FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

‘SuperCoordinator’ - Coordinator of Educational Subsystems in the context of Industry 4.0

Joaquim Daniel Rios da Cunha

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Jorge Miranda de Sousa

July 24, 2023

Resumo

Ensinar e aprender conceitos de automação e da Indústria 4.0 é uma tarefa exigente, principalmente devido ao número de elementos em jogo, à abundância de dados gerados com diversa natureza e à necessidade de lidar com várias questões ajustadas ao nível de abstração correto (segundo a pirâmide de automação). Na Faculdade de Engenharia da Universidade do Porto (FEUP), no departamento de Engenharia Eletrotécnica e de Computadores, várias unidades curriculares (UCs) abordam a automação e o controlo industrial. O maior objetivo do SuperCoordenador (SC) é controlar a montagem de um conjunto de pequenos Módulos de Simulação (MS) de chão-de-fábrica num simulador maior (significativo), adequado para fins pedagógicos ao longo das várias UCs da área de automação da FEUP.

O SC proposto está implementado em JAVA e consegue coordenar MS de eventos discretos que comunicam por Modbus TCP/IP. É esperado um certo grau de interoperabilidade entre os MS. O SC proposto não implica nenhum investimento financeiro por parte da FEUP, uma vez que utiliza *software* livre e a versão do Factory I/O de que o DEEC da FEUP já dispõe para os seus alunos. Os testes incluíram ainda um MS físico também já existente na FEUP. O SC intercepta as comunicações do controlador para a parte operativa - atualmente, escrito com Codesys. Um estudante inicial(1) na área da automação escreveria o controlo de uma MS simples, e o SC apenas faria o seguimento das peças e introduziria e/ou mostraria alterações limitadas no tempo. Posteriormente, um aluno(2) controlaria um MS maior, otimizaria a produção, etc. - o SC permite analisar os dados, provavelmente melhorando as escolhas de produção, como as opções de encaminhamento na presença de alterações dos tempos de produção, etc.; a alteração dos tempos de produção é feita pelo SC (sem o conhecimento do *software* do controlador) enquanto visualiza o ambiente do Factory I/O (o ambiente tem de ser alterado para acomodar o controlo do SC). Mais tarde, um estudante(3) poderá analisar dados de produção de sistemas com tempos aleatórios e/ou tendências temporais, avarias e reparações simuladas, rastreio de peças e (des)aparecimento de peças, etc. O SC também permite a ligação automática à base de dados e fácil visualização dos dados no painel de controlo. A base de dados atualmente utilizada para os testes é a MariaDB e os exemplos de gráficos foram feitos utilizando o Grafana. Efetivamente, a chegada e a partida de peças é simples, mas com a natureza aberta do SC, permite a produção de dados significativos. O transporte de peças do SC entre os MSs é configurável. Os tempos dos eventos, das peças e de avaria/reparação dos elementos são também altamente configuráveis (por meio de fórmulas e expressões), envolvendo o tempo de trabalho, a contagem de peças, etc., que podem produzir dados relevantes num ambiente simulado de Indústria 4.0, sem deixar de ter um simulador visual. O uso de uma semente aleatória proveniente do ficheiro de configuração permite resultados repetíveis que naturalmente ainda podem ser modificados pela interação direta com as peças no simulador visual Factory I/O.

O SC foi implementado para ser flexível e testado com: i) uma pequena linha de produção simulada no Factory I/O, ii) uma linha de produção simulada no Factory I/O com maior dimensão, e iii) um modelo maior que inclui o anterior e um MS físico (de uma pequena linha de produção)

- adaptado no âmbito deste trabalho para ser controlável por Modbus. Os dados produzidos por amostragem revelaram-se potencialmente interessantes, como o tempo médio de produção com tendência para aumentar até à avaria da máquina.

Abstract

Teaching and learning Automation and Industry 4.0 concepts is a complex task mainly due to the many elements at stake, the large amounts of data flowing of diverse nature and the need to deal with several issues at the correct abstraction level (in accordance to the automation pyramid). At the Faculty of Engineering University of Porto (FEUP), in the Electrical and Computer Engineering programmes, several courses address automation and industrial control. The broad goal of the SuperCoordinator (SC) is to control the assembly of a set of small shop floor Simulation Modules (SMs) into a larger (meaningful) simulator, suitable for educational purposes along the several courses of the automation area at FEUP.

The proposed SC is written in JAVA and is able to coordinate discrete event SMs that communicate with Modbus TCP/IP. A certain degree of interoperability among SMs is expected. The proposed SC implies no financial investment from FEUP as it uses free software and the version of Factory IO that the DEEC of FEUP already has for its students. Tests also included a physical SM also already at FEUP. The SC intercepts communications from the controller to the operative part - currently, the controller is written with Codesys. An initial student(1) to the automation area would write the control for a simple SM, and the SC would only do part tracking and introduce and or show limited timing changes. Subsequently, a student(2) would control larger SM, optimise production, etc - the SC allows analysing data, likely improving production choices such as routing options in the presence of changing production timings, etc; the change in the production times are done by the SC (without knowledge by the controller software) whilst visualising in the Factory I/O scene (the scene has to be changed to accommodate SC control). Even later on, a student(3) would want to analyse meaningful production data in systems that have random times and/or timing trends, simulated breakdowns and repairs, part tracking and part dis/appearance, etc. The SC also allows automatic database connection and easy dashboard data visualization. The database currently used for testing is MariaDB and sample plots have been made using Grafana. Admittedly, part arrival and departure is simple, but with the open nature of the SC, it is expected to still allow the production of meaningful data. SC part Transportation between SMs is configurable. Event times, part timings and element breakdown / repair times are highly configurable (through formulas and expressions) involving work time, part count, etc. that may produce data meaningful in a simulated Industry 4.0 environment whilst still having visual simulator. Using a random seed coming from configuration allows mostly repeatable results that naturally can still be modified by direct interaction with the parts in the Factory I/O visual simulator.

The SC has been implemented in order to be flexible and has been tested with: i) a small simulated production Factory I/O line, ii) a larger simulated production Factory I/O line, and iii) a larger model that includes the previous and a physical SM (of a small production line) - adapted in the scope of this work to be controllable with Modbus. Sample produced data revealed potentially interesting data such as the average production time trending upwards up until machine breakdown.

Acknowledgements

The first words of gratitude must go to my family, who have helped me on this journey from the initial moment I went to university, supporting me and always believing that I was capable of achieving successfully this goal.

Second, to Professor Armando Jorge Sousa, Professor at the Faculty of Engineering of the University of Porto and supervisor of this dissertation, first for selecting me and believing from the beginning that I was capable of carrying out this work, and then for monitoring and supporting me throughout its development.

To my girlfriend, who besides not being aware of the work complexity nor its objectives, has always supported me and stayed by my side, especially in the delicate moments when the strength and motivation were very low.

And finally, to my five cats, for the love they have given me, but for always choosing the perfect inconvenient moments (cats just being cats).

Joaquim Cunha

"Your work is going to fill a large part of your life,
and the only way to be truly satisfied
is to do what you believe is great work.
And the only way to do great work is to love what you do.
If you haven't found it yet, keep looking.
Don't settle.
As with all matters of the heart,
you'll know when you find it."

Steve Jobs

Contents

List of Figures

List of Tables

Abbreviations and Symbols

API	Application Programming Interface
CAD	Computer-Aided Design
CSV	Comma-Separated Values
ERP	Enterprise Resource Planning
FEUP	Faculdade de Engenharia da Universidade do Porto
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
I/O	Input/Output
IP	Internet Protocol
MES	Manufacturing Execution System
OOP	Object-Oriented Programming
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
OSI	Open System Interconnection
RFID	Radio Frequency Identification
SCADA	Supervisor Control and Data Acquisition
PLC	Programmable Logic Controller
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context

The Fourth Industrial Revolution has brought pragmatic changes to the sector, introducing new concepts and technologies in production automation and establishing new methodologies in process control.

These changes are continuous, and educational institutions need to keep up to date in order to stay at the forefront of innovation. One strategy passes through digitalization and simulation of real manufacturing processes. It makes it possible to maintain the link between these two domains and at the same time to bring students closer to an industrial environment by starting introducing real world scenarios and inherent concerns. In this way, simulation allows students to develop themselves in different areas like testing without physical implications, as equipment damages, or bring any sort of risk to people's safety. With today's computing power, it is possible to create simulations with a high level of realism, allowing students to gain experience with real systems. Depending on the simulation software, there are different limitations that can restrict its usability. Some of them are built using physics engines and do not allow multiple instances of the application. As a result, all the simulation items must be in the same scene. Licensing is also a limitation in the way that supporting multiple instances on the same hosting machine implies a robust licence validation system and other legal aspects. Nowadays, computing resources are more available, so this is a less common problem, but should be taken in account. Nevertheless, these resources offer many positive things such as ease of replication, no ageing, inherent security, among others. Furthermore, the cost of simulation is lower than that of real equipment and its maintenance.

In Electrical and Computer Engineering (EEC) at FEUP, Factory I/O is widely used in the automation courses. It is the most frequent option when students need to apply their theoretical knowledge by controlling an automated process related to the course objectives. In this way, FEUP has 85 educational floating licences for this simulator. However, Factory I/O is a closed simulation, which does not allow introduction of new machine types, has a limited number of pieces, similar in

format only distinguished by colour, and although it supports different communication standards, it is not possible to connect more than two simulation scenes.

Handling with production line real modules, with conveyors and/or machines, is beneficial in the learning phase. If it is a full-size model or a scale miniature is irrelevant, the point is the mechanical and electrical aspects that do not need to be ensured in the digital world. The real modules are usually more visually appealing, but they have a number of difficulties, such as price, maintenance, ageing (breakage, etc.), volume and struggles with long running times. Moreover, the access to these modules is not trivial, as they have to be shared among many students if they are to be used in the automation courses.

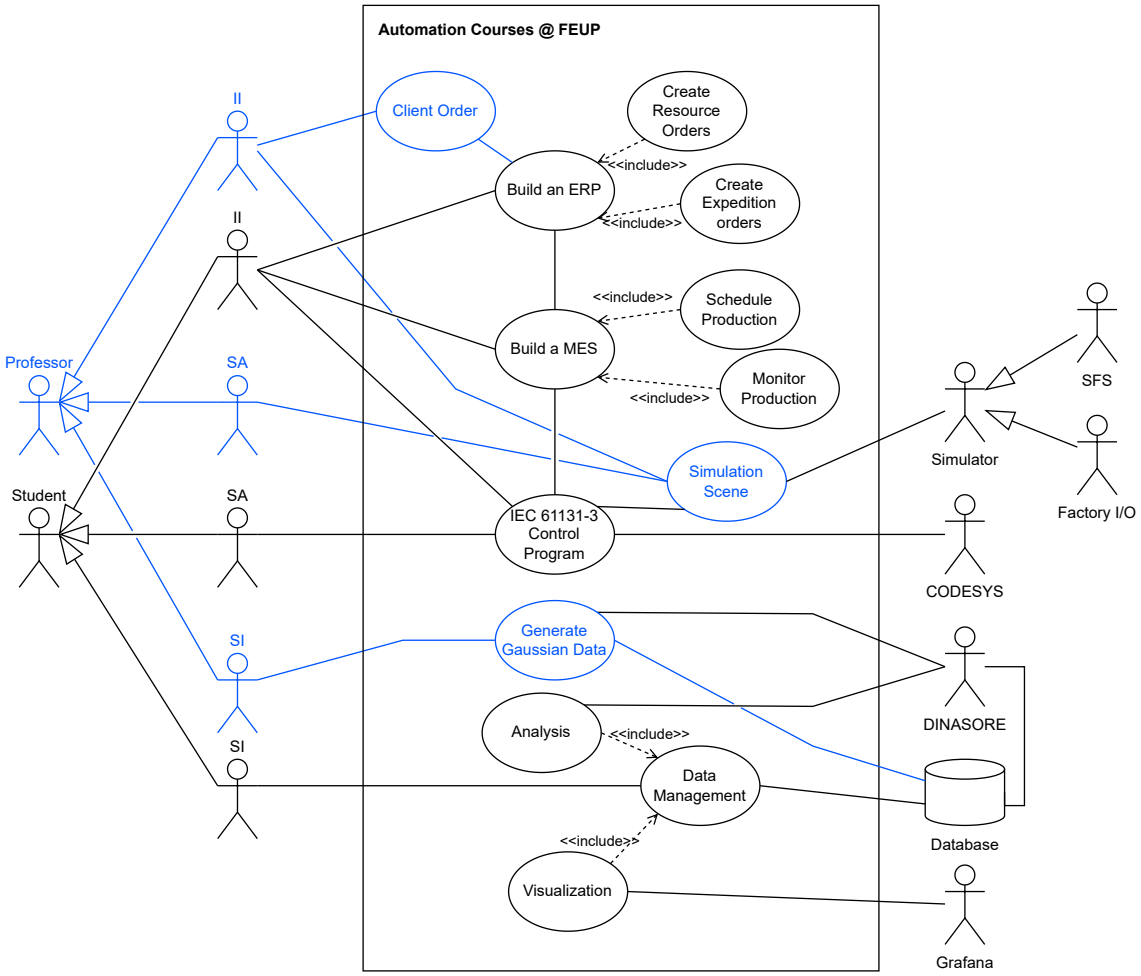


Figure 1.1: Automation Courses at FEUP (UML use case diagram)

Today, the automation courses stream is formed by Systems and Automation (SA), Information Systems (SI) and Industrial Informatics (II) (Figure 1.1). They are inserted in different years of the academic education, respecting the Industry 4.0 (I4.0) concepts and their complexity. SA is the first course and so is therefore oriented towards the basic concepts, nomenclature, standards, etc. The course project involves the design and implementation of a control program for

a given module that represents an automated process. SI focuses on data management, analysis and visualization. Students have access to simulated data, based on Gaussian distributions, of a real manufacturing process or specific equipment, such as the friction spot welding machine, for instance. II main objective is to test the students' knowledge on several aspects, covering the automation pyramid in a bottom-up approach. In addition to designing and implementing a more complete automated process, students have to implement a simplified solution to cover the upper levels of the automation pyramid. In other words, the course challenges the students with a semester project to build a fully automated factory that receives and fulfils orders from outside.

1.1.1 Introduction to Nomenclature

This purpose of this section is to clarify some of the nomenclature used throughout the document, to make it easier to understand. The prefix SFE stands for Shop Floor Educational, which can be detailed in Module (SFEM), Element (SFEE), or Item (SFEI) leading to:

- SFEM — A set of Elements; it may have one or more Elements. In the case of multiple Elements, they all of them share the communication settings. Associated with a simulation window or a real station;
- SFEE - A set of Items; ordered to complete Element function, (i.e., series of conveyor, machine, conveyor to perform an operation); It is the Element that have support for the "real" factors, for example, define the time between the first and last SFEE item;
- SFEI - Smaller piece, usually associated with conveyors or machines;

There are some special variations, due to object similarities in the scope of the application development. They relate to warehousing, transport in the way that they inherit or implement the general case in a particular way. The figure 1.2 is a simplified view of these notations and their relationship. Throughout this document, more complete and detailed instances will be presented, such as figure 3.36.

1.2 Motivation

This proposal seems to overcome some previous limitations and to expand the educational scenarios by improving the flexibility of the automation stream courses.

From a simple perspective, an industrial automated process can be summarised in three elements: sensors, actuators, and control unit. In most cases, these units are Programmable Logic Controllers (PLC), but they can also be electrical circuits or dedicated computers. There are several communication protocols and different ways of interacting options between these elements. The industrial simulators can be seen as the virtual sensors and actuators that need to be controlled by a dedicated unit. The key point is that these units support many communication standards and multiple simultaneous connections. For example, the PLC is responsible for controlling the field

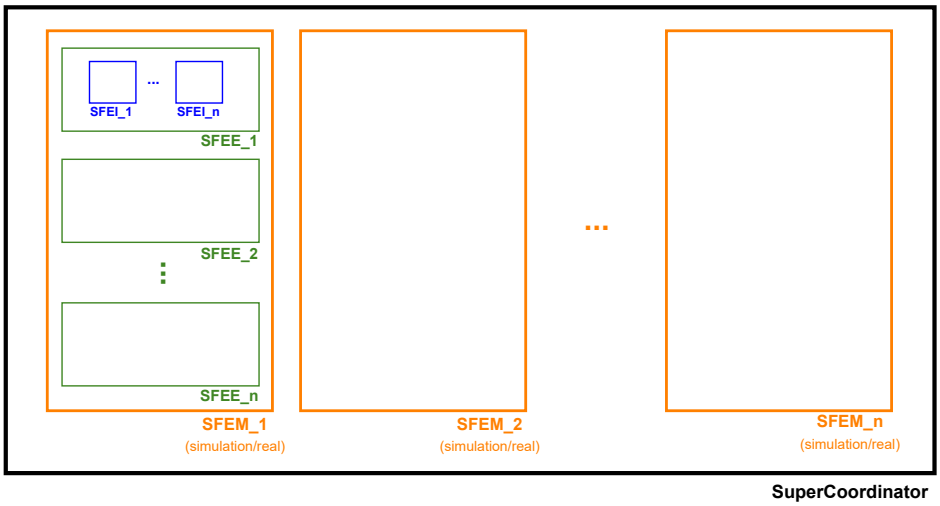


Figure 1.2: Shop Floor Educational components and their relation

level, but can also interact in real-time with other applications to share relevant information about the process and its status.

At this point, the PLC can provide the abstraction needed to handle real or simulated processes without any distinction. Thus, with a software application, capable of establishing and managing multiple connections in real time, it is possible to connect as many PLCs as needed and, as a consequence, build a larger, more complex and diverse factory. The boundary between what is simulated and what is real disappears, so it is possible to build a hybrid factory with simulated and/or real sections, with the moving parts (i.e., process outputs) constantly switching between the two worlds.

From the educational point of view, with a well-defined connection interface for the modules, it is possible to create different factory layouts, scaled in complexity levels according to the objectives and needs of the automation courses.

Frequently, it is a challenge to cover real-world concerns in a simulation, regardless of its context. For example, in a bottling production line, it is necessary to frequently remove a sample for testing purposes. Another example might be the load on a conveyor, that can vary the time between point A and B, or even the age of the conveyor, which can slow it down. These types of factors can be difficult to configure on a simulation, or not be supported, and the study of these factors is playing a more prominent role in the present, associated with process improvements or prevention engineering.

Consequently, by enabling the software application to supervise and coordinate the modules on the basis of defined metrics, it is possible, beyond the connection functions, to introduce on the simulations some abstractions of the "real" factors, leading to a closer digitalisation of a real production line.

1.3 Objectives

The SuperCoordinator (SC) is the software application needed to link and coordinate the different modules. However, the application must be aware of many other features to achieve these objectives (Figure 1.3).

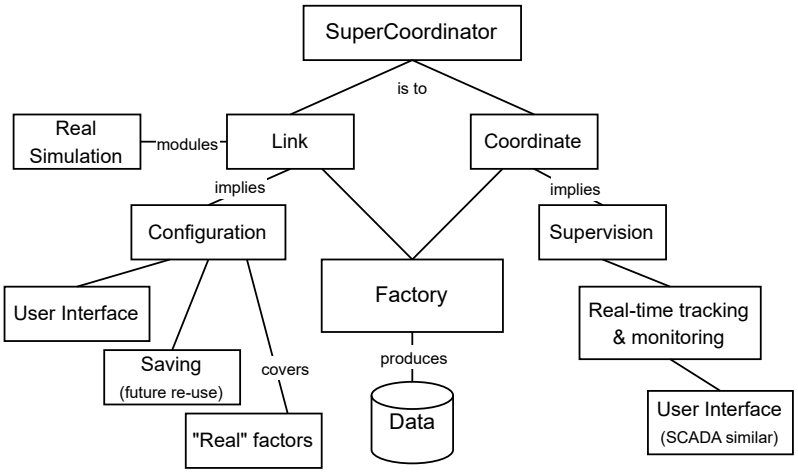


Figure 1.3: SuperCoordinator main objectives

The SC should be the connecting element between individual modules, which may be multiple simulation windows in different host machines or physical builds, called real modules. In this dissertation, Factory I/O will be the digital environment and the Staudinger compact module will validate the integration between the real and simulation domains.

Naturally, in order to give SC the ability to control the factory flow, it is necessary to pass the information about many aspects. The application should be informed about the order and type of inputs and outputs in the communication protocol, which connections are needed between the modules, if the user pretends to change the normal production process, by introducing "real" factors, and so on. In order to carry out the configuration, it is helpful t have an interface for interaction with the user. Depending on the number of modules to be connected and their sizes, the configuration can be extensive, so it should be stored in a plain text file to allow future changes without having to create a new one.

After the configuration and linking stages, the SC should interconnect the isolated modules, i.e. transport the moving parts among them. This involves communication between them at the moment of part entry and part exit, i.e. a module must accept commands to place/remove, or start/finish an operation, and must report about this particular event using the part characteristics, mainly the visual aspect. Thus, the coordination between the modules is highly dependent on the supervision, by periodically checking the states of the sensors and actuators as well as the production variables, to verify if "real" factors condition is verified. In this way, SC fits in with the I4.0 concepts of traceability and supervision, where each produced part has information about itself

and communicates it with all the relevant production elements. The other aspect correlated to the monitoring and supervision functions is the user interface that allows the user to take action when some unexpected event occurs, the called Supervisor Control and Data Acquisition (SCADA). The SC should therefore present a visual interface, adapted to each factory, where special events are highlighted and the evolution of the process is made transparent, in order to be integrated in the educational side and possibly used by the automation students.

To make the hybrid factory more independent, it is necessary to support the management of manufacturing process. To do this, the SC should work based on the basis of external request, such as production orders, and deliver them when finished. Typically, these functions are supported by the Enterprise Resource Planning (ERP) and the Manufacturing Execution System (MES). With this support, long runtime periods can be achieved.

By combining all these aspects, the hybrid factory can function properly. A lot of production data containing valuable information is generated during the uptime. At this point, considering the automation courses at FEUP, as detailed in section 1.1, by including the SC with the work developed by the students in the SA and II courses, it is possible to reuse the data for the SI course (Figure 1.4) in order to extract indicators or even detect some patterns in the data, instead of repeatedly generating Gaussian based data.

1.4 Document structure

This document consists of five chapters. The current chapter, Chapter 1, explains the context of the proposal, what are the motivations and objectives, and briefly clarifies the recurring terms. Chapter 2 presents a literature review including the main aspects related to the industrial domain, as communication and simulation environments. Chapter 3 includes some difficulties and respective decisions, the proposal objectives and have worked developed in accordance to the proposed solution architecture and its validation. Chapter 4 contains the main conclusions and suggestions for future work.

Appendix A have the repository reference as well as pointers to relevant videos covering both the configuration and operation of the application.

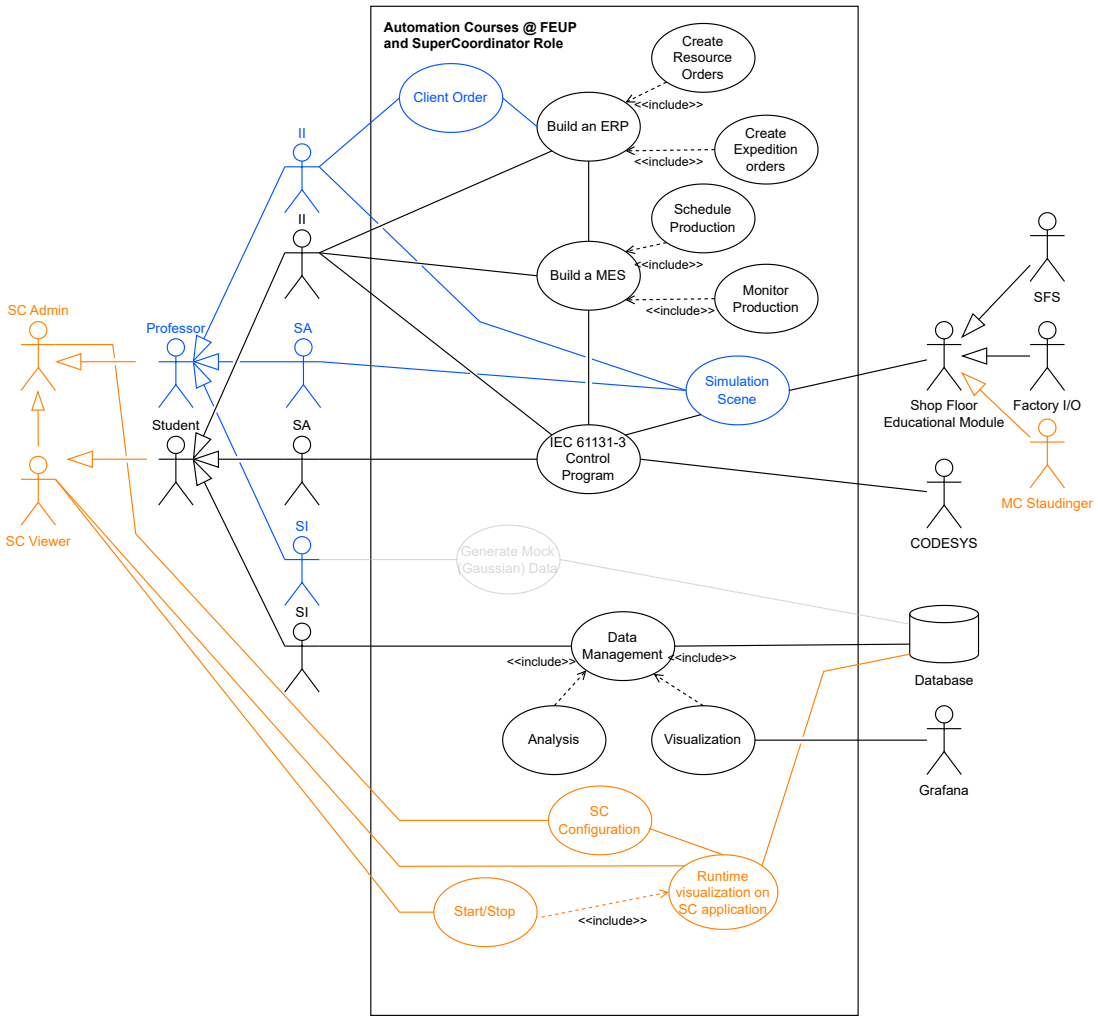


Figure 1.4: Automation Courses at FEUP with the SuperCoordinator (UML use case diagram)

Chapter 2

Background and State of the Art

This chapter gives relevant background on the technologies used in the automation field, an approach to some existing tools and previous related work.

Nowadays, there are several communication options for software applications. However, in the automation area, some of them are more interesting than others, but they all follow the I4.0 ideologies.

The same applies to simulators. The options available range from ready-to-use scenes, through automation-specific simulators, simulation platforms and digital twins, to game engines or physics engines. All are possible alternatives, but in the educational field some of them are preferable.

The SuperCoordinator can be thought of as an extra layer or application that would normally run in parallel with the others in a normal situation. Usually, when there is a simulation environment, there is also a control algorithm running on a SoftPLC. So, the concept design, implementation and testing of the control is not the objective or responsibility of the SC, but will be, most of the time, the student's task, however is required.

2.1 Industrial Context

This chapter provides an industrial context for the main theme of this dissertation, aiming to provide a comprehensive overview of two key aspects: the automation pyramid and the Industry 4.0 ideologies and innovations.

2.1.1 Automation Pyramid

The automation pyramid concept emerged in the 1980s in consequence of the multi-level vertical connection model between enterprise and control systems. The ISA-95 [6] establishes interfaces between the business model and the manufacturing process. Beyond that, it defines a hierarchy model that describes the functional levels and control domains related within manufacturing associations, as well as describes the functional and data stream in the data flow model. On the other

hand, it creates an object model that specifies the information that can pass through the boundaries of the enterprise and the control system.

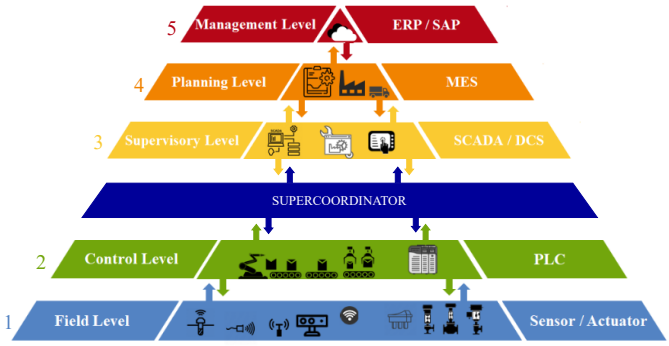


Figure 2.1: Automation Pyramid and its technologies (adapted from [1])

Taking a bottom-up approach, as shown in the figure 2.1, the first level in the pyramid regards the field level. This comprehends all the sensors and actuators in the production cells. In this layer, time has an important role because different types of manufacturing are supported, such as continuous or discrete or even batch production. The second level is about control. This includes the PLCs or similar equipment (microcontrollers or dedicated computers) that must be able to receive, process and communicate outputs as the result of several combinations of states and inputs related to the process. In terms of time, these operations take place in a few milliseconds. The above layer in the pyramid represents the boundary between the control and the supervisor. This layer should represent the relevant states of the process, so usually some low-level details or constraints are hidden in this layer. Sometimes, the supervising agent allows control actions (e.g. associated with a user) that will impact the control level and consequently the manufacturing process. Due to their position in the pyramid, these layers have different time abstractions in terms of whether they are passing information to the upper layers or to the control, so it can be seconds to tens of minutes. In most cases, a SCADA application and its concepts are used to implement the user-process interface. The business domain starts at the planning level. This layer contains the MES or Manufacturing Operation Management Systems. At this point, the main objective is to monitor the evolution of processes and compare them with the schedule using indicators such as Key Performance Indicators (KPI). This task is more time-independent than the previous layers, as the monitoring process can have different durations. The top level of the pyramid is the management level. There are several ERPs that manage all the resources needed for the processes. This level is concerned with supply, warehouse inventory and also keeps the accounting information. An ERP system is fundamental to automated processes because it saves records of the entire pyramid, allowing statistical reports and predictions to be made about the evolution of the production, as well as monitoring the state of the equipment. More recently, with the Industrial Internet of Things (IIoT), some authors defend that a new layer was added in this vertical model, the cloud layer. This layer is very useful in Big Data-Driven Supply Chain Management (BDD-SCM) because it

allows the communication between the several actors in the supply chain, ensuring that everything goes on time [6, 1].

