FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ambiente de simulação para o ensino de conceitos associados a aplicações do tipo ERP e MES

Pedro Miguel Pinto Silva

Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Paulo José Lopes Machado Portugal

23 de julho de 2023

© Pedro Miguel Pinto Silva, 2023

Resumo

O modelo subjacente à pirâmide de automação é um conceito muito relevante para organizar por camadas as diferentes funções, equipamentos e as suas interações num sistema de automação em contexto real.

Na vertente educacional, este modelo é de extrema importância, uma vez que, dada a sua estrutura clara e organizada, permite que os estudantes percecionem progressivamente as funções desempenhadas por cada camada. Assim, existe a necessidade de ferramentas que permitam demonstrar estes conceitos. Em particular, as aplicações comerciais que implementam as camadas superiores, como os ERP e os MES, não são adequadas para o meio educacional, pois possuem um grau de complexidade muito elevado para atenderem às necessidades existentes em contexto real.

Nesta dissertação, foram desenvolvidas duas aplicações *Java* com uma base de dados *PostgreSQL*, que permitem representar os conceitos principais dos ERP e MES, efetuando o planeamento e o controlo da produção, respetivamente, mas com funcionalidades mais reduzidas, denominadas *mini-ERP* e *mini-MES*.

Para atuar como as camadas inferiores, foi utilizada uma ferramenta de simulação desenvolvida numa dissertação prévia, consistindo num *Digital Twin*. Este *Digital Twin* é constituído pelo simulador *FactoryIO*, que simula os equipamentos mais comuns em ambientes industriais, e pelo *Codesys*, para o qual foi desenvolvida uma biblioteca de *Function Blocks* que implementa o controlo para cada um dos equipamentos.

Deste modo, foi desenvolvido um ambiente de simulação que possibilita aos estudantes contactar com um conjunto de aplicações que demonstram o funcionamento e as interações que ocorrem entre as várias camadas da pirâmide de automação.

Abstract

The underlying model of the automation pyramid is a highly relevant concept for organizing the different functions, equipment, and their interactions into layers within a real-world automation system.

In an educational context, this model is of utmost importance as its clear and organized structure allows students to progressively understand the functions performed by each layer. Therefore, there is a need for tools that allow these concepts to be demonstrated. In particular, commercial applications that implement the upper layers, such as ERP and MES systems, are not suitable for the educational environment, as they have a high degree of complexity that is required to meet the existing needs in a real-world context.

In this dissertation, two Java applications with a PostgreSQL database were developed to represent the main concepts of ERP and MES systems, focusing on production planning and control, respectively, but having reduced functionalities compared to their commercial counterparts, referred to as mini-ERP and mini-MES.

To act as the lower layers, a simulation tool developed in a previous dissertation was used, a Digital Twin. This Digital Twin consists of the FactoryIO simulator, which emulates common equipment found in industrial environments, and Codesys, for which a library of Function Blocks was developed to implement control for each of the equipment.

As a result, a simulation environment was created, allowing students to interact with a set of applications that demonstrate the operation and interactions that occur between the various layers of the automation pyramid.

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, Professor Paulo Portugal, pelos seus conselhos, disponibilidade, colaboração e motivação (e paciência, nos momentos mais críticos) durante este semestre.

Ao André Sousa, autor do trabalho utilizado como base para esta dissertação, que sempre se disponibilizou para me esclarecer dúvidas.

Ao Professor José António Faria, pelas ideias sugeridas na fase conceptual deste trabalho, e ao Professor Armando Jorge Sousa, que contribuiu na minha participação em atividades extracurriculares no meu percurso académico.

Aos meus amigos e colegas, César, Daniel, Mina e Sengo, pela companhia nos momentos de diversão, e entreajuda nos momentos de "desespero académico", durante estes 5 anos que passamos juntos.

Por fim, agradeço a toda a minha família. Em especial, aos meus pais, Amélia e Miguel, pela educação, motivação e suporte que sempre me deram em tudo, e pela excecional paciência em "aturar-me" todos os dias (que sei que não é fácil). Aos meus avós, Augusta e Inácio, que sempre estiveram presentes. Fico feliz por poderem estar comigo no momento em que finalizo esta etapa importante da minha vida.

Pedro Silva

"The only real mistake is the one from which we learn nothing."

Henry Ford

Conteúdo

1 Introdução **1**
- 1.1 Contexto . . . 1
- 1.2 Motivação . . . 2
- 1.3 Objetivos . . . 2
- 1.4 Organização do documento . . . 3

2 Conceitos Fundamentais **5**
- 2.1 Camadas da pirâmide de automação . . . 5
- 2.2 Trabalho prévio - A Digital Twin for Education With FactoryIO . . . 6
  - 2.2.1 FactoryIO . . . 7
  - 2.2.2 CODESYS . . . 9
  - 2.2.3 Arquitetura do Digital Twin . . . 10
  - 2.2.4 Comunicação entre aplicações . . . 11
  - 2.2.5 Comunicação entre Function Blocks . . . 12
  - 2.2.6 Estruturas de dados . . . 13
  - 2.2.7 Encaminhamento das peças . . . 14
  - 2.2.8 Conclusões sobre o Digital Twin . . . 15
- 2.3 Tecnologias utilizadas . . . 15
  - 2.3.1 JavaFX Scene Builder . . . 15
  - 2.3.2 PostgreSQL . . . 17
- 2.4 Conceitos base das aplicações a desenvolver . . . 17
  - 2.4.1 Enterprise Resource Planning . . . 17
  - 2.4.2 Manufacturing Execution System . . . 20
- 2.5 Estado da arte . . . 21
  - 2.5.1 Soluções existentes . . . 21
  - 2.5.2 Discussão das soluções encontradas . . . 24
- 2.6 Conclusões . . . 24

3 Arquitetura proposta **25**
- 3.1 Contextualização . . . 25
  - 3.1.1 Matérias-primas e produtos finais considerados . . . 25
  - 3.1.2 Divisão do layout e ordens de produção . . . 26
  - 3.1.3 Peças com defeito . . . 27
  - 3.1.4 Clientes e fornecedores . . . 27
- 3.2 Requisitos . . . 28
- 3.3 Arquitetura de alto nível . . . 30
- 3.4 Arquitetura do mini-ERP . . . 31
  - 3.4.1 Introdução de dados por parte do utilizador . . . 32

3.4.2 Encomendas de clientes . . . 33
3.4.3 Eventos que implicam a alteração do planeamento . . . 34
3.5 Arquitetura do mini-MES . . . 37
3.5.1 Pontos de interesse . . . 39
3.5.2 Definição dos caminhos . . . 42
3.5.3 Controlo da produção . . . 42
3.6 Conclusões . . . 47

4 Implementação e Testes **49**
4.1 Modelo de dados . . . 49
4.2 Ciclo de simulação . . . 52
4.3 Padrões de desenvolvimento . . . 53
4.3.1 Model-View-Controller . . . 53
4.3.2 Singleton . . . 54
4.4 Implementação do mini-ERP . . . 54
4.4.1 Interface inicial . . . 54
4.4.2 Interface interna . . . 55
4.4.3 Interface do fornecedor . . . 58
4.4.4 Interface do cliente . . . 59
4.5 Alterações ao *Digital Twin* . . . 60
4.5.1 *Feedback* dos comandos . . . 61
4.5.2 Limite no número de comandos enviados . . . 61
4.6 Implementação do mini-MES . . . 62
4.6.1 Mapeamento do *layout* da fábrica . . . 62
4.6.2 Gestor de percursos . . . 64
4.6.3 Thread de controlo . . . 65
4.6.4 Interfaces . . . 66
4.7 Testes e Validação . . . 68
4.7.1 Layout utilizado no *Digital Twin* . . . 68
4.7.2 Teste de utilização . . . 69
4.8 Conclusões . . . 70

5 Conclusões **71**
5.1 Trabalhos futuros . . . 72

A *Setup* da base de dados **73**

B Identificadores no *feedback* do *Digital Twin* **75**

C Repositórios de código **77**

D Montagem do layout de teste no Codesys **79**

Referências **81**

Lista de Figuras

1.1 Pirâmide de automação segundo a ISA95, retirado de [2] . . . 2

2.1 Itens leves (esquerda) e pesados (direita), retirado de [4] . . . 7
2.2 *Conveyor*, *Emitter* e *Remover*, retirado de [4] . . . 8
2.3 *Sorter*, *Turntable* e *Rollers*, retirado de [4][8] . . . 8
2.4 *Machining Center*, *Assembler* e *Warehouse*, retirado de [4] . . . 8
2.5 Exemplo de código ST, retirado de [4] . . . 9
2.6 Exemplo de um diagrama SFC, retirado de [4] . . . 9
2.7 Exemplo um FB para um *Conveyor*, retirado de [4] . . . 10
2.8 Arquitetura global, retirado de [4] . . . 10
2.9 Exemplo de *layout* no FactoryIO . . . 11
2.10 *Layout* mapeado para a biblioteca em CODESYS . . . 11
2.11 *Orders Manager*, retirado de [4] . . . 12
2.12 *Bus* de comunicação entre FB, retirado de [4] . . . 12
2.13 Comunicação sequencial entre FB, retirado de [4] . . . 12
2.14 *Scene Builder*, retirado de [14] . . . 16
2.15 Estrutura de um ERP, adaptado de [17] . . . 18
2.16 Módulos de um ERP para produção industrial, retirado de [18] . . . 19
2.17 Comparação entre *push* e *pull models*, retirado de [17] . . . 20
2.18 Simulação no 3DEXPERIENCE DELMIA, retirado de [21] . . . 22
2.19 Simulação de processo de fabrico no Simio, retirado de [22] . . . 23
2.20 Exemplo de resultados da análise de risco no Simio, retirado de [22] . . . 23

