descr]
END
```

This code leads to the output displayed by Figure 3.39, which corresponds to the Airflow Data tab in Workflow 397. Each of the items of the SELECT query affects what is shown in the Opera MES client. For example, *[ImageRef]* takes the save location of an image within the system and displays it in the client.

Clicking on the image of Figure 3.39 makes Opera MES call *LeggiDatiAirflow*, thus saving the data the machine collected into the Opera MES database.



Figure 3.39: Workflow 397 - *Controlli per Airflow*: Airflow Data Tab

```
ELSE IF (@idEvVar = 'airflow_data') BEGIN

    IF (@airflow_data IS NOT NULL) BEGIN
        SELECT 'airflow_test' AS idVar,
            '' AS idValue,
            '' AS value,
            '0' AS bDefValue,
            '0' AS bEvPropagation,
            '4' AS nMode

        EXEC [dbo].[CUST_LeggiDatiAirflow]
            @ma_codice = @ma_codice,
            @fa_id = @fa_id,
            @ol_codice = @ol_codice

    END
END
```

3.3.4 Additional Changes to Workflow 397

So far, two stored procedures have been created, *InviaDatiAirflow* and *LeggiDatiAirflow*. These two allow the communication between the Opera MES database and another external database that interacts directly with Airflow machines, thus permitting the exchange of data between Opera MES and the Airflow machines. This is essential to gather the data collected by the latter. The result of using these two stored procedures is that the Airflow machines have all the data they need to be executed and run the tests that gather the information Lincotek wants to retrieve, which, after the implemented modifications, is now saved in new tables in the database.

Furthermore, a new Workflow was created: Workflow 397 - *Controlli per Airflow*. This Workflow was generated from the code of Workflow 381, i.e. its four stored procedures (*OnWfCreate*, *ViewFillGrid*, *OnChangeValue* and *Commit*). These stored procedures were edited in the previous two subsections so that they would call *InviaDatiAirflow* and *LeggiDatiAirflow* when necessary. This means that the communication between the Opera MES database and the external database dedicated to Airflow machines is triggered by WF 397.

Naturally, and since WF 397 is so different from WF 381, many more modifications had to be done to the four stored procedures that originally came from WF 381. These additional changes aim to create a functional Workflow that works in an intuitive way and achieves all of the goals Lincotek had set. This subsection is dedicated to the most relevant changes that facilitated the gathering of data and that improved the usability for operators.

Firstly, Figure 3.40 shows Workflow 397 with its final tab selected, *Airflow Test*. In this tab, the operator is shown all available tests that match the selected work order, phase, serial numbers and configuration (set in previous tabs) and must then select which tests are to be run.

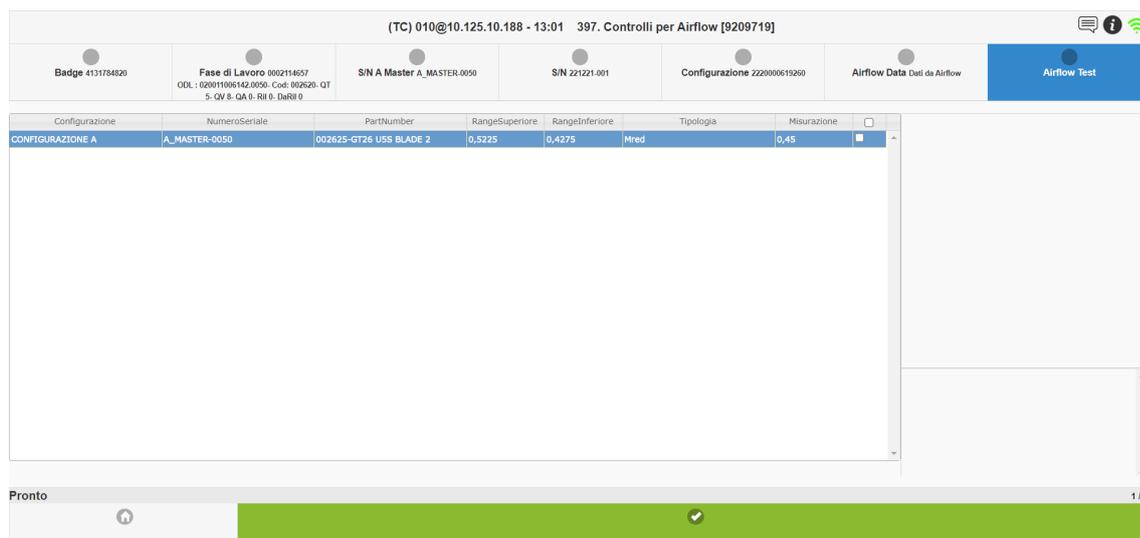


Figure 3.40: Workflow 397 - *Controlli per Airflow*: Airflow Test Tab

This is a more advanced version of Workflow 397 compared to the one displayed in Figure 3.37, where the *Configurazione* tab had just been added. The main difference between the two Workflow 397 versions is that the one exposed in Figure 3.40 has three additional tabs: *S/N A Master*, *Airflow Data* and *Airflow Test*. The last one replaced the tab *Prova*.

In previous Workflows, the operator had to select the serial number in the *S/N* tab, which is associated to the *sn_config* variable. In the case of WF 397, there is an additional tab called *S/N A Master*, which also sets the serial number. Why is this?

Usually, serial numbers identify each individual item that is produced during a phase. However, Lincotek requested that, when a phase required an Airflow machine, an additional type of serial number be created. Due to this, there now are two types of serial numbers when a phase is to be executed by an Airflow machine: the original “normal” serial numbers and the new ones, called master serial numbers. The former, as always, identifies each individual item. The latter encompasses all the individual ones that fall under that particular phase. There are two types of master serial numbers, one that is executed before the “normal” ones, *A_master*, and one that is executed after, *Z_master*.

In other words, every phase that uses an Airflow machine has an additional two master serial numbers that encompass all the original serial numbers that identify each individual item being worked on. In order to run the tests, the operator must first select the *A_master* serial number on the *S/N A Master* tab. Then, they must skip the *S/N* tab (i.e. leave it empty) and continue with the normal execution of the Workflow. Once this has been done, the operator can execute the tests of each individual serial number, which are selected in the *S/N* tab. Once all of them have been properly tested, final tests must be run by selecting the *Z_master* serial number in the *S/N* tab.

The purpose of the master serial number implementations is not the scope of this thesis project, but it was done to have more control over the testing process and thus generate more information. This was requested by Lincotek after the initial project had begun.

As mentioned above, there are two additional tabs that Workflow 397 possesses, *Airflow Data* and *Airflow Test*. The former is associated to the *airflow_data* variable and was discussed in subsection 3.3.3 and shown in Figure 3.39. As mentioned, it triggers the call to the *LeggiDatiAirflow* stored procedure. On the other side, *Airflow Test* is the tab where the operator must select what tests are going to be run (see Figure 3.40) when the green check mark is clicked and thus the *Commit* stored procedure is called.

The *Commit* stored procedure must take all the selected tests and consider them, and only them, when running. Any test that was not selected by the operator must be ignored. Despite the fact that many tests can be selected, only one response is given. If all of the tests have been passed, then the result of the test is positive. If one or more tests do not pass, then the result is negative and a *Non-Conformità* (Non-Compliance) report is generated, which the operator must manually attend to. This report is dealt with in Workflow 383 - *Gestione Non Conformità*. There, the operator must fill out some data that saves into Opera MES what went wrong with the tests. If this is not done, the work order cannot be advanced.

To individually check every selected test, the *Commit* stored procedure must iterate through all of them, just like a for loop. This is done by using a cursor. In SQL, queries often return a set of rows, and a cursor is a mechanism for working with one row at a time. The following is the code of the cursor:

```

DECLARE @pr_esito TipoChiave = 'Positiva'

DECLARE CursoreAirflow CURSOR LOCAL SCROLL FOR

SELECT * FROM [Opera6010_TC_DEV].[dbo].[CUST_AirflowDataHistory] WHERE
machine = @ma_codice AND ol_codice = @ol_codice AND fa_id = @fa_id AND
serial_number = @serial_number AND configuration = @sn_config

OPEN CursoreAirflow
FETCH FIRST FROM CursoreAirflow INTO @machine, @configuration,
@serial_number, @part_number, @measure, @lower_range, @upper_range,
@measure_type, @ol_codice, @fa_id, @timestamp

WHILE @@FETCH_STATUS=0 BEGIN

    -- Check if @measure is between the allowed ranged and set @pr_esito
    IF (@measure < @lower_range) OR (@measure > @upper_range) BEGIN
        SET @pr_esito = 'Negativa'
    END

    -- Check if serial number is Z Master
    IF (@serial_number like '%Z_MASTER%') BEGIN
        SET @isZMaster = 1
    END
    ELSE BEGIN
        SET @isZMaster = 0
    END

    FETCH NEXT FROM CursoreAirflow INTO @machine, @configuration,
@serial_number, @part_number, @measure, @lower_range, @upper_range,
@measure_type, @ol_codice, @fa_id, @timestamp
END