2.1.2 Industry 4.0

According to Teixeira [7], the well-known term "Industry 4.0" dates back to 2011, when the then German government initiated "a high-technology-based strategy to promote the digitalization of manufacturing". As Sung [8] suggests, some people prefer "the fourth industrial revolution than Industry 4.0 because the term "the fourth industrial revolution" is more appealing and familiar than Industry 4.0". Others do not consider this to be an industrial revolution because it only refers to routines and procedures that were carried out informally at the time.

Before 2011, the internet and information technology (IT) were already in use, as were robotics. The innovation step was the integration of these technologies with the electronics, leading to the cyber-physical systems (CPS). These systems reflect the integration of information and communication technology with physical and computational components, supported by the Internet of Things (IoT) and cloud computing for communication support.

The interconnection of multiples CPSs makes the Smart Factory. It can be defined as an advanced manufacturing facility that leverages emerging technologies and data-driven processes to achieve higher levels of automation, efficiency, flexibility, and customisation. It represents a significant paradigm shift in industrial production

A mart factory harnesses the power of connectivity and real-time data to create a highly networked ecosystem within the production environment. It involves the seamless integration of various components, including sensors, actuators, machines, production systems, and software applications, into an intelligent network of interconnected devices and systems.

Therefore, there are key characteristics associated with the smart factories inside I4.0 context, such as:

Interoperability - Fluid interaction and flexible collaboration between all players.

Virtualization - Creation of virtual models (known as Digital Twins) based on real processes and machine monitoring; Digital Twins allow different scenarios to be changed and tested before implementation without affecting the physical process; Reduces costs, increases efficiency and minimizes the risk of errors or disruptions.

Decentralization - Enables different systems within the smart factory to be independent, i.e. make individual decisions but still achieve the organisational objective.

Real-time Capability - Real-time data collections and process monitoring.

Service Orientation - Each component of the smart factory provides services in such a way that they can all be linked together in different ways to produce different and complex functions. This facilitates mass customisation by enabling for the efficient production of personalised products or small batch sizes without compromising efficiency.

Modularity - By designing and building modular production systems, smart factories are flexible. They can be expanded, improved or replaced with another system with minimal impact on the rest of the production process.

The concept of a smart factory within the Industry 4.0 perspective revolves around the use of digital technologies and data-driven insights to create highly automated, connected and intelligent manufacturing systems. By embracing the principles of connectivity, data analytics, automation, and flexibility, smart factories can revolutionise industrial production and drive significant improvements in efficiency, productivity, quality, and sustainability.

2.2 Communications

SuperCoordinator's ambition is to interact with different modules regardless of the communication protocol. To do this, it should have an interface to convert its instructions into the solution implemented by the module. Therefore, in this section covers two industrial communication protocols, Modbus TCP/IP and OPC UA, as well as the HTTP protocol because of Factory I/O support in the Ultimate version.

2.2.1 HTTP Web Server

The HyperText Transfer Protocol (HTTP) is implicit in the *World Wide Web* (*WWW*). HTTP has undergone many changes over the years to maintain its simplicity and increase its flexibility over the years.

According to Mozilla Contributors [9], the *WWW* has "built over the exiting TCP and IP protocols, consisted of 4 building blocks:

- A textual format to represent hypertext documents, the HyperText Markup Language (HTML);
- A simple protocol to exchange these documents, the HTTP;
- A client to display (and edit) these documents, the first web browser called the World Wide Web;
- A server to give access to the document, an early version of HTTP daemon (process that runs in background and plays the role of a server in a client–server model using the HTTP network protocol)."

The first version of HTTP/0.9, released in 1991 (Figure 2.2) was known as "the one-line protocol" because the requests had only one possible method, GET, and the path to the requested resource. The response followed the simplicity, returning only the HTML file. In a short time period, HTTP/1.0 appeared, which extended the type of plain text document that could be transmitted by the introducing headers in the message structure, leading to the POST and HEAD methods. In the meantime, a standardised version of the protocol was released, HTTP/1.1, in order to resolve

the ambiguities and establish fundamental aspects. This also introduced the concept of a persistent connection, allowing multiple requests to be made using a single connection. Many other methods were added, such as PUT, PATCH, DELETE, CONNECT, TRACE and OPTIONS, enriching the protocol. Special attention should be paid to the PUT method, which allows "any web application to let an API retrieve and modify its data without having to update the browsers or the servers" [9, 10].

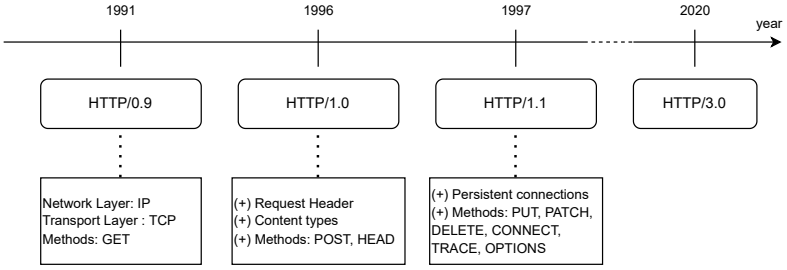


Figure 2.2: HTTP versions timeline

Within the scope of this dissertation, it is important to review the HTTP protocol, as the Ultimate version of Factory I/O can accept HTTP requests. However, it is not necessary to use a more powerful and also complex web Application Programming Interface (API), such as Apache HTTP, as the interactions with the application server are based on the HTTP/1.1, and it only accepts GET and PUT requests, responding with messages of type JavaScript Object Notation (JSON).

2.2.2 7-Layer OSI and TCP/IP models

These models grew out of the need to standardise the way the computer systems communicate over the network. The Internet as we know it is not based on the 7 layers Open Systems Interconnection (OSI) model, but on a simpler model, the Transfer Control Protocol/Internet Protocol (TCP/IP). However, the OSI model is widely used and has been adopted by the ISO/IEC 7498-1, because it makes it possible to see how networks operate, as well as helping to isolate and troubleshoot network problems.

Table 2.1: Comparison of the 4-layer TCP/IP model with the 7-layer OSI/IEC model (adapted from [5])

OSI model (ISO/IEC 7498-1)	TCP/IP model (RFC 1122)	Example Protocols
7 - Application layer	4 - Application layer	HTTP, DNS, Modbus
6 - Presentation layer		
5 - Session Layer		
4 - Transport layer	3 - Transport layer	TCP, UDP
3 - Network layer	2 - Internet layer	IP
2 - Data link layer	1 - Link layer	Ethernet, RS-232/485
1 - Physical layer		

An important aspect to highlight is the overhead imposed by each layer. For small messages, it could happen that the final packet may have more overhead than the bytes of content. So if it is possible to minimize the number of layers, the performance of a network will be improved due to smaller packet lengths. This is one of the reasons for using TCP/IP on the Internet, which is, after all, it is a real-time system.

In summary, the TCP/IP model is a collapsed version of the OSI model, although it is older. Despite its simplicity, the model has been supported in specific and standard protocols in order to solve specific communication problems. On the other hand, the 7-layer OSI model is a generic, protocol-independent model designed to describe all forms of network communication. Sometimes, in simple applications, some layers are not used [11].

2.2.3 Modbus TCP/IP

The Modbus protocol was created by Modicon in 1979 for industrial automation systems and their PLCs. "Modicon took the approach of openly publishing its specification and allowing its use by anyone without asking for royalties"[2]. Since then, "it became an industry standard method for the transfer of discrete/analogue Input/Output (I/O) information and register data between industrial control and monitoring devices"[3].

The Modbus TCP/IP is an application layer that allows sending Modbus packet to be sent over the TCP/IP protocol stack. A Modbus TCP network consists of several devices connected via a TCP/IP network using a client-server technique. First, a TCP/IP connection must be established between the server and the client. By default, a Modbus server listens on port 502 for new connection requests from its clients. "Connection establishment and management are handled by the TCP/IP protocol and occur independently of the Modbus protocol"[2]. Interactions then consist of requests from the client and responses from the server (Figure 2.3).

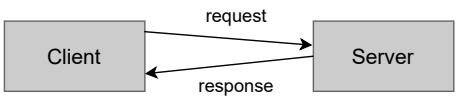


Figure 2.3: Client-server interaction model (adapted from [2])

Some Modbus server devices can support multiple connections to multiple clients at the same time and vice versa. The protocol also does not require a response to an old request before a new request, which means that the client can make many requests without receiving a response. A device can be both a client and a server, for example, in a complex network where a particular device is a client of a larger network and a master for its subordinate clients.

A client can be any peripheral device (I/O transducer, valve, network drive, or other measuring device) that processes information and sends its output to the server using the Modbus protocol over a TCP/IP connection.

The Modbus TCP frame (Figure 2.4) is a Modbus standard data frame, called Application Data Unit (ADU), embedded in a TCP frame without the Address and CheckSum fields, keeping the Function Code and the Data unchanged, forming the Protocol Data Unit (PDU).

- **Function Code** – Represents the purpose of the message, for example, read coils (0x03) or write single coil (0x05). Each code is 1 Byte long and is in hexadecimal format. The function codes are grouped into 3 classes, in terms of the minimum useful set of functions, class 0; the most commonly used class 1; and the more generic ones, that support all data transfer, class 2. There are 14 functions in total.

- **Data** – Contains the information about the client request to be handled by the server. It has a minimum length of 1 Byte and the maximum length depends on the network protocol used. For the TCP/IP, the packet can be 65535 Bytes including the headers.

The Modbus Application Protocol (MBAP) header is then added to the PDU, encoding information relevant to the recipient of the message, such as:

- **Transaction Identifier** – Since it is possible for the client to make many requests, it is important to assign an identifier to each one in order to recognise the server response when it arrives. For this reason, the client initiates this field and the server simply echoes the value.

- **Protocol Identifier** – For Modbus, this field is always valued at 0. Other values can be associated in the future.

- **Length** – This field is a byte count of the remaining fields. It includes the unit identifier byte and the length of PDU in bytes.

- **Unit Identifier** – This field is used to identify a remote slave connected to a Modbus server via a serial interface, acting like a gateway. In a typical Modbus TCP/IP server application, the unit identifier is set to 00 or FF, and it is ignored by the server or simply echoed back in the response.

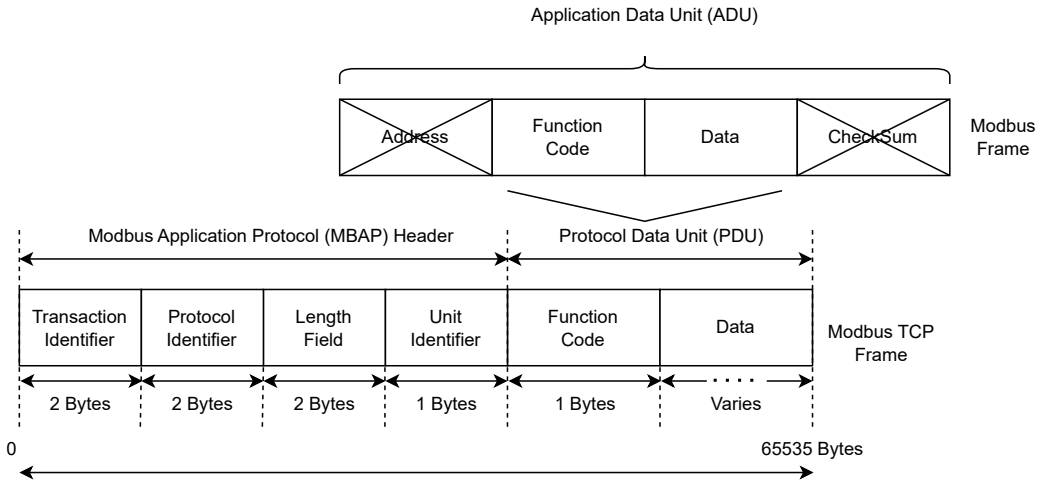


Figure 2.4: Modbus TCP frame (adapted from [2, 3])

The Modbus protocol has error handling in case the received PDU is in the wrong format and the action cannot be performed. To do this, when the server receives a request, it first validates the frame and if it is not recognised as a Modbus frame, the server does not respond. Then, if the frame is well formed, it will start to check the syntax, i.e. the validity of the function code, the memory address and the data values. If there are any errors, the server will respond with the error codes 1, 2 or 3, respectively. The actions are then executed. If an error condition occurs during this step, the appropriate code is generated and sent to the client in the Exception Response PDU format. In total there are 9 possible error codes.

In a normal response, the server simply returns the function code of the request in the function field of the response.

2.2.4 OPC and OPC UA

Open Platform Communications (OPC) is the interoperability standard for the secure and reliable data exchange in the industrial automation space and other industries. Developed by Microsoft for Windows operating systems, it is aimied at process control in industrial automation applications [12].

At the time of its first release, back in 1996, “its purpose was to abstract PLC specific protocols (such as Modbus and Profibus) into a standardized interface allowing Human-Machine Interface (HMI) and SCADA systems to interface with a “middle-man” who would convert generic-OPC read/write requests into device-specific requests and vice versa” [12, 13].

OPC Unified Architecture (OPC UA) combines the functionality of the individual OPC Classic specifications into an extensible framework. One of the most fundamental elements of OPC UA is the information modelling framework, which is used to define rules and describe the methods

to access the information model. It is therefore platform-independent and supports different stack implementations (Figure 2.5) to interoperate with each other in different programming and runtime environments, thus ensuring interoperability by supporting the most common operating systems, including the embedded ones [14, 15, 13].

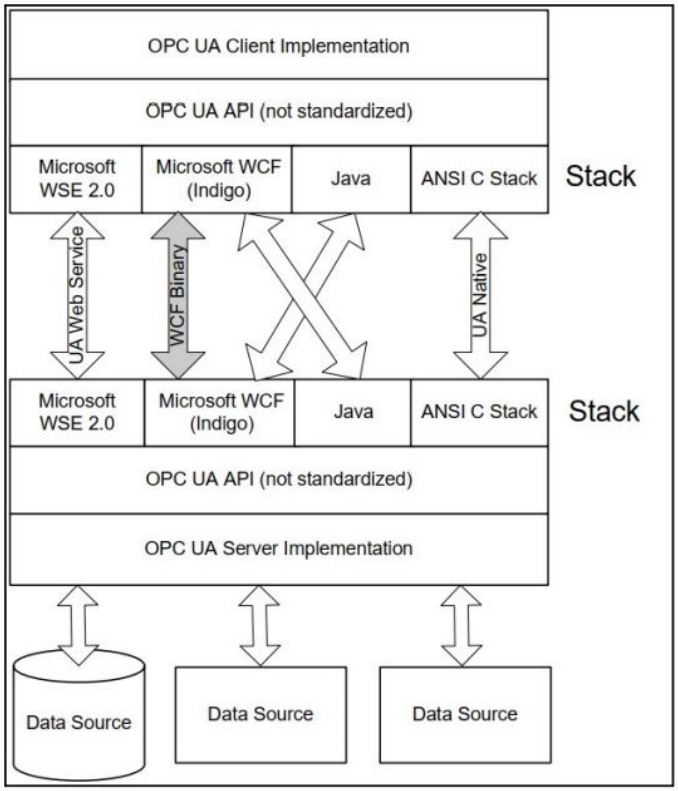


Figure 2.5: OPC UA communication stack [4]

The information model is defined by nodes and references (Figure 2.6). A node is a set of attributes and can be grouped into classes. A class specifies the type of node and refers to its attributes. In the standard, there are predefined base classes to represent real-world entities. A reference report to the relationship between nodes [15].

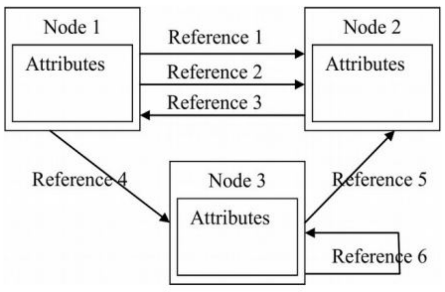


Figure 2.6: Example of a OPC UA information model [4]

The main disadvantage of OPC UA is its complexity. The full OPC UA protocol specification is extremely large. In the context of this work, it is important to have an overview of the standard definition. The theoretical details about the realisation of the concepts are not relevant, as long as the tools to be used have simple interfaces and support for this communication protocol.

2.3 Simulators

Simulation is a common practice in the industrial automation, as it leads to better decisions in a way that allows to see the impact of the changes beforehand, without having to implement them and support the inherited consequences. There are many options available on the market in order to fulfil the sector needs. Naturally, most of them are paid. In general, they have different licence options to highlight specific features, so they have different prices. Nevertheless, some developers offer trial versions that allow the user to test the software and its functionalities and compare them with the project requirements.

In this section a brief analysis of some simulation software will be made. The aim of this analysis is to find interesting features for the educational side and possible integration in the SuperCoordinator application, covering I4.0 principles and the automation pyramid area.

2.3.1 Visual Components

Visual Components[1] is a 3D manufacturing simulation technology. It has been designed for manufacturing with the objective of creating powerful, flexible and scalable platforms. Visual Components has different types of licences that share the resources and functionality between them (Table 2.2).

The Essentials version offers the basic functionality. It allows the creation of custom layouts with a few clicks, using the drag-and-drop and plug-and-play features, included in the eCatalog, as well as importing Computer-Aided Design (CAD) data for more realistic simulation. The eCatalog contains more than 2700 pre-defined and ready-to-use components from leading industrial automation brands. It offers easy robotics, i.e. easy programming of robot behaviour using teaching tools supported by the application.

In terms of connectivity, it supports OPC UA and some vendor-specific interfaces, allowing the simulation to be controlled externally.

The Professionals version relates to the customisation of the layout and its parts. This version allows defining properties for imported CAD models, create personalised libraries of components and use some pre-built wizards that automate the configuration for different component types in order to speed up the import process. It also has basic solid geometry tools and simplification, allowing users to build their own models using the available 3D geometry libraries and even simplify the imported models to improve simulation performance by reducing the file size.

[1] Visual Components https://www.visualcomponents.com/ (visited on 05/06/2023)

Table 2.2: Visual Components characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Essentials / Closed / Not available	No / Yes	Yes	Ready to use components (eCatalog) Expansible with CAD files Point Cloud facilities import Statistics of simulated data Offline robot programming (OLP) 3D simulations exports
Professional / Closed / Not available			Component modeling Pre-build wizards Basic solid geometry Geometry simplification
Premium / Closed / Not available			KUKA and Fanuc support Siemens S7 connectivity Path statement Paint process visualization Virtual reality and web viewer

The Premium version is dedicated to specific solutions, offering interfaces and support for some industry brands, such as Siemens S7 PLC connectivity. It has a visualization of a painting process in such a way that even the layer thickness can be seen. In this version the virtual reality concepts are more developed, so it is possible to interact in real time with different simulation elements, have remote access to the simulations and streaming.

The Visual Components require a ticket to the support team to obtain a trial version. However, special pricing is available for educational institutions and research laboratories.

2.3.2 Anylogic

AnyLogic[2] is a self-contained simulation environment that can run complex simulation over a long period of time. The aim of this simulator is to recreate a real world and find the advantages and disadvantages of the layout, the best configuration and relevant points.

The main feature of Anylogic is to manage and improve the design based on the simulation results (Table 2.3). As an example, consider the design phase of a building. Some evacuation standards have to be followed in case of an accident. This type of software allows the simulation of several numbers of people looking for the resources such as lifts, escalators, or stairs. This application identifies the critical points and the room for improvement. There is also a database resource that can store the data permanently. Another example could be the construction of a motorway to solve traffic problems. This simulation will determine the critical regions depending on the variables of the problem such as rush hours, different flows in the ways generate variations and which.

[2] Anylogic https://www.anylogic.com/ (visited on 05/06/2023)

Table 2.3: Anylogic characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Personal Learning / Closed / Free of charge	No / No	Yes	Multimethod modeling capabilities Limited resources Simulation in the cloud 2D/3D simulation Built-in database (excel and text files) Simulation and parameters variation experiments Public model export to AnyLogic Cloud Basic model debugging
University Researcher / Closed / One-time purchase			Technical support in model building Unlimited use of industry specific libraries External database integration Unlimited use of professional experiment framework Private model export to AnyLogic Cloud
Professional / Closed / One-time purchase			Professional database integration Export models to standalone applications Professional model debugging CAD drawing import

Anylogic offers a free personal licence for self-learning with its inherent limitations, as well as a 30-day trial of the university and professional versions.

As the software is based on Java, there are many customisation options available, as well as cross-platform support. However, it is a self-contained simulation, which means that the models and layouts must be created or adapted (in the case of imported configuration) in order to run the experiments. Thus, it is not possible to have external control and subsequent communication with the simulation and its variables. Therefore, within the scope of this work, this software does not support explicit interaction with other applications.

2.3.3 FlexSim

FlexSim[3] is a highly realistic discrete event simulation with 3D graphics that lets create and refine a model based on existing or proposed systems.

This software allows the import of user-defined 3D objects, making the simulation similar to the real system and providing realistic effects such as shadows and lightning by default. It also offers a standard object library with ready-to-use elements that can be configured as required, including C or C++ model commands for specific details. A relevant aspect is the association of different operating times for the machines, e.g. depending on the probability distribution (Poisson, normal, etc.). Although the FlexSim library provides many simulation models using the standard 3D objects, it is possible to customise the simulation model with 3D objects that look like the system to be modelled.

[3]FlexSim https://www.flexsim.com/ (visited on 05/06/2023)

Table 2.4: FlexSim characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
FlexSim / Closed / Not available	Yes /Yes	Yes	3D manufacturing simulation Warehousing and handling integration Supply chain model analysis Industry 4.0 oriented Digital Twin support PLC emulation Educational oriented

FlexSim can simulate PCL's within the software, no physical hardware is required. Based on this feature, this software is able to communicate with external applications, through Modbus and OPC UA, in order to achieve the virtualisation of the system.

An important theme of the work is to make a simulation as real as possible. To achieve this, it is essential to introduce downtime and repair times in the machines on the shop floor. FlexSim supports mean time before failure (MTBF) and mean time to repair (MTTR). The ability to configure the elements of the simulation with these concepts brings the model closer to reality, in that it is also possible to choose which behaviours are required to recover the machine from certain states.

Using the FlexSim, it is possible to simulate an entire company (Table 2.4), from the logistics levels don to the shop floor level, as it takes into account the supply chain simulation, the warehouse logistics and the manufacturing process virtualisation. This is useful in the way of getting the simulation going, throwing long time periods for data collection, along with the capabilities of the SuperCoordinator to store this information.

In respect of the available licences, FlexSim only provides one paid licence that includes all the software strengths. It is also possible to request a trial version after a registering on the official website page.

2.3.4 Simio

Simio[4] is the acronym for Simulation Modeling framework based on Intelligent Objects. The Simio presence is extensive and covers numerous industries such as healthcare, manufacturing, supply chain, etc. It offers two main software products, simulation and planning. Some common features are presented in table 2.5

The simulation software is completely object-oriented. It defines processes and objects step-by-step, graphically, without the need for programming. This leads to a quick and easy representation of the components, creating Digital Twins for all the elements involved in a particular operation to be studied and optimised, allowing the analysis and elimination of risks or potential

[4]Simio https://www.simio.com/index.php/(visited on 05/06/2023)

Table 2.5: Simio characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Different options/ close / Different plans	Yes / Yes	Yes	Industry 4.0 concepts cover Digital Twin modelling 3D visualization Data-driven strategies to monitor, simulate and evaluate production Real-time decisioning Risk-based planning and scheduling Educational resources

problems in complex systems. Simulation can therefore interact with MES/ERP systems in real time and optimise schedules and reschedule if necessary.

The scheduling software consists of a model that fully captures the detailed constraints and variations within the system and performs real-time risk analysis. The model can be used in two different ways: firstly, to produce a detailed schedule and plan executed in a perfect environment where machines do not break down, process times are always constant, and materials arrive on time; secondly, it can replicate the same simulation model, but this time with variation switched on. This allows a probabilistic analysis to be carried out to estimate the underlying risks associated with the schedule and, consequently, to control the risk.

2.3.5 Gazebo/Ignition

Gazebo [5] is a dynamic multi-robot simulator powered by Open Robotics [6].

Gazebo is widely used in the robotics domain due to its bridge to the Robotic Operating System (ROS). ROS is a powerful framework with several libraries implemented by the contributors to facilitate the inherently complex task of controlling a robot in the different environments that the real world suggests.

Table 2.6: Gazebo characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Lifetime / Open / Free	No / No	No	Distributed simulation Dynamic asset loading Tunable performance Cross-platform support ROS integration Realistic simulation Extensible Different interfaces supported

[5] Gazebo https://gazebosim.org/home (visited on 05/06/2023)
[6] Open Robotics https://www.openrobotics.org/ (visited on 05/06/2023)

As mentioned in the table 2.6, the Gazebo simulator addresses performance concerns by supporting distributed simulation, i.e. distributing the computational load across multiple servers to improve performance; dynamic asset loading, i.e. during simulation, the application is able to load or unload assets to improve performance, which is a good combination with distributed simulation; and tunable performance, i.e. adjusting the simulation time step size to run at real time, slower or faster compared to real time.

In terms of support, Gazebo runs on Linux and MacOS operating systems, while the Windows version is under development. It is possible to run the Gazebo application in the cloud, taking advantage of their cloud hosted server [7]. The most common use relates to ROS integration, since the Melodic version. This is achieved by the converting Gazebo's messages into ROS messages, which allows the use of the wide ROS framework with the integration of many sensors such as monocular and depth cameras, Light Detection and Ranging (LiDAR) sensors, Inertial Measurement Units (IMU) and others frequently used in robotics field. This integration is further enhanced by the 3D graphics provided by Gazebo Rendering, which offers the latest improvements in rendering. It is also extensible, allowing new models to be created and used in the simulation scene. Related to this, there are many industry models available from the community that represent warehouses and other elements of automation that could be useful for building an industrial factory.

The main disadvantage is the connection to the industry standards, as it do not directly support this integration, for example Modbus or OPC UA protocols.

2.3.6 Robosuite

Robosuite[8] is a simulation framework powered by the MuJoCo physics engine for robot learning. It also offers a set of benchmark environments for reproducible research.

Table 2.7: Robosuite characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Lifetime / Open / Free	No / No	No	Open-source XML based Dynamics and physics simulation

Robosuite is a recent software development that addresses the limitations of robot hardware, in terms of reproducibility and the limited accessibility. The main goals of Robosuite are to provide researchers with a standardised set of benchmark tasks for rigorous evaluation and algorithm development, a modular design that offers great flexibility in designing new robot simulation environments and a high quality implementation of robot controllers and standard learning algorithms to lower the barriers to entry. Although it is a powerful simulator, the Robosuite is a very recent software, so it has some limitations in the area of interest for this dissertation. For example, it

[7] App Gazebo https://app.gazebosim.org/ (visited on 05/06/2023)
[8] Robosuite https://robosuite.ai/docs/overview.html (visited on 05/06/2023)

does not have any kind of communication with external applications, for external control of the manipulators. In this way, it has not been supported by the industry, i.e. does not have pre-built models of conveyors or machines, so that the construction of a shop floor for an entire automated production line has to start from scratch. Despite these facts, it is open source and is based on Extensible Markup Language (XML), so it is possible to create any kind of layout and models for any application, investing the time and resources (Table 2.7).

2.3.7 Factory I/O

FactoryIO [9] is a simulation environment oriented to the PLC connection. The software has many supported technologies, divided in different versions, to satisfy the user needs. Some of them are listed in Table 2.8

Table 2.8: Factory I/O characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Different options / Closed / One-time purchase or annual	Yes / Yes	Yes	Industrial scenarios ready-to-use Warehouse simulation Works in all PLC brands Support most common automation technologies Industry 4.0 oriented PLC wiring Built based on education Web API (Ultimate version)

The Factory I/O environment has many pre-built model, such as parts, machinery, conveyor, splitters, mergers and even automated warehouses, making the configuration of a new shop floor less time-consuming. However, it does not support the introduction of new elements, which limits the options available for a customised simulation. For example, if a real simulation is needed to highlight the changes in the parts due to transformations in the machines, such as colour or dimension, this software does not allow such virtualisation. There are ways to overcome this inconvenience by appealing to visual abstraction. The Factory I/O has Radio Frequency Identification (RFID) and visual sensors. Then, by assigning different identifiers to the pieces, which is one of the tasks of the SuperCoordinator, it is possible to simulate a more complex process conditioning the control process with the codes of the pieces.

The Factory I/O is all about connectivity. The software licence types differ in terms of the connection technology to be used. In addition, it offers resources for simulating PLC operation to facilitate real-world integration. Similarly, the software can be connected directly to an existing PLC, with the I/O from the real world mapped to the simulation. It supports Modbus and OPC-UA from a simple approach, having easy to use interfaces for both.

[9] Factory I/O https://factoryio.com/ (visited on 05/06/2023)

Regarding the objectives of this proposal, related to programmatic fail injection, i.e. introduction of real world events such as parts removal from conveyors or damage, machine failures or maintenance and other behaviours. The Factory I/O Ultimate version allows a web API to be launched to interface with other applications. This makes it possible to externally manipulate the value of certain elements of the simulation, changing the sensor states in such a way that unexpected failures occur, by establishing an HTTP connection to the web server and sending GET or PUT requests.

The main advantage of using this software is that it is already being used in FEUP. This way the learning curve is less compared to other options.

2.3.8 SimTwo

The SimTwo[10] is a simulation environment developed by a FEUP professor, Paulo Costa.

Table 2.9: SimTwo characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Lifetime / Open / Free	Yes / No	No	XML based Easy to use Highly configurable External communications Realistic environment Permanent support

As the table 2.9 suggests, SimTwo is not an industrial simulation environment, unlike previous ones, but is a realistic simulator for robotics. It is based on the connection of elementary 3D blocks, such as cylinders, cubes, spheres, etc, to form a more complex object. The definition of these objects is XML based, so it is open source, highly configurable and reusable. There are special parts, such as sensors and actuators, in order to enable the simulation scenario.

After the defining the objects, the world is constructed by assigning the spatial position for each of them. It is also necessary to define the control function, i.e., the interaction between all the elements and the response of the global system to certain events.