3.1 Matérias-primas, retirado de [7] . . . 25
3.2 Tampas (em cima) e bases (em baixo), retirado de [7] . . . 26
3.3 Divisão do *layout* . . . 26
3.4 Arquitetura de alto nível . . . 31
3.5 Módulos do mini-ERP . . . 32
3.6 Algoritmo para nova encomenda . . . 34
3.7 Algoritmo para atrasos . . . 35
3.8 Algoritmo para o cancelamento de uma encomenda . . . 36
3.9 Processamento do feedback do MES . . . 38
3.10 Pontos de interesse . . . 39
3.11 Exemplo de uma zona de entrada . . . 40
3.12 Exemplo de uma zona de produção . . . 41
3.13 Exemplo de uma zona de expedição . . . 42
3.14 *Inbound Emitter*, *Inbound Buffer* e *Inbound Warehouse* . . . 43
3.15 Controlo da zona de entrada . . . 44

3.16 Controlo da zona de produção (Parte 1) . . . 45
3.17 Controlo da zona de produção (Parte 2) . . . 46
3.18 Controlo da zona de expedição . . . 47

4.1 Modelo de dados . . . 50
4.2 Ciclo de simulação . . . 52
4.3 Model-View-Controller, retirado de [23] . . . 53
4.4 Interface inicial . . . 54
4.5 Páginas *Add Client* e *Add Supplier* . . . 55
4.6 Páginas *Client Orders* e *Supplier Orders* . . . 56
4.7 Páginas *Pending Orders* e *Schedule* . . . 57
4.8 Páginas *Edit parameters* e *Order List* . . . 58
4.9 Página *New Order* . . . 59
4.10 Páginas *Pending Orders* e *Confirmed Orders* . . . 60
4.11 *Thread* de controlo . . . 64
4.12 Página *Settings* . . . 65
4.13 Página *Current Schedule* . . . 66
4.14 Página *History Log* . . . 66
4.15 Página *Warehouse* . . . 67
4.16 Página *Machine Overview* . . . 67
4.17 *Layout* de teste no FactoryIO . . . 68

Lista de Tabelas

3.1 Requisitos do mini-ERP 28
3.2 Requisitos do mini-MES 30

4.1 *equipment_type* obrigatórios 63
4.2 *equipment_type* dos *Machining Center* 63
4.3 *equipment_type* dos equipamentos de transporte 63
4.4 Fornecedores registados 69
4.5 Eventos 70

Abreviaturas e Símbolos

IoT	Internet of Things
IIoT	Industrial Internet of Things
DT	Digital Twin
L.EEC	Licenciatura em Engenharia Eletrotécnica e de Computadores
M.EEC	Mestrado em Engenharia Eletrotécnica e de Computadores
FEUP	Faculdade de Engenharia da Universidade do Porto
OPC UA	Open Platform Communications Unified Architecture
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
OOP	Object-Oriented Programming
SQL	Structured Query Language
ISA95	International Society of Automation
PLC	Programmable Logic Controllers
SCADA	Supervisory Control and Data Acquisition
HMI	Human-Machine Interface
IEC	International Electrotechnical Commission
FB	Function Block
POU	Program Organization Units
KPI	Key Performance Indicators
GUI	Graphical User Interface

Capítulo 1

Introdução

Neste capítulo é apresentada uma introdução ao trabalho realizado. Inicialmente, é efetuada uma contextualização sobre o seu tema, seguindo-se uma apresentação das motivações que levaram à sua realização e dos objetivos que se pretendem alcançar. Finalmente, a estrutura deste documento é descrita na última secção.

1.1 Contexto

A quarta revolução industrial, denominada Indústria 4.0, baseia-se na utilização de tecnologias digitais que permitem a constituição de sistemas ciberfísicos [1], possibilitando a existência de sistemas que monitorizam os processos de produção. Estes sistemas têm como objetivo promover linhas de produção mais eficientes e autónomas, melhorando a qualidade do produto final. É necessário que todos os elementos intervenientes na produção estejam conectados aos sistemas existentes, fornecendo e recebendo dados, ou seja, uma versão do conceito de *Internet of Things* (IoT) voltada para a indústria, a *Industrial Internet of Things* (IIoT). Com isto, os diferentes dispositivos que compõem o chão de fábrica, podem fornecer às camadas hierarquicamente superiores um grande volume de dados.

Para estruturar os sistemas referidos, foi desenvolvido um modelo denominado pirâmide de automação, representado na figura 1.1. Segundo a norma ISA95 [3], a pirâmide de automação é um modelo hierárquico que separa por camadas as funções de um sistema de automação e define a interação entre elas. Além de ser útil no processo de desenvolvimento e implementação deste tipo de sistemas num contexto real, tem também um grande valor ao nível educacional. Como é estruturada de forma clara e organizada, permite compreender progressivamente os conceitos de automação relativos a cada camada.

Dada a sua importância na área de automação, é necessário que existam ferramentas acessíveis no ambiente de ensino e que permitam aos estudantes percecionar estes conceitos.

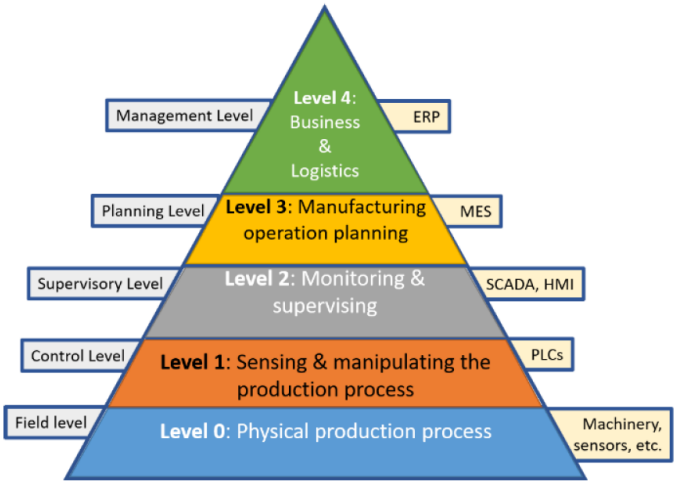


Figura 1.1: Pirâmide de automação segundo a ISA95, retirado de [2]

1.2 Motivação

No mercado, existem várias aplicações que implementam as funcionalidades das camadas do topo da pirâmide de automação como, por exemplo, os sistemas ERP (*Enterprise Resource Planning*) e MES (*Manufacturing Execution System*). No entanto, por serem soluções completas desenvolvidas para a indústria, não são adequadas para o meio educacional por serem complexas, tendo a sua utilização, inerentemente, uma curva de aprendizagem muito elevada.

Deste modo, existe a necessidade de desenvolver um novo tipo de aplicações, que possuam os conceitos principais dos ERP e MES comerciais, mas mais simples, e que sejam focadas na vertente educacional. Estas serão utilizadas em demonstrações nas unidades curriculares da área de automação, junto dos estudantes do L.EEC e M.EEC da FEUP. O estudo do seu funcionamento, bem como da partilha de informação que ocorre entre camadas, permite consolidar um vasto conjunto de conhecimentos sobre tecnologias estruturantes no atual panorama da Indústria 4.0.

1.3 Objetivos

Esta dissertação tem como objetivo a criação de um conjunto de aplicações que permite demonstrar aos estudantes todo o funcionamento da pirâmide de automação. Para esta finalidade, pretende-se utilizar o trabalho realizado na dissertação "*A Digital Twin for Education With FactoryIO*" [4], em que foi desenvolvida uma ferramenta de simulação, um *Digital Twin*, que pode atuar como as duas camadas inferiores da pirâmide, ou seja, dos equipamentos físicos existentes ao nível do chão de fábrica e do seu respetivo controlo.

Utilizando esta ferramenta como base, pretende-se desenvolver duas aplicações para implementar as duas camadas de topo da pirâmide. O ERP, que efetua e envia o planeamento da produção para o MES que, por sua vez, procede ao controlo da execução das operações de fabrico. Pretende-se que estas duas aplicações sejam uma versão minimalista das que são comercializadas, com funcionalidades mais reduzidas, mas representativas dos conceitos principais. Por este motivo, foram denominadas, no contexto deste trabalho e deste ponto em diante, como *mini-ERP* e *mini-MES*.