CLOSE CursoreAirflow
DEALLOCATE CursoreAirflow

```

The variable *@pr_esito* saves the result of the test. Its default value, before any of the tests have been changed, is *Positiva*, meaning that the the result is successful. If a single negative test is found, the value of the variable changes to *Negativa*.

Firstly, the cursor must be declared. Then, the SELECT statement fetches all the tests that have been selected by the operator which will individually go through the checks within the WHILE loop. When a row is selected by the cursor, it temporarily saves all of the values of its columns into variables that must be declared previously. This is done by the FETCH FIRST FROM command.

Then, the variables are used to execute queries or any other operation. In this case, the code checks for any test that does not comply with what is expected. As mentioned previously, a single test with a value outside of the permitted range saves *Negativa* in the *@pr_esito* variable and thus makes the whole result of the tests negative.

Within the WHILE loop, an additional IF statement is present. Its only purpose is to check if a *Z_master* serial number has been selected, because in this case the *Commit* stored procedure will redirect the operator into a different Workflow.

3.3.5 Final Comments

This marks the end of the most important changes done to the Opera MES software to allow the efficient and complete communication with Airflow machines in order to gather the valuable data they provide that Lincotek operators were having to insert manually, thus risking it being saved incorrectly.

Naturally, many more modifications were done to Workflow 397 to ensure that it worked properly and that operators would not experience any trouble when using it. However, many of these were small adjustments that consisted of many hours of debugging the code rather than large changes that drastically altered the way in which the entire software works. Due to this, only the more important modifications were included in this chapter, since these are the ones that allow the reader to understand how the objectives were achieved and how value was provided for Lincotek with the creation of a simple and effective way of communicating with the Airflow machines. Chapter 5 will analyse and delve deeper into the added value these implementations provide.

Robopac and Opcenter

This chapter is centred around the second project of the thesis, which spanned approximately six weeks. The first section explains what Opcenter is and how it works, while the second details what the project aimed to do and how it was ultimately achieved.

4.1 Opcenter

Opcenter is a manufacturing operations management (MOM) solution that enables a company to digitalise their manufacturing operations [53]. It is a software that covers and provides solutions for several areas, including:

- Advanced Planning and Scheduling
- Manufacturing Execution
- Quality Management
- Manufacturing Intelligence and Performance
- Research, Development and Laboratory

Within the Manufacturing Execution area, Siemens offers two products as part of its Opcenter package: Opcenter Execution Process and Opcenter Execution Discrete. The former is dedicated to process manufacturing while the latter is designed for discrete manufacturing.

Put simply, process manufacturing consists on following a formula or recipe to create a product, whereas discrete manufacturing is concerned with the assembly of products in a prescribed process [54]. Robopac falls under the second category and therefore uses Opcenter Execution Discrete.

Siemens claims that Opcenter Execution Discrete “provides powerful Manufacturing Execution capabilities that ensure greater process flexibility and efficiency, complete integration of regulatory and quality requirements, synchronised production processes for optimal supply-chain management and sustained reductions in maintenance and operation costs” [55].

When mentioning Opcenter throughout this chapter and the entire thesis in general, it will be referencing only its discrete MES part, Opcenter Execution Discrete, not the entire MOM solution that encompasses all Opcenter software.

In Opcenter, users with different roles interact with the application. These roles determine the level of authorisation a user has and what they have access to. A user is defined as a person who can access the application, and each one has a login and password [56].

Internally, Opcenter tracks production through processes and how they are sequenced. A process is an abstract representation of the sequence of operations and steps involved when a specific good is being produced; they serve as an outline that indicates the order in which certain activities should be executed. A process encompasses all operations, tools, machines and materials involved in the producing of a product. Each instance of a process is referred to as a work order [56].

A process is composed of the following elements, listed hierarchically:

- Sub-processes (optional)
- Process operations
- Process steps (optional)

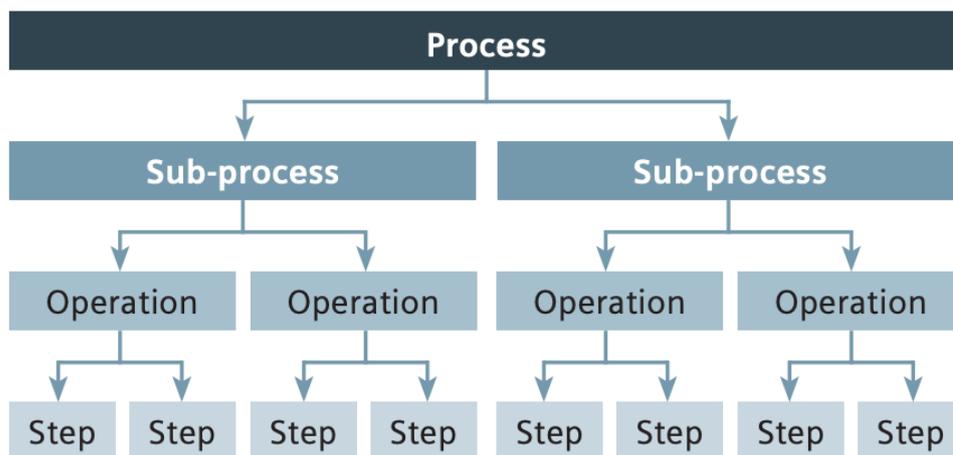


Figure 4.1: Hierarchical Structure Model of a Process

Operations are the building blocks of processes and their most important element. Sub-processes and steps are optional, meaning that the only element that is always present is operations. This means that a work order is always composed of operations at its core, even if it has been subdivided into sub-processes and/or steps.

Figure 4.2 shows the main window Robopac’s users interact with when using Opcenter. It is worth noting that this is not the default main window, since aizoOn has edited it ever since it started working alongside Robopac.

This window is used to select the activity the user wants to carry out. All of these activities are located towards the bottom of the page and are arranged by category. These categories are the ones visible inside the white squares in Figure 4.2 (Robopac, Robopac Landing, Production Coordination, etc.). If the user clicks on one of these white squares, the app automatically scrolls down to the selected category. This

means that clicking on a category button does not redirect the user to another page, it just scrolls down the appropriate amount (see Figure 4.3). The user can also manually scroll down or use the search bar in the top left to manually look for an activity.

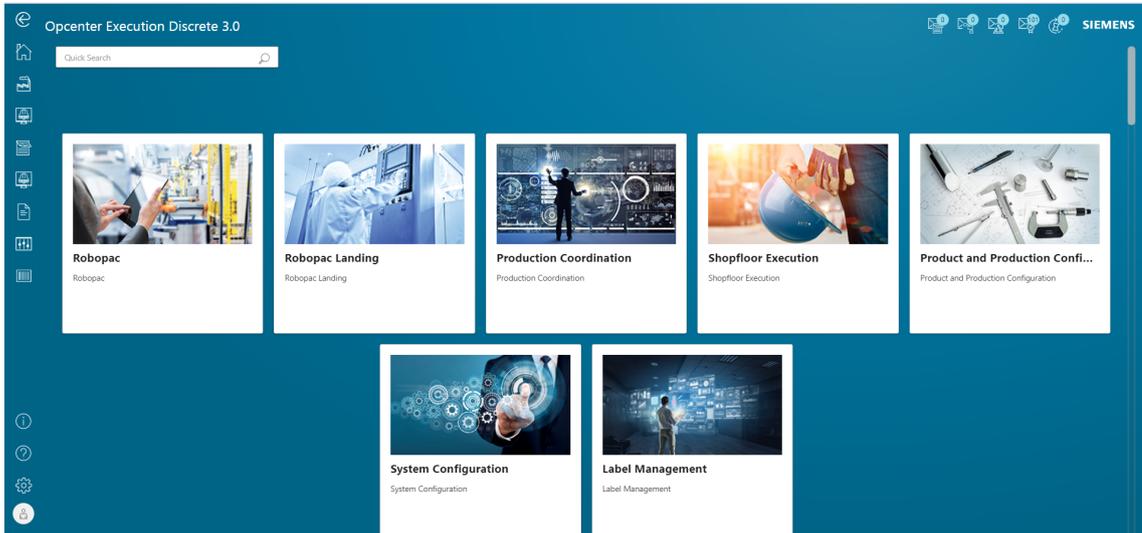


Figure 4.2: Opcenter Execution Discrete Main Window

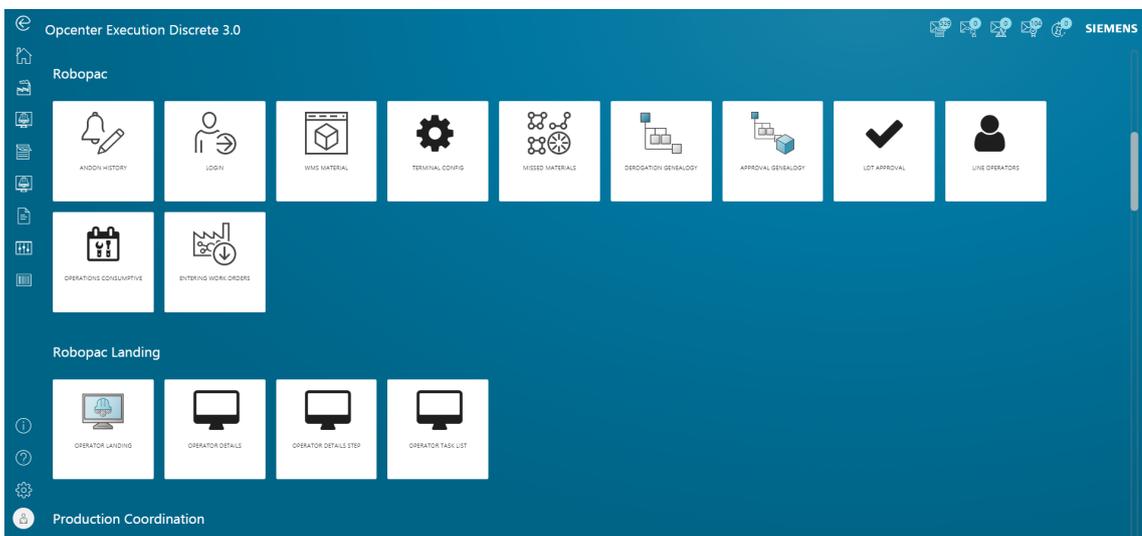


Figure 4.3: Opcenter Execution Discrete Main Window After Scrolling

Figure 4.3 shows some of the activities belonging to the Robopac and Robopac Landing categories. Naturally, clicking on one redirects the user to the corresponding activity's interface. One of the main activities is called Work Orders, shown in Figure 4.4. Its function is to display information about all work orders. Clicking on one of them redirects the user to a site that shows all kinds of information regarding the work order, as well as the operations it is made up of and a diagram that links the operation together (see Figures 4.5, 4.6 and 4.7).

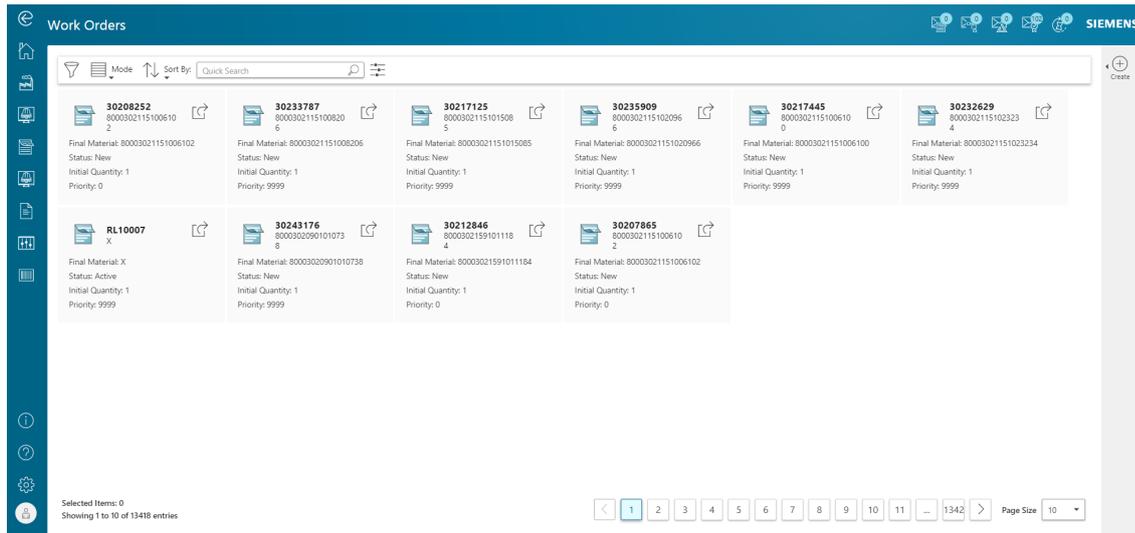


Figure 4.4: Opcenter - Work Orders Window

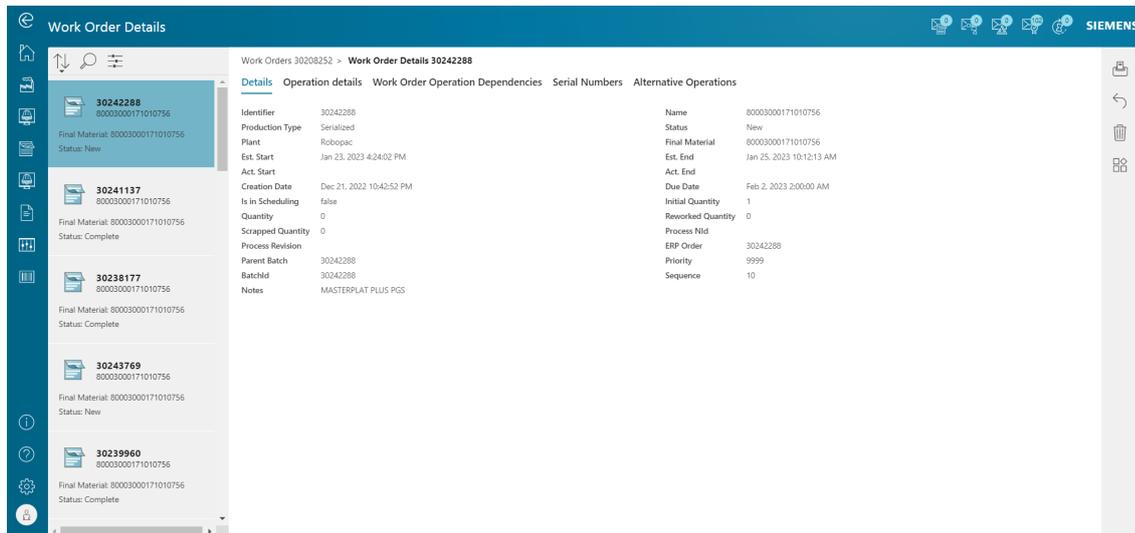


Figure 4.5: Opcenter - Work Order Details

In this activity, the user can access all relevant information about a particular work order. However, the activity does not allow the user to control production in real-time and see how a certain operation advances. This is done in another activity called Operator Landing, the interface of which can be seen in Figure 4.8. Here, the user can start an operation, pause it or even skip it if necessary. Furthermore, the user can also notify the MES system that there have been defects and add notes.

So far, it has been explained how a user can access information about a work order as well as track how far it has been advanced and what the current operation is. If the user is in the Operator Landing Page, they would need to go to the Work Orders activity to visually see what operations have been completed, which are yet to come and how the dependencies work. The dependencies refer to the order in

Work Order Details

Work Orders 30208252 > Work Order Details 30242288

Details Operation details Work Order Operation Dependencies Serial Numbers Alternative Operations

Act. Start	Act. End	Name	Optional	Sequence	Status	Identifier	Type	Is Ready
		SUPERMARKET P...	<input type="checkbox"/>	10	Open	0026684458	OptionalType	<input type="checkbox"/>
		MONTAGGIO PAL...	<input type="checkbox"/>	10	Open	0026684418	Standard	<input checked="" type="checkbox"/>
		BUFFER BASAME...	<input type="checkbox"/>	20	Open	0026684459	Standard	<input type="checkbox"/>
		MONTAGGIO PAL...	<input type="checkbox"/>	20	Open	0026684419	Standard	<input type="checkbox"/>
		PICKING BASAME...	<input type="checkbox"/>	30	Open	0026684451	Standard	<input type="checkbox"/>