A realistic environment is achieved by decomposing the robot into a system of rigid bodies, hinged or prismatic joints, optionally actuated by electric motors.

In terms of communications, the SimTwo supports serial User Datagram Protocol (UDP) and serial communication, as well as Modbus TCP/IP.

The main advantage of this software is the proximity of support when needed. However, the simplicity it offers allow creating almost any object or industrial environment by investing the time needed to define the object.

[10] SimTwo https://github.com/P33a/SimTwo (visited on 05/06/2023)

2.3.9 SFS

The Shop Floor Simulator[11] is a simulation software developed by the FEUP professor André Restivo, in order to be used in the automation courses.

Table 2.10: SFS characteristics

License / Source / Pricing	Modbus / OPC UA	Industry Support	Characteristics
Lifetime / Open / Free	Yes / No	No	Java based Easy to use Simple simulator Highly configurable Stable version

According to the table 2.10, the SFS is based on Java 2D and communicates with other software applications via Modbus. It has a very versatile implementation so that a plant floor can be configured as required. In a simple way, the layout definition only requires changing the configuration file and placing the type of element in the desired position.

The simulator supports 3 conveyor types, linear, rotary and sliders. Beyond that, it has only one machine type to perform operations on the parts (only colour changes). However, each machine supports many tools and each tool can perform a specific operation. In addition, it integrates warehouse functionalities, by storing pieces, and even release zones [16].

2.4 Automation Discrete Events Systems

A PLC is a device specifically designed to control different manufacturing processes such as assembly lines, production robots, automated equipment, and more. Meanwhile, a SoftPLC is defined as software that is capable of emulating PLC functionalities using a traditional CPU, but has the advantage of coexisting with other applications, that is, not fully dedicated to process control [17].

From the SuperCoordinator's point of view, the aim is to monitor and supervise a process and not to control it. Therefore, the control algorithm should be implemented using any automation discrete event system. After that, it is only needed to establish a connection to the application so that it can play its functions.

In this section, some automation discrete events systems are approached that are able to develop and execute the control program and also interact with the SC.

2.4.1 CODESYS

CODESYS [18] is, according to the manufacturer, an integrated development environment for programming controller applications according to the International Industrial Standard (IEC) norm,

[11] SFS https://github.com/arestivo/sfs (visited on 05/06/2023)

IEC 61131-3 [19]. This is a crucial point in the decision of the programming tool in order to all the different parts of the system compatible as they follow and support the standards in the automation field.

CODESYS also has a good support from the manufacturers as well as continuous updates. Indeed, the latest version achieves with the new agreement of the IEC 61131-3 user by defining the Instruction List (IL) as an obsolete language, ending with it maintenance in future version [18]. However, the programming environment will continue to support the language for the older implementations.

The programming software supports object-oriented programming, fully compliant with the 3rd edition of the IEC 61131-3 Standard in all available editors without any additional tools, which helps in the implementation of complex control algorithms.

Related to this dissertation and the previous simulator overview, the CODESYS has support for many communication protocols, including OPC UA and Modbus, and it is a free tool.

2.4.2 Beremiz

Beremiz [20] is a free software that supports the IEC 61131 and other standards, likewise Modbus and OPC UA.

This software was born with the objective of standardising the PLC program to eliminate the need to write the control program for specific brands. Although the IEC 61131 is an open standard, back to the time of the software development, there wasn't enough heterogeneity between the PLC brands, leading to each of them adopt a specific way of programming their control units.

With Beremiz, it is possible to write a unique control program, according to independent standard specifications, compile it and use the computer itself as a PLC. In the end, it lets you "turn any processor into a PLC" [20].

2.4.3 FEUPAutom

FEUPAutom [12] is a SoftPLC for Windows, mainly used at FEUP used in the Systems and Automation course. As it is free, it does not require any installation or licensing, so students ca use it in the laboratory or at home, if necessary.

The software has been developed for educational purposes, mainly by prof. Armando Sousa, so it has some interesting features such as syntax highlighting, completion, debugging through time traces, debugging through variable inspection and other debugging options. As an introduction tool for automation basics, it follows the IEC 61131-3 definition and implements an IEC 60848 adaptation in order to better suit the needs of the subjects to be taught. The controller can be programmed in Structured Text (ST) or GRAFCET.

In terms of communication, the FEUPAutom supports Modbus TCP/IP, making it possible to connect to Factory I/O, SimTwo and other simulators.

[12] FEUPAutom https://sites.google.com/gcloud.fe.up.pt/asousa/feupautom (visited on 05/06/2023)

2.4.4 Node-RED

Node-RED is a programming tool for wiring together hardware devices, APIs and online services [21]. Unlike the other programming tools, the Node-RED provides a browser-based editor that illustrates the relationship between the nodes through the wiring. It also offers a wide range of nodes that can be deployed to its runtime in a single click [21]. This feature plays a relevant role from an educational point of view. It is easier to understand the concepts of an automated control system and the interaction between the elements when there is an elucidative Graphical User Interface (GUI).

This tool also supports JavaScript functions. These can be created in the text editor and be saved for future use. It has a lightweight runtime based on Node.js, taking full advantage of its event-driven and non-blocking model. This makes it ideal for running at the edge of the network on low-cost hardware such as Raspberry Pi[21].

The flows created in Node-RED are stored using JSON, which can be easily imported and exported for sharing with others. The concept generalises the applications of this software to any control system, including the automation industry. Based on the online repository, flows for Modbus and OPC UA communication are available from user contributions, making this programming tool as powerful as the others.

2.4.5 DINASORE

The DINASORE[13] is the acronym for Dynamic INtelligent Architecture for Software MOdular REconfiguration and is compliant with the 4DIAC platform that implements the IEC 61499 standard. The framework aims at executing digital twins in CPS while supporting the latest advances in machine learning [22].

The basic idea behind DINASORE is to distribute a pipeline of blocks across a set of computers or CPS from a local platform. All that is needed to run the DINASORE on a dedicated machine before you can start deploying your pipelines. Although it was initially designed for industrial applications, being based on Python, it can execute any Python code, taking advantage of the simplifications introduce by the IEC 61499 standard to better implement the algorithms [23].

The framework has an embedded implementation of OPC UA in the function blocks so that any data passing through them is automatically provided in a OPC UA server for better vertical system integration [22].

This framework is currently being used in the Information Systems (SI) course at FEUP.

2.5 Summary

This section gives a brief overview of two fundamental aspects, the communication between the lower level of the automation pyramid and the available simulators. The aim of this analysis was

[13] DINASORE https://github.com/DIGI2-FEUP/dinasore (visited on 07/06/2023)

to cover both the industrial and the educational vision.

For the simulations, as resumed in table 2.11, the Factory I/O as the benefit of already being in use, due to acquired licences. SimTwo and SFS are very straightforward simulators, with considerable flexibility to have made custom elements, such as machines, or different shop floor layouts, with the major of have been developed in the institution, so it has a more accessible support.

Regarding the automation discrete event system (ADES) for the control program implementation for the simulation scenes, CODESYS is widely used, given its multi-platform support.

The research carried out did not reveal any previous work with similar aims to this proposal. In this way, the educational tool will be created from scratch.

Table 2.11: Simulators summary

Simulator	Advantages	Disadvantages
Visual Components	Ready-to-use scenes Diverse pre-built models	No Modbus interface Windows only
Anylogic	Long runtime periods Accelerate time Free personal use license OS independent	No Modbus/OPC UA interface Closed simulation
FlexSim	MTBF/MTTR notions Educational oriented SoftPLC implementation	Registration need for trial
Simio	Digital Twin Industry 4.0 coverage Real-time features Educational oriented	Licences acquisition
Gazebo	Widely used in robotics Community support Several pre-existent libraries Open-source	No Modbus/OPC UA interface No Windows support
Robosuite	Extensible Open-source	No Modbus/OPC UA interface Out of I4.0 scope
Factory I/O	Already in use at FEUP (small learning curve)	No 3D models import Limited simulation resources Few piece types
SimTwo	Open-source Reusable Highly configurable Easily accessible support	No OPC UA interface Time-consuming for new scenes
SFS	Open-source Highly configurable Easily accessible support	No OPC UA interface 2D simulation Limited piece types

Chapter 3

SuperCoordinator architecture

This chapter presents the proposed solution for the SuperCoordinator application, taking into account the proposed framework within the educational side, the Industry 4.0 principles, and the objectives and constraints presented in the previous chapter.

Figure 3.1 gives an overview of the different SC components, organised in four main categories: connectivity, configuration, production and visualisation, and will be used as a guideline for the presentation of the solution details in this chapter.

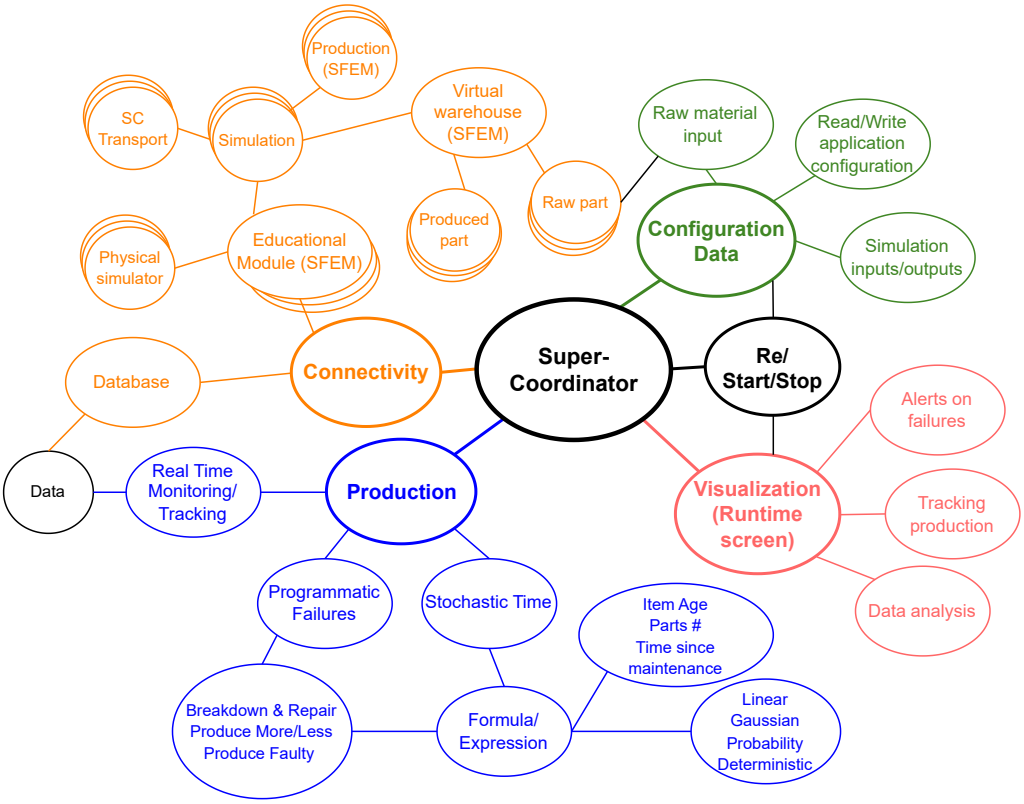


Figure 3.1: SuperCoordinator application overview

3.1 Problem Statement

3.1.1 Objectives

The SC main objectives can be grouped into four main categories: factory configuration, production monitoring and supervision, connectivity and visualization.

Factory configuration

- Support for importing I/O from a standard file format;
- Instantiate SFEMs, SFEEs and SFEIs (flexible and configurable);
- Configure special connection modules (transport modules);
- SFEE custom time definition (stochastic time dependency);
- "Real" factors definition [1];
- Configuration interface (user-friendly, simple and systematic);
- Save the configuration in plain text format for manual modifications and future reuse;

Production monitoring and supervision

- Real-time tracking (knowing at all times where each part is in the factory and its movements);
- Highlight asynchronous events ("real" factors);
- Long runtimes periods (data collection).

Connectivity

- Support for Modbus TCP/IP and OPC UA protocols;
- Database connection to store relevant data;
- Entry/Exit warehouse concepts to store parts from the virtual trucks (simplified version of the business-to-business models).

Visualization

- Simple and intuitive user interface, where the movement of parts and the events are visible;
- Dashboard for extracting indicators and evaluating data from the various configured factories.

[1] Human intervention or unexpected breakdown

3.1.2 Characteristics, Constraints and Engineering Decisions

During the project development there were some limitations, mainly related to licencing and free software restrictions, which influenced the work progress and had to be taken into account in the solution.

As explained in 2.3.7 Factory I/O has some restrictions. Another is that it does not support multiple instances of the same application per host machine. One possible alternative is to have virtual machines (VM) with the software, however this leads to an increasing number of licences being used for a common purpose. Another option is to have multiple machines connected on a local network, but the licencing issue remains. Since the number of licences at the institution is limited (based on floating seats), logouts could happen for some students during course time, which is undesirable. Having these aspects in consideration, an intermediate solution was implemented to minimise the impact of the SC. In a first step, to implement the connectivity between the modules, VM's running more Factory I/O applications were adopted. After that, it has chosen to include all the separate modules in the same simulation scene, leading to use only one licence of the available. The use of virtual machines can be more extensive, in the sense of creating a large factory that is partially simulated, with some real simulators interconnected, never forgetting that all the modules are interchangeable, that is, a fully simulated module can be replaced by a physical simulator. In order to build this factory, it is therefore necessary to have several hosting machines for each module. In this way, if you have enough processing power and enough screens to split the GUIs, a machine hosting many VMs can solve the problem.

Regarding the Ultimate version of the simulator, which supports the web HTTP server, it can receive higher level commands that overlap the instructions received by the communication protocol. The web server can be seen as a duplication of the simulation items variables, so that any changes to them are reflected in the simulation flow. From the SC's perspective, this is exactly what is necessary to condition SFEE's processing time and "real" factors introduction. Once again, this implies to having as many special licences as the number of desired modules to be connected, which is difficult to estimate and has an inherited cost associated as FEUP's licence is for the Modbus version.

In order to minimise the previous drawbacks, a solution was designed to keep the costs low by using the available resources, without the need to acquire new licences. For this, it is necessary to support it in CODESYS functionalities. Usually, to control a simulation it is necessary to map the inputs and outputs in ADES. At the runtime, the program constantly reads the inputs states and decides what outputs should turn on or off based on the program's logic. So far, this is what the students are usually asked to do in the course projects. For the SC operation, it is necessary to create a bridge for inputs and outputs, so that they can be duplicated in another Modbus TCP/IP connection (Figure 3.2).

In this way, the SC API will periodically check the state of the input, run the algorithm and write the corresponding outputs. Then, in the next PLC cycle, values from address area B are written to address area A, calling a special program unit to copy them manually. The factory

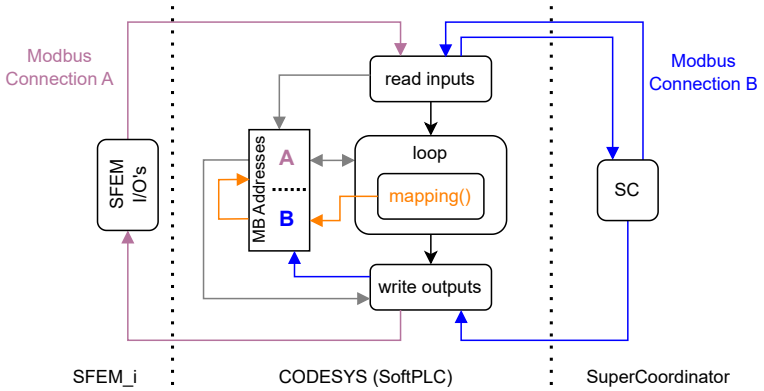


Figure 3.2: SFEMs - SoftPLC - SuperCoordinator interaction

control program should therefore be adapted to give higher priority to SC instructions, i.e. if the SC command it to stop a given conveyor and the logic has the instruction to move it, then the conveyor must obey the SC request. Thus, for each Factory I/O instance, it is mandatory to have a CODESYS instance too. This task should be performed by the SC admin, so it must not be in the domain of the students.

The CODESYS is a free software, available after a registration in the official website. Yet, the runtime for the Windows operating system has a timeout after two hours of being started. One of the objectives is to achieve long running times in order to connect as much data as possible from the hybrid factories. With this limitation, the maximum continuous time available period is two hours. In the end, it is necessary to restart the runtime, which means stopping the SC application, as the open connections are immediately terminated when the time expires. Unfortunately, no workaround for this constraint has been implemented, so that every two hours it is necessary to restart the operation of all system components.

Another strategic design decision to reduce the complexity of the solution was to consider only one entry point for parts into the factory. For multiple part types, a separation system is required to direct the parts to the correct position. However, there can be multiple exit points.

With regard to part manipulation, i.e. controlling the insertion/removal of parts within the production elements for the simulation, it was decided to place a set of Factory I/O emitter and remover components (as shown in Figure 3.7). This consideration allows the introduction of the referred events just simply by mapping more inputs and outputs, and does not require other higher functionalities that are only available in the Ultimate version. Admittedly, this scene change is necessary in order to allow SC working.

Due to the scalability, interconnectivity and interchangeability between modules, variety and flexibility required for the SuperCoordinator application and shop floor layouts, an object-oriented approach simplifies the solution complexity. The programming language chosen was JAVA. Moreover, JAVA has several frameworks for the development of graphical user interfaces, from the Abstract Window Toolkit (AWT) to JavaFX, among others, which meets with some of the objectives

of the proposal.

3.1.3 Physical module considerations

In the Department of Electrical and Computer Engineering (DEEC), at FEUP, exits a physical simulator of a real production line, an at scale miniature, named Staudinger compact flexible process line (more details in section 3.7.3 and in appendix A.3). However, at the time the compact line was built, a PLC was needed to run the model. Today, there are alternative solutions.

One of the SC's objectives is to interconnect the factory modules, whether real or simulated. Moreover, these modules should be interchangeable, so that it should be possible to switch from a physical simulator to its digital twin without too many changes from the SC point of view. In this way, the existing Staudinger process line was used in this work as the real module for development, testing and validation, after some updates to its older technology.

The SuperCoordinator requires a Modbus TCP/IP connection for each module. The compact line requires a control system that supports the physical wiring of the inputs and outputs. To meet these requirements, the solution represented in figure 3.3 was adopted.

The control logic is 24VDC due to PLC standards. It therefore requires an optocoupler isolation circuit for the inputs and a relay-based voltage boost circuit for the outputs. Moreover, it is required 2 external power supplies, one of 24VDC for the compact line logic and one of 5VDC for the relay module, as it consumes more current than the maximum Arduino output for that voltage. So the microcontroller cyclically reads the inputs and updates the outputs based on the packet received via the serial port (Modbus RTU), using the library libmodbus [2]. The source code for this task is available under A.1.

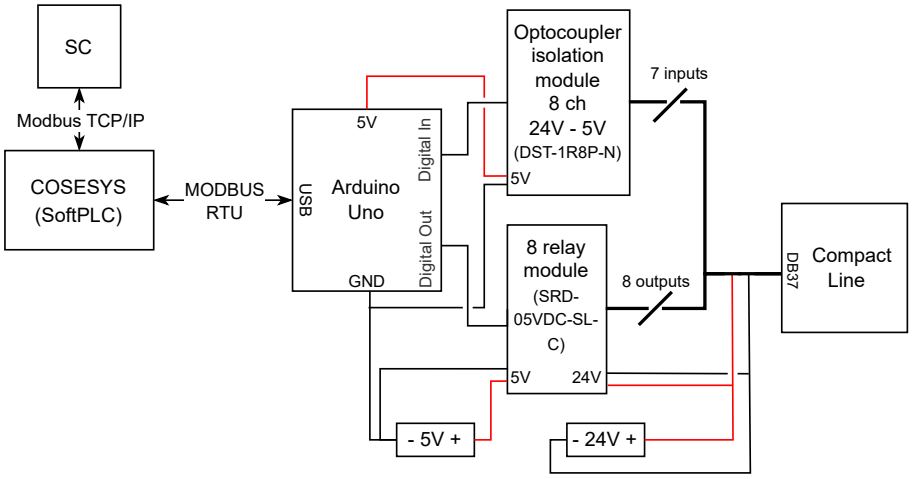
The connector on the physical simulator is a DB-37 male connector. Due to its age, there was no cable available with a female connector at one end and the wires at the other. However, there are only 19 pins in use on the connector (7 inputs, 8 outputs and 4 wires for power supply), so a cable was made by hand using a 25 wires cable and a new DB-37 female connector. The wiring mapping for this cable can be found in the appendix C.2.1. All the electrical connections for the physical simulator were made exclusively for this work.

3.2 Connectivity

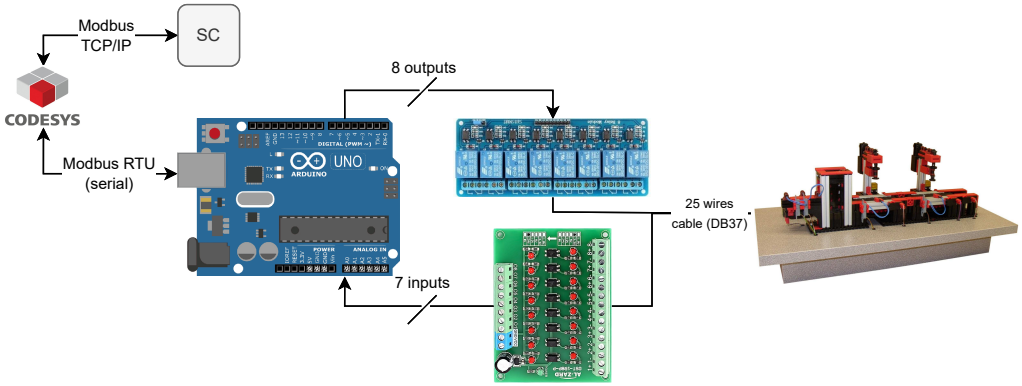
3.2.1 Educational modules

As previously introduced in subsection 1.1.1, there are three groups, SFEMs, SFEEs and SFEIs. Each has a role from an application point of view. The figure 3.4 presents the attributes and the relationship between them from a simplified perspective. Then some other particular classes will inherit the base class and add other relevant attributes for their function.

[2] libmodbus https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino (visited on 16/06/2023)



(a) Physical module wiring scheme



(b) Physical module circuit elements

Figure 3.3: Physical module electrical connections

The SFEEs have a crucial role because they register the elements inputs and outputs as well as the SFEIs that belong to them. The SFEIs are recorded within each SFEE as an organized list, respecting the order in which the components appear in their environment. The SFEIs, as the smallest in the hierarchy, are instantiated for each particular representation of a simulation or real component, such as the conveyors or the machines. The Java *Treeset* collection is used because it implements a sorted array according to the natural order of its elements, based on specified criteria, in this case, the part *id*. This helps in tracking parts within the factory, as the first part to enter an SFEI is also the first to leave.

Due to the role possibilities of SFEM, the generic class does not integrate the SFEEs because it depends on the generalisation (Figure 3.5). The *SFEM_production* class contains the Elements of the factory, typically groups of conveyors and/or machines, so it has a list, similar to the composition of the SFEIs in the *SFEE* class. The *SFEM_transport* and *SFEM_warehouse* are special cases: the transport module establishes the link between elements, so as regards the input and output elements there are several combinations, represented by the *configuration* enumeration ; the

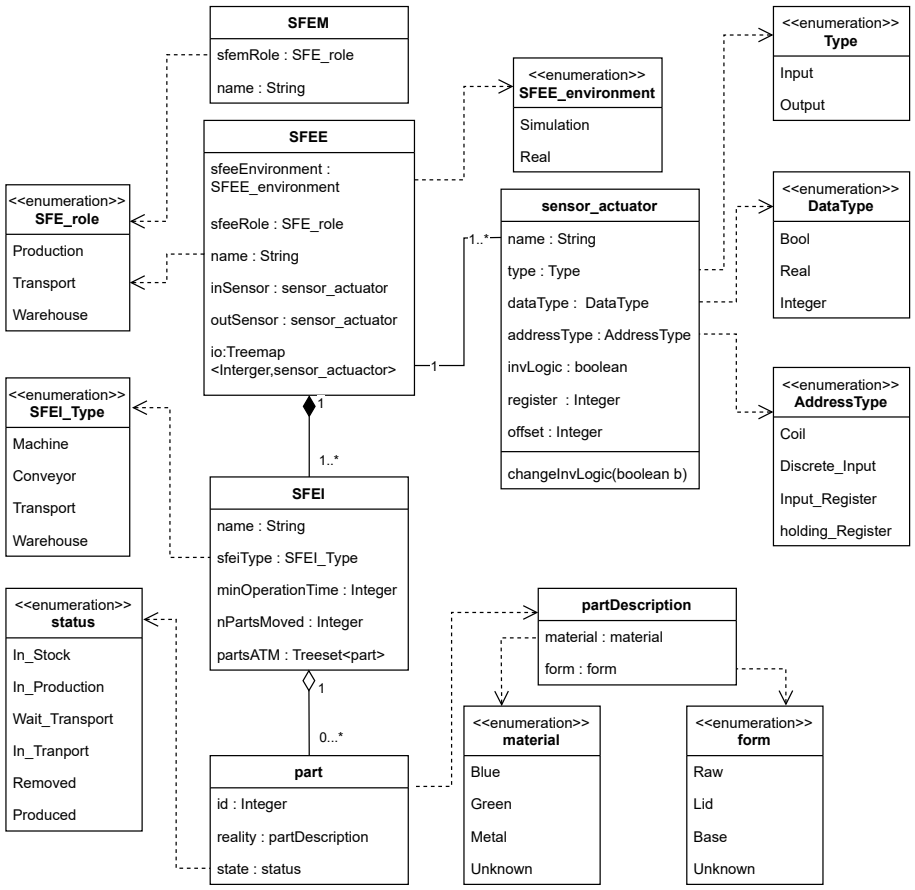


Figure 3.4: Modules, Elements and Items simplified (UML class diagram)

warehouse module has a configuration parameter that makes it possible to interfere in the parts' arrangement on the ordered array. If the option is random, then when new parts arrive at the factory, they will be stored randomly, ignoring their properties, otherwise if the option is sequential, the parts will be stored by colour, that is, a metal one, a green one, a blue one, until there are no more parts, and so on.

As for SFEI's, there are four specialisations (Figure 3.6), conveyor, machine, transport and warehouse, where the last two being special cases. Beyond the inherited attributes from the base class, some may need more specific attributes depending on the environment and/or the supposed operation mode (i.e. whether they implement "real" factors or not). Each SFEI needs an input and output sensor to know when a new part arrives and an old one leaves.

The *SFEI_conveyor*, beyond the conveyor actuator definition, can be configured with an Emitter and a Remover, both Factory I/O components (Figure 3.7), in the middle of the conveyor. By adding these elements, it is possible to interfere with the normal flow by adding or removing parts as pretended.

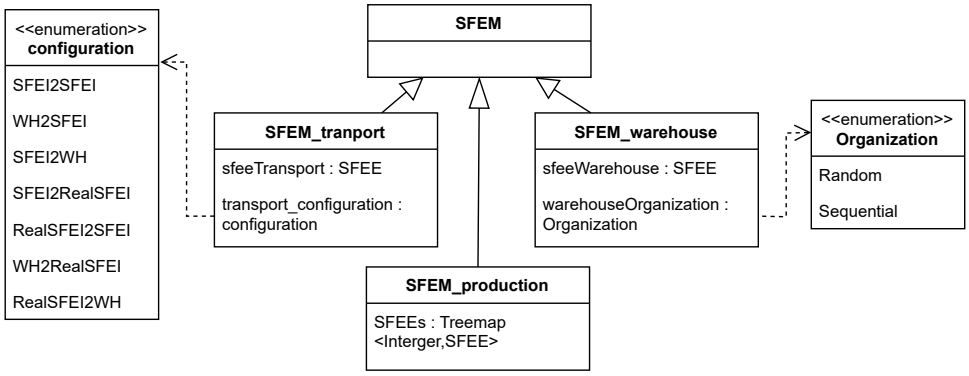


Figure 3.5: SFEM generalization (UML class diagram)

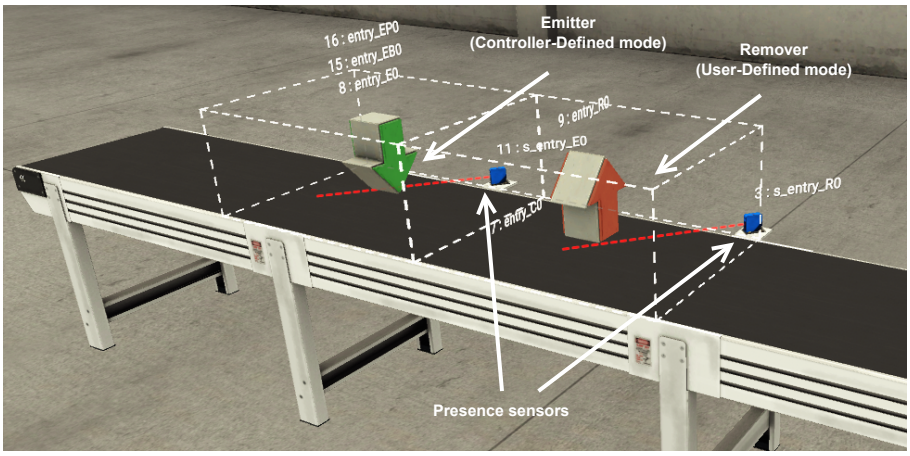


Figure 3.7: Factory I/O conveyor with emitter and remover components

The *SFEI_machine*, beyond the stop actuator definition, can be configured in relation to the Factory I/O machine centre component. For this purpose, it is necessary to associate a *partDescription*, which means what type of transformation is to be performed on each part colour, the sensor that indicates that the machine is working (sDoor) and the produce actuator, which defines if it is to produce a base or a lid.

The *SFEI_transport* connects any two A and B Elements within the factory. When both are simulated, there will be a Remover in the last Item of SFEE A (which is the transport module input item) and an Emitter in the first Item of SFEE B (which is the transport module output item). So, similar to the *SFEI_conveyor*, these attributes should be configured. If one of the input/output items of the transport module does not belong to the factory itself (it can be a real module or the virtual warehouse), then some adaptations are made and some attributes do not need to be specified.