De um modo geral, pretende-se que o mini-ERP efetue o planeamento da produção de uma linha fabril, tendo em conta as encomendas que recebe dos clientes, os materiais existentes no armazém e as encomendas realizadas a fornecedores. O mini-MES, por sua vez, recebe o planeamento e controla a produção, interagindo com a ferramenta desenvolvida na dissertação anterior, executando as ordens de produção pretendidas e recolhendo dados sobre o processo de fabrico. Ambas as aplicações devem possuir interfaces gráficas na qual o utilizador possa introduzir e visualizar dados sobre o processo.

1.4 Organização do documento

Este documento é constituído por 5 capítulos. Além do capítulo 1, os restantes encontram-se organizados da seguinte forma:

- Capítulo 2: Apresentação dos conceitos fundamentais que permitem perceber o trabalho que foi realizado.
- Capítulo 3: Apresentação e discussão da arquitetura proposta para as aplicações.
- Capítulo 4: Metodologia utilizada para a implementação das aplicações e a sua validação.
- Capítulo 5: Conclusões finais e trabalhos futuros.

Capítulo 2

Conceitos Fundamentais

Neste capítulo são apresentados os aspetos relevantes para a contextualização deste trabalho. Inicialmente são detalhadas as funções de cada camada da pirâmide de automação. De seguida, é explicada a ferramenta que foi desenvolvida numa dissertação anterior e que serviu de base a este trabalho, pelo que a compreensão do seu funcionamento é essencial. Além desta ferramenta, são apresentadas as tecnologias que foram utilizadas no desenvolvimento das aplicações deste trabalho. São descritos os conceitos principais relativos aos ERP e MES e, no final, são apresentadas soluções existentes no mercado que podem ser utilizadas num contexto educativo semelhante ao pretendido para este trabalho.

2.1 Camadas da pirâmide de automação

Nesta secção são analisadas as funções das 5 camadas, ou níveis, da pirâmide de automação (apresentada no capítulo 1), recorrendo à nomenclatura da figura 1.1 [2][5]:

- *Field Level*: É a camada da base da pirâmide, constituída pelos dispositivos físicos, incluindo os sensores (de temperatura, humidade, pressão, entre outros) e os atuadores (máquinas, motores, válvulas, por exemplo) que interagem diretamente no processo industrial. As interações presentes nesta camada ocorrem na menor escala temporal da pirâmide (microssegundos a milissegundos), sendo, por isso, considerado que atua no processo industrial em tempo real.

- *Control Level*: A camada de controlo coordena os dispositivos presentes na camada da base recorrendo a microcontroladores ou autómatos (*Programmable Logic Controllers*), por exemplo. Estes recebem os dados provenientes dos sensores e enviam comandos para os atuadores conforme a lógica de decisão implementada.

- *Supervision Level*: A camada de supervisão é implementada por sistemas como os *Human-Machine Interface* (HMI) ou os *Supervisory Control and Data Acquisition Systems* (SCADAs). Estes sistemas permitem a monitorização dos elementos de controlo e dos dispositivos de campo, apresentando os dados (devidamente processados) relativos aos mesmos.

- *Planning Level*: Esta camada controla o processo de produção através de sistemas como os MES. Estes sistemas definem quais operações devem ser executadas ao nível do processo de produção e organiza-as, de modo a otimizar o uso de recursos. Além disto, fornecem indicadores que permitem monitorizar o desempenho do processo de produção, *Key Process Indicators* (KPI).
- *Management Level*: É a camada de topo da pirâmide, em que as interações ocorrem nas maiores escalas temporais (horas, dias, meses), onde é efetuada a gestão global do sistema de automação. Sistemas como os ERP possibilitam gerir as encomendas, as matérias-primas, produtos intermédios e finais existentes, o que é necessário produzir, entre outros aspetos de alto nível relacionados com a produção.

No contexto deste trabalho, não foi considerada a camada de supervisão. Os sistemas ERP e MES serão discutidas com mais detalhe na secção 2.4, por serem o foco deste trabalho. Finalmente, a ferramenta desenvolvida na dissertação anterior permite implementar as camadas inferiores da pirâmide, ou seja, a camada de campo e a de controlo.

2.2 Trabalho prévio - A Digital Twin for Education With FactoryIO

Como já foi referido, nesta dissertação pretende-se utilizar o trabalho que foi realizado numa dissertação anterior, na edição 2021/22 do M.EEC (FEUP), denominado "*A Digital Twin for Education With FactoryIO*" [4]. Esta secção tem como finalidade apresentar o trabalho que foi efetuado no decorrer da dissertação referida. É de salientar que, por uma questão de coerência, é utilizada a nomenclatura definida pelo autor da ferramenta em questão.

O termo *Digital Twin* é amplamente utilizado, pelo que é necessário a sua contextualização para ser possível a compreensão da ferramenta que foi desenvolvida. O conceito de *Digital Twin* consiste numa réplica digital de uma entidade física [5]. Por ter as mesmas caraterísticas da sua contraparte física e encontrar-se no domínio digital, os *Digital Twins* permitem a realização de simulações fidedignas dos processos reais e também da expansão dos cenários de teste. Com isto, os resultados obtidos nessas simulações e a sua análise permitem aumentar a eficiência dos setores de produção reais [6].

Para além de ser útil ao nível prático, um *Digital Twin* possui, também, um grande valor para a componente educacional, uma vez que pode ser utilizado para substituir certos conjuntos de equipamentos que de outra forma seriam inacessíveis aos estudantes. Exemplificando, o *Digital Twin* pode ser utilizado para replicar uma linha de produção de uma fábrica real. Desta forma, torna-se possível o contacto dos estudantes com elementos constituintes de uma linha de produção, através da simulação, disponibilizando uma interface para a programação, controlo e supervisão dos mesmos.

Existindo esta necessidade na vertente educacional, foi desenvolvida, na dissertação anterior [4], um *Digital Twin* que permite simular e controlar um *layout* fabril. Esta ferramenta permite, por

si só, demonstrar o funcionamento das camadas de campo e controlo da pirâmide de automação, mas também pode ser utilizada como base para a implementação das camadas superiores.

2.2.1 FactoryIO

Para representar a camada de campo no domínio digital, é necessário um ambiente que simule os equipamentos presentes numa fábrica real, como tapetes de transporte, armazéns, máquinas, bem como os produtos que se pretendem produzir (transformação de matérias-primas em peças trabalhadas, por exemplo).

Para este efeito, foi utilizado o FactoryIO [7]. O FactoryIO é um simulador 3D de uma fábrica para o ensino de tecnologias de automação. Foi desenvolvido para ser de fácil utilização, permitindo rapidamente construir uma fábrica virtual selecionando um conjunto de elementos comuns em ambientes industriais. Devido à necessidade de implementar o controlo para cada elemento (que será explicado na secção seguinte), apenas foram considerados alguns dos elementos do vasto conjunto que o FactoryIO oferece. Assim, os elementos contemplados no *Digital Twin* são os seguintes:

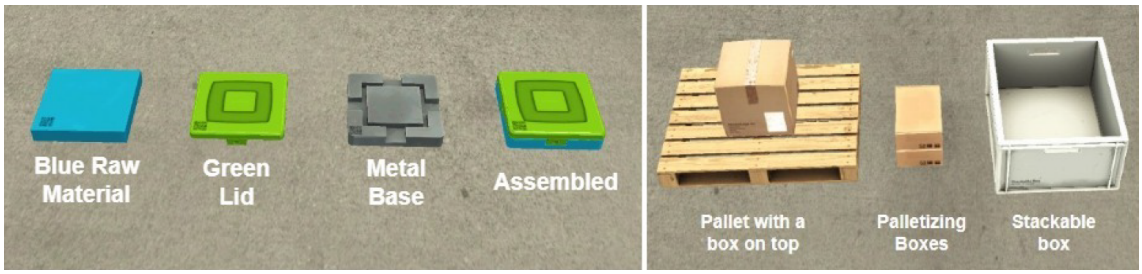


Figura 2.1: Itens leves (esquerda) e pesados (direita), retirado de [4]

- *Items* - Os elementos que fazem parte deste conjunto (figura 2.1) correspondem aos objetos, ou peças, que vão ser transportados ao longo da linha de produção, e podem ser agrupados pelo seu peso, definido pelo simulador. Os itens leves representam as matérias-primas e os produtos finais que se pretendem produzir na linha de produção. Existem 3 tipos de matérias-primas, azuis (*Blue*), verdes (*Green*) e metálicas (*Metal*). As matérias-primas podem ser transformadas em tampas (*Lid*) ou bases (*Base*), ou seja, peças processadas. Uma tampa e uma base podem ser colocadas uma em cima da outra, originando uma peça montada (*Assembled*). Por outro lado, os itens pesados, como as caixas e as paletes, são normalmente utilizados como base de transporte para os itens leves.

- *Emitters e Removers* - Os *Emitters* são utilizados para introduzir todos os *Items* pretendidos na linha de produção, enquanto os *Removers* servem para os retirar (figura 2.2).