		MONTAGGIO CA...	<input type="checkbox"/>	30	Open	0026684455	Standard	<input type="checkbox"/>
		MONTAGGIO PAL...	<input type="checkbox"/>	30	Open	0026684420	Standard	<input type="checkbox"/>
		SMKT MOTORID...	<input type="checkbox"/>	30	Open	0026684467	OptionalType	<input type="checkbox"/>
		MONTAGGIO PAL...	<input type="checkbox"/>	40	Open	0026684421	Standard	<input type="checkbox"/>
		MONTAGGIO BA...	<input type="checkbox"/>	40	Open	0026684452	Standard	<input type="checkbox"/>

Selected Items: 0
Showing 1 to 10 of 51 entries

Page Size: 10

Figure 4.6: Opcenter - Work Order and its Operations

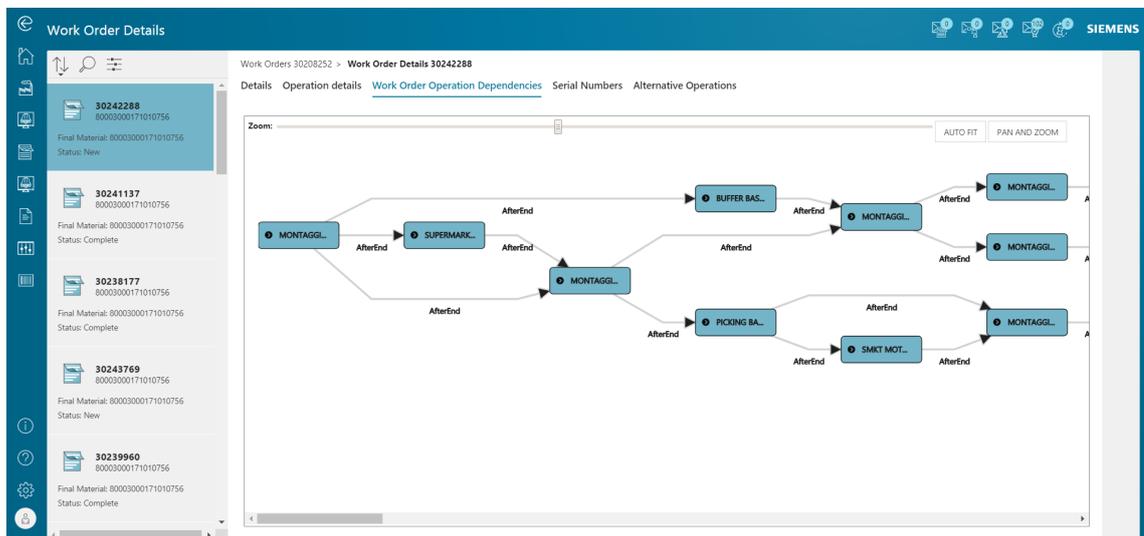


Figure 4.7: Opcenter - Work Order Diagram of Operations

which operations must be executed, as well as what operations must be completed in order to start another one.

Opcenter is a very complete and complex software that allows a company to control much more than what has been covered so far. However, additional features will not be covered in this thesis, since they go beyond its scope and do not help the reader understand why Robopac wanted to extend the Opcenter software and why it was important, as well as what customisations were done.

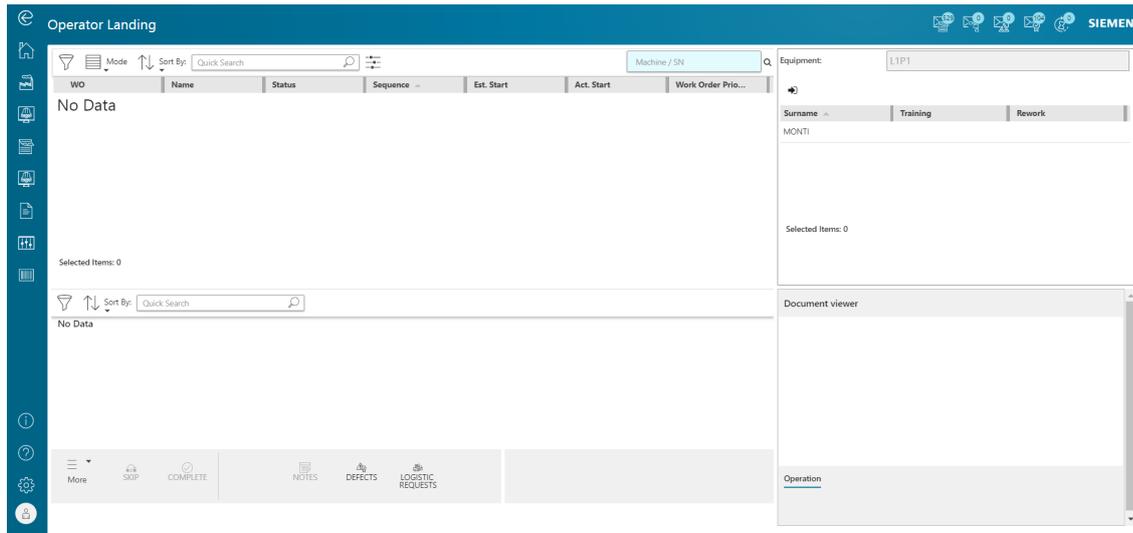


Figure 4.8: Opcenter - Operator Landing Page

4.2 Robopac’s use of Opcenter

As mentioned in subsection 2.3.3 of the Background chapter, Robopac has a manufacturing plant in San Marino that produces and assembles machinery. Their main product is a robot that does packaging operations. Since these robots require the assembling of pieces, their production process falls under the discrete manufacturing category and therefore the Discrete version of Opcenter Execution is used, as previously explained.

The San Marino manufacturing plant has three production lines. Two of them, lines 2 and 3, are not automated. In them, there are workers at each station ready to operate machines and take care of the assembling process. Workers must also insert everything manually into the MES software to keep track of everything, just like Lincotek operators do with Opera MES.

The other line, however, is completely automated. This is Line 1 and it is the kind of production line one would find in a smart factory. In this line, machines communicate directly with the MES software and do not require the worker to insert any data or intervene in the process unless it malfunctions. Robopac has developed a site that shows the status of Line 1 in real-time, which is shown in Figure 4.9. As evidenced there, Line 1 has a lot of stations (machines) that the product must go through. It also displays a timer in the top right corner, which indicates how much time there is left in the current cycle. Each cycle lasts 14 minutes and it is the time each operation has available to complete. After it reaches 0, the products move to the next station. Figure 4.10 shows a part of Line 1 and some of its stations.

Just like one would expect, Line 1 is the most productive line of the manufacturing plant. This is because it has a much higher uptime and because the highly specialised machinery is very efficient at doing its particular job. Furthermore, the

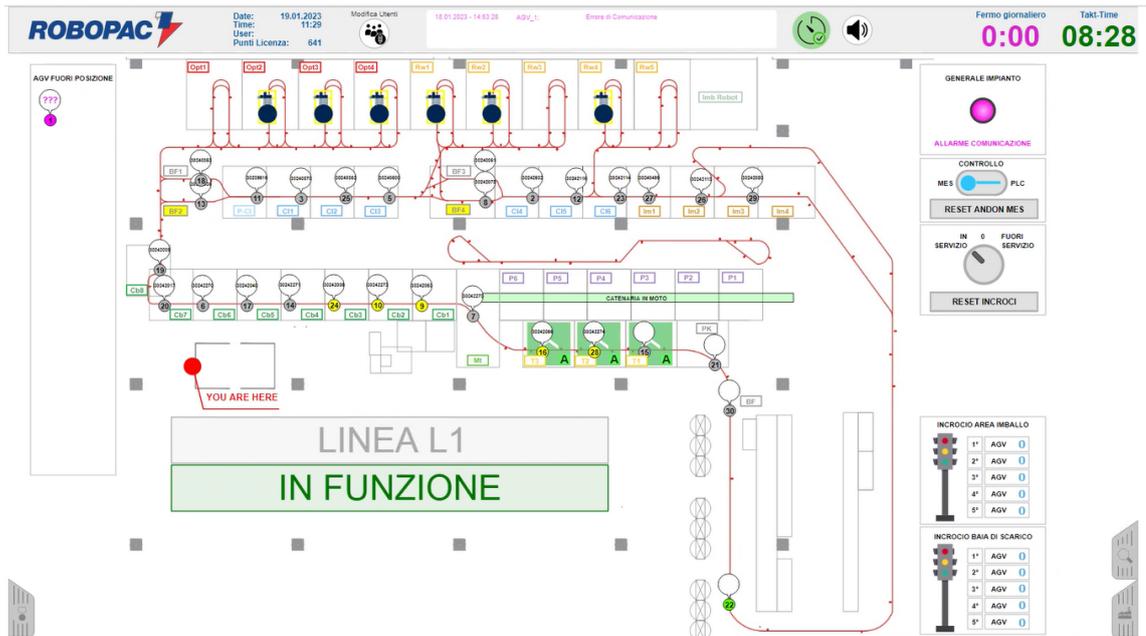


Figure 4.9: Robopac’s Smart Line in Real-Time



Figure 4.10: Robopac’s Smart Line

MES software communicates directly with the machines and vice versa, meaning that no human intervention is needed.

This setup allows operators to focus on overseeing the manufacturing process and ensuring smooth operations. They utilise monitors to access the Opcenter MES software, primarily interacting with the Operator Landing Page (refer to Figure 4.8) during active production. However, this approach presents a practical challenge. Workers find it impractical to access information from other activities, such as the

Work Order Details activity (see Figures 4.5, 4.6 and 4.7). Robopac recognised this as a significant issue since efficient real-time monitoring, a critical aspect of a smart line in a smart factory, was hindered by the inconvenient location of crucial information.

4.3 Novel Contribution: Enhanced Work Order Visual Representation

4.3.1 Objectives

As mentioned above, the way in which certain features were laid out across Opcenter was a problem for Robopac. In particular, they felt that operators were not being able to access all the information they needed in the activity that was almost always on display in the monitors surrounding Line 1 and its machines. Therefore, to assist its operators and the manufacturing process, Robopac requested that the Work Order Diagram of Operations (see Figure 4.7) be visible in the Operator Landing Page.

However, there was also an issue with the diagram itself. Robopac believed the flowchart was not very helpful, since it was poorly organised, dull and provided little information about the work order and the operations it was composed of. They also claimed it lacked a colour scheme that quickly informed operators of the state of each operation and the work order in general. Therefore, Robopac also requested a new diagram that fixed these issues by being more clear, informative and visually appealing.

Consequently, the following two objectives were defined:

1. Integration of the Work Order Diagram of Operations into the Operator Landing Page to improve operator access to critical information.
2. Enhancement of the clarity, informativeness and visual appeal of the work order flowchart to optimise understanding of operations and their statuses.

By modifying the Opcenter software and achieving these two objectives, Line 1 would become more productive and would have less errors due to improved accessibility, information clarity and visual representation of work orders and operations. In order to attain these objectives, the following steps were established:

1. Development of an interface within the Operator Landing Page to display the redesigned diagram.
2. Creation of the diagram, incorporating the required additional information.

The subsequent subsections will explain what were the necessary modifications to accomplish each step.

4.3.2 Implementing the Interface

The default Operator Landing Page contains a lot of information, meaning there is not much room to add more data without crowding everything and making it harder to understand. Figure 4.11 shows it when no order has been selected, and it looks like there is plenty of space. However, when selecting and starting an operation, this changes drastically, as highlighted by figure 4.12.

Clearly, adding the new diagram to this overloaded interface was not a good idea, since it would be far too small to be useful or would cover everything up. Because of this, it was decided to create a retractable panel that could be activated by pressing a button. This solution is further supported by the fact that operators do not require constant visibility of the diagram, but rather only when specific information is needed.

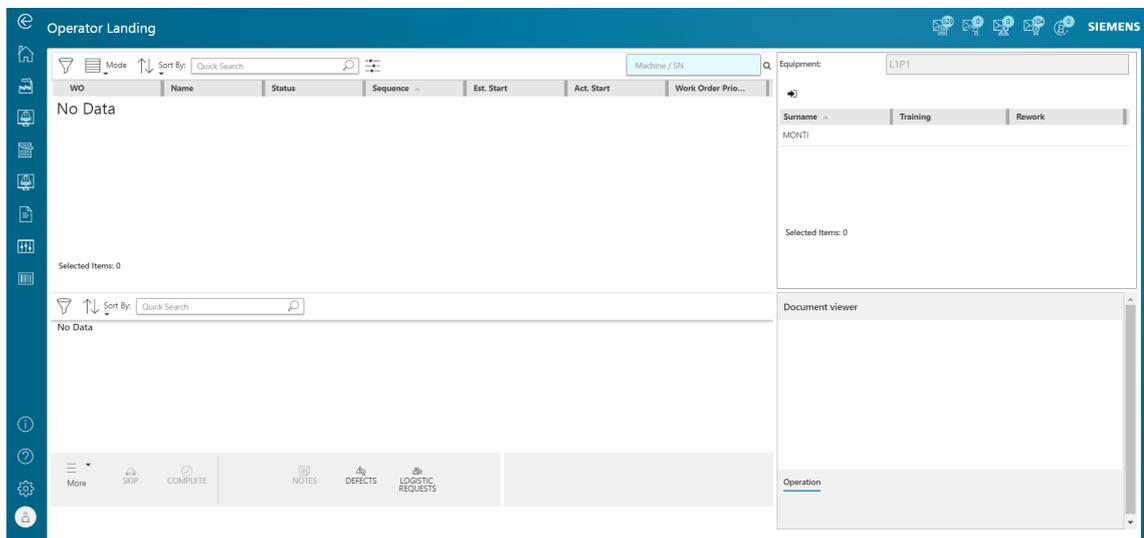


Figure 4.11: Opcenter - Operator Landing Page

The choice was made to add a button to show and hide the panel in the lower bar, next to the already existing *Logistic Requests* button. To do this and many other things, Siemens provides an additional program called Solution Studio. In it, an authorised user can modify activities and their layouts by adding, deleting or rearranging components. Figure 4.13 shows the Solution Studio interface, called Mashup, when modifying the Operator Landing Page interface.

To add the desired button, it is enough to add an additional component between the *Button Bar* and *Send to Rework Area* components located at the bottom of the Mashup. Figure 4.14 exhibits this. Once the button has been added to the Mashup and saved, the Operator Landing Page will display it. Figure 4.15 shows the same situation as Figure 4.12 but with the new button, which was called Layout. Naturally, at this point the button did not do anything.

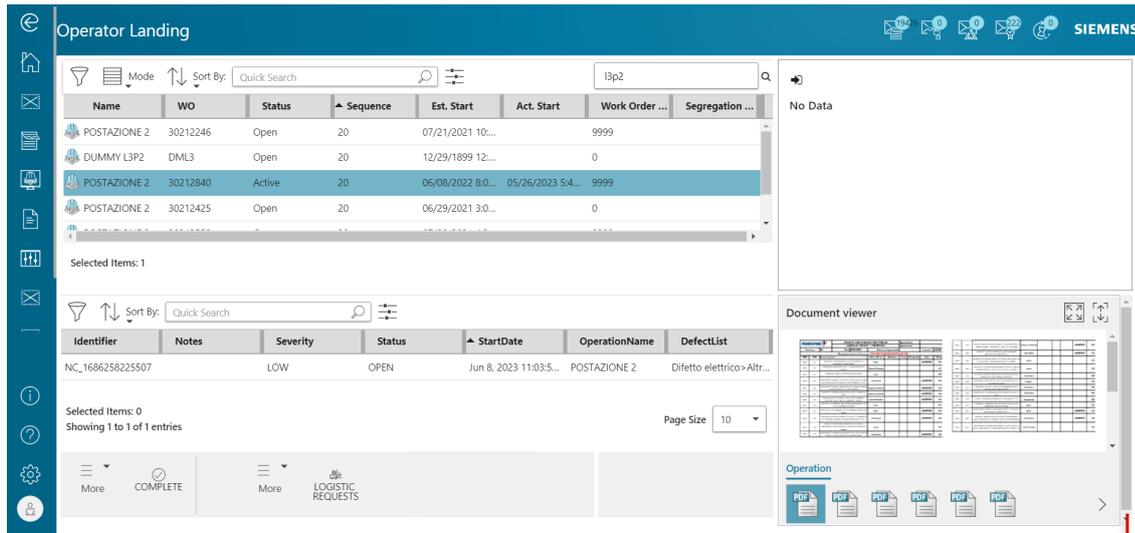


Figure 4.12: Opcenter - Operator Landing Page with Selected Operation



Figure 4.13: Solution Studio - Operator Landing Edit Page

Two JavaScript files were created in the application's frontend to interact with the new Layout button: ShowLayout.js and Layout.js. The former connects the button itself, in the Opcenter interface, with the code that makes the frontend work; it defines its visibility and specifies which functions are executed when it is clicked. In other words, whenever the button is clicked, ShowLayout.js is called. The latter controls the panel, including its contents and dimensions. Furthermore, a file called Treant.js was also added. Its purpose will be explained in the next subsection. The tree on the following page shows the organisation of relevant files within the folder where all extensions to the software are saved.



Figure 4.14: Solution Studio - Operator Landing Page Edit Page with New Component

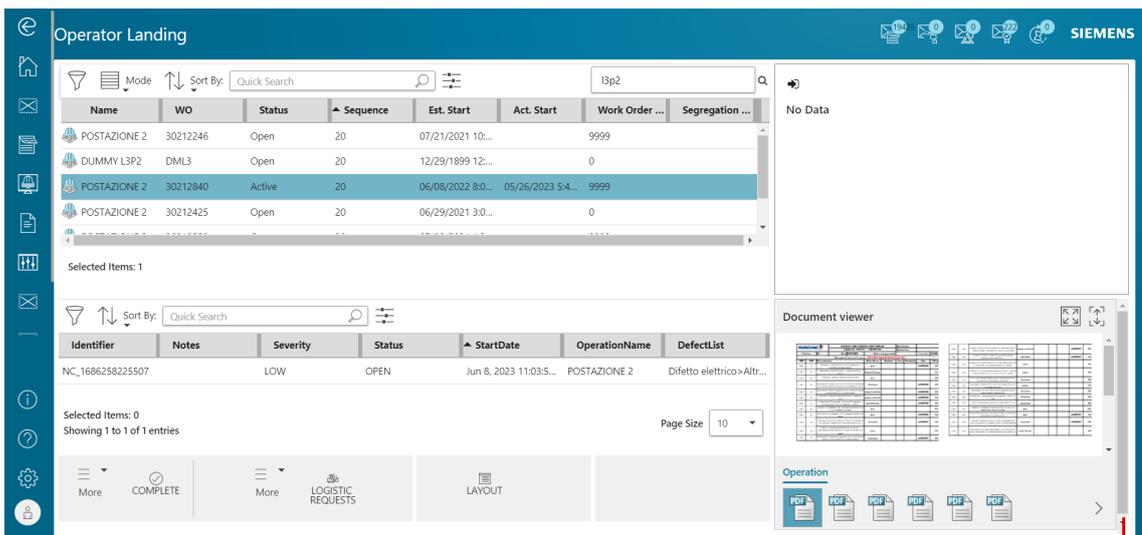


Figure 4.15: Opcenter - Operator Landing Page with Layout Button