The *SFEI_warehouse* generalisation is based on following the code hierarchy and pattern, because it does not need any other attribute. So, each instance will only store parts.

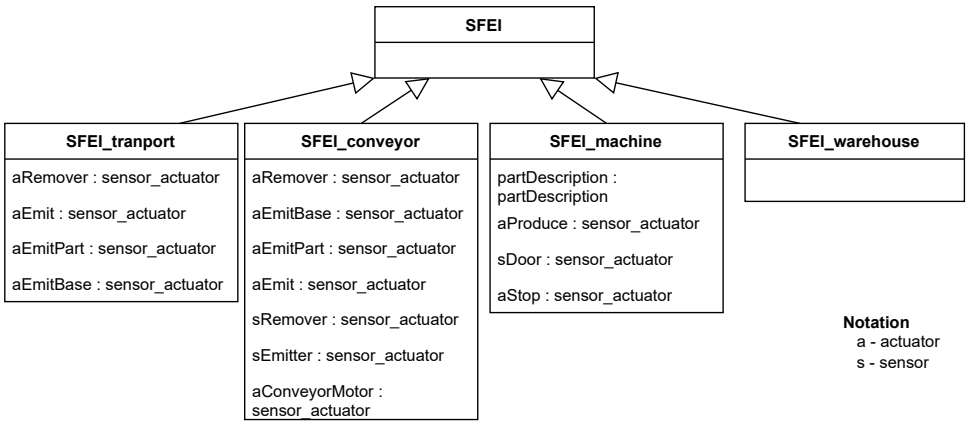


Figure 3.6: SFEI generalization (UML class diagram)

On the SuperCoordinator application, by default, there is only support for one *SFEM_warehouse* which has one Element with two *SFEI_warehouse's*, acting as an abstraction for an entry storage for raw materials arriving at the factory, and another for production results waiting to be shipped.

3.2.2 Educational elements connection

The user defines the Elements (and theirs I/O's) and the communication parameters. Since only Modbus TCP/IP has been implemented, using the library JAMOD,[3] only the IP, port and slave identifier are required. The SC will then consult all defined SFEEs before establishing the connections. In the case of duplicate parameters (IP and port), only one connection is open and it is shared between the elements. So a Modbus connection can be shared by n SFEEs. In other words, for each Modbus connection, a *Modbus* object instance is created, running on a dedicated thread, with a 50ms period, which implements PLC technique, i.e. reading inputs and writing outputs to and from local memory, respectively (Figure 3.8). This memory is thread-safe because it is based on atomic operations, so no race conditions. An atomic operation refers to a indivisible and uninterruptible operation that is performed as a single, coherent unit. It is a fundamental concept in concurrent programming and is used to ensure data consistency and prevent race conditions or conflicts when multiple threads or processes are accessing and modifying shared data simultaneously.

The SuperCoordinator is a real-time application because it monitors, supervises and intervenes in a real-time process. When using network communication protocols, there is an intrinsic delay between the client request and the server response. Therefore, in order to mitigate the impact on the application execution time, it is a good practice to split the operation into separate threads. With this approach, before the SFEE controller executes the control routine, it consults the input states and passes them to all methods within it, so that all see the values per execution cycle. At the end of the cycle, the controller writes to the specific Modbus thread memory.

[3] Java Modbus https://jamod.sourceforge.net/ (consulted on 08/06/2023)

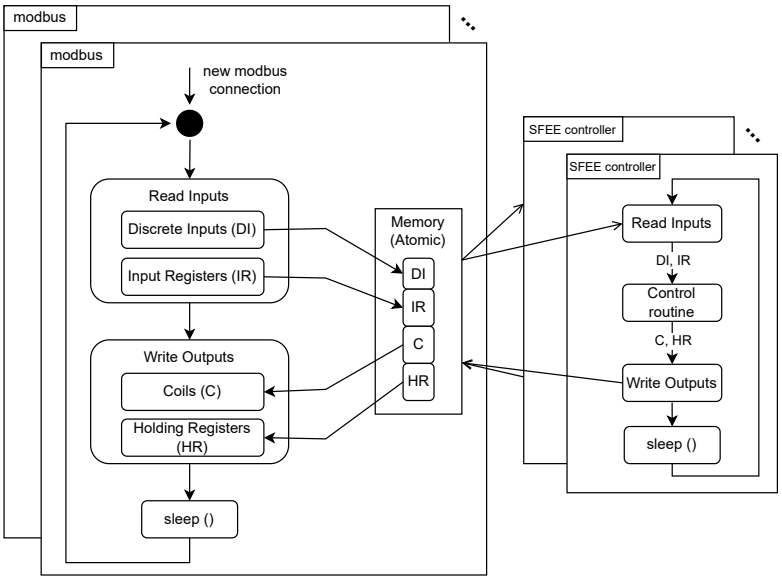


Figure 3.8: Modbus TCP/IP and Element threads interaction

In addition, to make the writing procedure more efficient, there is a trigger that detects when an output is modified, i.e. when the new value is different from the old one. When the trigger is activated, the write calls are executed, otherwise each Modbus thread cycle only runs the read methods, which can reduce the execution time.

3.2.3 Database connection

The database integration was supported using the JDBC[4]library. The database server was configured locally using the XAMPP[5] open source package, which contains MariaDB (a backward-compatible, enhanced version of MySQL).

The database creation is dynamic and associated with the factory configuration, provided they have different names. When the user creates a new shop floor layout in the SC application, they give it a name which is used as the database name on the server. The database is then created, the Data Definition Language (DDL) is executed and the tables are set (Figure 3.9). On the other hand, when loading an existing configuration, no changes are made to the *time_stamp* table, except for the *sf_configuration* field. This field is updated to always register the last access, with modifications, to this particular database. It also provides the SC applications with the number of parts stored in the warehouse, both with the status *Produced* and *In_Stock*, corresponding to the diagram 3.4.

[4]Java Database Connectivity https://docs.oracle.com/javase/tutorial/jdbc/index.html(consulted on 08/06/2023)

[5]XAMPP https://www.apachefriends.org/

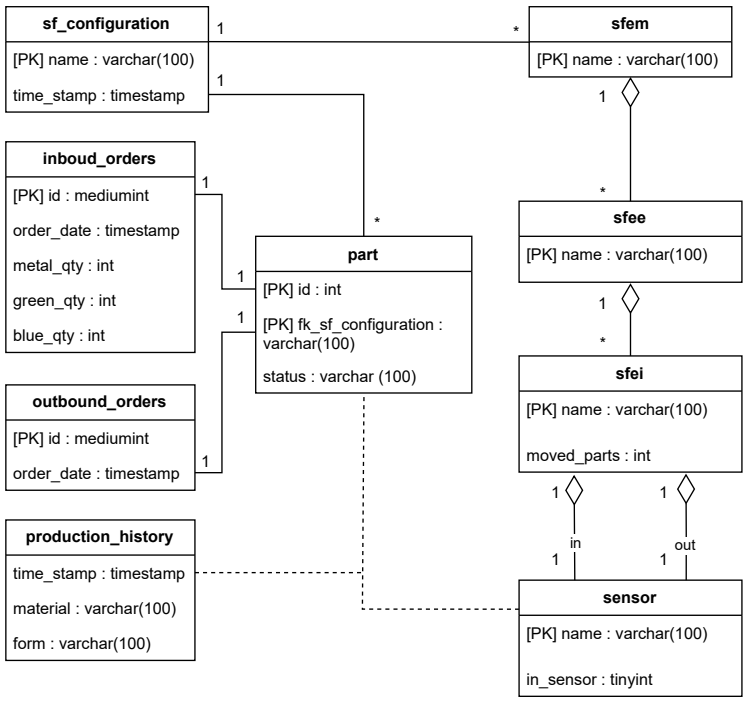


Figure 3.9: Database UML class diagram

The database integration is to record configured factory information, mainly when a part A passes in front of a sensor B. In this way, the database can have two parts, one for static information and the other for records during the run operation. The right part of the figure 3.9 is essentially made up of the static information, since it depends on the factory configuration (except for the *moved_parts* field, which is always incremented when a part passes the SFEI output sensor); the left part registers the dynamics during the operating time, as the arrival of new parts or the expedition of produced parts, or registers the instant, in the *production_history* table on the *time_stamp* field, when the part passes in front of the sensor.

The choice of this database model was an engineering decision in order to store the relevant data only with minimal complexity for the relational model. However, it has been designed to support future extension (as addition of new modules) without adding new columns or new tables, which is a simpler and less error-prone approach. For this reason, adding new elements to the database is equivalent to adding a new row, i.e. if a new module is added to the factory, the update to the database consists of a new addition to the table *sfem*. In addition, there are some specific cardinalities, such as the *sensor-sfei*, with two 1 to 1 connection to integrate both input and output sensors of each item. In addition, the *part-production_history-sensor* relation is critical as it stores important data for various analyses and indicators extraction for the SC viewers.

For the same reasons mentioned in subsection 3.2.2 regarding the delay introduced when there is communication between agents, the *dbConnection* instance was also dedicated to a periodic ex-

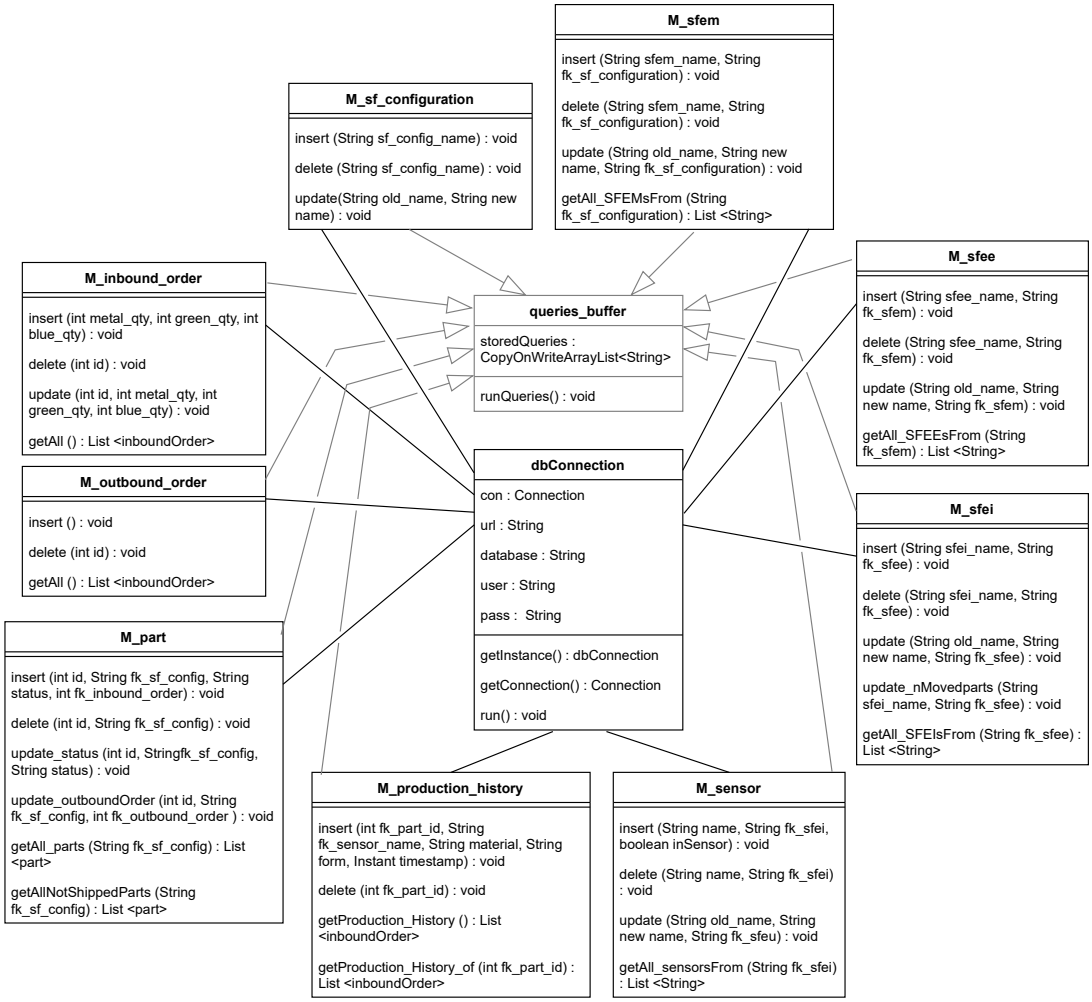


Figure 3.10: *dbConnection* class and relation with database tables mediators (UML class diagram)

ecution thread, with a period of one second. However, since this instance is unique, i.e. there are no multiple connections to databases, the singleton pattern was adopted to share it along all application objects. Thus, for each class shown in figure 3.10, a mediator class (prefix *M_*) was created that implements the CRUD (Create, Read, Update and Delete) operations for the corresponding table. In this way, the application objects that want to register some data in the database only need to get a dbConnection instance, select the object that refers to the relevant table and perform the appropriate operation. This operation will store the built query in the *storedQueries* array. Then, in the next *dbConnection* cycle (*run()* method), all queries in the buffer are executed.

A note should be made for the array type *CopyOnWriteArrayList* because it is thread-safe without the need for synchronisation. When using one of the modify methods (such as *add()* or *remove()*), the entire content of the *CopyOnWriteArrayList* is copied into the new internal copy. Because of this simple fact, it is possible to safely iterate over the list, even if there is a concurrent

modification.

In the appendix A.2 there are the DDL's regarding the table creation and deletion operations.

3.3 Communications intersection and responsibility sharing

As introduced in section 1.3, there are different actors with different roles for the SC application. So before getting into the details of the configuration, let's first clarify who the actors are and how they can interact with the software.

There are two users, the administrator and the viewer, with the administrator inheriting the viewer functions. Typically, in the educational scenario, the SC administrators are the automation course professors and the SC viewers are students. Depending on the automation course and its objectives, both the professor and the student may have other responsibilities (Table 3.1).

Table 3.1: SC actors and responsibilities (related with figure 1.4)

	Automation Stream		
Actors	Systems and Automation (SA)	Information Systems (SI)	Industrial Informatics (II)
Professor	Build a simple factory with one simulation cell	Generate mock data based on Gaussian distributions	- Replicate industry clients orders - Build a more complex simulation scene (similar to a real production line)
SC Administrators	Setup SC application by: - Specify SFEM's inputs and outputs - Provide connection parameters, for the Modbus TCP/IP and MySQL database connections - Perform the pretended configuration for the SC, by defining operation modes, custom times for the Elements, "real" factors, etc.		
SC Viewers	- Re/start and stop SC application - Access to real time interface - Consult data during runtime		
Student	- Implement the control program, applying IEC 61131-3 standard - Focus on the single simulation cell - Storage of simulation cell data to be available for analytics future use	- Data analysis (extrapolate indicators) - Dashboard creation (Grafana)to visualize and perceive the data - Focus on data intensive applications (IIoT domain)	- Build an ERP using OOP concepts - Build a MES using OOP concepts, able to interact with the ERP and the control program to meet the schedule - Focus on factory and logistics level control - Storage of simulation cell data to be available for analytics future use

As indicated in the table 3.1 and in the figure 1.3, the start consists in opening the different agents in the system, from the Factory I/O simulation scenes (that can coexist on multiples host machines), to the CODESYS applications, due to the relationship also referred before and the SC application. In the case of real modules, there must be a previous physical setup, carried out by the SC administrator, which ensures the logical and electrical wiring. The stop can occur from two sources, after two consecutive hours of operation, due to the CODESYS limitation or by the user.

In both cases, a manual restart is required, since the CODESYS runtime environment must reset the control program to its initial state. The restart is therefore very similar to an initial start.

The main change introduced by the SC in the automation stream is the reuse of data. This eliminates the need for the SI professors to generate mock data by giving the SC to the other actors to include in their work.

3.4 Configuration parameters

3.4.1 Configuration UI

The configuration interface has been implemented on the terminal. This means that users have to run the Java application from the terminal. The first screen is shown in figure 3.11.

```
****************************
***** SuperCoordinator *****
****************************

1 - New configuration
2 - Load configuration
> 1
Configuration name
>
```

(a) New configuration option

```
****************************
***** SuperCoordinator *****
****************************

1 - New configuration
2 - Load configuration
> 2
Path to XML file
>
```

(b) Load configuration option

Figure 3.11: SuperCoordinator configuration interface

For the first option, creating a new configuration involves a series of consecutive steps, as shown in the figure 3.12.

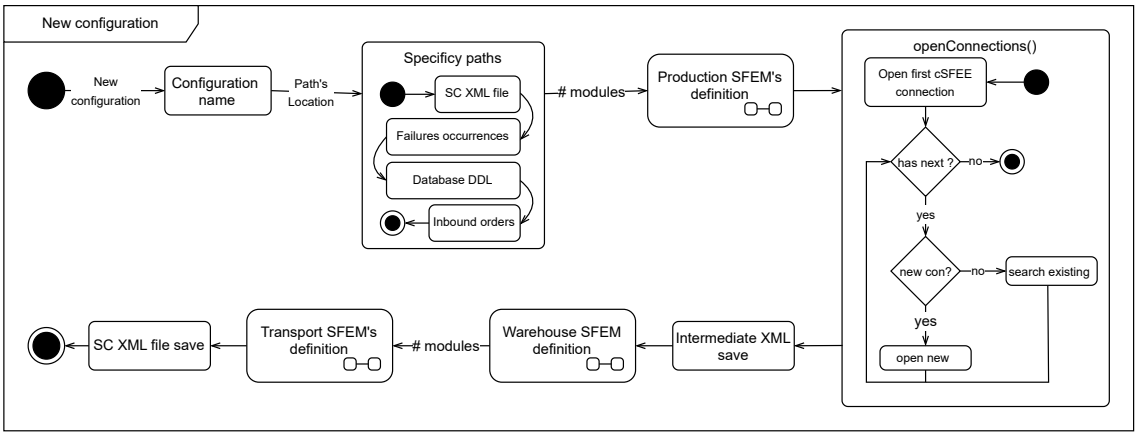


Figure 3.12: New configuration (UML state machine diagram)

First of all, a name and path for some folders are specified. The path for the SC XML file is the desired location for saving the generated file at the end; the path for the failures occurrences is for saving a specific XML file with the history of the failures (i.e. the "real" factors) that have