- *Loads* - Servem para transportar os *Items* e, como tal, possuem a mesma divisão por peso. Para transporte de itens leves tem-se os *Conveyors* (figura 2.2) e os *Sorters* (figura 2.3), e os

seus equivalentes para pesados são os *Rollers* e as *Turntables* (figura 2.3). Os *Conveyors* e os *Rollers* são tapetes que podem ter 3 tamanhos (2, 4 ou 6 metros) e permitem transportar *Items* nos dois sentidos. Os *Sorters* e as *Turntables* são utilizados para conectar múltiplos equipamentos de transporte, permitindo variar os percursos.

- *Stations* - São equipamentos para processar os *Items*. Os *Machining Centers* transformam as matérias-primas (*Raw Materials*) em peças (*Lids* ou *Bases*), e os *Assemblers* permitem a montagem das peças (*Assembled*) (figura 2.4).
- *Warehouses* - Permitem armazenar até 54 *Items* (figura 2.4). No entanto, por limitações do FactoryIO, apenas é possível guardar itens pesados, pelo que, para guardar as matérias-primas ou as peças (que são o foco deste trabalho) é necessário colocá-las sobre uma palete, para que sejam armazenáveis.

Figura 2.2: *Conveyor*, *Emitter* e *Remover*, retirado de [4]

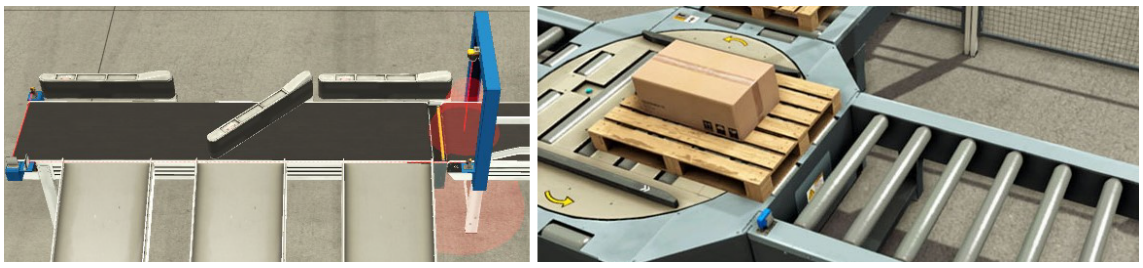


Figura 2.3: *Sorter*, *Turntable* e *Rollers*, retirado de [4][8]

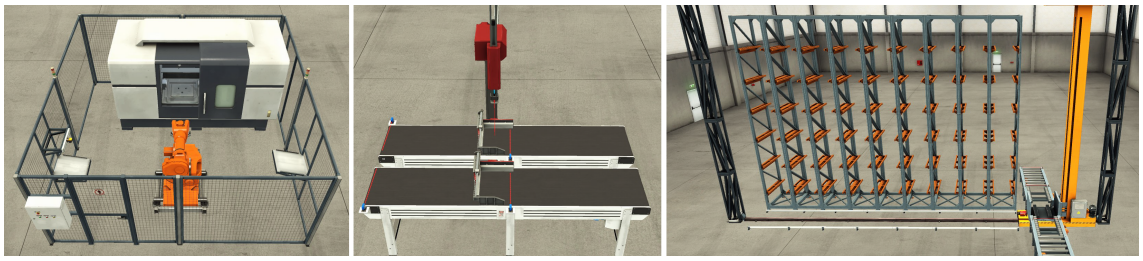


Figura 2.4: *Machining Center*, *Assembler* e *Warehouse*, retirado de [4]

2.2.2 CODESYS

Para implementar a camada de controlo, foi utilizada a ferramenta CODESYS [9]. O CODESYS é um ambiente de desenvolvimento para PLC que suporta as linguagens de programação definidas segundo a norma IEC 61131-3. No trabalho em questão, foram utilizadas as seguintes:

- *Structured Text* (ST): É uma linguagem de alto nível baseada em texto, semelhante a *Pascal* [4], podendo ser utilizada programas com funções aritméticas complexas, manipulação de vetores [10], entre outros. Na figura 2.5 está representado um excerto de código ST.

```
FOR I:=0 TO 2 DO
    IF (blocks.B[index[i]-1].block_id <> 0) AND (blocks.B[index[i]-1].INST = 0) AND (index[i]=5) THEN
        IF pers.COMUNIC_MES_OK THEN
            blocks.B[index[i]-1].INST:=1;
        ELSE
            blocks.B[index[i]-1].INST:=0;
        END_IF
```

Figura 2.5: Exemplo de código ST, retirado de [4]

- *Sequential Function Chart* (SFC): É uma ferramenta gráfica que permite a representação de processos e algoritmos sequenciais [4], através de estados que podem possuir múltiplas ações, e as respetivas transições entre estados (figura 2.6).

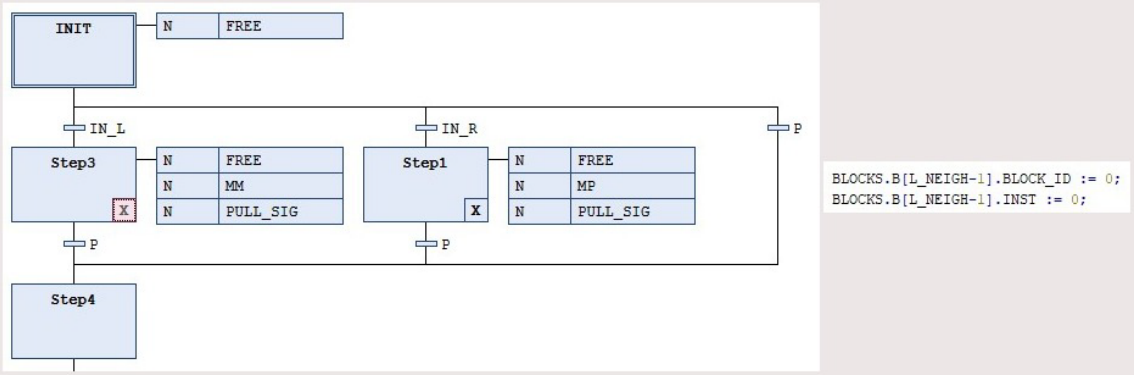


Figura 2.6: Exemplo de um diagrama SFC, retirado de [4]

- *Function Blocks* (FB): São *Program Organization Units* (POU) que possuem memória interna [4]. Isto permite que, para as mesmas entradas, sejam obtidas saídas diferentes [4][10]. Os FB seguem os pilares da programação orientada a objetos (abstração, encapsulamento, herança e polimorfismo) [4].

No contexto deste trabalho foi desenvolvida uma biblioteca de FB em que cada FB corresponde a um dos equipamentos do FactoryIO apresentados, como representado na figura 2.7. Para cada FB, foi desenvolvido um algoritmo sequencial utilizando SFC, cujas ações e transições foram programadas em ST.

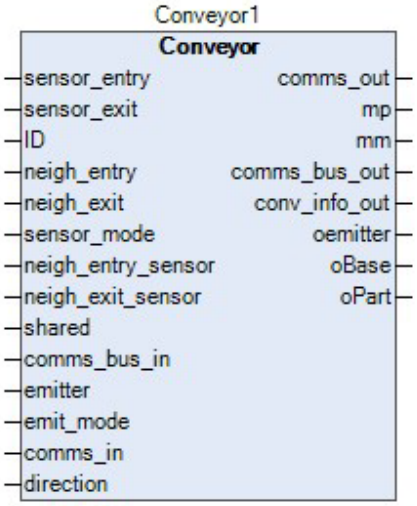


Figura 2.7: Exemplo um FB para um *Conveyor*, retirado de [4]

2.2.3 Arquitetura do Digital Twin

A arquitetura global do *Digital Twin* encontra-se representada na figura 2.8, seguindo o modelo da pirâmide de automação, em que cada camada é implementada pelas aplicações descritas nas secções anteriores.

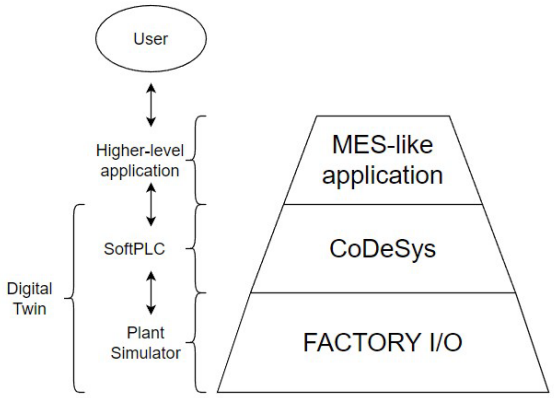


Figura 2.8: Arquitetura global, retirado de [4]

A camada de controlo, ou seja, a biblioteca de FB, apresenta uma arquitetura *plug-and-play*. Isto significa que, após a montagem dos componentes no FactoryIO, é possível montar o mesmo *layout* no CODESYS, selecionando a versão em FB dos equipamentos utilizados. Arrastando os FB e conectando-os aos seus adjacentes de tal forma que fiquem com a mesma disposição da fábrica, a camada de controlo é implementada sem a necessidade de modificação ou adição de código por parte do utilizador.