```

Siemens.SimaticIT.RobopacExtApp
├── components
│   └── ShowLayout
│       ├── ShowLayout.js
│       └── ShowLayout.html
├── scripts
│   └── Treant.js
├── views
│   └── Layout
│       ├── Layout.js
│       └── Layout.html

```

The complete contents of the ShowLayout.js and Layout.js files will not be displayed in this document as it would not contribute to the reader's comprehension of the modifications. However, relevant and useful portions will be presented. The following code is responsible for connecting the frontend code and the Layout button:

```
function initCommandBar() {
  self.buttonSL = {
    id: "btnBrShowLayout",
    type: "Command",
    name: "ShowLayout",
    unauthorizedBehavior: "show",
    svgIcon: "common/icons/cmdListBox24.svg",
    onClickCallback: showLayoutFunction,
    visibility: true,
    disabled: false,
    align: "left"
  };

  self.buttonBarConfig = {
    barType: "Tool",
    bar: [
      self.buttonSL
    ]
  };
}
```

Essentially, this code serves the purpose of locating the Layout button within Opcenter and establishing a connection to the code. Then, it enables the button's visibility control and associates a callback function to execute when the button is clicked. This function is responsible for triggering the opening of the panel and its code is included on the following page.

The function first checks that the operator has selected an operation. If they have not, then nothing happens. However, if an operation has been selected, then the function gathers the data from the work order and the operation and opens the panel. These data are stored in the global *\$state* variable that is used for this purpose.

In summary, the aforementioned actions resulted in the creation of a user interface button named Layout. When pressed, and if an operation has been selected by the operator, an empty panel is opened. The information regarding the selected operation and the work order it belongs to is stored in a global variable that will be accessed in the following step to generate the diagram and display it inside the panel.

```

function showLayoutFunction() {
  var currentOperation = null;
  var ctx = u4dmSvc.data.cache;

  if (self.selectWoo) {
    if (currentOp) {
      ctx.setCurrentWorkOrderOperation(currentOperation);
    } else {
      currentOperation = ctx.getCurrentWorkOrderOperation();
    }
  }

  if (currentOperation) {
    $state.go(self.layoutState, {
      "wooId": currentOperation.Id,
      "woId": currentOperation.WorkOrder_Id,
      "woNId": currentOperation.WorkOrder.NId
    });
    u4dmSvc.ui.sidePanel.open("e");
  }
}
}

```

4.3.3 Generating the New Diagram

When brainstorming the best way to go about creating the new diagram, two possible approaches were discussed:

- Use the default library provided by Siemens and customise it to add the desired functionalities.
- Search for a new library that supported the features Robopac requested.

After serious consideration, the first approach had to be ruled out. This was because the diagram-generating system created by Siemens was very inflexible and did not allow the level of customisation required to add the necessary features. Additionally, even if it had been more modification-friendly, the amount of necessary changes would have been extensive and very time consuming. Due to all of this, the decision was made to pursue the second approach. Thus, the search for a suitable library began.

After conducting a thorough search and evaluating several libraries, one was identified as the most suitable option: Treant.js. This “is a Javascript library for creating tree structure charts” [57]. To use it, one simply needs to add the Treant.js file, mentioned in the previous subsection, to the project folder. Then, after importing the file, the Treant object has to be initialised with a *chart.config* parameter which contains all of the tree information, either in an array or in JSON format. The following is a simple example:

```
// JSON approach
chart_config = {
  chart: {
    container: "#tree-simple"
  },

  nodeStructure: {
    text: { name: "Parent node" },
    children: [
      {
        text: { name: "First child" }
      },
      {
        text: { name: "Second child" }
      }
    ]
  }
};
```

```
// Array approach
config = {
  container: "#tree-simple"
};
parent_node = {
  text: { name: "Parent node" }
};
first_child = {
  parent: parent_node,
  text: { name: "First child" }
};
second_child = {
  parent: parent_node,
  text: { name: "Second child" }
};
chart_config = [
  config, parent_node,
  first_child, second_child
];
```