been occurred during this running period (the SC application always creates a new file by adding a sequential number at the end of the file name, regarding the number of existing files in this directory); the database DDL path is to have access to the creation and deletion DDL as detailed on 3.2.3; the inbound orders path refers to the location where the input orders are for the runtime. Without these last files, the factory will have no raw material in the warehouse, so it will not run as expected. It is important to highlight that the structure of the XML file (Figure 3.13) should not be modified, only the values, otherwise the application will generate an unrecoverable error that will immediately close the application.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inboundOrder id="0" metalQty="4" greenQty="3" blueQty="3"/>
```

Figure 3.13: Example of an inbound order XML file

Thereafter, the production SFEMs (in more details on subsection 3.4.1.1) are configured and the Modbus connections are established. Each SFEM has at least one SFEE, so the first connection is always opened. After that, the remaining Elements are checked for coincident connections (i.e. a Module with many Elements within the same Factory I/O scene, for example), as described in subsection 3.2.2. Then an intermediate save is made, simply because at that moment many Modules could be defined and it is always good to back up the data. Next, the warehouse and transport modules are defined, as described in the following subsections.

Finally, the configuration done is saved, before staring entering the runtime. XML was chosen as the file type, as it is easier to read and adapt manually. To do this, JAXB[6] library was used.

The second option, loading a previous configuration, is shown in figure 3.14.

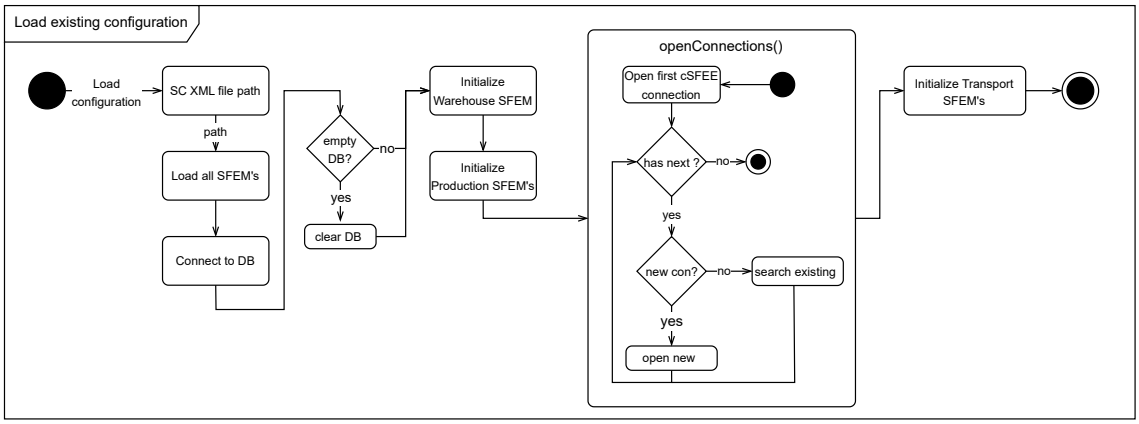


Figure 3.14: Load configuration (UML state machine diagram)

With the path for the SC XML file, it is possible to reverse the process of saving to a file using the same library. Then, after connecting to the database, it is possible to delete all the

[6]Java Architecture for XML Binding https://javaee.github.io/jaxb-v2/

data from previous runtimes. Initializing the warehouse SFEM depends on this decision, as it is the difference between having an empty warehouse or already having parts on it, which leads to the resumption of the process where it was stopped. The next step is to initialise the production SFEM, since this is where the connections parameters are saved in order to establish the Modbus connections with the modules. The last step must be the transport modules, as they need all the connections opened to move the parts between the SFEE's.

3.4.1.1 Production SFEM's definition

The production modules definition is iterative, within a loop limited by the number of modules to be defined. For each module, several elements can be instantiated and for each element, many items can be instantiated (Figure 3.15).

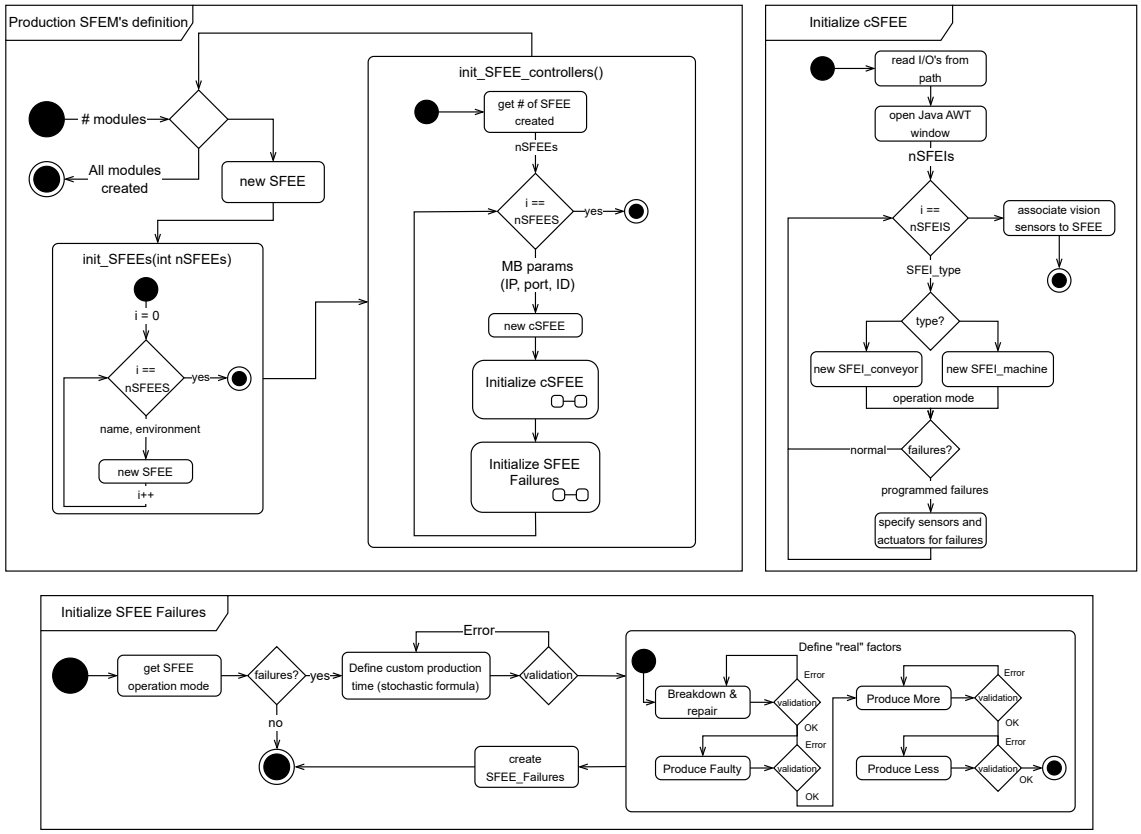


Figure 3.15: Production SFEM's definition (UML state machine diagram)

For each module, n SFEE instances having the name and the environment (real or simulation) attributes are created. Each SFEE must have a controller, that implements the methods and interacts with other controllers. The controller initialization is divided into three steps, the Modbus connection, the SFEI declaration and the possible failures support in case of "real" factors or custom production times implementation.

Starting with the items, they can be a conveyor or a machine, and in case of support of "real" factors and/or custom production time for the element where they are inserted (operation mode), they must have extra attributes setting. When all items of a SFEE are defined, it is necessary (only for the simulation items) to associate a vision sensor, to be placed on the exit conveyor of the element, to identifying the part material and form, and to update this particular part history.

To facilitate the input/output mapping during the definition, a window is opened for each element containing a table of all the imported I/Os from the given path. So, if a sensor or actuator is required, it is sufficient to specify the corresponding line number. The I/O import it is simply the loading of a CSV file (using the OpenCSV[7] library), usually exported directly from the Factory I/O simulation. In the case of real modules, a similar CSV should also be created.

The *SFEE_Failures* is an object of the SFEE controller that targets the custom production times and the "real" factors. The custom production times, sometimes called stochastic time, controls the time that a part takes since it enters and leaves the element. It is defined by a formula and has two possible operators, *Gaussian* or *Linear*. For both, the arguments passed can be an expression relating to runtime variables or not:

- n - number of parts moved on that run period;
- a - item age (in minutes);
- m - elapsed time since last maintenance (in minutes);

The generic formula shape is: $operator\ [\ X\ (;\ Y\ )\ ]$. The X is a mandatory expression field, unlike the Y field, which is only required if the operator is *Gaussian*. An example formula is:

$$gauss\ [\ 65\ +\ (\ 0.01\ \times\ n\ )\ ;\ 2\ +\ (\ a\ /\ 360\ )\ ] \tag{3.1}$$

or:

$$linear\ [\ 70\ ] \tag{3.2}$$

The blank space between characters is fundamental, as it is a restriction of the string parser used. The formula 3.1 is interpreted as the time of a part inside an element following a Gaussian distribution, with the mean value increasing by one second every 100 parts produced and the variation value increasing by one second every 6 hours (360 minutes). The formula 3.2 sets a linear value (that is constant for this case, but can be a linear dependency with any of the runtime variables) of 70 seconds per part produced. An important aspect is the intrinsic operating time of the items in their environment, i.e. if an 8 metre conveyor takes 8 seconds to move a part and a machine takes 33 seconds to perform an operation, an element with a series of a conveyor, a machine and a conveyor will not be able to have production times less than 49 seconds. These are the values that Factory I/O components take at the normal speed, so they are written by default in the SC XML file and can be changed manually.

[7] OpenCSV https://opencsv.sourceforge.net/

For the "real" factors, the approach is different. For each of the available options (breakdown with repair time, produce faulty, produce more or produce less), there is an expression for each runtime variable. There are 3 supported operators for the expressions: gaussian (*gauss*), probabilistic (*prob*) and deterministic (*det*). The SC administrator can associate up to 3 expressions for the same "real" factor, causing it to occur for different reasons. It is also possible not to declare any expression, simply by entering *no*. Unlike the stochastic time formula, where the formula parameters could be associated with the runtime variables, for these expressions all the parameters must be numeric, so each variable can have its own expression.

When the module is real, the configuration is slightly different as the programmed failure operation mode is not supported, so it will always work normally.

3.4.1.2 Warehouse SFEM definition

The warehouse module definition consists of three steps (Figure 3.16): define the organisation option, instantiate the controller based on the frequency to check new orders under the inbound orders directory and enumerate the items that connect to the warehouse. These items are the first item of the factory (where the raw parts will be placed) and the items at the end of the production lines.

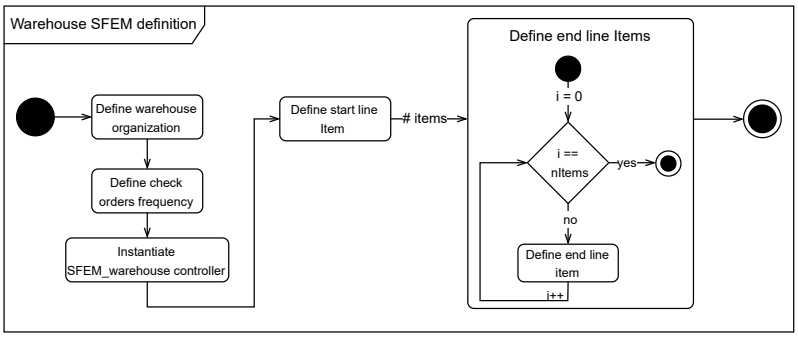


Figure 3.16: Warehouse SFEM definition (UML state machine diagram)

3.4.1.3 Transport SFEM's definition

The definition of the transport module is iterative and consists in enumerating an input and an output SFEI and the time needed for the transport (Figure 3.17).

The input is the end of the upstream SFEE, so it requires an output mapping independent of the environment. In the simulation case, a Factory I/O remover and corresponding sensor, in the real case, a sensor representing the disappearance of the part. An equivalent for the output Item (the start of the downstream SFEE), if it is a simulated conveyor, an emitter (on controller-defined mode) and corresponding sensor, if it is a real Item, the actuator that starts the movement in the station.

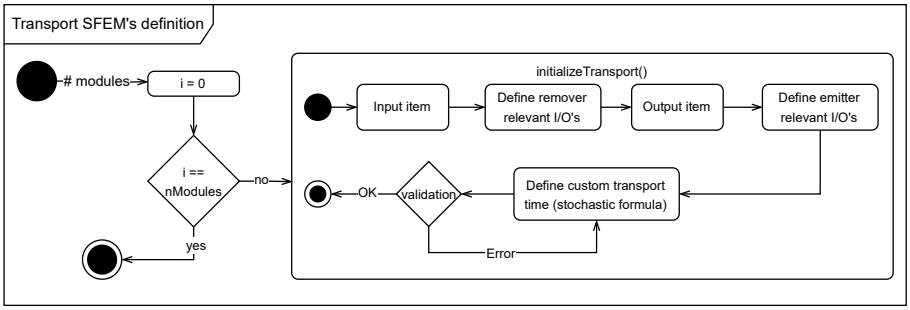


Figure 3.17: Transport SFEM's definition state diagram

The transport time has been implemented in the same way as the stochastic time in the production SFEM, taking advantage of the inheritance of the objects. Thus, it is possible to influence the transport time based on runtime variables as explained in subsection 3.4.1.1.

3.5 Production

The previous sections covered the configurations and setup, the offline part. From now on, it is the online part, the runtime scenario, that will be discussed.

As considered before, the SuperCoordinator must be flexible and extensible, to keep up with multiple events at the same time. To achieve this, the module controllers have been separated according to their functions, resulting in a multithreaded application, where each f the production and transport module controllers has its own thread, as does the warehouse module controller. Their executions are cyclic and variable. The production and transport controllers need to be more responsive, so they run at every 100 milliseconds compared to the 1 second period of the warehouse.

Each production SFEM controller (*cSFEM_production*) has multiples production SFEE controllers (*cSFEE_production*) under its responsibility. Each *cSFEE_production* (containing only one SFEE) can perform two functions, monitoring its SFEIs (*SFEE_Monitor* object) and interfering with the normal operation (*SFEE_Failures* object), if defined (Figure 3.18). Both methods of the *SFEE_Monitor* and *SFEE_Failures* objects are executed within the *cSFEE_production* periodic execution.

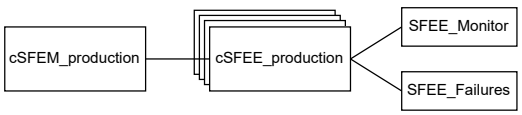


Figure 3.18: Production controllers interaction

The transport module controller (*cSFEM_transport*) follows the same production pattern, with a monitor and a failure object, but implements different methods, with only one element controller (*cSFEE_transport*) per module. The warehouse module controller (*cSFEM_warehouse*) has only one element controller (*cSFEE_warehouse*) and has only the monitor object, as it does not support interference with parts storage.

3.5.1 Monitor and Tracking

The *cSFEE_production* monitor main task is to monitor and track the parts' movement within the SFEE, i.e. to move instances of parts from SFEI to SFEI as they move around the factory. The number of possible layouts is large and requires a complex algorithm. For this reason, an algorithm 1 has been implemented that supports two shop floor layouts, serial items and parallel items with one item in common that feeds the others. In both cases, each item must have an input and output sensor, as illustrated in figure 3.19

(a) Series option (default)

(b) Parallel option

Figure 3.19: SFEE's layouts

By default, the SC uses the 3.19a layout. To change for the 3.19b option, during the SFEE configuration, it is sufficient to define the first item as ***lineStart***, as it appears on the XML configuration file. Similarly, and regardless of the layout, if an item is the ***lineEnd***, that is, after it the part is complete and should go to the warehouse, then it should also be defined during the SFEE configuration. This step is fundamental because it is the ***lineEnd*** items that change the part status to *Produced*, allowing it to be shipped in the future.

The *cSFEE_transport* monitor can have distinct roles regarding the transport configuration (as referred to in figure 3.5). For the warehouse-SFEI connection, it should remove parts from the warehouse and place them on the corresponding item, or vice versa for the SFEI-warehouse; for the SFEI-SFEI connection, it should remove parts from the input and place them on the output item. The operations performed by this controller are not registered in the database, unlike the production monitor.

The *cSFEE_warehouse* monitor has two main tasks, receiving and sending orders. Receiving orders is basically checking the inbound orders directory for new files, loading the parts in the virtual warehouse and registering the number of receive parts per colour on the database. Sending orders is the reverse process, i.e. creating a new outbound order record in the database and associating the parts with the *Produced* state to it. There is also a method that runs once at the beginning

Algorithm 1 Production element monitor and track algorithm

Input: sensorState
Input and Output: partState, partPos
Ensure: $isLineStart \leftarrow false$
for each SFEI from SFEE **do**
if first $SFEI\_idx$ AND $SFEI$ == LineStart **then**
$isLineStart \leftarrow true$
else if first $SFEI\_idx$ **then**
if $\uparrow inSensor$ **then**
$update\ production\_history\ DB\ table$
end if
if $\uparrow outSensor$ **then**
$movePartFromTo(PartPos,\ SFEI\_idx,\ SFEI\_idx\ +\ 1)$
$update\ production\_history\ DB\ table$
end if
else if before last $SFEI\_idx$ **then**
if $isLineStart$ **then**
if $\uparrow inSensor$ **then**
$movePartFromTo(PartPos,\ 0,\ SFEI\_idx)$
$update\ production\_history\ DB\ table$
end if
if $\uparrow outSensor$ **then**
$markForTransportFromTo(partState,\ In\_Production,\ Wait\_Transport)$
$update\ production\_history\ DB\ table$
end if
else
if $\downarrow outSensor$ **then**
$movePartFromTo(PartPos,\ SFEI\_idx,\ SFEI\_idx\ +\ 1\ )$
$update\ production\_history\ DB\ table$
end if
end if
else if last $SFEI\_idx$ **then**
if $\uparrow outSensor$ **then**
$markForTransportFromTo(partState,\ In\_Production,\ Wait\_Transport)$
$update\ production\_history\ DB\ table$
end if
end if
end for

of the application to load the warehouse with parts from possible previous executions, when the user selects the option *"Load configuration"* (Figure 3.11).

The possible connection of the SC with to an external resource planning software can be done exactly here, under the inbound order directory. The *receiveOrders()* method has been implemented to check the number of files in each run and select the next one, until reaching the end and restarting. So there are two options, one is to generate new XML files, always with the name *"orderX.xml"*, where X is the incremental order number; the other is, for example, to have only two files and change the values in them. From the perspective of the SuperCoordinator, both options are valid.

Figure 3.20 shows an example of the different states that a part has inside the factory, from

the arrival to dispatch. Moreover, it represents the SFEMs involved in the process, which the controllers have previously addressed. In the figure, it is represented the special cases when a part is removed or inserted inside the production, i.e. producing less and producing more "real" factors, respectively. When producing more runs, the inserted part has the status *In_Production*. The "real" factors are only available for production modules.

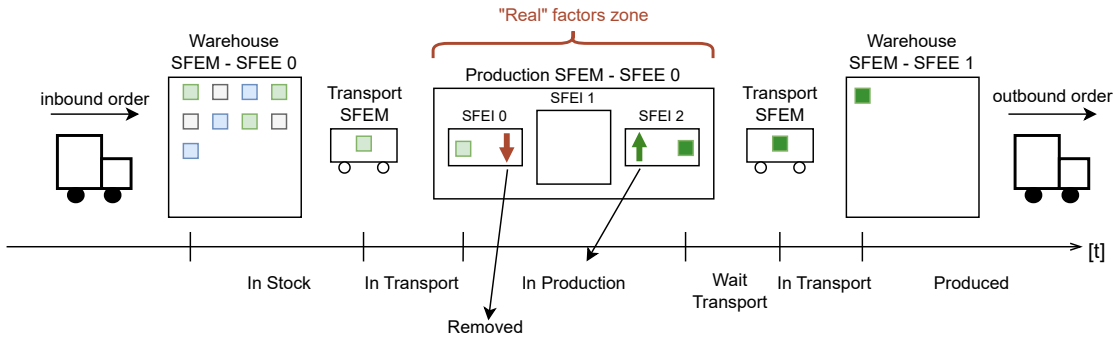


Figure 3.20: Part's status evolution

3.5.2 Custom production time and "real" factors

On the previous subsection, it was approached the monitor objects for all the modules controllers that perform objects instance's movement on memory and interact with the database. In this subsection, the Failure objects that interact with the control program (CODESYS) to execute higher level instructions are described in detail

During the configuration, the SC administrator can define whether the element supports failures or not. In the positive case, the user-defined production time (or stochastic time) is a must, while the "real" factors can be defined partially, completely or not at all. The stochastic time is associated to a formula, as presented in subsection 3.4.1.1, and belongs to the *SFEE_Failure* object.

When formula's operator is gaussian, there exists a fixed length array containing a discrete Gaussian distribution. In addition, a method is called that consists of a loop to calculate the *nextGaussian()* value, supported by Java Random class [8], and adjust it for the pretended parameters, as the method returns a double value with a mean of 0.0 and a standard deviation of 1.0. The Random object seed was set to **3587214**, in order to allow repeatability to the results. This parameter is also in the config file and can be changed if necessary.

So when a new part arrives at the element, the algorithm in figure 3.21 is run and a new *stochasticTime* task is created and stored in an array. The task only runs once, because it is unique to each part, as the delay and the SFEI chosen to introduce it are dynamic. To introduce the delay in the production line, only the conveyors are selected, as the part remains visible all the time.

[8] Java Random https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Also, the delay is introduced more or less in the middle of the item, when the part activates the remover sensor (Figure 3.7).

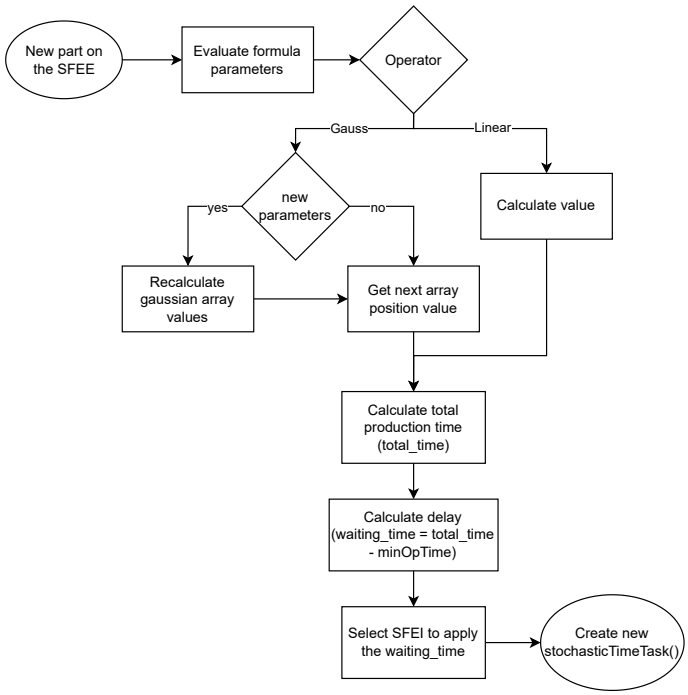


Figure 3.21: Stochastic time task creation fluxogram

The *stochasticTime* task algorithm (that runs for each task created, based in figure 3.21) is shown in the figure 3.22. At the end, the *SFEE_failures* object removes the executed tasks, i.e. those that are in the END state. The variable *waiting_time* refers to the calculated delay (in seconds) in the creation of the stochastic time tasks.

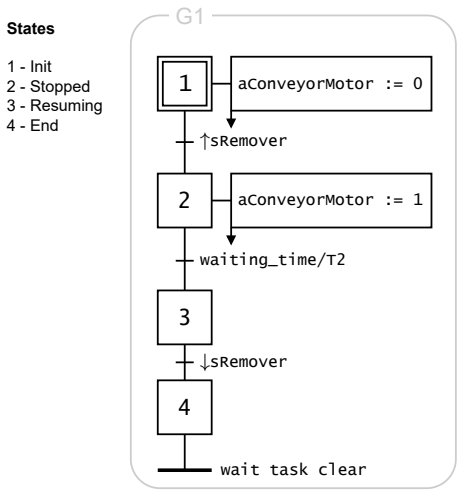


Figure 3.22: Production elements stochastic time task algorithm (GRAFCET)

The stochastic time presented so far is one state among others on the *SFEE_Failures*. In fact, it is the lowest priority state, i.e. it will only run if no other state is active. The other states are Breakdown and Repair (*BDwR*), Produce Faulty (*P_Faulty*), Produce More (*P_More*) and Produce Less (*P_Less*), in that particular order. Similarly to the selection of an element item to delay the production, for the "real" factors there is also a selected SFEI to implement it, done at the start of the application. All failures are executed on conveyors except the produce faulty, which implies injecting a specific error on the Factory I/O machine centre.

The transition between the states depends on each failure expression. So, before going into the details of each "real" factor, it is important to have an overview of expressions and their interpretation. In figure 3.23 is a "real" factor definition, only for demonstration purposes. The breakdown with repair failure can happen for 3 reasons, number of moved parts, age of the item or time since last maintenance, all related to the selected item.

```
BREAKDOWN WITH REPAIR
   BREAK
  > n : gauss [ 100 ; 5 ]
  > a : prob [ 360 ] >= 90
  > m : det [ 600 ]
   REPAIR (time, in minutes, since breakdown occurred): gauss [ 2 ; 1 ]
```

Figure 3.23: "Real" factor breakdown with repair definition example

For the first variable, and similarly to what happens with the stochastic time formula when the operator is gaussian, an array is stored with values of a gaussian distribution with a mean 100 and a deviation of 5. So when the number of parts is equal to the value on array position *i*, the conveyor stops for the time specified on the repair expression. After this time, the conveyor is "repaired", it resumes its normal operation and the next value will point to position *i+1*, and so on.

For the second variable, the normal interpretation is: the probability of breakdown after 6 hours since the machine was manufactured is greater than or equal to 90%. However, in terms of implementation, the value calculated is an integer from [0;100], based on the Java Random class and the *nextInt()* method. Then the first member of the expression is replaced by this number, turning the expression into a boolean result that determines whether the expression is verified or not. So the real interpretation is more like, the probability of breakdown after 6 hours (360 minutes) since the machine was made is less than or equal to 10%, because the boolean only takes a positive value if the number is greater than 90.

The last variable, has the deterministic operator, and it is interpreted as, the breakdown will occur at every 10 hours (600 minutes) since the last maintenance moment.

In the specific case of the breakdown, there is an additional expression for the repair time, in other words, the time that the item will be "broken". This expression is deliberately limited to the gaussian operator in order to introduce some variability, as is the case with the real word. For this example, the repair time will follow a gaussian distribution with a mean of 2 minutes and a variance of 1 minute.

As illustrated in figure 3.24, the process always evaluates the expressions for each failure type, one for each runtime variable as mentioned above, and whenever one is verified, the corresponding failure is triggered.

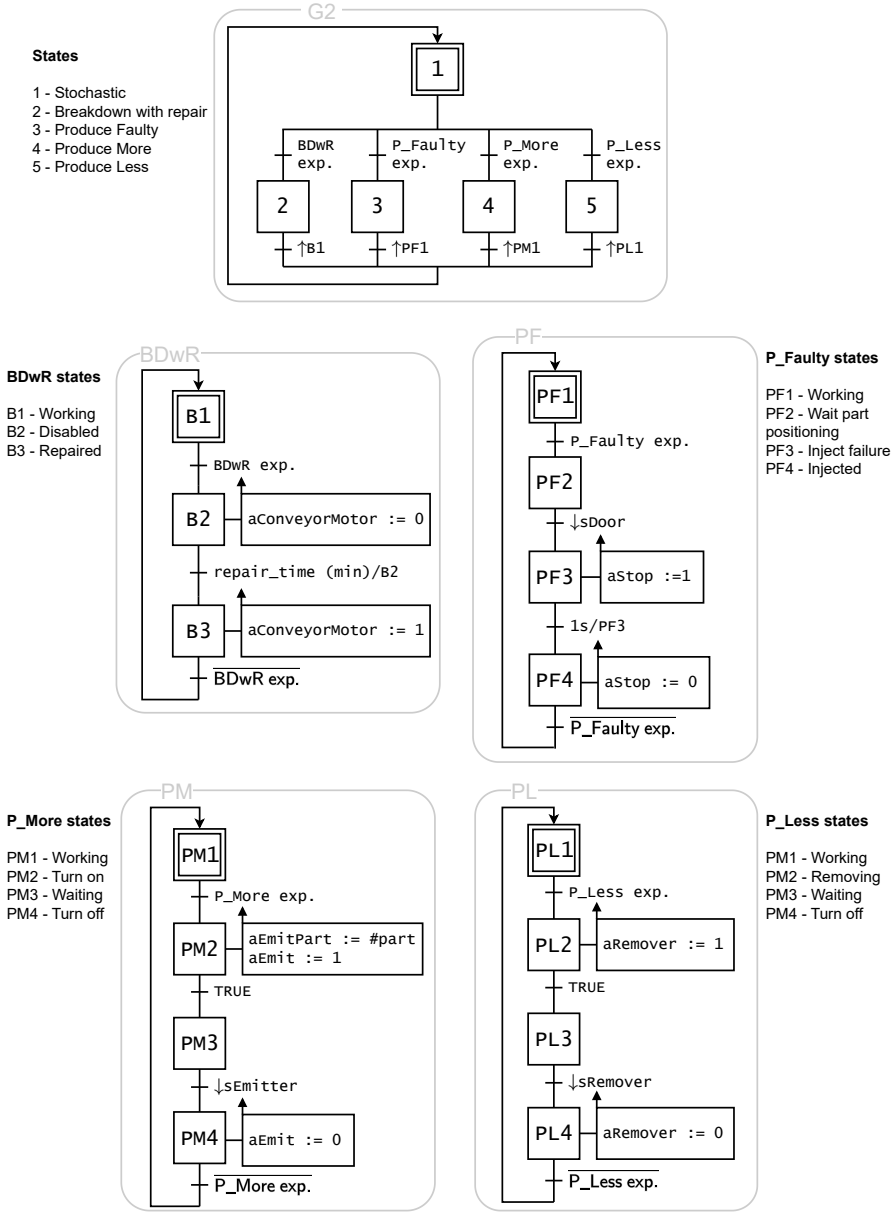


Figure 3.24: Supported "real" factors (GRAFCET)

This approach can lead to a situation where multiple expressions are valid on the same instance. In that case, the one with the higher priority will be executed, then in the next cycle the expressions will be evaluated again and the process repeated. So it may happen that a particular programmed failure is not executed as scheduled by the SC administrator. Considering a case where the produce more failure, can be triggered by two variables, a and m. The expression for

the item age is *det* [15] and for the time since the last maintenance *det* [30]. As 30 is a multiple of 15, the produce more parts failures will not be triggered by *a* if the elapsed time is a multiple of 30, because as a consequence it is also a multiple of 15. Another case can be that a higher priority failure coincides with a lower one. As expected, the lower priority failure will not occur.

Regarding the figure 3.24, there are some relevant aspects to highlight. In the case of produce faulty part, the failure is injected into the Factory I/O machine centre. Due to the simulator's intrinsic setup, if the goal is to stop the machine without ruining the part, it is mandatory to activate the stop bit after the transformation is complete plus a safe window time of 2 seconds, otherwise the part will leave the centre as it entered. In order to produce a faulty part, it is therefore necessary to stop the machine after it has just started the operation, i.e. after the door open sensor has passed from high to low level. In the case of produce more, it is necessary to command Factory I/O emitter component, on controller defined mode, which have 2 holding registers, one for the part and other for the base and 1 coil for to emit the part. For the SC, only the part register is important, so when it is necessary to emit a part, it is also necessary to write the number on the register, that codifies the material and form. Moreover, the part introduced inside the production line is not registered in the database, unlike the "normal" ones. However, it will be counted on the number of parts moved by the item, so by comparing the SFEI's of the SFEE it is possible to extract that something has happened. In the produce less failure, the part disappears from the production line, so it will be an incomplete *production_history* record for that particular part.

As previously explained, the *cSFEE_transport* has one *SFEE_Failure* object instance, and that is to control the transport time. The definition of the stochastic time for this element is very similar to the production definition, approached in subsection 3.4.1.1, so it follows an adaptation of the algorithm described earlier for the production elements (Figure 3.25), as there is no conveyor, instead there are emitters and removers to command, when the transport is from SFEI-SFEI. For the other cases, there are variants of the illustrated algorithm. For example, when the transport element input is virtual, there are no removers to operate, so the algorithm is only from state 3 to 5; the opposite case, when is the output, on this situation is taken the states from 1 to 3.

The first transition, from state 1 to 2, considers the presence of the part on the input item waiting for transport as well as confirming that it is the correct one by the part id and the id passed when at the stochastic task creation.

An important aspect related to the failures occurrences is the limitation to the 2 hours runtime period. The failures' execution relies on their expressions that use 2 time instants, the age of the machine and the time since the last maintenance. Both parameters are in the XML configuration file and both correspond to the time instant when the configuration was made. The cyclical execution of the failures is only monitored during the runtime, so when the application is closed, this information is store for an extra XML file (*failuresOccurrencesX.xml*) but is not recovered on the next execution. Thus, when the application is restarted, many expressions that depend on time will be triggered as the elapsed time makes the condition true. In this way, it is normal for multiple failures to occur when the SC application is restarted. One possible way to avoid this

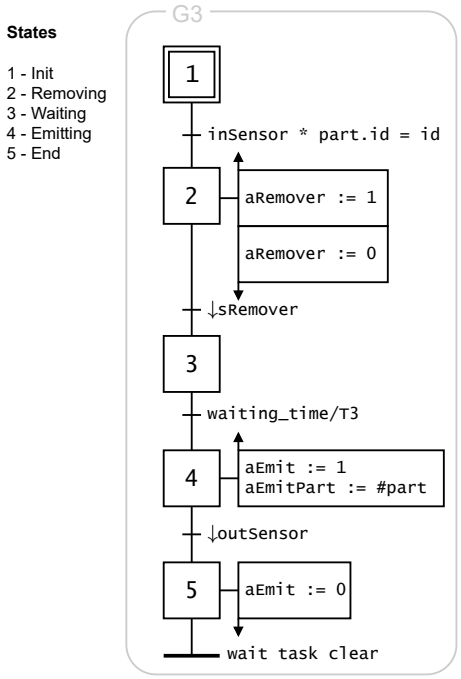


Figure 3.25: Transport elements stochastic time task algorithm (GRAFCET)

inconvenience is to manually update the time in the configuration file to the current time, which will make subsequent runs more like first runs.

3.6 Visualization

3.6.1 User Interface

The SuperCoordinator application has user interface objectives, both for configuration and runtime moments. Initially, the intention was to build a front-end interface based on JavaFX (JFX) to interact with the core application, that monitors, supervises and intervenes on the factory modules. The JFX UI was divided into 2 options for the configuration, creating a new one or loading an existing one. However, it turns out to be very time consuming compared to the results obtained. In the appendix B there are some developed frames.

Therefore, a more pragmatic solution was adopted: create a terminal interface for the configuration part (as presented in subsection 3.4.1) and create a dynamic UI for the runtime, based on Java AWT, to represent the flow inside the factory, give some information about the number of parts moved by items and highlight the failure occurrences.

So, when the SC user starts the application, a window similar to figure 3.26 is launched. Each white box represents an element; on the left side, there are the element items, in the information in the middle and the green border represents where are the part within the that same element; on the right side, there are the "real" factors supported, the grey colour indicates that a particular failure

was not defined, the black colour indicates that the failure was defined but is inactive and the red colour indicates that the failure is active.

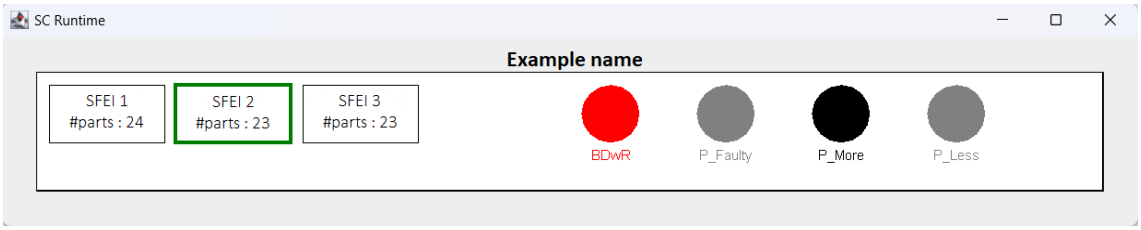


Figure 3.26: Runtime user interface example (adapted from a real configuration)

3.6.2 Data analysis dashboard

For the SC viewers, in addition to the runtime interface, it is also important to have a data analysis during at the same time, to confirm that the generated data matches with the configuration. For the specific case of the SI students, it is required to extract more indicators and to build a more complex and complete dashboard. The figure 3.27 illustrates a simple dashboard made on Grafana environment, with sample data from a test run.

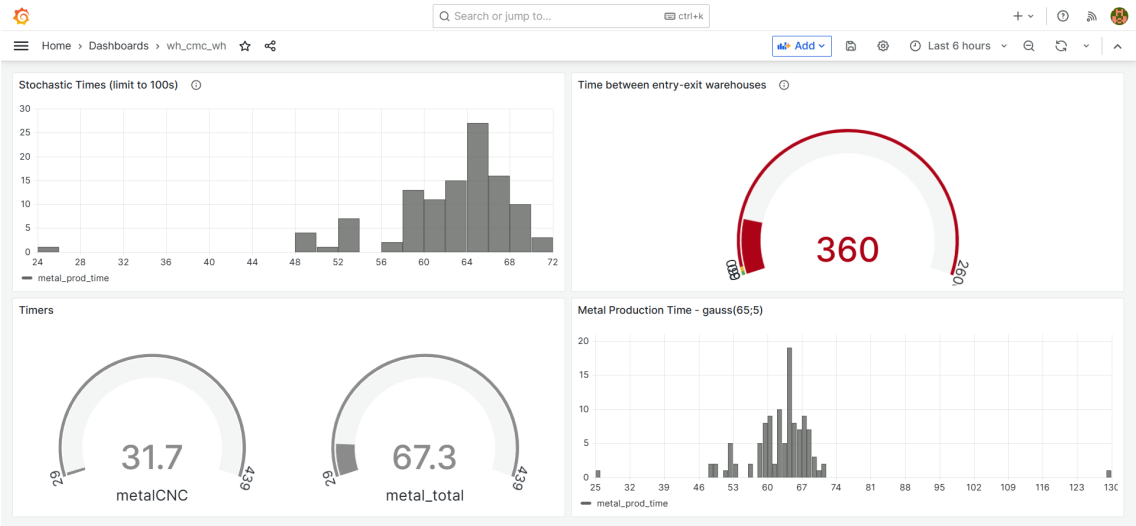


Figure 3.27: Grafana dashboard example

3.7 Tests and validation results

Although the Factory I/O has been used extensively for the development and testing of the simulation modules, the SC architecture has been designed and implemented in a scalable and standardised way to allow the application connection to be extended to other simulation environments

such as the SimTwo.

The creation of the control program for the modules it is not the aim of this proposal, but it was necessary to create different modules designed for the development of the application and the testing purposes. In this way, three simple modules were considered: the Sorting Station, which is a group of conveyors and turning belts to separates the parts according to their colours; the Staudinger miniature line, which consists of a series of a storage unit and two machines, connected by conveyors; and finally, the Conveyor-Machine-Conveyor, which is a series of a conveyor, a machine centre and another conveyor. Of these elements, only the Staudinger is a real module, the others were designed on the Factory I/O. Appendix C shows the GRAFCET's with the control algorithms behind the implementation executed under the CODESYS programming environment, essentially written on Sequential FlowChart (SFC) language, that belongs to the IEC 61131-3 standard.

This section presents the scenarios in which the SC application has been tested. Several aspects are checked to confirm that the objectives are met, such as:

- Definition and flexible configuration of the factory layout (stored in the saved file);
- SFEM's control and supervision and the connection between them (regardless of their environment);
- Customised production times on the elements (visible on the dashboard);
- "Real" factors events (confirmed by the failures occurrences generated file);
- Real-time monitor (using the runtime interface);

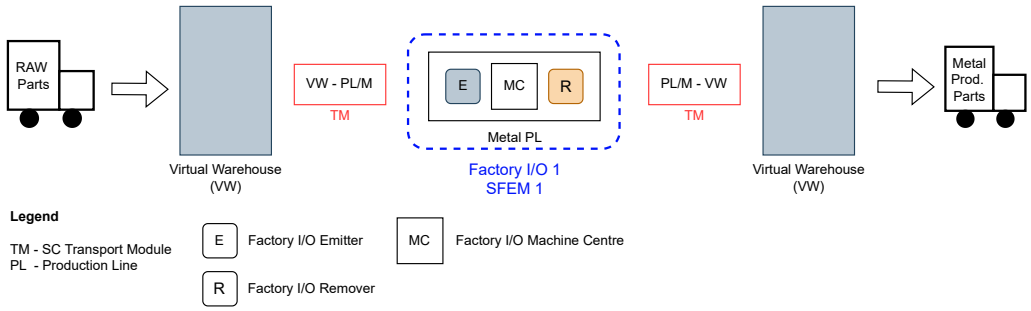
In this respect, 3 different scenarios were considered, with different factory layouts and sizes, with scalable complexity and possibly oriented to each course needs, in order to verify all the previous points and then to achieve the objectives of this proposal, as will be addressed in the next subsections.

Although not considered a test scenario, the connection between simulation modules hosted on different machines was tested and worked as planned. An example of this is the separation of the Factory 2 (described in subsection 3.7.2) into two modules, one with the sorting station and the other with the 3 production lines of type CMC.

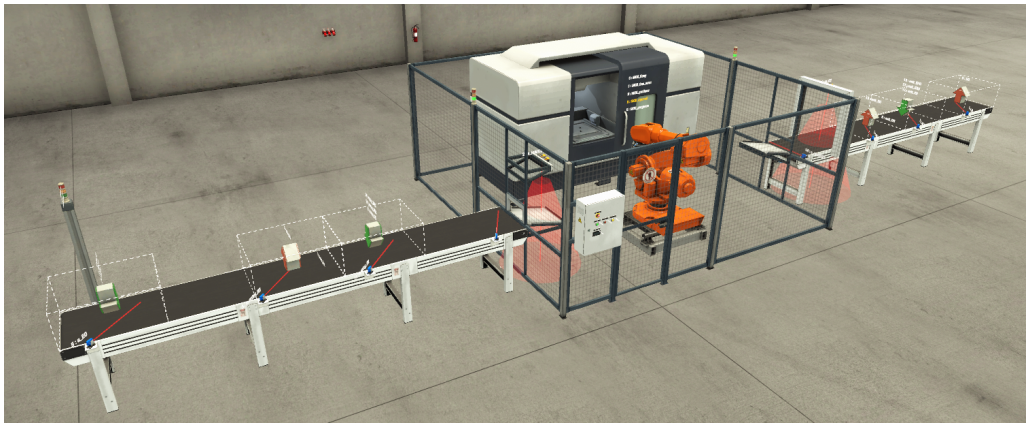
3.7.1 Factory 1

The first scenario is the simplest and is an example to validate the interaction between the warehouses and the production line of type CMC (Figure 3.28). The CMC production line consists of 1 SFEM, with 1 SFEE with 3 SFEIs.

The production (P) and transport (T) configuration requirements are defined in the table 3.2. For the warehouse module, the check for inbound orders has been set to a period of 5 minutes. The complete configuration file is available in the appendix A.3.



(a) Factory 1 layout



(b) Factory I/O CMC type module

Figure 3.28: Factory 1 layout and Factory I/O module

Table 3.2: Factory 1 stochastic time and "real" factors configuration

SFEE	Stochastic time	"Real" factors
(T) warehouse to production line	*linear* [2]	not supported
(P) metal PL	*gauss* [65 ; 5]	P_Less n :*gauss* [29 ; 3]
(T) production line to warehouse	*linear* [3]	not supported

The runtime was almost 4 hours. As only one element was configured, the runtime interface has only one region (Figure 3.29). From the dashboard (Figure 3.30), it is visible the gaussian distribution for the element operation time (duration between a part enter in SFEE's first item until it exists SFEE's last item), with $\mu \approx 65s$ and 95% of the data between $\mu \pm 2\sigma$. As the running time increases, more data is generated, so the curve becomes more like a gaussian distribution, filling the empty bins.

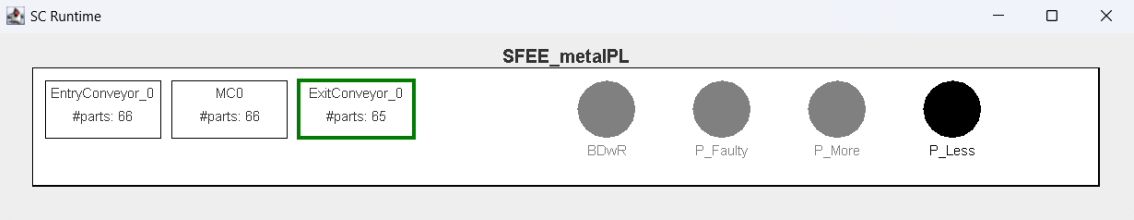


Figure 3.29: Factory 1 runtime interface frame

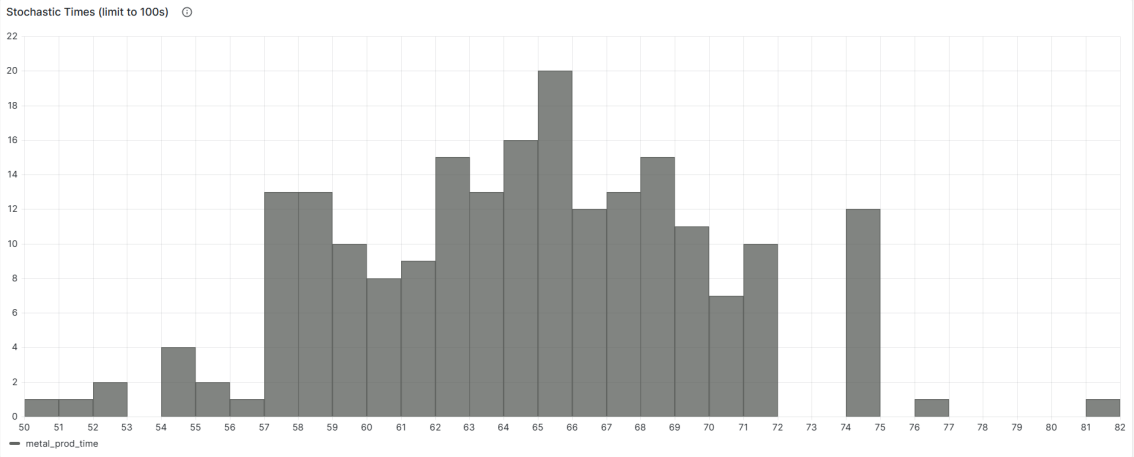


Figure 3.30: Factory 1 stochastic time histogram (group of all parts production time in seconds)

The "real" factor introduced in this element was only to produce less, i.e. to remove parts from the production line. The defined expression has the gaussian operator with $\mu = 29$ and $\sigma = 3$ and was related to the number of parts moved (n). As explained earlier in the 3.5.2 section, the runtime variables are reset when the application is restarted, so at least 1 restart is required for 4 hours. The figure 3.31 therefore contains a set of 2 files confirming that the programmed "real" factors occurred at the correct time, i.e. the difference between occurrences follows the defined gaussian (first at 29 moved parts, second at 28 and third at 24). In both cases, the failures occurred at exactly n because the seed that supports the method used has not changed, which confirms the repeatability of the results.

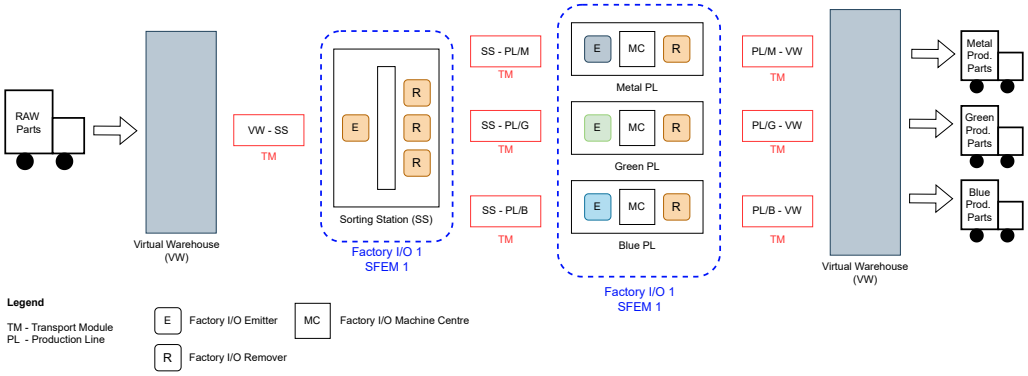
```
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="29" startT="2023-06-16T14:02:13.655154500Z" endT="2023-06-16T14:03:00.058603100Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="57" startT="2023-06-16T14:34:53.368144200Z" endT="2023-06-16T14:35:39.863609400Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="81" startT="2023-06-16T15:01:26.361556100Z" endT="2023-06-16T15:02:14.059015900Z"/>