Exemplificando, na figura 2.9 encontra-se representado um *layout* simples, constituído por um *Emitter* e um *Conveyor* de entrada, um *Machining Center*, um *Conveyor* de saída e um *Remover*.

O seu mapeamento no *Digital Twin* é equivalente ao representado na figura 2.10.

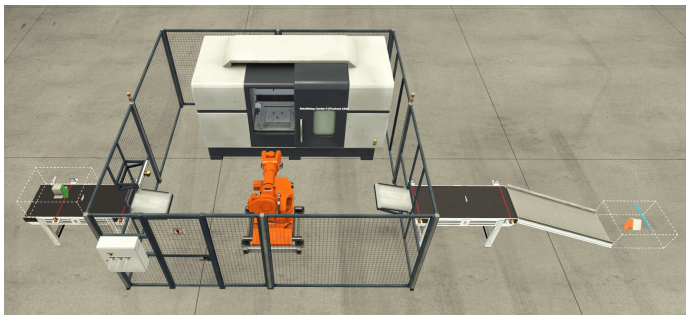


Figura 2.9: Exemplo de *layout* no FactoryIO

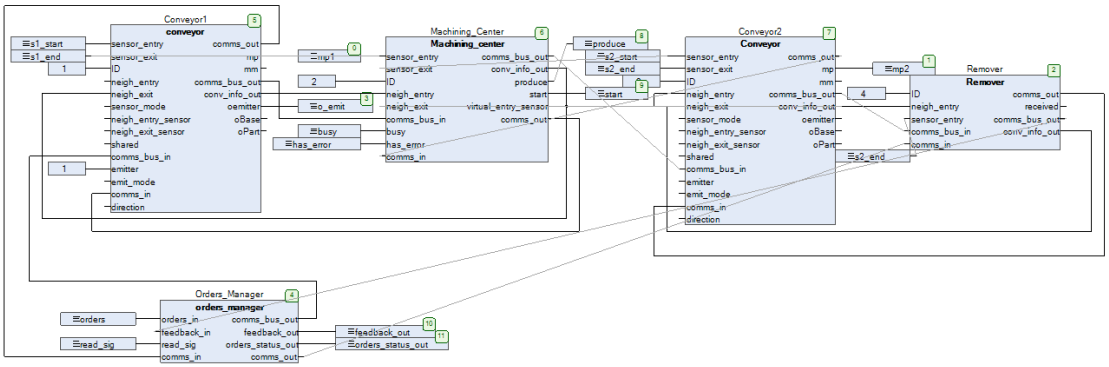


Figura 2.10: *Layout* mapeado para a biblioteca em CODESYS

2.2.4 Comunicação entre aplicações

A comunicação entre as aplicações do sistema é realizada através de uma interface de comunicação *Open Platform Communications Unified Architecture* (OPC UA) [11], que é um protocolo desenvolvido tendo em mente a unificação das comunicações industriais, seguindo uma arquitetura cliente-servidor. As aplicações que compõem o *Digital Twin* suportam OPC UA, facilitando a sua utilização do ponto de vista de desenvolvimento.

A ferramenta CODESYS disponibiliza uma interface para criar um servidor OPC UA, enquanto o FactoryIO pode funcionar como um cliente. Deste modo, foram definidas estruturas de dados no servidor existente no Codesys com dois objetivos: a criação de variáveis representativas dos sensores e atuadores, possibilitando o mapeamento dos equipamentos por parte do cliente existente no FactoryIO e, por outro lado, estruturas que possibilitam à aplicação superior enviar comandos ao *Digital Twin* e receber um *feedback*. As estruturas de dados em questão serão detalhadas na secção 2.2.6.

Além disto, na biblioteca de controlo desenvolvida, foi incluído um FB denominado *Orders Manager*, que não representa nenhum equipamento do FactoryIO, e tem apenas como objetivo a gestão das comunicações entre o *Digital Twin* e a aplicação superior (MES).

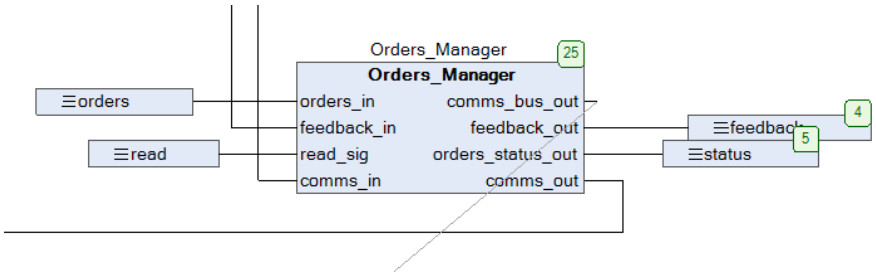


Figura 2.11: *Orders Manager*, retirado de [4]

2.2.5 Comunicação entre Function Blocks

Dada a arquitetura *plug-and-play* da biblioteca de FB, foram implementadas duas interfaces que possibilitam a comunicação entre FB.

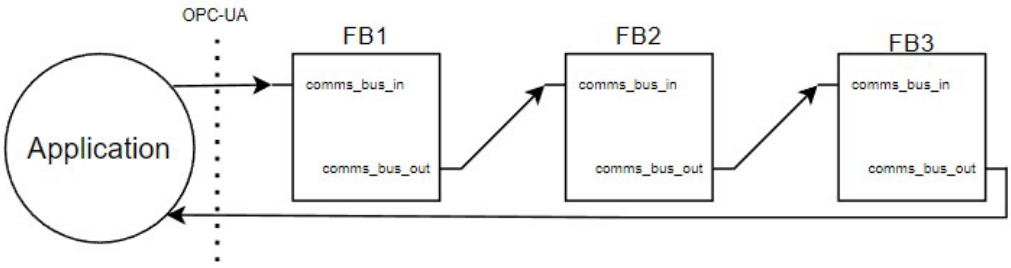


Figura 2.12: *Bus* de comunicação entre FB, retirado de [4]

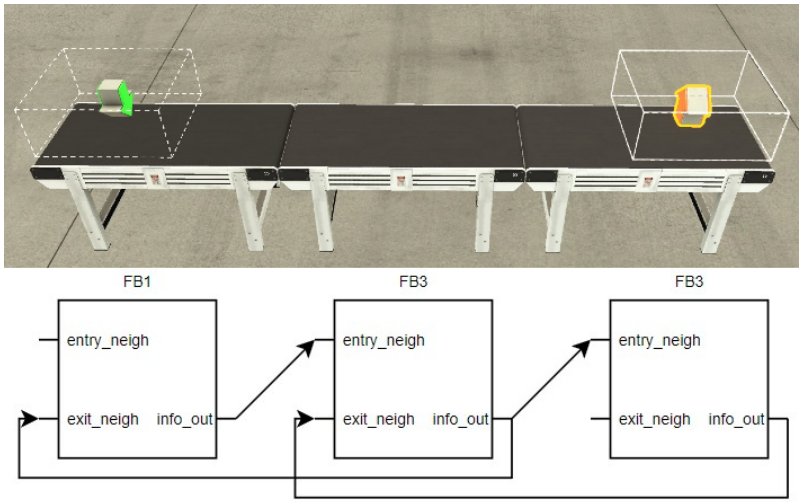


Figura 2.13: Comunicação sequencial entre FB, retirado de [4]

A primeira consiste num *bus* de comunicação partilhado por todos os FB. Quando a aplicação superior envia um comando para o *Digital Twin*, ou seja, para a estrutura de dados respetiva presente no servidor OPC UA, o FB *Orders Manager* (abordado na secção 2.2.4) envia as informações relativas a esse comando para o FB adjacente, que por sua vez envia informações para o seguinte

(figura 2.12). Este mecanismo permite que a aplicação superior envie comandos especificamente para um determinado FB.

A outra interface encontra-se relacionada com o transporte de *Items* do FactoryIO de um equipamento para outro. Durante a simulação, cada equipamento pode estar vazio, ocupado, a receber ou a enviar um *Item*. Para isto ser possível, foi implementado um mecanismo que permite a um FB obter o estado dos equipamentos adjacentes e também transmitir o seu estado (figura 2.13).

2.2.6 Estruturas de dados

Nesta subsecção serão apresentadas as principais estruturas de dados, e apenas serão mencionados os parâmetros relevantes para a dissertação atual. A descrição de todos os parâmetros, bem como dos seus possíveis valores, encontra-se detalhada no documento da dissertação anterior [4].