```
// Tree creation
var chart = new Treant(chart_config, function() {
  alert("Tree Loaded")
}, $);
```

Figure 4.16 shows the tree that the previous code creates:

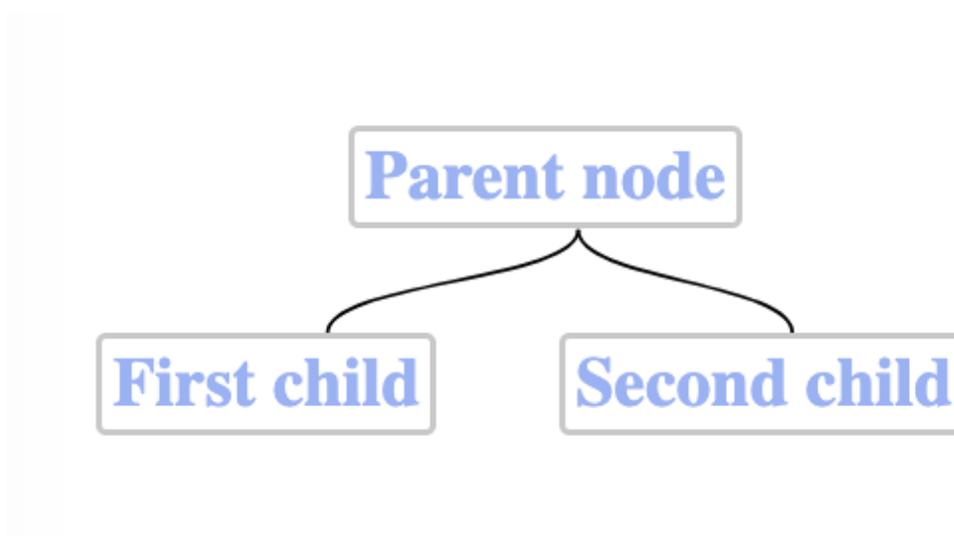


Figure 4.16: Treant.js Basic Example Tree

Treant.js is highly customisable and allows the implementation of all of the features Robopac desired. However, before actually designing the graph, the data that would be used to generate it had to be gathered. In any graph, there are two basic components: nodes and arcs. In the case of a work order graph, the nodes are the operations and the arcs are the dependencies that indicate the order in which these operations must be executed. Therefore, the first step is obtaining the operations of a work order and the order in which they must be executed.

Fortunately, Opcenter already has functions that facilitate the obtaining of these data. This means that the new code, which is all located in `Layout.js`, just had to call said functions to populate the two arrays that were created to store the information, one for operations and the other for dependencies.

Once the arrays had the necessary information, the next step was to get any additional data either requested by Robopac or necessary to generate the graph. This is all done by the function `createGraph`, which in turn invokes other functions to do specific tasks.

The `createGraph` function takes three parameters: the current operation, the operations array and the dependencies array. Essentially, the function takes each operation and obtains some additional data, which is then stored in JSON format. Finally, each operation is saved in a new array called *graph_config*.

```

function createGraph(selectedOperation, operations, dependencies) {
  var graph_config = [];

  operations.forEach((element) => {
    var elementParent = getParent(element, dependencies);
    var nodeColour = obtainColour(element, selectedOperations);
    var duration = obtainDuration(element.EstimatedDuration_Ticks);

    var currentOperation = {
      id: element.Id,
      HTMLclass: nodeColour,
      text: {
        name: element.Name,
        title: "",
        desc: duration,
        datafoo: element.Id
      }
    }

    if (elementParent != "") {
      currentOperation["parent"] = elementParent;
    }

    graph_config.push(currentOperation);
  });

  return transformGraph(graph_config);
}

```

The first step is creating the new empty array. Then, a loop that iterates through all operations begins. Within the loop, several functions are called. The first one is called `getParent` and it is very simple: given an operation, the function returns its parent. The following is the code:

```

function getParent(operation, dependencies) {
  var parent = "";
  dependencies.forEach((dep) => {
    if (dep.ToWOO_Id == operation.Id) {
      parent = dep.FromWOO_Id;
    }
  });
  return parent;
}

```

The second function invoked by `createGraph` is `obtainColour`. As its name implies, it determines the colour that the node should have in the graph. It also gives an orange border to the node that represents the operation that the work order is currently in, which is why it takes a second parameter corresponding to this. The colour scheme that Robopac desired was based on the status of the operation, so this is what the function uses to determine the correct colour. The code is the following:

```
function obtainColour(operation, selectedOperation) {
  var colour = "";

  if (operation.Status["StatusNId"] == "Open") {
    colour = "green";
  }
  else if (operation.Status["StatusNId"] == "Active") {
    colour = "blue";
  }
  else if (operation.Status["StatusNId"] == "Partial") {
    colour = "yellow";
  }
  else if (operation.Status["StatusNId"] == "NotExecuted") {
    colour = "red";
  }
  else {
    colour = "gray";
  }

  if (operation.Id == selectedOperation) {
    colour += "-border selected";
  }

  return colour;
}
```

Operations have a field that stores the duration of said operation in a unit called ticks; one tick equals one hundred nanoseconds. The job of the third function, `obtainDuration`, is to convert ticks into hours, minutes and seconds. Its code, as well as the code of the function it calls to create a more readable output, is included on the following page.

After `obtainDuration` returns its value, all necessary information is available. The next step is to put it all together in a single variable, `currentOperation`, which is then added to the `graph.config` array. The colour must be saved into the `HTMLclass` attribute because the only way to colour a node with `Treant.js` is to do it through the HTML file, which will be shown shortly. After every operation has been added to the array, the output of the `transformGraph` function is returned. This function takes the array that was just created and transforms it into a format that the `Treant.js` library can read and convert into a tree.

```
function obtainDuration(ticks) {
  var totalSeconds = ticks / Math.pow(10, 7);
  var hours = Math.floor(totalSeconds / 3600);
  var minutes = Math.floor((totalSeconds / 60) % 60);
  var seconds = totalSeconds % 60;

  var result = pad2(hours) + ":" + pad2(minutes) + ":" + pad2(seconds);
}

function pad2(number) {
  number = "0" + number;
  return number.substr(number.length - 2);
}
```

As previously mentioned, there are two approaches to represent the information that is used by the library to initialise the tree: array format and JSON format. In the former, each node is stored into a separate variable. In order to identify the parent node of a given node, the name of the variable where the parent is stored must be provided. However, due to the dynamic nature of the project, where the number of operations is not known beforehand (since it varies depending on the work order), the array approach was not suitable. Therefore, the JSON approach was deemed to be the only viable option, since it allows more flexibility when dealing with dynamic data.

The `transformGraph` function first iterates through each of the items it receives as parameters, which are the operations with their associated data, and saves each one with a format that complies with the JSON approach into a new dictionary.

After this, the output dictionary is modified to keep track of a node's children. In the JSON approach, all nodes that are children of a parent node are part of an array saved into the *children* field of the parent. The code included on the following page shows these two first steps, where the *output* dictionary is created and then the *children* field is added to the nodes that need it.

```

function transformGraph(graph) {
  var output = {};
  graph.forEach((elem) => {
    output[elem.id] = {
      id: elem.id,
      HTMLclass: elem.HTMLclass,
      text: {
        name: elem.text["name"],
        title: elem.text["title"],
        desc: "Durata Stimata: " + elem.text["desc"],
        datafoo: elem.text["datafoo"]
      }
    }
    if ("parent" in elem) {
      output[elem.id]["parent"] = elem.parent;
    }
  });

  var nodeStructure = "";

  for (var op in output) {
    var entry = output[op];
    var parent = output[output[op].parent];

    if (parent == undefined) {
      nodeStructure = entry;
    }

    else {
      if (!(children in parent)) {
        parent["children"] = [];
      }
      parent["children"].push(entry);
    }
  }
}

```

The result of this, as previously mentioned, is that all nodes have a field called *children* that contains the JSON objects containing the information of their children. The variable `nodeStructure` stores the first node (i.e., the entry node to the entire graph).

The last step is to create the JSON object that defines some of the settings for the generation of the graph. Then, this new object is returned alongside the `nodeStructure` variable that stores the entire tree. The following is the code:

```

var config = {
  container: "#layout",
  connectors: {
    type: "step"
  },
  node: {
    HTMLclass: "nodeExample1"
  },
  rootOrientation: "WEST",
  callback: {
    onTreeLoaded: function () {
      const $oNodes.on("click", function (oEvent) {
        const $oNode = $(this);

        var opId = $oNode[0].children[3].innerText;
        getEquipment(opId).then(function (eq) {
          $oNode[0].children[1].innerText = eq;
        });
      });
    }
  }
}

var chart_config = [
  config,
  nodeStructure
]

return chart_config;
}

```

A callback function was implemented to handle the event when a node in the diagram is clicked. This callback function retrieves the equipment associated to operation that the node represents and then displays it. This approach was selected for efficiency reasons, since retrieving the equipment of a given operation must be done through a query. Due to this, triggering the query for every operation when building the diagram would have been inefficient. This approach improves the performance of the system.

Finally, all that remains is calling the createGraph function that will construct the graph by invoking everything else. Also, and since many operations can be part of a single work order, a horizontal scrollbar was set up.

Subsequently, the interface implemented in subsection 4.3.2 displayed the Work Order Diagram of Operations. The remaining relevant code resides in the Layout.HTML file, specifically responsible for modifying the colour of a node and setting its border to orange when necessary. A section of this HTML file is included on the following page, as well as the aforementioned call to the createGraph function.

```

vm.chart_config = createGraph(workOrderOperationId, vm.allOperations,
                               vm.dependencies);

var chart = new Treant(vm.chart_config, function () {
  var opSelected = document.getElementsByClassName("selected");
  var graphElement = document.getElementById("layout");

  var selectedOffset = opSelected[0].offsetLeft;

  if (selectedOffset > 1200) {
    graphElement.scrollLeft += selectedOffset - 600;
  }
}, $);

```