<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="29" startT="2023-06-16T16:01:24.343987900Z" endT="2023-06-16T16:02:10.940476700Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="57" startT="2023-06-16T16:32:52.246347Z" endT="2023-06-16T16:33:38.939211600Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_LESS" nPartsMovedAtTime="81" startT="2023-06-16T17:00:07.547953800Z" endT="2023-06-16T17:00:55.437999850Z"/>
```

Figure 3.31: Factory 1 failures occurrences merged files

3.7.2 Factory 2

In the second scenario, the complexity increases. In this case, the aim is to test the SC application with a cell that has parallel elements, such as the sorting station, beyond the transport between the elements and the warehouses. In addition, there are more production lines, to transform parts of different colours. This scenario (Figure 3.32) is embedded in the same Factory I/O scene (to simplify the simulation) but has one module, the Sorting Station (SS), which has an element with 4 items representing 4 conveyors; and the production zone, which has 3 CMC type elements, the metal, green and blue production lines (PL).



(a) Factory 2 layout

(b) Factory I/O Sorting Station plus CMC's elements

Figure 3.32: Factory 2 layout and respective Factory I/O module

The SS role is only to separate the parts by colour, so the emitter-remover pair has not been set to support any "real" factors in this element. This element will therefore operate normally and the SC will only monitor its operation.

In this way, the configuration requirements (Table 3.3) are only for the production (P) and transport (T). From these configurations, a particular attention is paid to the blue PL, which has an average increase of 1 second per 10 parts moved. For the warehouse module, the check for inbound orders has been set to 5 minutes period. The complete configuration file is available in the appendix A.3.

Table 3.3: Factory 2 stochastic time and "real" factors configuration

SFEE	Stochastic time	"Real" factors
(T) warehouse to SS	*gauss* [5 ; 1]	not supported
(T) SS to metal, green, blue PL's	*gauss* [2 ; 1]	not supported
(P) metal PL	*gauss* [65 ; 4]	BDwR a : *det* [50] repair : *gauss* [1 ; 0] P_Faulty n : *prob* [35] >= 50 a : *gauss* [23 ; 5] P_More n : *det* [19] P_Less n : *det* [31]
(P) green PL	*gauss* [65 ; 6]	BDwR n : *gauss* [20 ; 5] m : *det* [30] repair : *gauss* [1 ; 0]
(P) blue PL	*gauss* [70 + (0.1 × *n*) ; 2]	P_Faulty n : *det* [23] P_More n : *gauss* [45 ; 5]
(T) metal, green, blue PL's to warehouse	*linear* [5]	not supported

At this time, the runtime was larger (about 15 hours) compared to Factory 1 because there were many more defined events to happen, besides it is one of the goals of the SC. The figure 3.33 represents a frame of the runtime interface during this period, which coincided with a "real" factor on the blue PL. For this reason, there are multiple failures occurrences files, as there were several restarts of the application (about every 2 hours).

With respect to the custom production times, the expected result is 3 Gaussians, 2 of which are centred at 65, but the metal gaussian will be narrower than the green one due to the different deviations. For the blue one, as the mean value increases, the resulting distribution is an overlap of multiple Gaussians centred at different values. Thus, after running the factory 2 configuration for 15 hours, the resulting distribution is shown on figure 3.34. The time axis is limited to 100 seconds to remove the outliers due to "real" factor events.

To validate the results, one file was selected from the available under the directory. The full contents are available in the appendix A.3. However, only a few lines were selected in order to validate the failures incidents (Figure 3.35).

For the first element, the metal PL, all kinds of "real" factors were set. Concerning the breakdown with repair event, there were 2 register events activated by the runtime variable *a*, separated in time by about 50 minutes (from 11:35:13 to 12:25:14 of *2023-06-02*). In both cases the repair was done about 1 minute later. The produce faulty parts' event could be caused by 2 variables, *n* and *a*. There was a probability expression related to the number of parts moved, so after 35 parts it

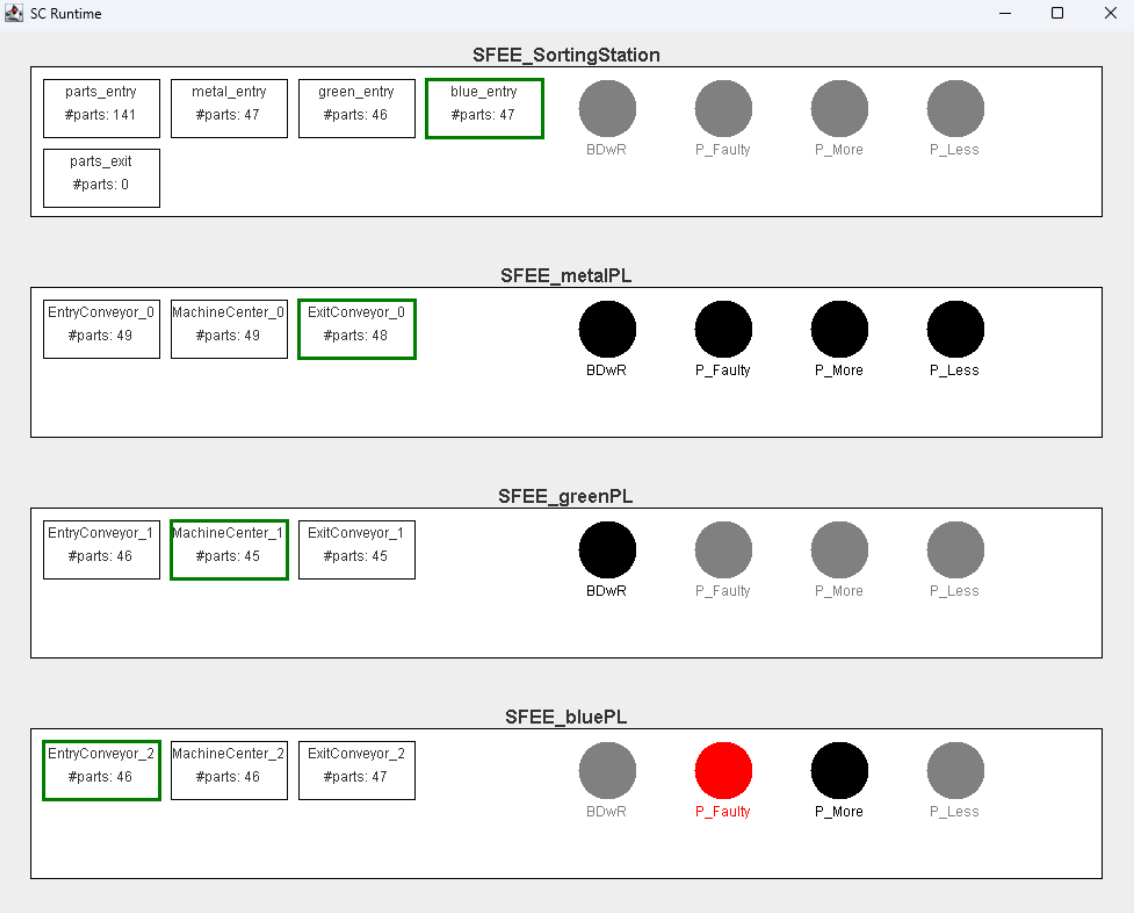


Figure 3.33: Factory 2 runtime interface frame

had a 50% change of happening. According to the results, it did not happen with part number 35, but happened with part number 70. For the age of the equipment, a transformation was schedule to fail according to a gaussian expression with $\mu = 23$ and $\sigma = 5$. In fact, the elapsed time between these 2 events was about 24 minutes, which corresponds to the prediction. For the production of more parts than planned, the event was correlated with the number of parts moved and, the records are consistent with the definition, with multiple occurrences separated by 19 parts. Unfortunately, there were no records for produce less failures. There are 2 possible reasons, first, as it is the less priority, all the attempts to execute were taken by other events, or, there could exist an error on the algorithm implementation.

For the other elements, the analysis is similar. Overall, all the "real" factor events that were carried out occurred as planned, with the exception of produce less.

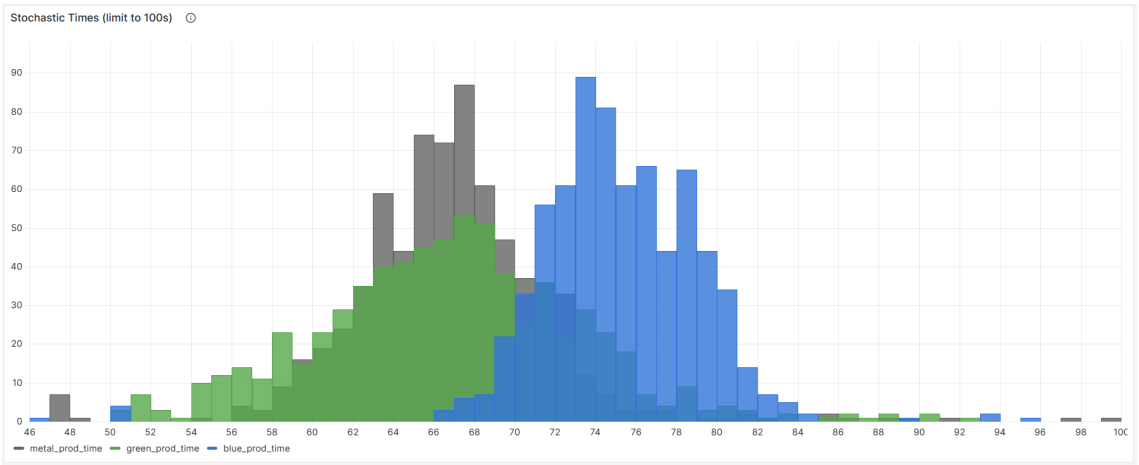


Figure 3.34: Factory 2 stochastic time histogram (group of all parts production time in seconds)

```
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_MORE" nPartsMovedAtTime="19" startT="2023-06-02T11:12:57.066905300Z" endT="2023-06-02T11:12:58.266570700Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="A" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="37" startT="2023-06-02T11:35:13.859894100Z" endT="2023-06-02T11:36:13.865062600Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_MORE" nPartsMovedAtTime="38" startT="2023-06-02T11:36:43.261186100Z" endT="2023-06-02T11:36:44.554552200Z"/>
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="N" failureType="PRODUCE_MORE" nPartsMovedAtTime="57" startT="2023-06-02T11:58:19.364840300Z" endT="2023-06-02T11:58:20.567671100Z"/>
(...)
<failuresOccurrences SFEI="EntryConveyor_0" activation_variable="A" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="79" startT="2023-06-02T12:25:14.360778800Z" endT="2023-06-02T12:26:14.469058900Z"/>
(...)
<failuresOccurrences SFEI="MachineCenter_0" activation_variable="A" failureType="PRODUCE_FAULTY" nPartsMovedAtTime="26" startT="2023-06-02T11:23:00.364908900Z" endT="2023-06-02T11:23:01.469061Z"/>
<failuresOccurrences SFEI="MachineCenter_0" activation_variable="A" failureType="PRODUCE_FAULTY" nPartsMovedAtTime="55" startT="2023-06-02T11:57:20.966611800Z" endT="2023-06-02T11:57:22.056063700Z"/>
(...)
<failuresOccurrences SFEI="MachineCenter_0" activation_variable="N" failureType="PRODUCE_FAULTY" nPartsMovedAtTime="70" startT="2023-06-02T12:15:53.463842600Z" endT="2023-06-02T12:15:54.555052100Z"/>

(...)
<failuresOccurrences SFEI="ExitConveyor_1" activation_variable="M" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="4" startT="2023-06-02T10:55:30.857341600Z" endT="2023-06-02T10:56:30.966281Z"/>
<failuresOccurrences SFEI="ExitConveyor_1" activation_variable="N" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="29" startT="2023-06-02T11:25:03.468335800Z" endT="2023-06-02T11:26:03.760151300Z"/>
<failuresOccurrences SFEI="ExitConveyor_1" activation_variable="M" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="29" startT="2023-06-02T11:26:04.368836700Z" endT="2023-06-02T11:27:04.867203800Z"/>
<failuresOccurrences SFEI="ExitConveyor_1" activation_variable="N" failureType="BREAKDOWN_WITH_REPAIR" nPartsMovedAtTime="44" startT="2023-06-02T11:44:20.061990800Z" endT="2023-06-02T11:45:20.168585900Z"/>