Como já foi referido, foi implementada uma estrutura que representa os *Items* existentes no FactoryIO, denominada *part*. Neste documento, a utilização do termo matérias-primas ou peças diz respeito a esta estrutura [4]:

- *id*: Identificador único da *part*, definido pela aplicação superior, no momento em que é emitida.
- *path*: Sequência de identificadores que indica ao *Digital Twin* qual será o caminho, ou seja, os equipamentos de transporte a serem percorridos por esta *part*. A sequência deve terminar com o valor -1, de modo a sinalizar o fim do trajeto.
- *type_base* e *type_part*: Permitem guardar qual é o tipo da *part* em questão (*BlueRawMaterial*, *MetalProductLid*, *GreenProductBase*, etc) e o tipo da base que a suporta (*Pallet*, *Box*, etc).
- *final_conveyor*: Identificador do último equipamento do trajeto, para o caso do encaminhamento automático.
- *store_position*: Posição do armazém em que deve ser colocada a *part*.
- *op*: Permite indicar qual a operação a ser executada pelos Machining Centers na *part* (transformação para base ou para tampa).
- *order_id*: Contém o identificador do comando que está a ser executado na *part*.

Para permitir a interação entre a aplicação superior e o *Digital Twin*, foi implementada uma estrutura que codifica os comandos que podem ser enviados, denominada *order*, constituída pelos seguintes parâmetros [4]:

- *id*: Identificador único do comando em questão, atribuído pela aplicação superior.

- *order_type*: Tipo do comando enviado. Os principais consistem na emissão de *parts* (*EMIT*), definição de novo caminho (*NEW_PATH*), remoção de uma peça do armazém (*OUT_WH*) e atualização das propriedades de uma *part* (*NEW_PROPS*).
- target_group: Identificadores dos equipamentos em que se pretende a execução do comando em questão.
- *new_data*: Variável que indica ao *Digital Twin* que um novo comando foi enviado pela aplicação.
- *order_status*: O estado de execução do comando, que permite à aplicação superior saber se o mesmo foi finalizado com sucesso.
- *remove_part*: Permite indicar quais *parts* devem ser removidas do armazém.
- *part_info*: Estrutura do tipo *part*, que permite à aplicação definir as propriedades da matéria-prima ou peça que se pretende emitir.

Quando uma *order* é finalizada, é colocada num *buffer* denominado *orders_status_out*, com o *order_status* correspondente a finalizado (*FINISHED*). Este *buffer* de *orders* finalizadas apenas possui 100 posições, pelo que é necessário que a aplicação superior envie um sinal para ser efetuada a sua limpeza antes que o seu limite de memória seja excedido. O sinal consiste no *rising edge* da variável *read_sig*.

Finalmente, na estrutura de dados *equip_info*, utilizada no transporte entre equipamentos mencionado na secção 2.2.5, destacam-se os seguintes parâmetros [4]:

- *id*: Identificador do FB do equipamento em questão.
- *free*: Variável que indica se o equipamento se encontra disponível para a receção de uma peça.
- *shared_resource*: Permite indicar aos equipamentos adjacentes se este se trata de um equipamento com recursos partilhados, ou seja, que possui mais do que uma entrada e/ou saída.
- *part*: Estrutura do tipo *part* que contém as informações relativas à matéria-prima ou peça que se encontra a ocupar o equipamento.

2.2.7 Encaminhamento das peças

O encaminhamento das peças (*parts*) pode ser efetuado de duas formas diferentes, conforme o pretendido pela aplicação superior. É importante salientar que, qualquer que seja o modo de encaminhamento, quando o percurso for terminado, a *part* em questão fica à espera de um novo comando, ocupando o último equipamento do percurso.

A primeira forma é a definição manual, por parte da aplicação superior, de toda a sequência de identificadores que codificam o trajeto a ser percorrido, preenchendo o vetor *order.part_info.path*. Este método pode ser utilizado para qualquer *part* que se encontre em espera na linha de produção.

A segunda opção é a definição automática do trajeto, onde a aplicação superior apenas preenche o valor *order.part_info.final_conveyor* com o identificador do equipamento de destino e a *part* é encaminhada pelo percurso mais curto existente. Isto é possível uma vez que todos os FB se encontram conectados na forma de um grafo, tendo sido implementado, para este efeito, o algoritmo de Dijkstra [12]. Este algoritmo permite encontrar o caminho mais curto entre dois vértices de um grafo, ou seja, neste contexto, o percurso mais curto entre dois equipamentos presentes na linha de produção. Apesar deste modo ser de extrema utilidade, verificou-se que apenas foi implementada a sua utilização para o momento em que uma *part* é emitida, ou seja, só é possível a sua utilização no primeiro trajeto que for percorrido.

2.2.8 Conclusões sobre o Digital Twin

Para a dissertação atual, o estudo detalhado do *Digital Twin* permitiu estabelecer de que forma o mini-MES (desenvolvido na dissertação atual) deverá efetuar o seu controlo. Em resumo, o mini-MES deve possuir um cliente OPC UA, de modo a escrever os valores correspondentes para os comandos pretendidos na estrutura de dados *order*. Para a receção de *feedback* da sua execução, o mini-MES deve consultar o estado das *orders* enviadas.

2.3 Tecnologias utilizadas

Nesta secção são apresentadas as tecnologias que foram utilizadas para a implementação do mini-ERP e do mini-MES, bem como os motivos da sua escolha.

2.3.1 JavaFX Scene Builder

A principal caraterística considerada na escolha da linguagem de programação a ser utilizada foi que a mesma suportasse o modelo de programação orientada a objetos (OOP). Esta caraterística facilita a implementação de aplicações que sejam estruturadas através de um modelo de dados que reflete a separação clara de diferentes entidades e das suas relações. No contexto deste trabalho, este modelo será apresentado na secção 4.1.

Para esta finalidade foi escolhida a linguagem Java [13]. Comparativamente a outras linguagens de diferentes níveis, o Java possui o equilíbrio desejado para este trabalho. Em comparação com a linguagem C++, de mais baixo nível, o Java possui uma gestão automática da memória, evitando possíveis erros nesta vertente. Por outro lado, a complexidade das aplicações a desenvolver não justifica a utilização de uma linguagem de tão alto nível como o Python. Além disto, garante uma grande portabilidade às aplicações desenvolvidas, uma vez que são executadas através da *Java Virtual Machine* (JVM), independentemente do sistema operativo.

Dado ser necessário a existência de uma interface gráfica para ambas as aplicações, foi utilizado a ferramenta *JavaFX Scene Builder* [14]. Esta ferramenta consiste num editor visual que permite a criação de uma interface gráfica sem a necessidade de código [14], arrastando os diversos componentes pretendidos para o *layout* da página em questão (figura 2.14). O código relativo à interface é automaticamente gerado em formato FXML (código 2.1), que depois é interpretado pelo compilador. Todo o resto da aplicação, incluindo os controladores para os componentes para cada interface, os modelos de dados e os algoritmos desenvolvidos são implementados em código Java.

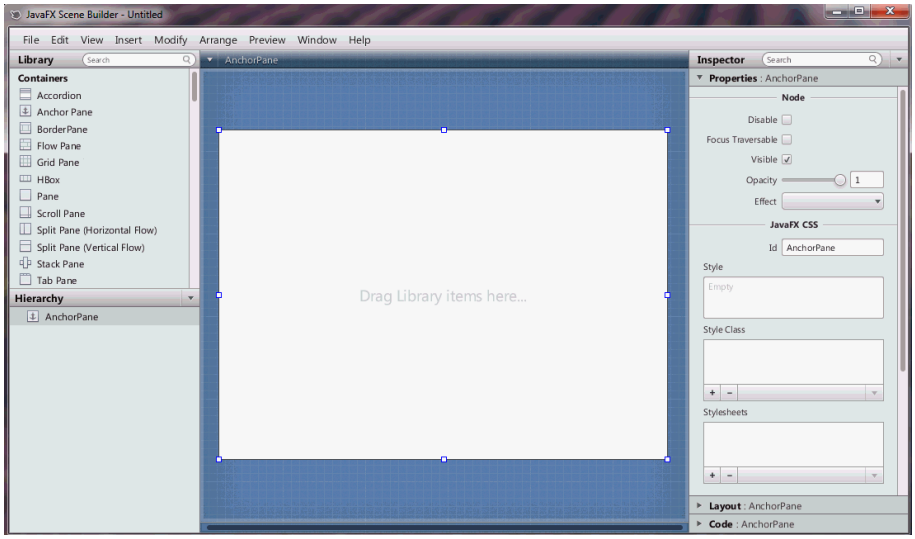


Figura 2.14: *Scene Builder*, retirado de [14]

```
<?xml version="1.0" encoding="UTF-8"?>
<GridPane fx:controller="fxmlexample.FXMLExampleController"
    xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10"
        vgap="10"
    styleClass="root">
    <padding><Insets top="25" right="25" bottom="25" left="25"/></padding>
    <Label text="User Name:" GridPane.columnIndex="0"
        GridPane.rowIndex="1"/>
    <TextField GridPane.columnIndex="1" GridPane.rowIndex="1"/>
    <Label text="Password:" GridPane.columnIndex="0"
        GridPane.rowIndex="2"/>
    <stylesheets>
      <URL value="@Login.css" />
    </stylesheets>
</GridPane>
```

Código 2.1: Exemplo de código FXML gerado, retirado de [14]

2.3.2 PostgreSQL

O PostgreSQL [15] é um sistema *open source* que permite a utilização da linguagem *Structured Query Language* (SQL) para a gestão de bases de dados relacionais. Esta linguagem foi utilizada para implementar o modelo de dados deste trabalho numa base de dados, através de comandos para a criação de novas tabelas (*CREATE*), a introdução de novas entradas (*INSERT*), consulta de dados por estruturas relacionais (*SELECT* e *JOIN*), entre outras operações.