```

<style>
  .green {
    background-colour: #3FCA67;
  }
  .green.border {
    background-colour: #3FCA67;
    border: 3px solid #F3962F;
  }
</style>

```

The new Work Order Diagram of Operations is depicted in Figure 4.17, showcasing the enhanced colour scheme and improved visual elements. When a node is clicked, additional information is shown, as highlighted in Figure 4.18. Finally, Figure 4.19 shows the diagram of a different work order, showcasing some of the other node colours.

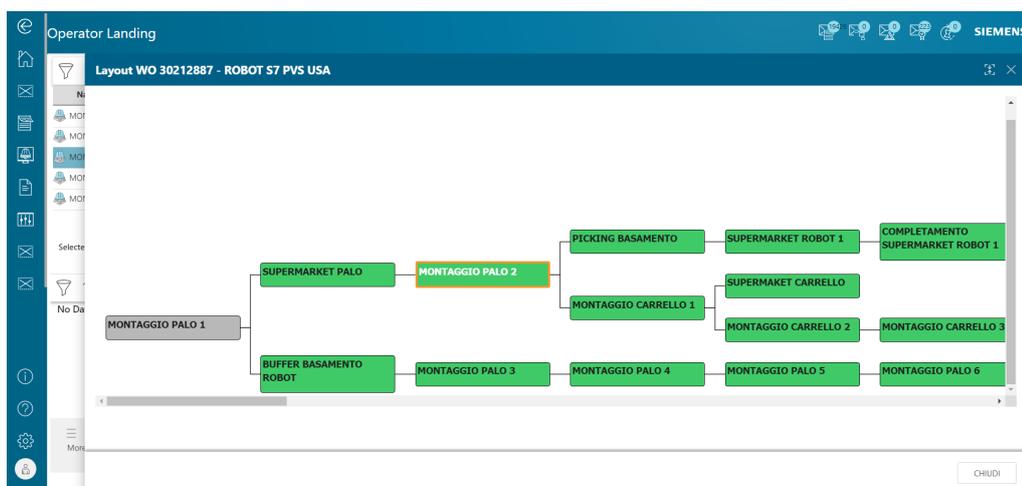


Figure 4.17: Updated Work Order Diagram of Operations

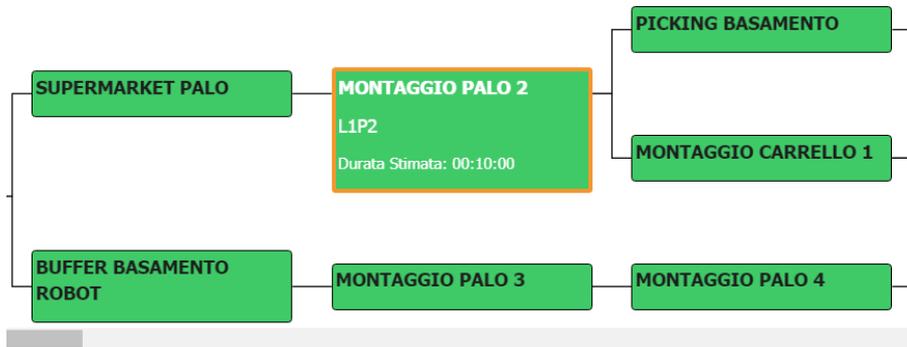


Figure 4.18: Updated Work Order Diagram of Operations when Node is Clicked

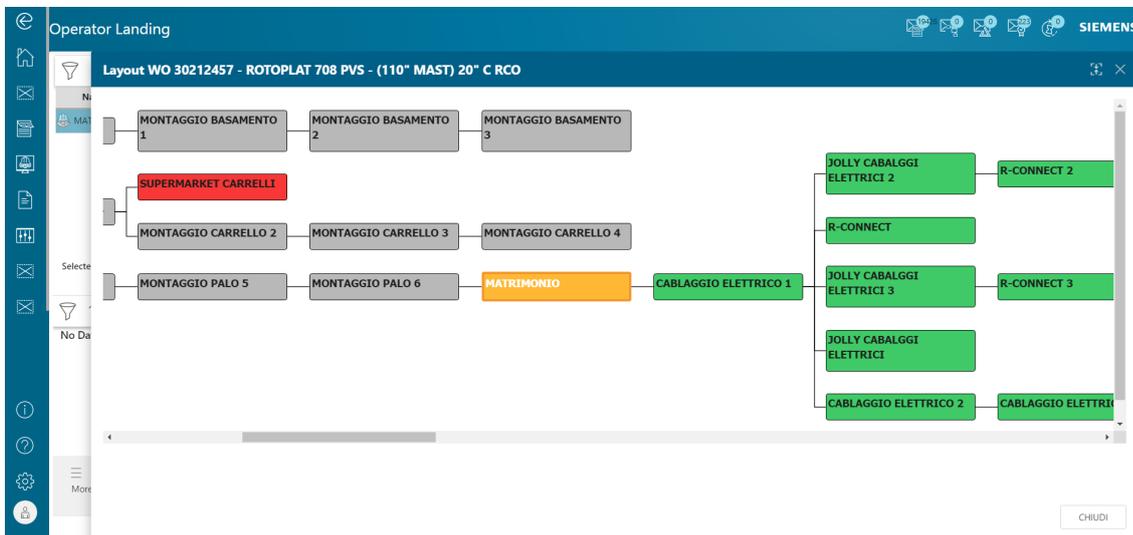


Figure 4.19: Another Example of the Updated Work Order Diagram of Operations

4.3.4 Final Comments

This denotes the completion of the most important modifications done to Siemens Opcenter Execution Discrete to satisfy Robopac’s desire to have a more convenient, practical and aesthetically appealing way of visualising the status of a work order through the Work Order Diagram of Operations. Robopac was very pleased with the outcome.

The next chapter, Chapter 5, will further discuss and delve deeper into the value these modifications provided both to Robopac and its workers.

Discussion

This chapter will analyse what the actual added value was in each of the two projects by comparing the initial situation to the final one. Additionally, the two MES software programs will then be compared, highlighting advantages and disadvantages of each one in the context of smart factories. The intention is to evaluate strengths and weaknesses of each software in relation to their applicability and effectiveness in this Industry 4.0 world.

5.1 Added Value

5.1.1 Lincotek and Opera MES

Before the project, there was a big problem regarding data transfer between Opera MES and the Airflow machines, which are part of the manufacturing process of some work orders. The issue was that all communication between these two entities had to be done manually by operators, meaning that data transfer was both inefficient and subject to human error. Figure 5.1 visually represents this.

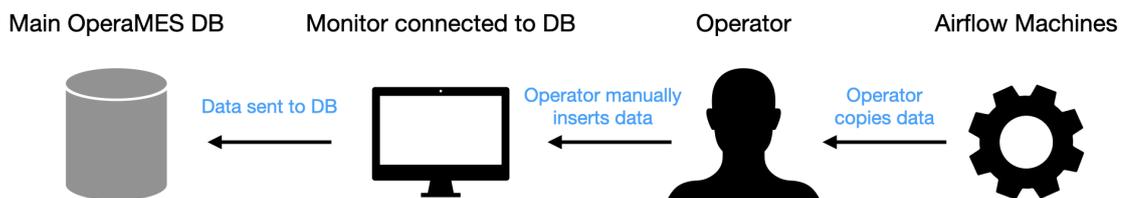


Figure 5.1: Opera MES - Original Flow of Data Exchange

The data, generated after running quality control tests, were crucial to Lincotek since they were used, among data from other machines and tests, to generate a document called Certificate of Conformity (COC), which reports to clients the results of quality tests. Essentially, this document serves as a guarantee that the manufacturing process is living up to the standards set by customers. Due to this, it is evident that ensuring the integrity of the data is of utmost importance.

To achieve this goal, a new architecture was devised. The new system permitted the direct communication and data transfer between the software that controlled and kept track of everything, Opera MES, and Airflow machines. By doing this, data integrity is assured and operators are relieved of the uncomfortable task of manually copying data from one machine to another. Figures 5.2 and 5.3 visually represent the new flow, including the stored procedures that trigger data exchange.

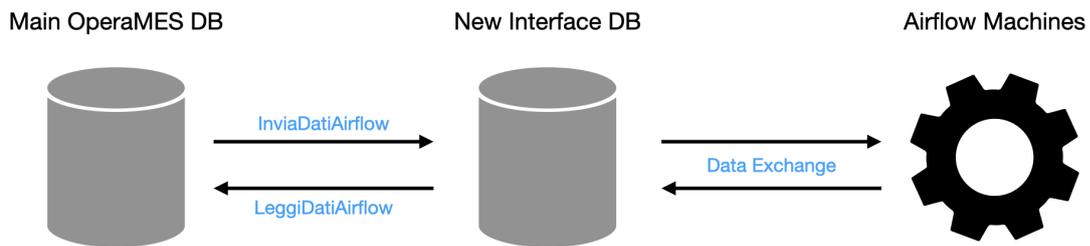


Figure 5.2: Opera MES - New Flow of Data Exchange

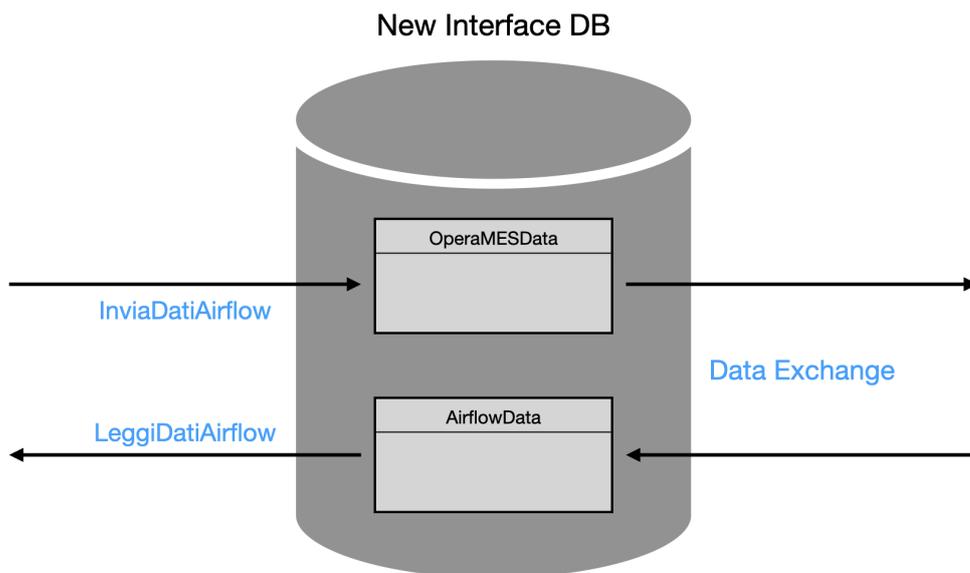


Figure 5.3: Opera MES - Zoom in on the New Flow of Data Exchange

5.1.2 Robopac and Opcenter

Siemens Opcenter Discrete, the MES software used by Robopac in their San Marino manufacturing plant, offers its users a diagram to visualise the flow of operations that make up any given work order. However, Robopac believed this diagram to be dull, counter-intuitive and even uninformative, as well as poorly placed within the software. This was all a problem, since the software was failing to aid the comprehension of operators in this particular aspect.

The project presented in this thesis devised and introduced a new diagram to address this issue. The new diagram offers enhanced clarity, intuitiveness and displays a greater amount of information, as well as being located in a more convenient interface of the software. Now operators do not need to leave the interface they use to interact with work orders to get information about the progression of a given work order, which is what they had to do previously. Furthermore, a quick glance at the new diagram provides them with more information that careful inspection of the previous one would have. Finally, the information is not only presented with improved clarity but also more detail is included.

Both diagrams, old and new, are shown in Figures 5.4 and 5.5. It is clear that the new one has a cleaner design and facilitates operator comprehension with its colour scheme.

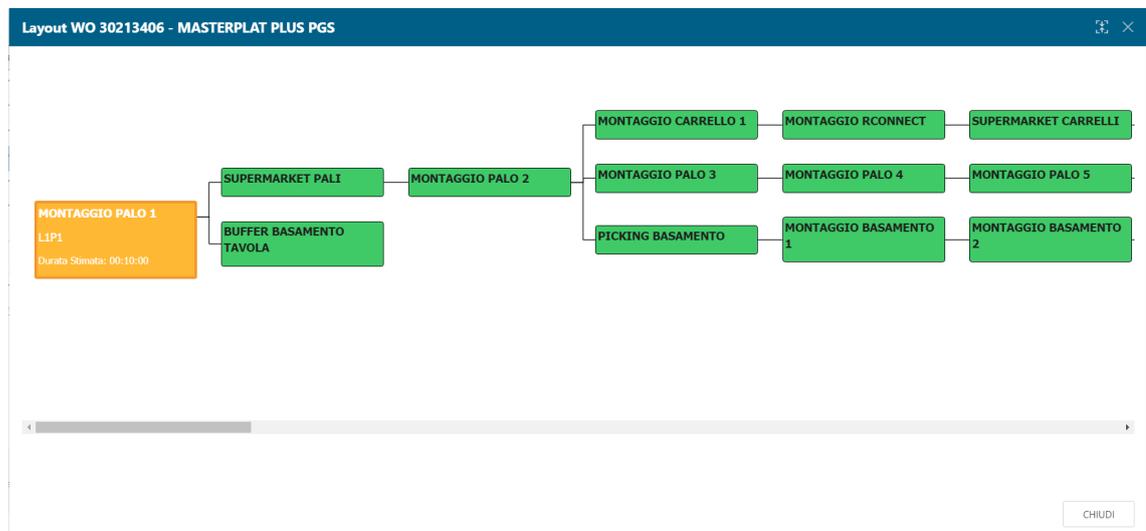


Figure 5.4: Opcenter - New Work Order Diagram of Operations

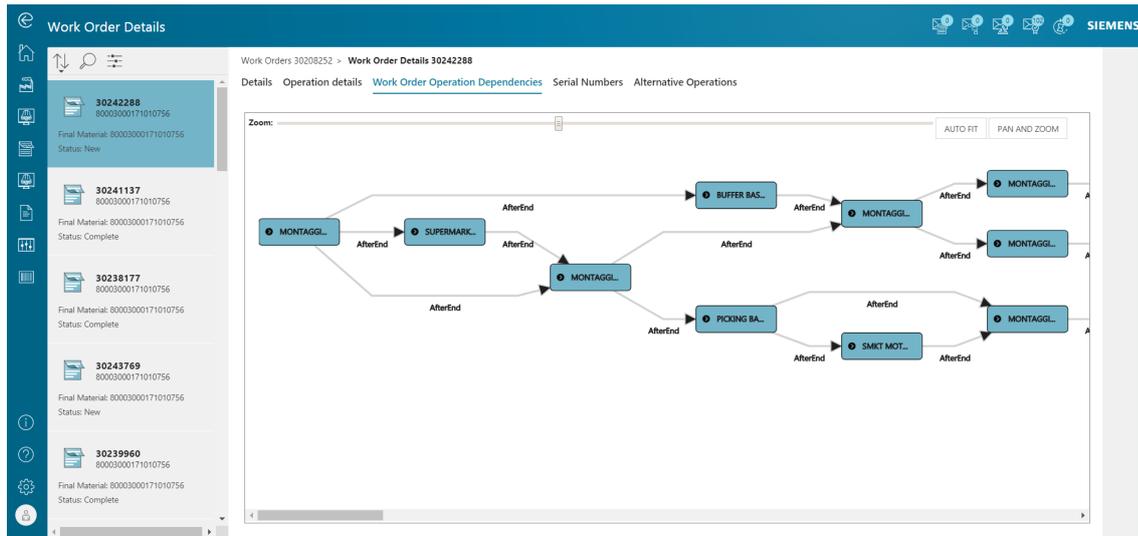


Figure 5.5: Opcenter - Old Work Order Diagram of Operations

5.2 Comparing Opera MES and Opcenter

Opera MES and Siemens Opcenter are two MES software programs that aim to keep track of everything production-related in a manufacturing plant, thus providing the company with a lot of information they would not have otherwise. Ideally, MES software should also facilitate the functioning of smart factories by communicating directly with machinery.

Both Opera MES and Opcenter offer capabilities for managing all operations in the manufacturing process, but they differ in terms of features and flexibility.

Opera MES offers a very user-friendly interface that is highly customisable and easily adaptable to customer needs, thus enabling a company to tailor the software to their specific requirements and streamline their operations effectively. From a programming perspective, Opera MES offers the advantage of easy modification and quick implementation and testing of new features. This is because, unlike other systems or programs, Opera MES does not require rebuilding the entire project for each change, thus facilitating rapid testing without significant downtime. This flexibility allows developers to evaluate changes to the software quickly and efficiently.

However, this flexibility can come at a cost. A potential downside of the Opera MES approach is that there is an increased risk of introducing bugs or errors, since the implementation can be very quick and there is not much revision of the code. Without the rebuilding process, there is a higher probability of unintended and unexpected code behaviour. Due to this, the developer has to be careful to thoroughly test any new features they might be implementing, since no safety net is present.

On the other hand, Siemens Opcenter is a more robust system that uses a more traditional rebuilding process to push any new changes. This means that flexibility

is reduced and testing times drastically increase, especially due to the downtime associated with the entire building process. However, this process has additional filters and barriers, meaning that the code is checked much more thoroughly before actually being added to the program. This approach is a lot more inefficient whenever a small change needs to be implemented or whenever a small error needs addressing, since checking whether it was solved can be very time-consuming. On the other hand, it also provides more safety and is more resilient, since it is able to deal with any potential human error in a better way, preventing these errors from being integrated.

Another significant difference between the two software programs is the way in which workflows are actually created and modified. In Opera MES, everything is done through SQL, as described in Chapter 3. This means that all interaction with the database is very direct, enabling more flexibility when managing and manipulating data. Additionally, SQL is built to interact with databases, so the developer can make the most of the language and retrieve data efficiently. However, SQL does not offer the same features that languages like JavaScript do, meaning that workflows in Opera MES might be slightly limited due to this.

In contrast, Opcenter manages workflows through JavaScript and C#. For this thesis, and due to the nature of the required modifications, only the former was used, as discussed in Chapter 4. Using these two programming languages allows for great flexibility since they are highly versatile and offer extensive libraries for development, like the Treant.js library used for this project. Moreover, these languages provide easy and seamless integration with other technologies, meaning that communication with other systems is effortless. However, using JavaScript and C# means that interaction with the database is done in indirect fashion, adding an additional layer of complexity that might impact performance. Also, performance may be impacted further since these are higher level languages that entail additional processing when compared to direct SQL interactions.

The work done for this thesis showcases some of the strengths of both Opera MES and Opcenter. The Lincotek project showed the advantages Opera MES has when connecting directly with a database due to the fact that workflows are created on SQL, facilitating the implementation of the new architecture that ensured efficient data collection and storage. Additionally, the ease with which a new workflow was created depicts the software's flexibility. These are all desirable qualities for a software program that controls a smart factory, since the company can tailor the process to their needs to streamline production.

On the other hand, the Robopac project illustrated the advantages of working with languages such as JavaScript, which offer seamless integration with countless libraries. This capability greatly enhances the extensibility of a software program because it massively increases the features that can be implemented while minimizing the work needed to do so. So, even if Opcenter is slightly less flexible than

Opera MES in terms of workflow creation and editing, the nature of the programming languages makes up for it. Once again, this is a desirable quality for a software program that interacts with a smart factory environment.

In conclusion, both MES software programs have advantages and disadvantages both to the manufacturing company and the developers implementing any potential required modifications. Both systems are adaptable and have good scalability, meaning they would be a good fit for small, medium or even large manufacturing operations. There is no right or wrong answer when asking which of the two programs is better, and which one should be used ultimately comes down to the requirements of the manufacturing process and the company.