(...)
<failuresOccurrences SFEI="MachineCenter_2" activation_variable="N" failureType="PRODUCE_FAULTY" nPartsMovedAtTime="23" startT="2023-06-02T11:20:03.463749200Z" endT="2023-06-02T11:20:04.566076300Z"/>
<failuresOccurrences SFEI="MachineCenter_2" activation_variable="N" failureType="PRODUCE_FAULTY" nPartsMovedAtTime="46" startT="2023-06-02T11:48:56.665431800Z" endT="2023-06-02T11:48:57.766627300Z"/>
(...)
<failuresOccurrences SFEI="ExitConveyor_2" activation_variable="N" failureType="PRODUCE_MORE" nPartsMovedAtTime="36" startT="2023-06-02T11:35:42.667511800Z" endT="2023-06-02T11:35:45.963426900Z"/>
<failuresOccurrences SFEI="ExitConveyor_2" activation_variable="N" failureType="PRODUCE_MORE" nPartsMovedAtTime="75" startT="2023-06-02T12:25:29.967647900Z" endT="2023-06-02T12:25:33.057846900Z"/>
```

Figure 3.35: Factory 2 failures occurrences simplified file

3.7.3 Factory 2 plus physical module

Initially, the pretended factory layout for correctly testing all the SuperCoordinator functionalities, since connecting modules (simulation-simulation or simulation-real), passing through exception behaviour injection, up to monitoring all element types (as an assembly line, for example) is represented in figure 3.36.

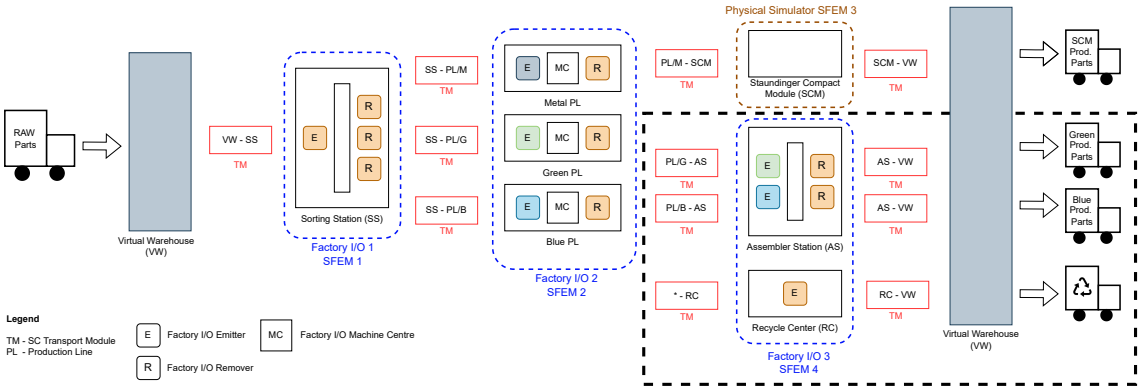


Figure 3.36: Complete factory layout

However, due to time constraints, it was not possible to build the Factory I/O assembly line, nor its control algorithm. Therefore, direct SC support for this type of element was not implemented or tested. Instead, a simplified version of the layout was tested (the complete factory without the black box hatched parts) to ensure the connection between the different environment modules.

This test scenario consists of the previous one (this time without "real" factors) with the physical simulator (see 3.1.3) in series with the metal production line. As with the previous tests, the full configuration file is also available in A.3. To recall the last paragraph of subsection 3.4.1.1, the real modules have no support for custom production time or "real" factors, so they run as programmed. However, compared to Factory 2, there are 2 new SC Transport modules to provide the link between the simulation and the real module. For the first transport SFEM, that connects the metal PL to the physical simulator (as known as MC Staudinger), the stochastic time expression was *gauss* [5 ; 2]; for the second, which connects MC Staudinger to the virtual warehouse, the stochastic time was similar to the previous one, *linear* [5] . For the warehouse module, the check for inbound orders was set to a period of 5 minutes.

An important aspect is the need for 2 hosting machines to perform this test. To control the simulation module and the physical simulator, 2 SoftPLCs are required, which implies 2 CODESYS instances. Thus, the host machine ran the SC application and the control of the real station, and a virtual machine ran the simulation environment.

The MC Staudinger only has a storage for 3 parts, so a manual process is required to move the parts from the end to the storage unit. For this reason, the runtime (about 30 minutes) for this

scenario was shorter than the previous one. Figure 3.37 shows a frame from the runtime interface for this test factory.

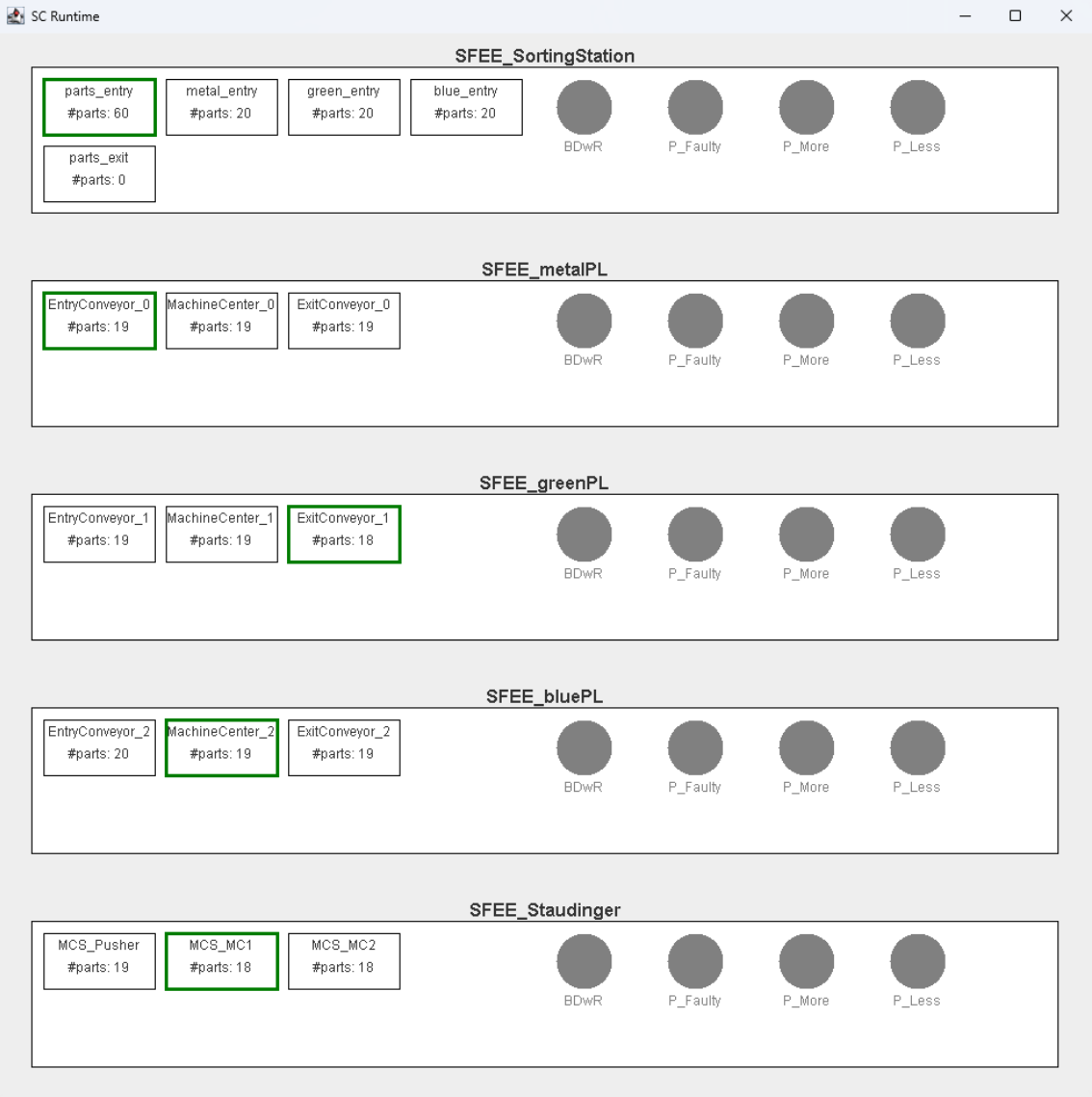


Figure 3.37: Factory 2 plus physical simulator runtime interface frame

The validation procedure is also different as there are no failure records. Therefore, after 30 minutes of work, the production time for each module is shown on the following dashboard 3.38. Naturally, the gaussian curves are raw due to the small number of samples recorded. However, the point of this test was to connect modules from distinct environments, and the purple curve refers to the physical simulator. Theoretically, the production time for this should be constant, but as it depends on human intervention, it is difficult to ensure that the part is always picked at the same instant. In the appendix A.3 there is a demo video of the whole factory operation.

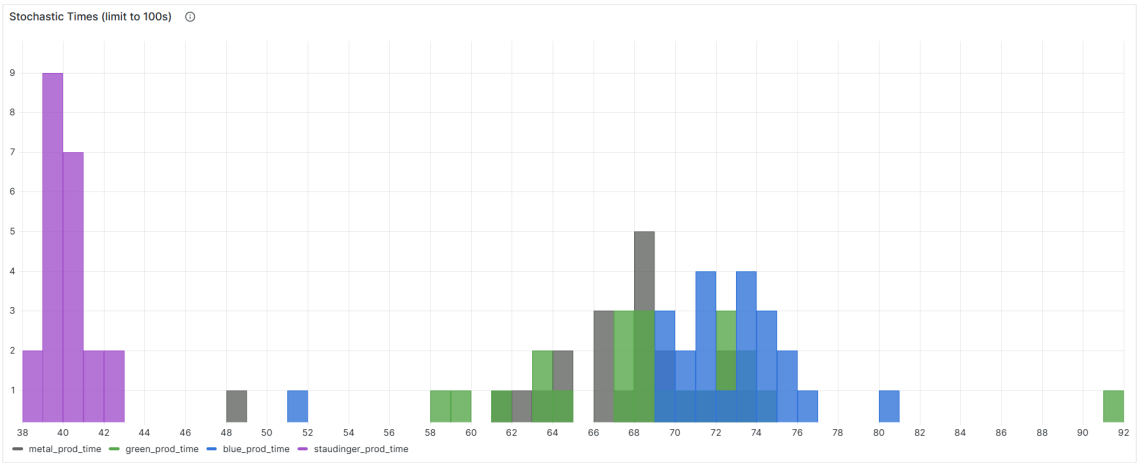


Figure 3.38: Factory 2 plus physical simulator stochastic time histogram (group of all parts production time in seconds)

3.8 Summary

The SuperCoordinator application has been designed to meet these objectives, but also concerning about possible extensions for new functions. This is reflected in the number of objects and classes created (to implement and support special cases) that allow a unique, hierarchical and independent control routine. This was only possible with a distributed application, which requires race condition avoidance, communication and synchronisation between all system agents. The number of threads depends on the configuration, but the concept is reflected in the figure 3.39. The minimum number of concurrent running threads is 7: 1 for the Modbus TCP/IP connection; 1 for the database; 1 for the production and other for the warehouse modules; 2 for the transport modules; 1 for the runtime user interface and finally the JDK main thread that starts or ends the SuperCoordinator's execution.

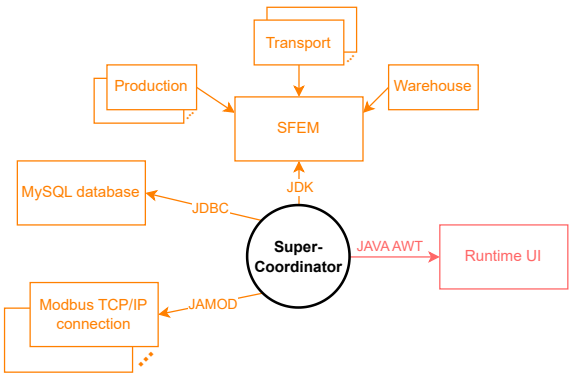


Figure 3.39: SC processes distribution

With this framework, SC is able to interconnect different modules, interfere with the normal

operation, both in terms of time or asynchronous events, but always aiming at accessibility for the educational environment, with user interfaces and support for standard communication protocols to connect with external actors, such as CODESYS.

In short, the SuperCoordinator is a complex and vast application that aims to be simple, extensible, user-friendly and useful for the automation courses, helping with the educational process.

Chapter 4

Conclusions

The SuperCoordinator is to connect several individual modules, real or digital simulators, based on a standard interface, creating a corporate wide simulation of the production environment, from the configuration of the shop floor layout to the storage of production data, oriented towards educational subsystems and supported by Industry 4.0 concepts.

With SuperCoordinator's developed application it is now possible to create a hybrid factory, monitor the natural evolution of the production or intervene in some key aspects that change the normal way of working, always concerning in data saving, both in terms of data generated during the runtime process but also in terms of factory definition. For the students of automation courses, SC can be useful during the development of the courses' projects, having an external application running simultaneously with those they have to develop, can reduce the debugging time and facilitate the introduction to the characteristics of real-time systems. In addition, the stored data provides input for event analysis, pattern extraction and indicator extrapolation. For automation course instructors, the SC's configurability and reusability features can improve course efficiency by enabling different shop floor designs with corresponding levels of complexity, reducing the initial time required to plan course projects.

Despite the solution presented is based on the technologies used today in the home institution, the architecture is broader and more abstract to support future integrations, both in terms of functionalities, communication protocols and external applications to be connected.

4.1 Contributions

The work described in this dissertation led to the following contributions:

- Design and architecture for the SuperCoordinator (SC) as a coordinator of educational subsystems of diverse nature, in the sense that modules are interconnected; the educational module is to be seen as a part of a larger Industry 4.0 mindset to be learned by students.

- Proof of concept and initial software for the SuperCoordinator.

- All configurations, including plant design, interconnections, etc. are read from the config file.

- Three educational modules: one is a miniature physical model and two others are implemented in Factory I/O — one is a simple module and the other is a collection of modules running Factory I/O across several host machines.

- The SC is able to "create" parts in modules and virtually transport them from the output of one to the input of another; additionally, a simple approach to part arrival and part departure has been implemented that is enough to validate global ideas.

- During the working of the modules, the SC ensures the visual failure of machines according to predefined rules (formulas or expressions); the data produced is usable in data analytics to discover data patterns and indicators (drifts, etc) in the spirit of Industry 4.0.

- Using a random seed from config files allows data analytics to be repeatable and generated data can even be visually inspected.

- Customised production times, breakdown and repair times, faulty part production, or part tempering (removing or introducing parts in the middle of the production line), all events are logged (at program exit).

- Factory I/O scene customisation by introducing the emitter and remover pair in the centre of the conveyor to support part tempering, using existing simulator licences.

- Part tracking is done within the SC so that future simulators and real miniatures can be used in this scenario where the SC manages large plants; this information is visible in a runtime GUI.

- All relevant data is sent to the database for easy and open access, also in the spirit of Industry 4.0.

- Direct human intervention on the modules, such as holding a part movement, affects the production time for that part and it is recorded in the database.

- When loading an existing plant design, factory operation is resumed from a previous save point.

4.2 Future Work

Although the SuperCoordinator is already functional and covers most of the defined objectives, there are some limitations or improvements that can be pointed out in order to broaden the scope of application and make it more robust.

As noted in section 3.5.1, the monitor and track algorithm it is based on a serial arrangement of items. For those with a broad knowledge of the implemented code, it is possible to perform some strategic configurations to support other item layouts, being aware that this could lead to conflicts by not respecting some engineering decisions or restrictions. Thus, taking advantage of Factory I/O RFID tags, it is possible to implement a more complex and solid method, based on these sensors, to track parts, under any circumstances within the simulation environment.

Within the scope of this work, only custom production and "real" factor injection have been supported on simulation modules. An interesting objective is therefore to adapt and extend this architecture to all the modules, starting with the virtual ones, such as the warehouse or the transport, and covering the physical simulators. In addition, as mentioned in subsection 3.5.2, when the application is restarted, it does not recover previous failures, although it has the failure records. Implementing this recovery should therefore eliminate the inconvenience of running the failures on restart. Furthermore, the extension to support other simulators, such as SimTwo, is also relevant as it increases the diversity of the corporate wide simulation of the production environment, which gives higher value to the whole system.

In terms of SuperCoordinator connectivity, it was an engineering decision not to implement support for the communication protocol OPC UA at this stage. However, it is known that this support allows more connections to external applications, thus extending the coverage of the SC, with a significant change in the connection management of the modules as a consequence of the service-oriented architecture.