A integração do PostgreSQL nas aplicações desenvolvidas em Java foi implementada utilizando a biblioteca *Java Database Connectivity* (JDBC) [16]. Esta biblioteca permite efetuar a conexão das aplicações Java à base de dados PostgreSQL, possibilitando que estas executem todas as operações SQL pretendidas.

2.4 Conceitos base das aplicações a desenvolver

Nesta secção serão apresentados os conceitos principais presentes nos ERP e MES comerciais, com o objetivo de perceber o seu funcionamento e determinar quais as funcionalidades que necessitam de estar presentes nas aplicações educativas a serem desenvolvidas neste trabalho.

2.4.1 Enterprise Resource Planning

Os *Enterprise Resource Planning* (ERP) são sistemas que auxiliam a gestão de uma empresa, integrando os processos na área da produção e manufatura, finanças e contabilidade, vendas e marketing, bem como recursos humanos [17]. Deste modo, as informações que anteriormente se encontravam dispersas pelos sistemas dos vários departamentos, passam a ser armazenados numa única aplicação que possui uma base de dados centralizada [17]. A centralização de todos os sistemas é vantajoso em duas vertentes. Por um lado, como têm acesso a todas as informações, ou seja, ao panorama geral da empresa, possibilitam a realização de planeamentos eficientes. Por outro lado, numa empresa, todos os departamentos interagem entre si, logo, a existência de um único sistema que capaz de integrar e automatizar os processos que exigem interações entre os diferentes departamentos, é de extrema utilidade.

Em [17], é ilustrado de que forma a receção de uma nova encomenda desencadeia uma sequência de ações em toda a empresa que podem ser automatizadas por um sistema como os ERP. Por exemplo, após a confirmação da encomenda, é necessário a reserva dos produtos no armazém, agendar a sua expedição e o envio da fatura ao cliente, por parte da contabilidade. De seguida, é preciso repor o *stock* em falta, pelo que pode ser necessário encomendar matéria-prima ao fornecedor e planear o seu processamento, de acordo com a disponibilidade da linha de produção. Dado existirem várias etapas e processos a decorrerem em paralelo, se a gestão for realizada por sistemas separados, a suscetibilidade a erros é elevada.

2.4.1.1 Estrutura

Para integrar as funcionalidades dos diferentes departamentos de uma empresa de forma coordenada, os ERP possuem, além de uma base de dados centralizada, uma estrutura composta por vários módulos de *software*, tal como representado na figura 2.15. Cada módulo é projetado de forma a atender às necessidades específicas do respetivo departamento. Em linhas gerais, os aspetos contemplados para cada um podem ser os seguintes [17]:

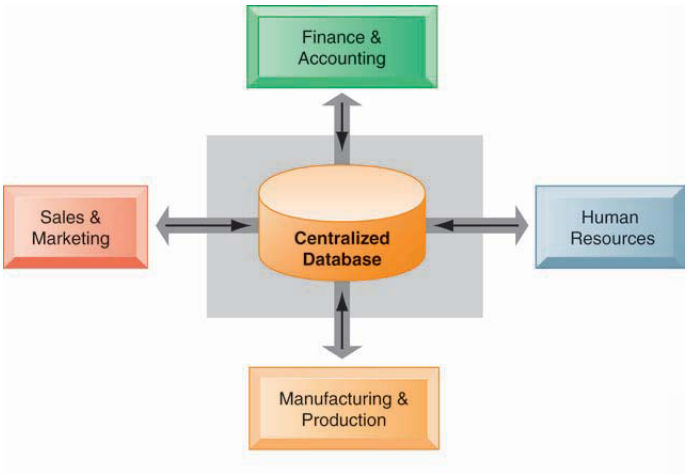


Figura 2.15: Estrutura de um ERP, adaptado de [17]

- Vendas e *marketing*: Encomendas, previsão de vendas, pedidos de devolução e mudanças de preços.
- Finanças e contabilidade: Dinheiro em caixa, contas a receber e rendimentos.
- Recursos humanos: Horários de expediente, custos de mão de obra e competências profissionais.
- Fabrico e produção: Matérias-primas, planeamento da produção, datas de expedição e capacidade de produção.

Por norma, um ERP tem de ser flexível a ponto de ser vendido por módulos. Assim, uma empresa não necessita de comprar o pacote completo, podendo escolher os módulos de *software* que mais se adequam ao seu modelo de negócio [18]. Por exemplo, uma indústria de produção terá, tipicamente, os módulos e as necessidades representadas na figura 2.16, como a gestão da linha de produção e do armazém, gestão financeira e corporativa e a gestão dos recursos humanos. Por outro lado, uma empresa que não possua um departamento de produção, pode optar somente pelos módulos de finanças, compras e vendas, e inventário [18]. Com isto, pretende-se que seja possível mapear o funcionamento da empresa no ERP, selecionando as funcionalidades necessárias [17].

Operations and Supply Chain		
Plant Maintenance	Purchasing	Quality Management
Sales and Distribution	Shop Floor Management	Transportation Management
Manufacturing	Warehouse Management	Advanced Planning
Financial Accounting		
General Ledger	Cash Management	Accounts Payable
Accounts Receivable	Fixed Assets	Financial Consolidation
Management Accounting		
Cost Center Accounting	Product Costing	Budgeting
Profit Center Accounting	Activity-Based Costing	Profitability Analysis
Human Capital Management		
Personnel Management	Payroll	Learning Management
Time and Attendance	Benefits	Recruitment Management

Figura 2.16: Módulos de um ERP para produção industrial, retirado de [18]

Utilizando a informação e os recursos dos diferentes módulos, o ERP realiza um planeamento de produção, sob a forma de ordens, que são enviados para as aplicações das camadas inferiores (por exemplo, um MES). As ordens contemplam procedimentos de armazenamento, produção e expedição de produtos.

2.4.1.2 Cadeia de abastecimento

Dado ser o responsável pelo planeamento da produção, o ERP tem um papel fundamental na cadeia de abastecimento, ou seja, em todos os intervenientes na produção e venda de um produto. Assim, é necessário perceber a intervenção do ERP na cadeia de abastecimento.

As incertezas na cadeia de abastecimento são causadas por falta de informação. Numa situação real, não é possível encomendar a quantidade exata de matéria-prima, para que esta chegue no dia exato do início da produção e seja realizada a expedição da quantidade exata de produto para o cliente assim que a produção termine [17]. Devido a isto, o ERP tem de realizar previsões e adotar estratégias para lidar com as incertezas, como a existência de um *stock* de segurança. O *stock* de segurança consiste numa quantidade adicional de inventário que uma empresa mantém em armazém para situações em que existe uma procura acima da média. Como é referido em [17], o *stock* de segurança atua como *buffer* pela falta de flexibilidade na cadeia de abastecimento.

Existem dois modelos para estruturar a cadeia de abastecimento, tendo cada um as suas vantagens e desvantagens (figura 2.17). O *push-based model* é um modelo que se baseia na previsão da procura dos produtos, e a produção é planeada de acordo com essa procura [17]. A principal vantagem é permitir um planeamento da produção com antecedência e uma diminuição do tempo de espera do cliente para a compra do produto. No entanto, encontra-se dependente do rigor das previsões realizadas, o que pode originar incerteza. O outro modelo é o *pull-based model* que possui uma menor incerteza relativamente ao anterior, pois o planeamento da produção é despoletado

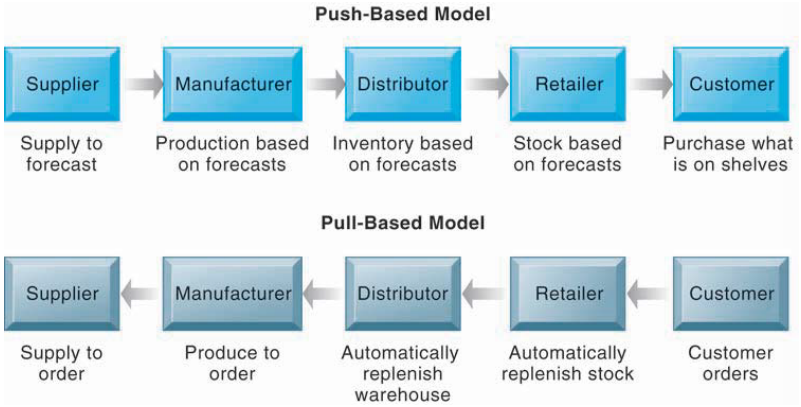


Figura 2.17: Comparação entre *push* e *pull models*, retirado de [17]

quando um cliente realiza uma encomenda [17]. Porém, neste modelo, é necessária uma espera por parte do cliente e variações da procura podem causar sobrecarga da produção, a ponto de não ser possível atender a todos os pedidos.