Conclusions

In the rapidly evolving landscape of manufacturing, manufacturing execution systems (MES) software has assumed an increasingly leading role. This software plays a crucial part in facilitating production planning, tracking and control, monitoring the transformation of raw materials into finished products. This has made MES indispensable in the era of Industry 4.0. As smart factories continue to emerge, MES software will continue to be a key component to ensure efficient operations. Due to all of this, this thesis was focused on implementing functionalities to extend two different MES software programs, as requested by two distinct companies, Lincotek and Robopac, as they navigate the evolving demands of modern manufacturing in the context of Industry 4.0.

The initial phase of the project was centred on addressing Lincotek's need for an efficient data exchange process with Airflow machines, which are involved in their manufacturing process. Prior to the project, data transfer was performed manually by operators. To solve this, a novel architecture was devised and implemented. The solution involved the development of a dedicated database specifically designed to interact with Airflow machines. Furthermore, a new workflow was integrated into the Opera MES software, establishing a connection between the new database and the primary database, thus enabling efficient data exchange.

Following the successful implementation of the above, data exchange between Opera MES and Airflow machines has become automatic and seamless. This has eliminated the need for manual copying by operators into monitors, thus enabling integration of Airflow machine data into Opera MES. As a result, operators can now use their time more efficiently whilst data accuracy is ensured.

The second phase of the project consisted on redesigning the default diagram available in Opcenter to show the flow of operations that make up a work order. Robopac identified limitations in the existing diagram, which lacked sufficient information and also presented it in an unintuitive way, thus hindering the ability of an operator to quickly understand the current stage and production status. Moreover, the diagram was located in an inconvenient interface, making it difficult for operators to access the information swiftly.

To solve these issues, a new visually appealing and informative diagram was developed using a JavaScript library, taking advantage of the extensive range of libraries

available in the language. The diagram was placed in a convenient user-friendly panel within the interface that operators primarily interact with when overseeing the manufacturing process. With the integration of the implemented features, operators can now quickly access and interpret information regarding the status of operations within a work order. The novel diagram not only provides a clearer visual representation but also incorporates new data, allowing workers to plan the manufacturing process more effectively.

The work done in this thesis for Lincotek and Robopac shows the versatile and flexible nature of MES software. The ability to successfully extend both programs, Opera MES and Opcenter, demonstrates the great adaptability that MES software has, as it can keep up with the ever evolving needs of manufacturing environments. Because of this quality, managers can effectively improve the production processes they oversee by tailoring the MES software to their operators' needs and requirements, thus optimising manufacturing.

To effectively adapt MES software to a firm's needs, company collaboration is a very valuable tool. The partnership Lincotek and Robopac have with aizoOn allows them to swiftly evolve their MES software by implementing new functionalities that streamline their manufacturing processes. The partnership is also very beneficial for aizoOn, since the specific requirements of real-world clients provide valuable insight into the manufacturing world and developers are faced with practical challenges, thus improving the service the company provides its clients.

Furthermore, this thesis illustrates the importance of customising the manufacturing process to drive the development and advancement of smart factories. This is because, as stated in Chapter 2, a smart factory requires flexible and adaptive production processes in a world of increasing complexity [32]. The focus of this thesis of continuously updating and tailoring MES software to align with a company's manufacturing process plays a key role in this adaptation, thereby serving as an essential step towards establishing a smart factory.

The fine-tuning of MES software enables the enhancement of the manufacturing process and the increase in operational efficiency, thus catalysing the transformation of a production plant into a smart factory. The ability to continuously adapt and optimise processes through technology integration and automation empowers a company to achieve new levels of agility and responsiveness, aiding it to be at the forefront.

The impact of this thesis goes beyond the specific implemented features, either MES software program and the involved companies. The most important lesson from the entire project is the importance of continuously improving and optimising the production process to enable the advancement of smart manufacturing practices. This mindset of ongoing betterment is relevant not only in the manufacturing industry but also in various aspects of professional and personal life. The principle of constant improvement transcends the boundaries of this thesis and serves as a reminder of the

importance of striving for growth and development in all aspects of life. Embracing this mindset propels a company or individual to always push and work towards excellence.

In summary, this thesis has consisted of the extension of two MES software programs through the implementation of various functionalities. The Lincotek and Robopac projects have served as examples of how MES software can be enhanced to meet specific needs: The former upgraded data exchange capabilities while the latter improved visual appeal and representation of information regarding work orders. Through these modifications, the versatile and adaptable nature of MES software has been showcased, which allows it to adjust to the evolving requirements of the fast-paced manufacturing world. Due to this, managers can swiftly address the needs and demands of their operators to continuously improve production processes.

Furthermore, this thesis has highlighted and justified the importance of continuous improvement and process optimisation in smart factories. The integration of emerging technologies and the refinement of production processes are key elements that empower a company to transform their production plants into efficient smart factories.

This mindset of continuous improvement is an important lesson that this thesis carries that goes beyond the implemented features and the software programs used. A culture of perpetual betterment and striving for development is an important concept in several facets of professional and personal life which unlocks the potential for success and growth.

To conclude this thesis, the profound impact of MES software in the process of transforming a production plant into a smart factory should be restated. Additionally, it is essential to have a mindset of ongoing betterment to tailor the software to the needs of managers and operators, thus improving and optimising the manufacturing process. As industries embrace digitalisation and automation, the experiences shared in this thesis can serve as a valuable guideline to companies undertaking the journey of transforming their manufacturing plants into smart factories.

Bibliography

- [1] Y. Ersoy, “The advantages and barriers in implementing of industry 4.0 and key features of industry 4.0,” *The Journal of International Scientific Researches*, vol. 7, pp. 207–214, 10 2022.
- [2] M. Nardo, D. Forino, and T. Murino, “The evolution of man–machine interaction: the role of human in industry 4.0 paradigm,” *Production & Manufacturing Research*, vol. 8, no. 1, pp. 20–34, 2020.
- [3] S. Mantravadi and C. Møller, “An overview of next-generation manufacturing execution systems: How important is mes for industry 4.0?,” *Procedia Manufacturing*, vol. 30, pp. 588–595, 2019. Digital Manufacturing Transforming Industry Towards Sustainable Growth.
- [4] M. Hermann, T. Pentek, and B. Otto, “Design principles for industrie 4.0 scenarios,” in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pp. 3928–3937, 2016.
- [5] A. Shojaeinasab, T. Charter, M. Jalayer, M. Khadivi, O. Ogunfowora, N. Raiyani, M. Yaghoubi, and H. Najjaran, “Intelligent manufacturing execution systems: A systematic review,” *Journal of Manufacturing Systems*, vol. 62, pp. 503–522, 2022.
- [6] S. Mantravadi, C. Li, and C. Møller, “Multi-agent manufacturing execution system (mes): Concept, architecture ml algorithm for a smart factory case,” pp. 477–482, 01 2019.
- [7] M. McClellan, *Applying Manufacturing Execution Systems*. CRC Press, 1997.
- [8] L. Mayer, N. Mehdiyev, and P. Fettke, “Manufacturing execution systems driven process analytics: A case study from individual manufacturing,” *Procedia CIRP*, vol. 97, pp. 284–289, 2021. 8th CIRP Conference of Assembly Technology and Systems.
- [9] MESCenter, “Mes - manufacturing execution system.” <http://mescenter.org/en/articles/108-mes-manufacturing-execution-system>. Accessed: 2022-12-15.
- [10] I. Wright, “What is industry 4.0, anyway?.” <https://www.engineering.com/story/what-is-industry-40>. Accessed: 2022-12-16.
- [11] J. Mokyr and R. H. Strotz, “The second industrial revolution, 1870-1914,” *Storia dell’economia Mondiale*, vol. 21945, no. 1, 1998.
- [12] A. Atkeson and P. J. Kehoe, “The transition to a new economy after the second industrial revolution,” 2001.
- [13] P. Magal, “Timeline of revolutions.” <https://manufacturingdata.io/newsroom/timeline-of-revolutions/>. Accessed: 2022-12-16.

-
- [14] P. Troxler, “Making the 3rd industrial revolution,” *Fab Labs: Of Machines, Makers and Inventors*, Transcript Publishers, Bielefeld, 2013.
- [15] C. Roser, “The four industrial revolutions.” <https://www.allaboutlean.com/industry-4-0/>. Accessed: 2022-12-16.
- [16] S. Muhammad, Y. Pan, C. Magazzino, Y. Luo, and M. Waqas, “The fourth industrial revolution and environmental efficiency: The role of fintech industry,” *Journal of Cleaner Production*, vol. 381, p. 135196, 2022.