The user interfaces built are very simple and contain only the most relevant information. As a suggestion, building a single graphical user interface, for both the configuration and runtime phases, would significantly increase the value of SuperCoordinator. For the runtime UI, it would be worth adding more relevant aspects from the production side, such as the direct access to the failures' history from the UI, as this would avoid waiting for the file generated only at the end of the program.

Regardless of the suggestions for improvement, the implemented code has been extensively tested and debugged, to provide a final application with as few conflicts as possible.

Appendix A

Repository

A.1 Source code

The SuperCoordinator application is available under
https://github.com/SuperCoordinator/SuperCoordinator-app/.

A.2 Database files

The database files, creation and drop DDL's, used in the application are available under
https://github.com/SuperCoordinator/SuperCoordinator-app//tree/main/database.

A.3 Test scenarios

In this section are the links for the XML configuration files used for testing the application, presented in section 3.7 as well as the files resulting of the failures occurrences.

Factory 1
https://github.com/SuperCoordinator/SuperCoordinator-app//tree/main/blocks/WH_CMC_WH/saves

Factory 2
https://github.com/SuperCoordinator/SuperCoordinator-app//tree/main/blocks/WH_SS_3CMC_WH/saves

Factory 2 plus Staudinger
https://github.com/SuperCoordinator/SuperCoordinator-app//tree/main/blocks/WH_SS_3CMC_MCS_WH/saves

To demonstrate all the test scenarios' operation, there is a video where it is visible the start-up and loading of previous configurations as well as the SC application on runtime environment:
www.fe.up.pt/asousa/sc/

A.4 Deployment instructions

In order to present to the SC user the functionalities and the corrects steps to perform a new configuration for a shop floor, there is a deployment video, where it is done the setup for the Factory 2 covered in detail in subsection 3.7.2: `www.fe.up.pt/asousa/sc_deploy/`

Appendix B

JavaFX UI

sectionConfiguration UI This section presents the various windows developed in the JavaFX. As previously stated, there are not presented on the final version, as their building are inefficient based on this proposal objectives. The first figure B.1 is the application initial screen, followed by the figure B.2 that have the multiple windows supporting the new configuration option and then the figure B.3 with the screens that follow the existing configuration loading option.

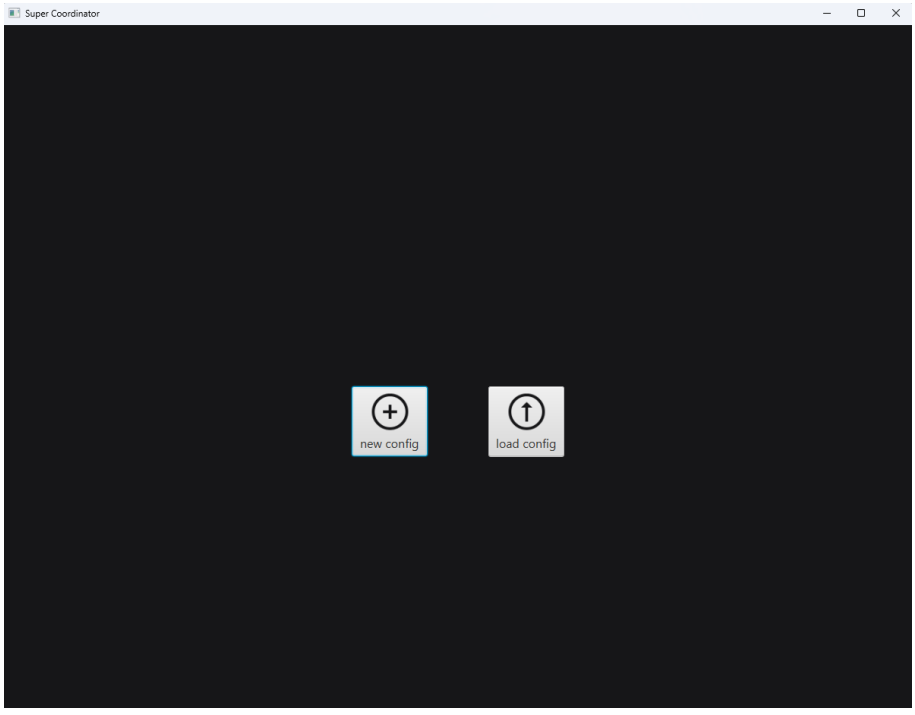
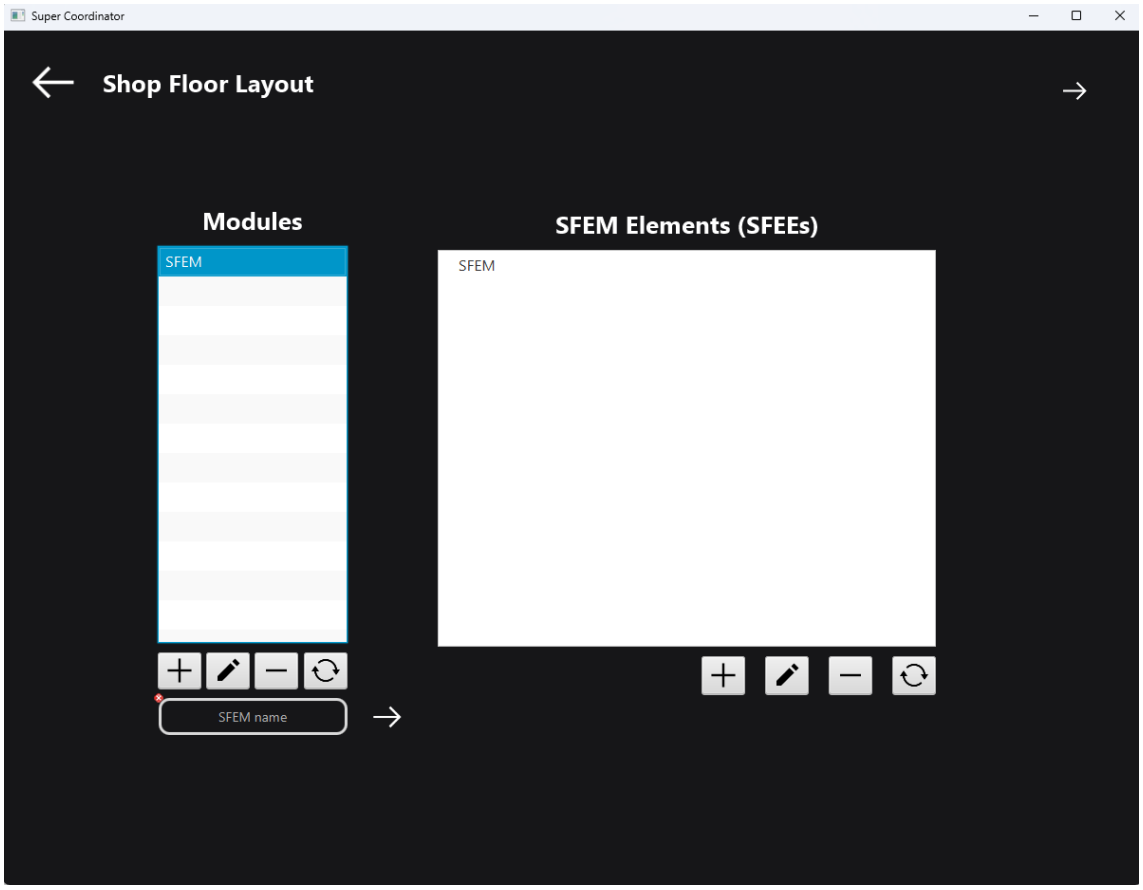
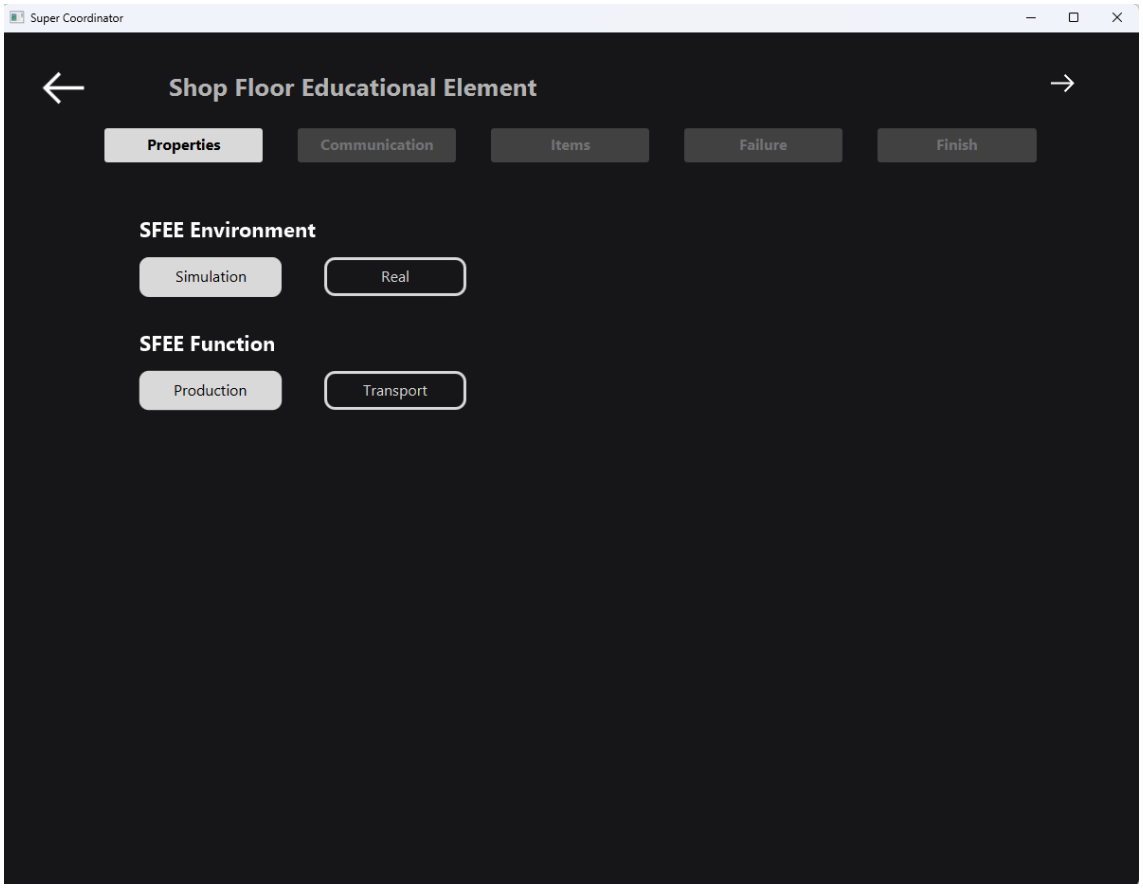


Figure B.1: SuperCoordinator initial screen

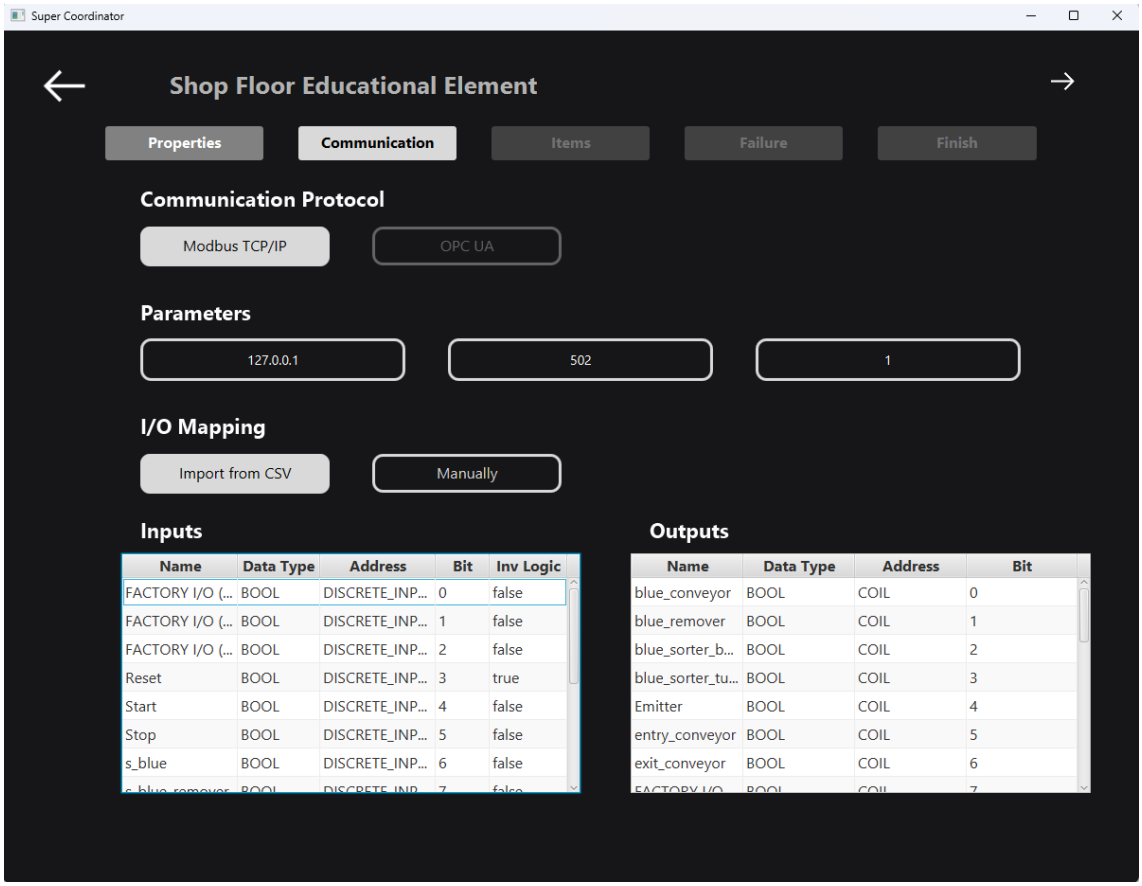


(a) New module definition window

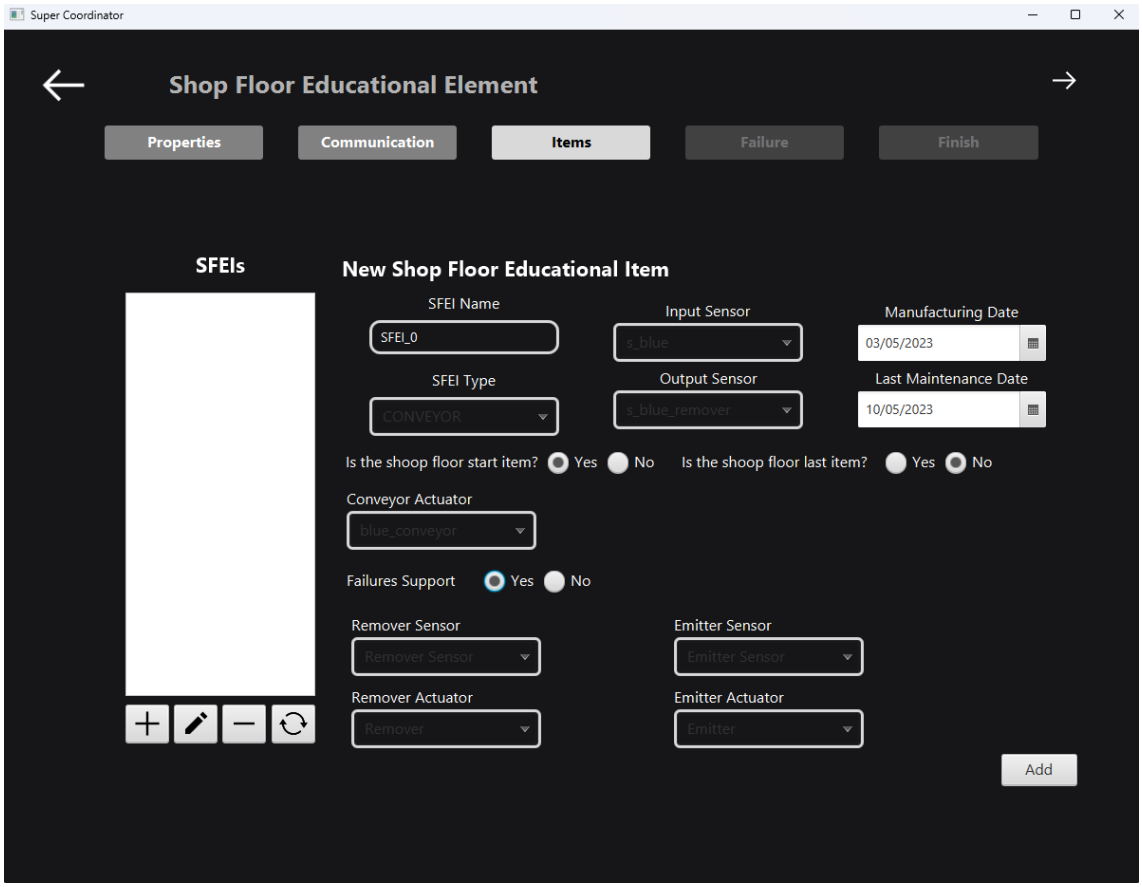


(b) Element properties definition window

Figure B.2: JavaFX's configuration interface sketch for new configuration option

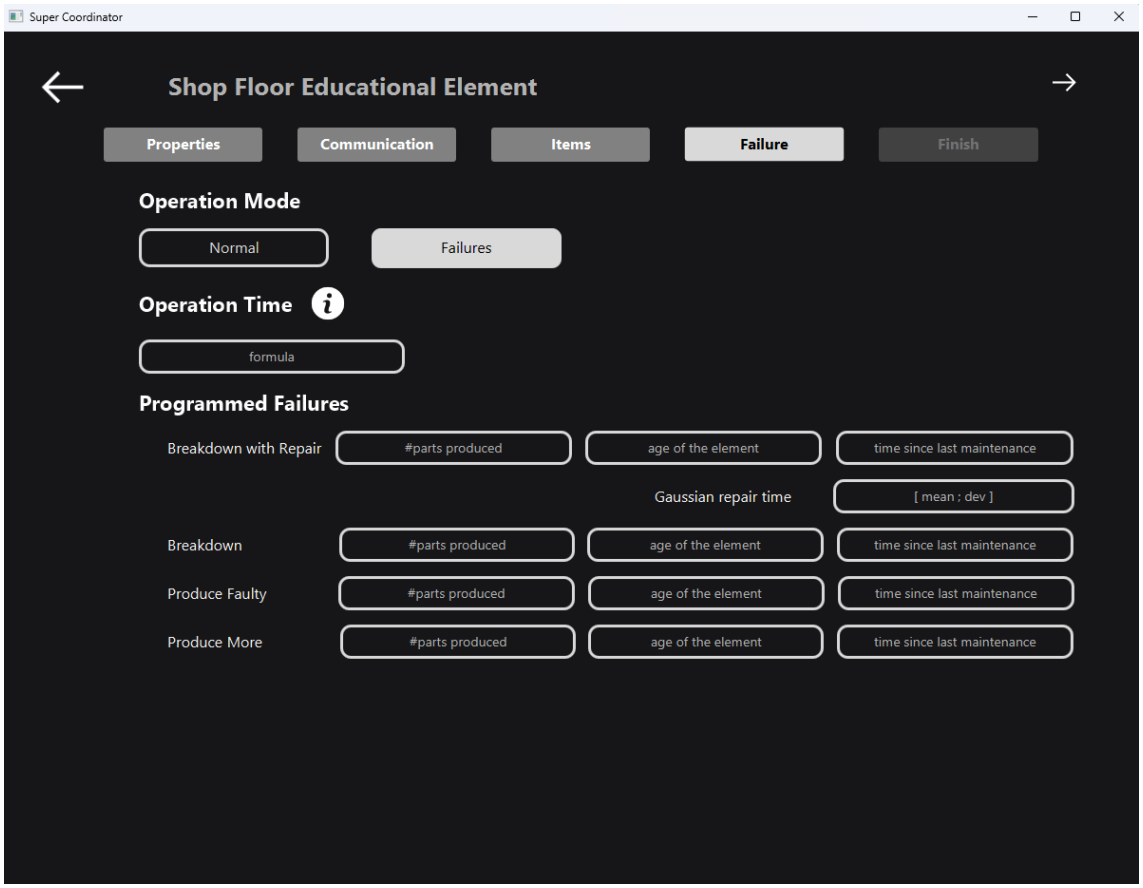


(c) Transport Modules definition window

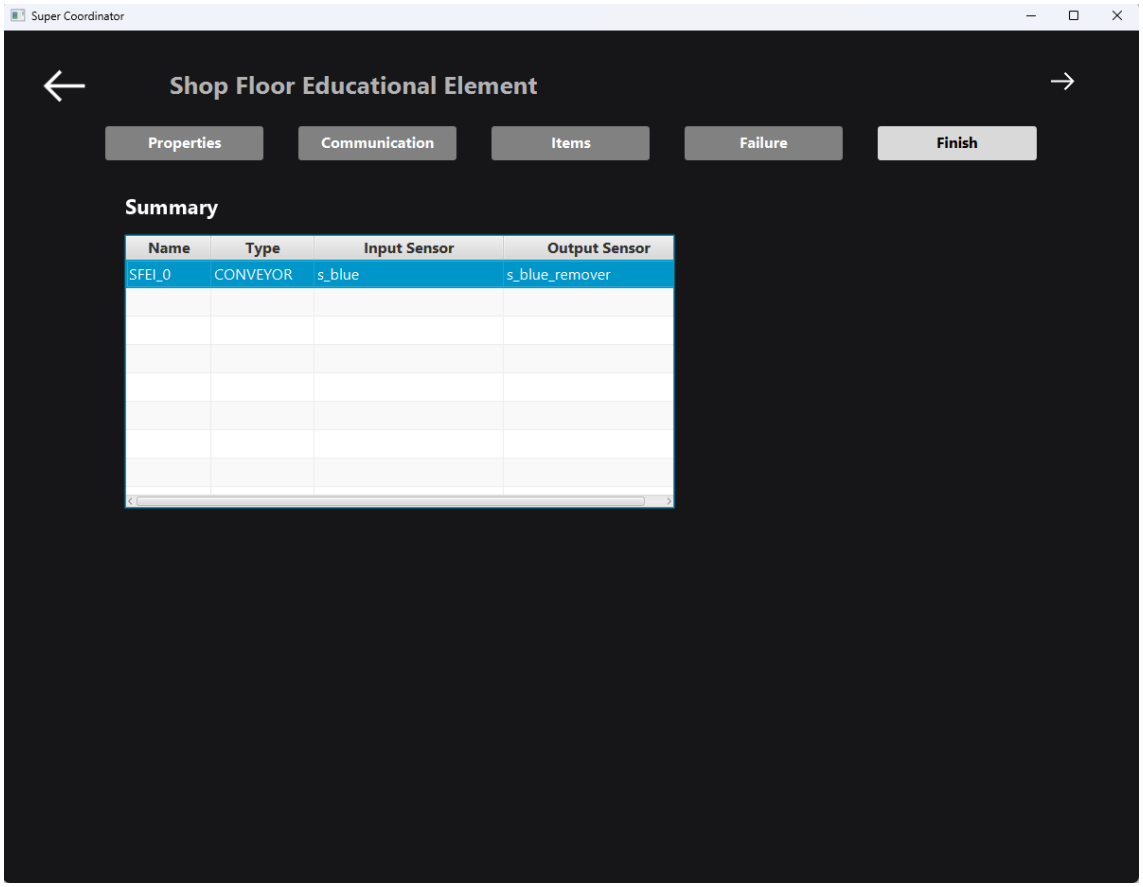


(d) Element communication and I/O's parameters definition window

Figure B.2: JavaFX's configuration interface sketch for new configuration option (cont.)

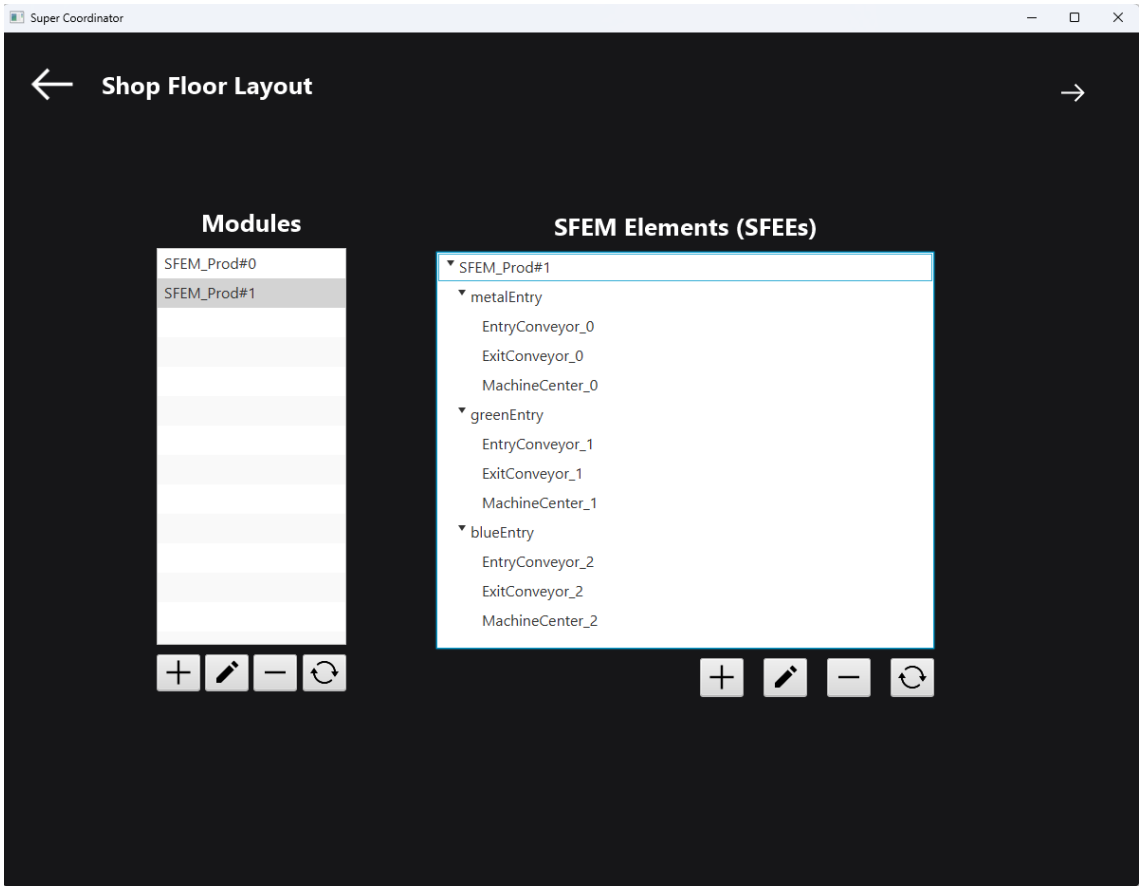


(e) Element custom time and "real" factors definition window

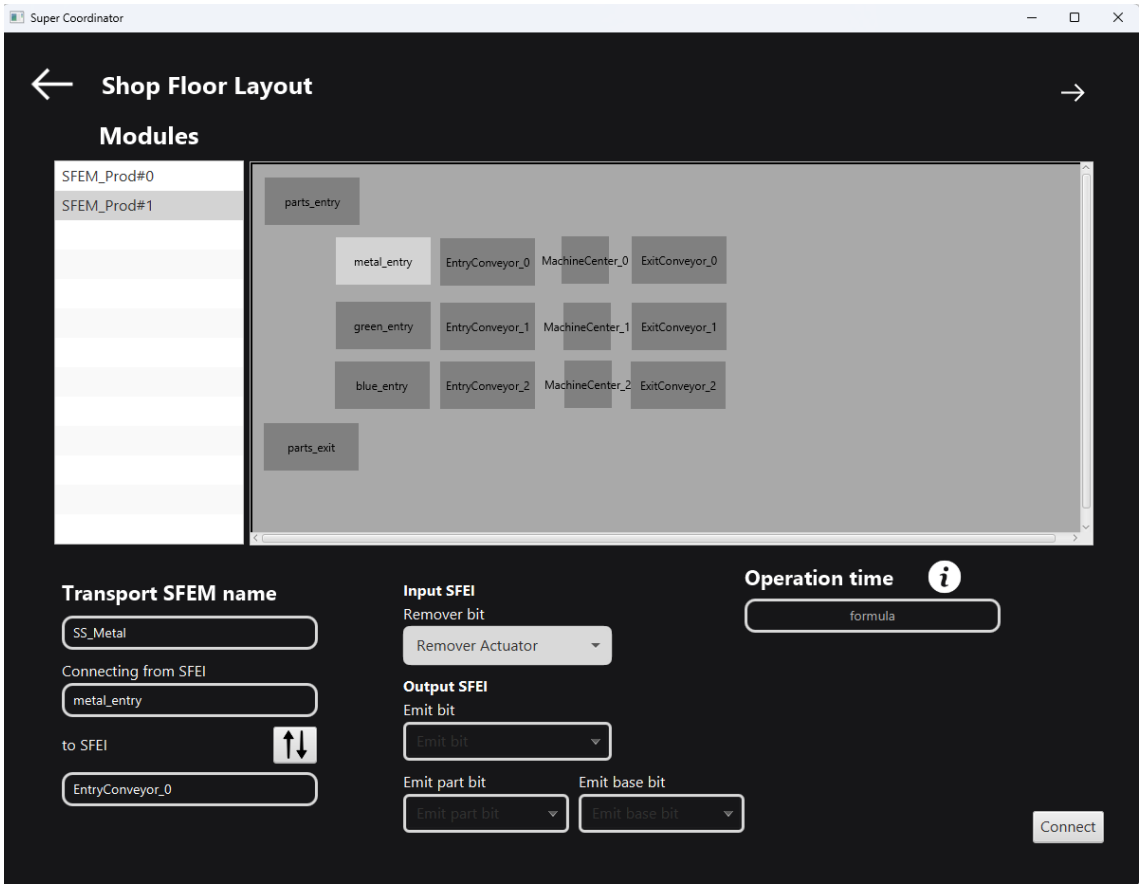


(f) Element defined items summary window

Figure B.2: JavaFX's configuration interface sketch for new configuration option (cont.)



(a) Defined production modules, elements and items summary window



(b) Transport Modules definition window

Figure B.3: JavaFX's configuration interface sketch for loading option

Appendix C

Example modules

C.1 Sorting Station module

This section contains the control algorithm (Figure C.2), implemented in SFC language, for the sorting station element (Figure C.1), which was used in two test scenarios.

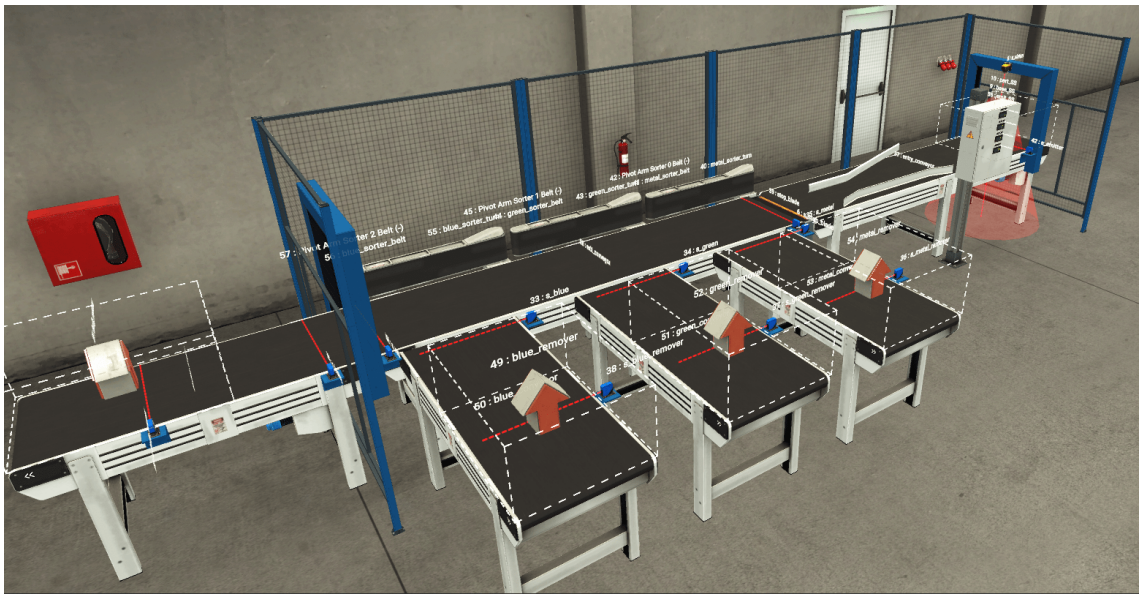
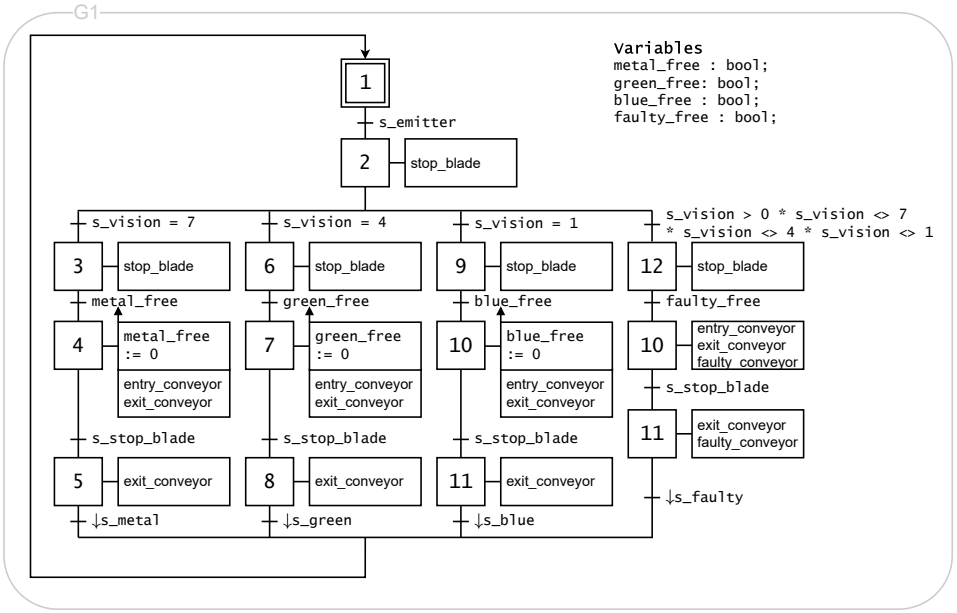
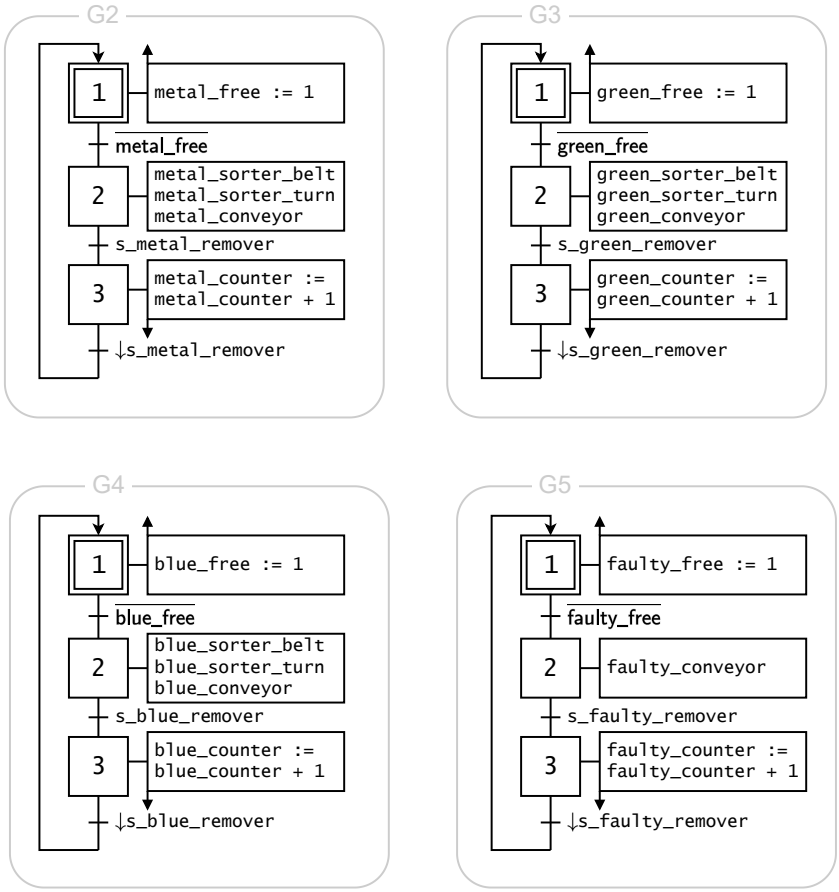


Figure C.1: Sorting Station element



(a) Entry and Exit common conveyors



(b) Other specific conveyors

Figure C.2: Sorting Station control algorithm (GRAFCET)

C.2 Staudinger Compact module

C.2.1 Wiring mapping

The table C.1 shows the correspondence between the wire of the cable with the pin of DB the connector, as well as the corresponding mapping with the inputs and outputs for the control program on the CODESYS (column *SC I/O map*). The spare pins are welded into the connector but are not used at the moment. The 18, 19 and 36, 37 pins are to be connected to the external 24VDC power supply.

Table C.1: DB37 connection cable wiring mapping

DB37 PIN	SC I/O map	Cable wire colour	Function
1	I0	White	Pusher Register Storage engaged
2	I1	Green	Pusher Register Storage extended
3	I4	Orange	Reed Switch at Conveyor Belt 1
4	I5	Blue	Reed Switch at Conveyor Belt 2
5	I2	Orange-Red	Hand Key at Conveyor Belt 1
6	I3	Orange-Green	Hand Key at Conveyor Belt 2
7	I6	Orange-Black	Hand Key at Storage Place
8		White-Red-Black	Spare
9		Red-White-Black	Spare
10		Green-White-Black	Spare
18		Black	Power Supply 0V
19		Black-White	Power Supply 0V
20	O0	Blue-Red	Pusher engage
21	O6	Blue-Black	Pusher extend
22	O7	Red-Green	Conveyor Belt 1
23	O3	Red-White	Conveyor Belt 2
24	O4	White-Red	Machine Tool 1
25	O5	White-Black	Machine Tool 2
26	O1	Green-White	Indicator Lamp 1
27	O2	Green-Black	Indicator Lamp 2
28		Blue-White	Spare
29		Black-Red	Spare
30		Black-White-Red	Spare
36		Red	Power Supply Motors 24V
37		Red-Black	Power Supply Sensors 24V

C.2.2 Control algorithm

This section contains the control algorithm (Figure C.5), implemented in SFC language, for the Staudinger physical simulator element (Figure C.3) used in the last test scenario.

The *G1* is the storage unit and the GRAFCET states 2-4 are to position the pusher arm correctly, i.e. to align it with the back sensor. Such singularities do not occur in the simulation. The *G2* is the first machine and the*G3* is the second one. For both, the "drilling" phase has 15 seconds.

Figure C.4 is the real electrical circuit implemented in order to control the physical simulator via Modbus and also to be monitored by the SC. The details of this circuit are described in subsection 3.1.3

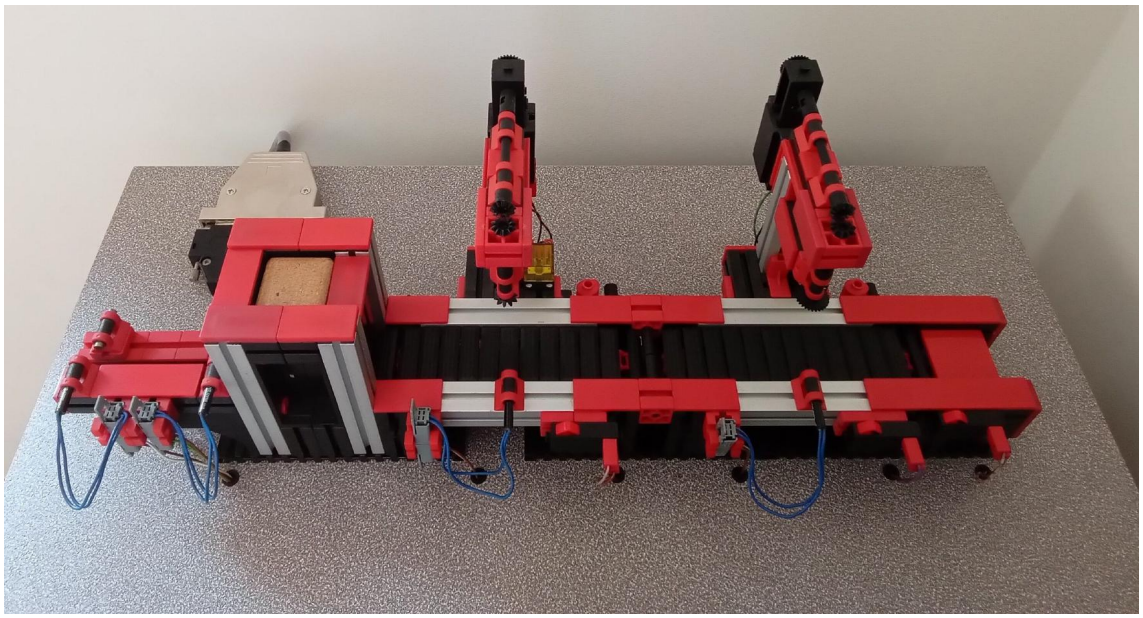


Figure C.3: Physical Staudinger simulator element

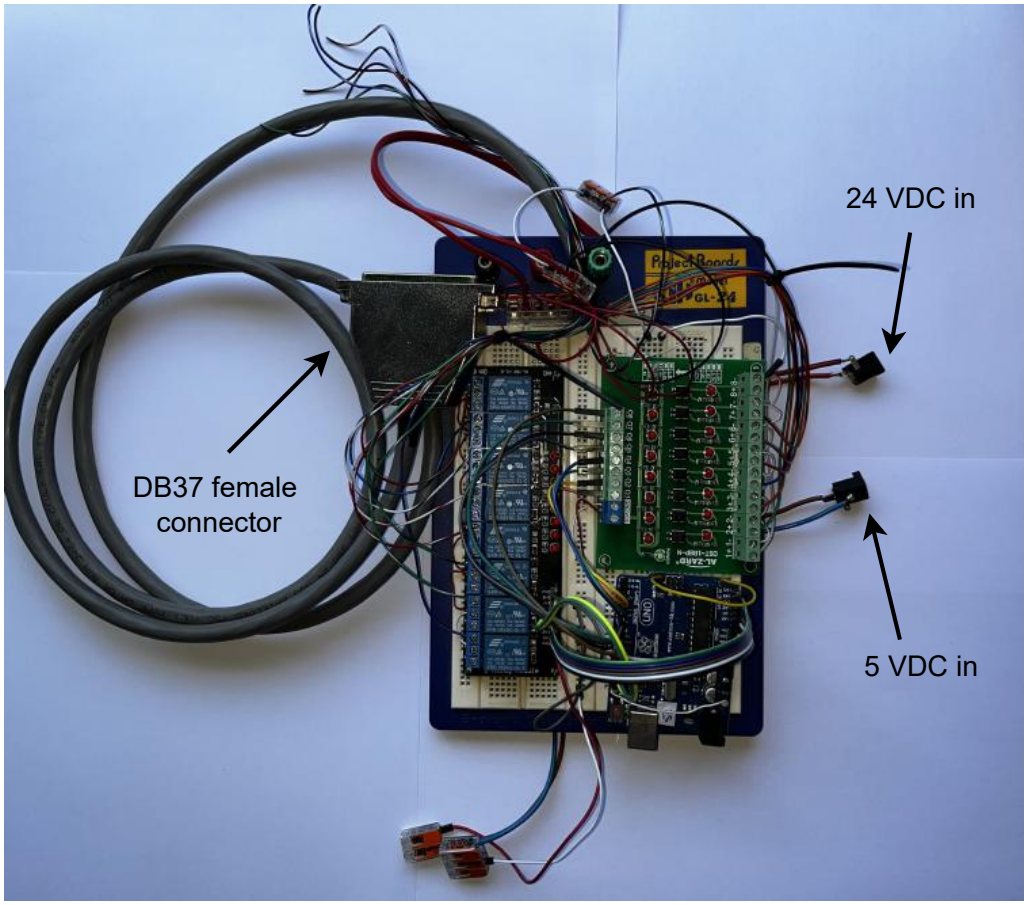
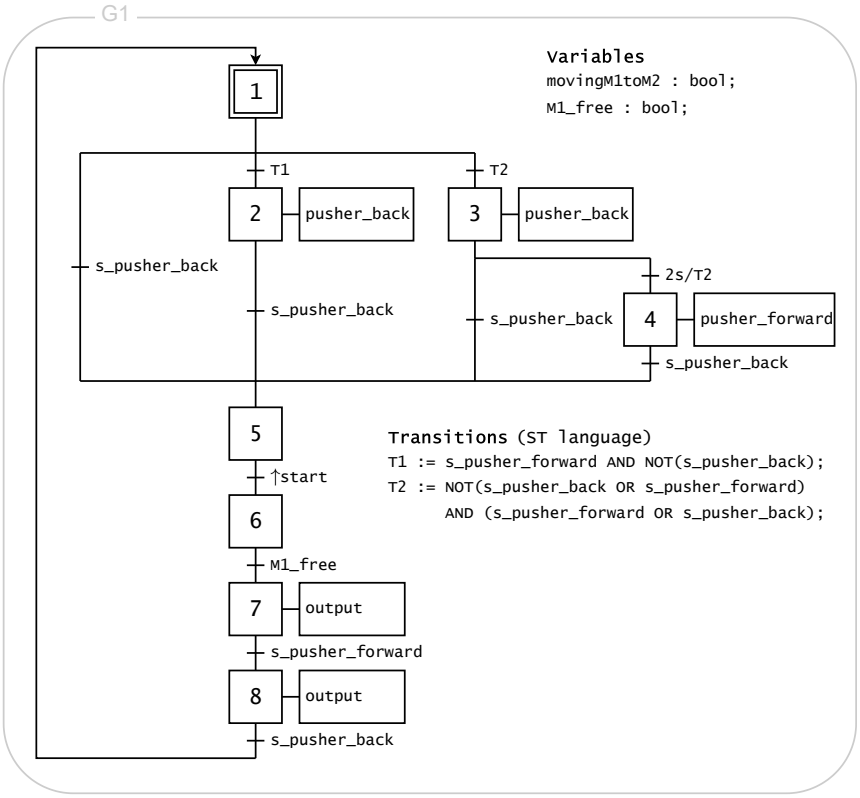
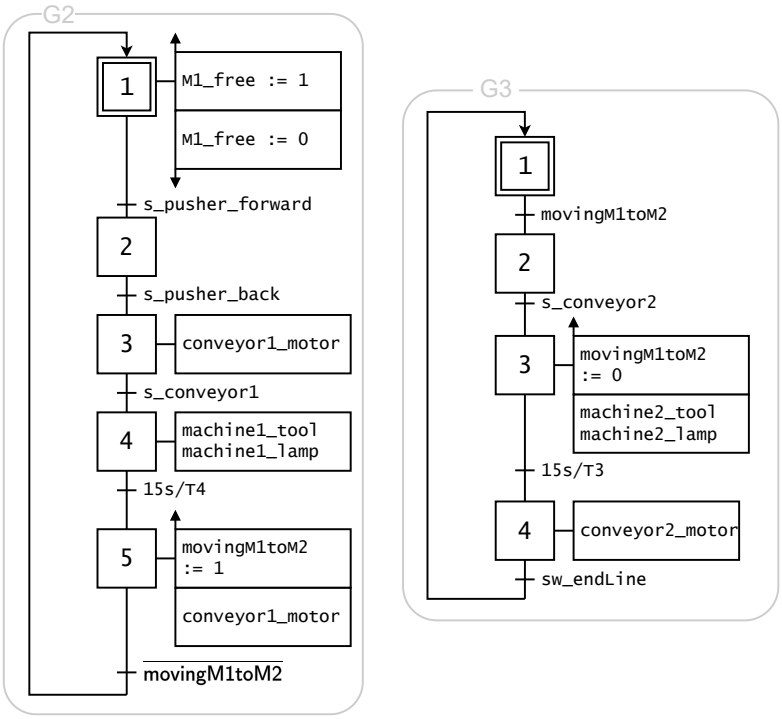


Figure C.4: Physical Staudinger electrical circuit



(a) Storage unit



(b) Machines units

Figure C.5: Physical Staudinger simulator control algorithm (GRAFCET)

C.3 Conveyor-Machine-Conveyor (CMC) module

This section contains the control algorithm (Figure C.7) implemented in SFC language, for the conveyor-machine-conveyor element (Figure C.6) used in all test scenarios. The *G1* corresponds to the entry conveyor, *G2* to the machine centre and *G3* to the exit conveyor.

Figure C.6: CMC element

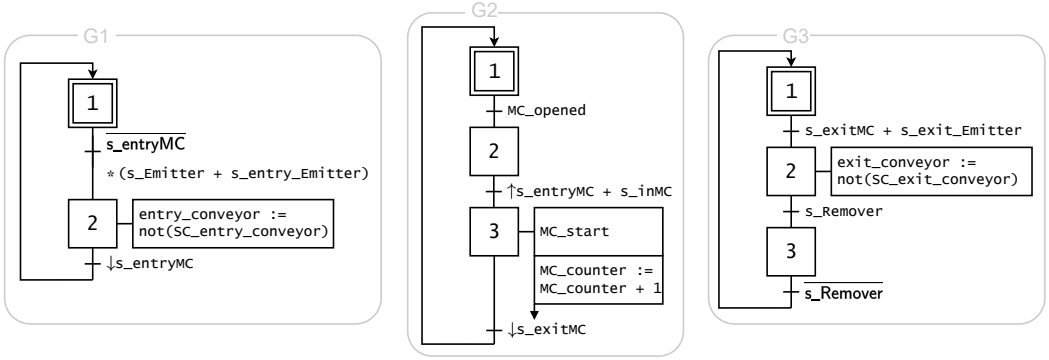


Figure C.7: CMC control algorithm (GRAFCET)

References

[1] Daniel Cortes, Jose Ramirez, Luis Villagomez, Rafael Batres, Virgilio Vasquez-Lopez, and Arturo Molina. Digital pyramid: an approach to relate industrial automation and digital twin concepts. pages 1–7. IEEE, 6 2020. URL: https://ieeexplore.ieee.org/document/9198643/, doi:10.1109/ICE/ITMC49519.2020.9198643.

[2] Paulo Portugal and Mário de Sousa. *Industrial Communication Systems*, chapter 36, pages 36–1, 36–16. CRC Press, 1st edition edition, 10 2018. URL: https://www.taylorfrancis.com/books/mono/10.1201/9781315218434/industrial-communication-systems-bogdan-wilamowski-david-irwin, doi:10.1201/9781315218434.

[3] Acromag Inc. *BusWorks® 900EN Series 10/100M Industrial Ethernet I/O Modules w/ Modbus Technical Reference-Modbus TCP/IP INTRODUCTION TO MODBUS TCP/IP*, 2005. URL: www.public.modicon.com. [last accessed 2022-12-20].

[4] Stefan-Helmut Leitner and Wolfgang Mahnke. Opc ua-service-oriented architecture for industrial applications. *ABB Corporate Research Center*, 48(61-66):22, 2006.

[5] Jörg Schwenk. The internet. pages 1–11, 2022. URL: https://link.springer.com/chapter/10.1007/978-3-031-19439-9_1, doi:10.1007/978-3-031-19439-9_1.

[6] Lütfi Apilioğulları. Digital transformation in project-based manufacturing: Developing the isa-95 model for vertical integration. *International Journal of Production Economics*, 245:108413, 3 2022. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925527322000068, doi:10.1016/j.ijpe.2022.108413.

[7] Josélia Elvira Teixeira and Ana Teresa Tavares-Lehmann. Industry 4.0: the future of manufacturing from the perspective of business and economics-a bibliometric literature review. *Competitiveness Review: An International Business Journal*, 33:458–482, 2023. URL: https://www.emerald.com/insight/1059-5422.htm, doi:10.1108/CR-07-2022-0091.

[8] Tae Kyung Sung. Industry 4.0: A korea perspective. *Technological Forecasting and Social Change*, 132:40–45, 7 2018. doi:10.1016/J.TECHFORE.2017.11.005.

[9] Mozilla Contributors. Evolution of http, 12 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#invention_of_the_world_wide_web [last accessed 2022-12-20].

[10] Vinicius Fulber Garcia. Http: 1.0 vs. 1.1 vs 2.0 vs. 3.0, 11 2022. URL: https://www.baeldung.com/cs/http-versions [last accessed 2022-12-20].

[11] Imperva. What is osi model | 7 layers explained, 2022. URL: https://www.imperva.com/learn/application-security/osi-model/ [last accessed 2022-12-20].

[12] OPC Foundation. What is opc? URL: https://opcfoundation.org/about/what-is-opc/ [last accessed January 2023].

[13] Woonggy Kim and Minyoung Sung. Standalone opc ua wrapper for industrial monitoring and control systems. *IEEE Access*, 6:36557–36570, 7 2018. doi:10.1109/ACCESS.2018.2852792.

[14] OPC Foundation. Unified architecture. URL: https://opcfoundation.org/about/opc-technologies/opc-ua/ [last accessed January 2023].

[15] Seung Yong Lee and Minyoung Sung. Opc-ua agent for legacy programmable logic controllers †. *Applied Sciences (Switzerland)*, 12, 9 2022. doi:10.3390/APP12178859.

[16] Sebastião José Fernandes de Oliveira Cunha Reis. Upgrade do simulador da linha de produção flexível do laboratório de automação. 3 2020. URL: https://repositorio-aberto.up.pt/handle/10216/126573.

[17] Alex Matheson. Soft plcs vs hard plcs - which is the future? - atlas. URL: https://weareatlas.com/resources/articles/soft-hard-plcs/ [last accessed January 2023].

[18] CODESYS GmbH. Codesys development system v3. URL: https://store.codesys.com/en/codesys.html [last accessed January 2023].

[19] Wikipedia. Codesys. URL: https://en.wikipedia.org/wiki/CODESYS [last accessed January 2023].

[20] Beremiz. Beremiz. URL: https://beremiz.org/about [last accessed January 2023].

[21] OpenJS Foundation and Node-RED contributors. Node-red. URL: https://nodered.org/ [last accessed January 2023].

[22] Eliseu Moura Pereira, João Pedro Correia dos Reis, and Gil Gonçalves. Dinasore: A dynamic intelligent reconfiguration tool for cyber-physical production systems. pages 63–71. URL: http://ceur-ws.org/Vol-2739/#paper_9.

[23] Dinasore — a tool for distributed function-block-based systems | by josé rafael fidalgo fonseca matias | medium. URL: https://medium.com/@jrffmatias/dinasore-a-tool-for-distributed-function-block-based-systems-f2613a37e1ca [last accessed June 2023].