- [17] M. Shahbaz, M. A. Nasir, E. Hille, and M. K. Mahalik, “Uk’s net-zero carbon emissions target: Investigating the potential role of economic growth, financial development, and rd expenditures based on historical data (1870–2017),” *Technological Forecasting and Social Change*, vol. 161, p. 120255, 2020.
- [18] M. Ghobakhloo, “Industry 4.0, digitization, and opportunities for sustainability,” *Journal of Cleaner Production*, vol. 252, p. 119869, 2020.
- [19] S. Corfe, “4ir and the environment: How the fourth industrial revolution can curb air pollution and decarbonise the economy,” *Social Market Foundation*, 2020.
- [20] S. S. Kamble, A. Gunasekaran, and S. A. Gawankar, “Sustainable industry 4.0 framework: A systematic literature review identifying the current trends and future perspectives,” *Process Safety and Environmental Protection*, vol. 117, pp. 408–425, 2018.
- [21] M. Hermann, T. Pentek, B. Otto, *et al.*, “Design principles for industrie 4.0 scenarios: a literature review,” *Technische Universität Dortmund, Dortmund*, vol. 45, 2015.
- [22] X. Yao, H. Jin, and J. Zhang, “Towards a wisdom manufacturing vision,” *International Journal of Computer Integrated Manufacturing*, vol. 28, no. 12, pp. 1291–1312, 2015.
- [23] M. M. Mabkhot, A. M. Al-Ahmari, B. Salah, and H. Alkhalefah, “Requirements of the smart factory system: A survey and perspective,” *Machines*, vol. 6, no. 2, 2018.
- [24] P. Taylor, “Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025.” <https://www.statista.com/statistics/871513/worldwide-data-created/>. Accessed: 2022-12-20.
- [25] X. Xu, “From cloud computing to cloud manufacturing,” *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75–86, 2012.
- [26] K. Rose, S. Eldridge, and L. Chapin, “The internet of things: An overview,” *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.
- [27] A. A. Author, B. B. Author, and C. Author, “Supply chain management and industry 4.0: A theoretical approach,” *Brazilian Journal of Operations and Production Management*, vol. 10, no. 2, pp. 49–53, 2005.
- [28] M. Ben-Daya, E. Hassini, and Z. Bahroun, “Internet of things and supply chain management: a literature review,” *International Journal of Production Research*, vol. 57, no. 15-16, pp. 4719–4742, 2019.
- [29] R. Baheti and H. Gill, “Cyber-physical systems,” *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

- [30] E. A. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.
- [31] E. Hozdić, "Smart factory for industry 4.0: A review," *International Journal of Modern Manufacturing Technologies*, vol. 7, no. 1, pp. 28–35, 2015.
- [32] A. Radziwon, A. Bilberg, M. Bogers, and E. S. Madsen, "The smart factory: exploring adaptive and flexible manufacturing solutions," *Procedia engineering*, vol. 69, pp. 1184–1190, 2014.
- [33] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart factory of industry 4.0: Key technologies, application case, and challenges," *IEEE Access*, vol. 6, pp. 6505–6519, 2018.
- [34] O. Didenko, "Industry 4.0: the real value of a smart factory [full guide]." <https://www.altamira.ai/industry-4-0-smart-factory/>. Accessed: 2022-12-20.
- [35] A. Pereira and F. Romero, "A review of the meanings and the implications of the industry 4.0 concept," *Procedia Manufacturing*, vol. 13, pp. 1206–1214, 2017. Manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain.
- [36] S. Erol, A. Jäger, P. Hold, K. Ott, and W. Sihn, "Tangible industry 4.0: A scenario-based approach to learning for the future of production," *Procedia CIRP*, vol. 54, pp. 13–18, 2016. 6th CIRP Conference on Learning Factories.
- [37] DATAZero, "The fourth industrial revolution." <https://dat4zero.eu/what-is-industry-4-0/>. Accessed: 2022-12-21.
- [38] Wonderware, "Cos'è il mes, manufacturing execution system?." <https://www.wonderware.it/cose-il-mes-manufacturing-execution-system/>. Accessed: 2022-12-21.
- [39] S. Henderson, "The benefits of mes: from the field," 2016. 6th CIRP Conference on Learning Factories.
- [40] M. McCLELLAN, "Introduction to manufacturing execution systems," in *MES Conference & Exposition, Baltimore, Maryland*, pp. 1–7, 2001.
- [41] K. Tzedef, "The evolution of manufacturing execution systems (mes)." <https://www.contel.com/news-item/the-evolution-of-manufacturing-execution-systems-mes/>. Accessed: 2022-12-21.
- [42] S. Jaskó, A. Skrop, T. Holczinger, T. Chován, and J. Abonyi, "Development of manufacturing execution systems in accordance with industry 4.0 requirements: A review of standard- and ontology-based methodologies and tools," *Computers in Industry*, vol. 123, p. 103300, 2020.
- [43] A. Deuel, "The benefits of a manufacturing execution system for plantwide automation," *ISA Transactions*, vol. 33, no. 2, pp. 113–124, 1994.
- [44] Iriday, "Software mes cos'È?." <https://mes.iriday.it/software-mes-cose/>. Accessed: 2022-12-23.
- [45] aizoOn, "aizoon technology consulting." <https://www.aizoongroup.com/home.aspx#intro>. Accessed: 2022-12-26.
- [46] Lincotek, "Lincotek." <https://www.lincotek.com/>. Accessed: 2022-12-27.
- [47] FlowSystems, "Flowsystems." <https://www.flowsystemsinc.com/>. Accessed: 2022-12-27.

- [48] Robopac, “Robopac: About us.” <https://www.robopac.com/en/about-us>. Accessed: 2022-12-28.
- [49] RobopacMachinery, “Robopac machinery.” <https://www.robopac.com/en/business-units/robopac-machinery>. Accessed: 2022-12-28.
- [50] O. MES, “Opera mes.” <https://www.operames.net/index.php/en/operames/operames-overview>. Accessed: 2022-12-29.
- [51] O. Data, *Opera v.6 - User Manual*. Open Data, 40050 Funo di Argelato (BO) - Italy, 1st ed.
- [52] FlowSystemsInc, “Portable air flow test stand.” <https://www.flowsystemsinc.com/portable-air-flow-test-stand/>. Accessed: 2023-01-04.
- [53] Siemens, “Opcenter.” <https://www.plm.automation.siemens.com/global/en/products/opcenter/>. Accessed: 2023-01-23.
- [54] Sage, “What is discrete manufacturing?.” <https://www.sage.com/en-us/blog/glossary/what-is-discrete-manufacturing>. Accessed: 2023-01-23.
- [55] Siemens, “Opcenter execution discrete.” <https://www.plm.automation.siemens.com/global/en/products/opcenter/discrete-manufacturing.html>. Accessed: 2023-01-23.
- [56] Siemens, *Siemens Digital Industries Software: Opcenter Execution Discrete*. Siemens, 5800 Granite Parkway, Plano, Texas, United States, 1st ed.
- [57] Treant.js, “Treant.js.” <https://fperucic.github.io/treant-js/>. Accessed: 2023-06-12.

List of Figures

2.1	The Four Industrial Revolutions [15] (by Christoph Roser at AllAboutLean.com)	4
2.2	Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (by Petroc Taylor at statista.com)	5
2.3	Smart Factories [34] (by Olha Didenko at altamira.ai)	6
2.4	Industry 4.0 [37]	7
2.5	Features of Manufacturing Execution Systems [44]	9
2.6	Rubbiano Manufacturing Plant	11
2.7	Robobac Machinery Manufacturing Plant in San Marino	12
3.1	IT architecture were Opera works best	13
3.2	Opera's Factory Layout Configuration	14
3.3	Opera's main window	16
3.4	Opera - Hovering over a Workflow	16
3.5	Opera - <i>Riepilogo Commessa</i> Workflow	17
3.6	Opera - <i>Riepilogo Commessa</i> Workflow When a Work Order is Selected	17
3.7	The Four Stored Procedures	19
3.8	First Ten Entries of MnrVars Table	19
3.9	Opera MES Example - <i>Inizio Piazzamento</i>	20
3.10	Opera MES Example - <i>Inizio Piazzamento</i> : Badge Selected	21
3.11	Opera MES Example - <i>Inizio Piazzamento</i> : <i>Macchina</i> Tab	22
3.12	Opera MES Example - <i>Inizio Piazzamento</i> : <i>Macchina</i> Tab After Scrolling	22
3.13	Opera MES Example - <i>Inizio Piazzamento</i> : <i>Ordine di Lavoro</i> Tab	23
3.14	Opera MES Example - <i>Inizio Piazzamento</i> : Item Containing Work Order Information	23
3.15	Opera MES Example - <i>Inizio Piazzamento</i> : <i>Fase di Lavoro</i> Tab	24
3.16	Opera MES Example - <i>Inizio Piazzamento</i> : Items in <i>Fase di Lavoro</i> Tab	24
3.17	Opera MES Example - <i>Inizio Piazzamento</i> : <i>Programma/Scheda componente</i> Tab	25
3.18	Opera MES Example - <i>Riepilogo Commessa</i> After Committing the <i>Inizio Piazzamento</i> Workflow	25
3.19	Opera MES Example - <i>Fine Piazzamento</i> Workflow	26
3.20	Opera MES Example - <i>Inizio Attività</i> Workflow	26

3.21	Opera MES Example - <i>Inizio Attività: Attrezzature</i> Tab	27
3.22	Opera MES Example - <i>Inizio Attività: Componenti</i> Tab	27
3.23	Opera MES Example - <i>Controlli</i>	28
3.24	Opera MES Example - <i>Controlli: Correct Fase di Lavoro</i> Selection	28
3.25	Opera MES Example - <i>Controlli: Programma/Scheda componente</i> Tab	29
3.26	Opera MES Example - <i>Controlli: Prova</i> Tab	29
3.27	Opera MES Example - <i>Controlli per Prova</i> Workflow	30
3.28	Opera MES Example - <i>Controlli per Prova: Test Successfully Passed</i>	30
3.29	Opera MES Example - <i>Fine Attività</i> Workflow	31
3.30	Opera MES Example - <i>Riepilogo Commessa</i> After Phase is Complete	31
3.31	First Fifteen Entries of the <i>Macchine</i> Table	32
3.32	Airflow Machine Interface	33
3.33	Airflow Machine Interface - Operation Data Inserted	33
3.34	Airflow Machine Interface - Operation Description Menu	34
3.35	External Airflow Machines Database	36
3.36	Tables Created in the External Database	37
3.37	Workflow 397 - <i>Controlli per Airflow: Configuration</i> Tab	41
3.38	MntrWV Table Showing <i>airflow_data</i> Variable	46
3.39	Workflow 397 - <i>Controlli per Airflow: Airflow Data</i> Tab	47
3.40	Workflow 397 - <i>Controlli per Airflow: Airflow Test</i> Tab	48
4.1	Hierarchical Structure Model of a Process	54
4.2	Opcenter Execution Discrete Main Window	55
4.3	Opcenter Execution Discrete Main Window After Scrolling	55
4.4	Opcenter - Work Orders Window	56
4.5	Opcenter - Work Order Details	56
4.6	Opcenter - Work Order and its Operations	57
4.7	Opcenter - Work Order Diagram of Operations	57
4.8	Opcenter - Operator Landing Page	58
4.9	Robopac's Smart Line in Real-Time	59
4.10	Robopac's Smart Line	59
4.11	Opcenter - Operator Landing Page	61
4.12	Opcenter - Operator Landing Page with Selected Operation	62
4.13	Solution Studio - Operator Landing Edit Page	62
4.14	Solution Studio - Operator Landing Page Edit Page with New Component	63
4.15	Opcenter - Operator Landing Page with Layout Button	63
4.16	Treant.js Basic Example Tree	67
4.17	Updated Work Order Diagram of Operations	73
4.18	Updated Work Order Diagram of Operations when Node is Clicked	74
4.19	Another Example of the Updated Work Order Diagram of Operations	74
5.1	Opera MES - Original Flow of Data Exchange	75
5.2	Opera MES - New Flow of Data Exchange	76
5.3	Opera MES - Zoom in on the New Flow of Data Exchange	76
5.4	Opcenter - New Work Order Diagram of Operations	77
5.5	Opcenter - Old Work Order Diagram of Operations	78