No contexto deste trabalho, pretende-se modelar o mini-ERP de modo que este efetue um reescalonamento na ocorrência de certos eventos. Como as encomendas de clientes fazem parte do conjunto de eventos de interesse, o modelo a ser utilizado será o *pull-based model*.

2.4.2 Manufacturing Execution System

Os *Manufacturing Execution Systems* (MES) são sistemas (aplicações) industriais que permitem a monitorização e o controlo da execução da produção [19]. No contexto da pirâmide de automação, estes sistemas atuam como a camada intermédia entre os ERP e a camada que implementa o controlo dos processos de fabrico [19].

2.4.2.1 Interação entre camadas

Como foi explicado na secção anterior, o ERP efetua o planeamento da produção, definindo as respetivas ordens de produção. A função do MES é receber este planeamento e controlar a produção de modo que as ordens planeadas sejam executadas pelas camadas inferiores. No sentido contrário, o MES envia para o ERP o *feedback* do estado do processo de produção.

A complexidade do MES está na comunicação com as camadas inferiores, ou seja, no controlo de forma coordenada do chão de fábrica. Para isto ser possível, é necessário que no MES seja mapeado o ambiente da produção, como as máquinas, os recursos humanos, recursos operacionais (ferramentas, meios de transporte), materiais, produtos intermédios e as suas respetivas descrições [20]. Deste modo, o MES consegue traduzir as ordens de produção do ERP (que têm um nível de abstração maior, relativamente ao chão de fábrica) numa sequência de comandos mais simples que podem ser enviados e executados pela camada de controlo.

Por ter este contacto direto em tempo real com a camada de controlo durante a produção, o MES é uma das principais fontes de informação do processo para as camadas superiores. Durante o processo de produção, são recolhidas informações como o estado das máquinas, dos equipamentos, entre outros aspetos, permitindo fornecer indicadores de desempenho (*Key Performance Indicators*) [20].

2.4.2.2 Otimização

Por possuírem acesso aos dados específicos sobre a produção, os MES permitem implementar algoritmos de decisão que tornam o chão de fábrica mais eficiente, realizando o controlo da produção de forma otimizada [19]. É importante salientar que geralmente não são utilizados algoritmos genéricos para todas as fábricas.

Nos contextos reais, os algoritmos de otimização são definidos e implementados de acordo com um estudo efetuado individualmente para cada *layout*, daí a importância dos dados de produção fornecidos pelos MES. Com isto, podem ser considerados fatores como os componentes alternativos de cada produto, deslocações e tempos de espera no processo de produção, alocação dos recursos de produção e prioridades nos tempos de entrega [20].

2.5 Estado da arte

No mercado, existem algumas soluções de *software* de simulação que podem ser utilizados no contexto educativo de demonstração das funções das camadas da pirâmide de automação, permitindo aos estudantes interagir com o ambiente simulado de modo a consolidar os conceitos pretendidos. Nesta secção são apresentadas algumas das soluções encontradas.

2.5.1 Soluções existentes

Na pesquisa que foi efetuada, encontraram-se soluções de *software* que incidem em diferentes vertentes. Para o ensino de conceitos associados aos ERP e de gestão empresarial, a principal ferramenta é o simulador *ERPsim*. Não foram encontradas aplicações que incidam especificamente sobre os MES, tal como o ERPsim está para os ERP. Existem simuladores que permitem a montagem de um *layout* fabril, tal como é efetuado pelo *Digital Twin* utilizado neste trabalho, e onde o utilizador pode controlar e otimizar o fluxo da linha de produção, como o *3DEXPERIENCE DELMIA*. Por outro lado, na vertente de contemplar tanto o planeamento efetuado pelos ERP, quanto o controlo, escalonamento e simulação da linha de produção, existe a ferramenta *Simio*, que tem uma complexidade superior às outras mencionadas.

2.5.1.1 ERPsim

O *ERPsim*[1] é um jogo de simulação empresarial que permite aos utilizadores gerir uma empresa virtual utilizando um ERP semelhante às soluções comerciais distribuídas pela SAP[19], que é uma das empresas mais relevantes no ramo dos ERP.

O ERPsim apresenta todas as informações da empresa virtual em tempo real e possibilita que o utilizador tome decisões ao nível da gestão empresarial. O ambiente simulado permite a visualização do impacto das decisões tomadas, além de ser feita uma avaliação das mesmas com base em dados recolhidos em contexto real. Isto promove a aprendizagem de vários conceitos de gestão empresarial através de uma metodologia *learn by doing*, assim como a perceção do funcionamento de um ERP real.

2.5.1.2 3DEXPERIENCE DELMIA

O *3DEXPERIENCE DELMIA for Education*[2] é uma plataforma educativa que permite aos estudantes interagir com um ambiente industrial virtual (figura 2.18), com o objetivo de adquirir competências relevantes para o seu futuro na indústria.

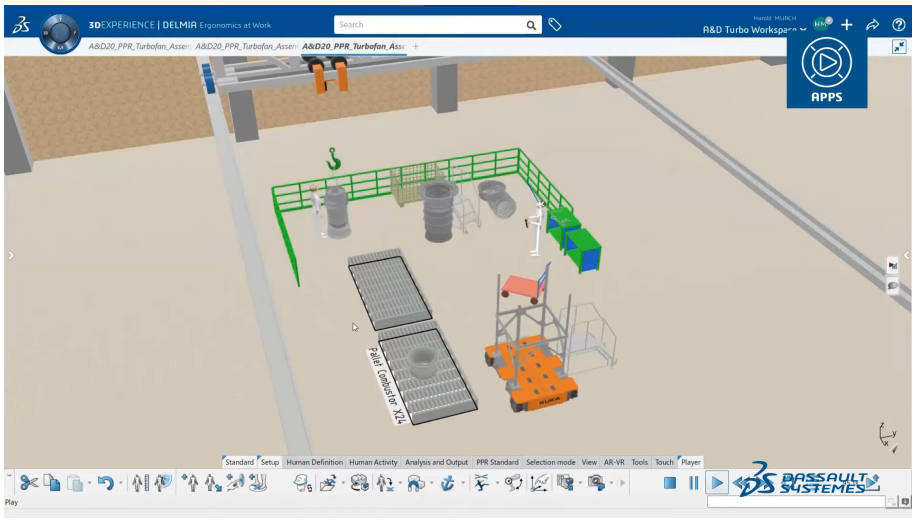


Figura 2.18: Simulação no 3DEXPERIENCE DELMIA, retirado de [21]

Esta ferramenta contempla a montagem e a simulação de uma linha de produção, a automação e o controlo dos seus equipamentos e, em particular, a programação e validação de sistemas de montagem robóticos, pretendendo demonstrar a relação entre o processo de fabrico, os produtos que se pretendem produzir e a alocação dos recursos existentes. Ao nível do controlo da produção, possibilita que os estudantes experimentem técnicas para a otimização dos processos que decorrem

[1] https://erpsim.hec.ca/en/erpsim

durante a simulação e que são utilizadas em contexto real, com foco na redução dos custos e desperdícios associados.

2.5.1.3 Simio

O Simio[3] é uma ferramenta completa que permite simular o funcionamento de sistema de fabrico desde a representação da linha física, ao seu controlo e planeamento das ordens de alto nível (figura 2.19). Esta ferramenta foi desenvolvida com o intuito de possibilitar a construção e execução de modelos dinâmicos representativos de sistemas reais, para avaliar o seu desempenho.

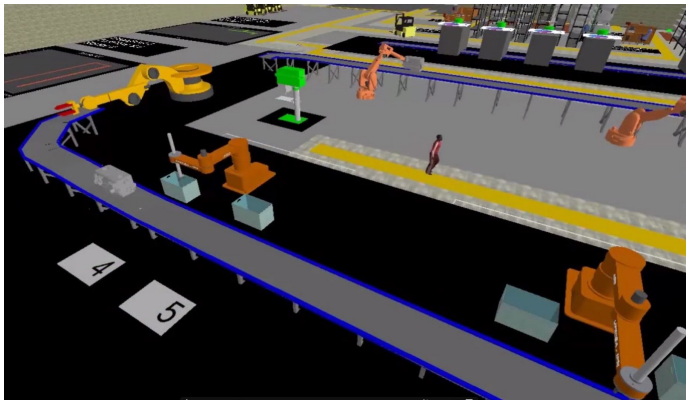


Figura 2.19: Simulação de processo de fabrico no Simio, retirado de [22]

Assim, permite a montagem da linha de produção pretendida recorrendo a uma biblioteca de equipamentos, contemplando diferentes industrias como a de fabrico, cadeia de abastecimento, cuidados de saúde, transportes, entre outros. Tendo o sistema construído e a sua lógica definida, podem ser implementados os modelos de planeamento e escalonamento da produção a serem executados durante a simulação. É disponibilizada muita informação que possibilita avaliar o desempenho em diferentes vertentes como, por exemplo, através de uma análise baseada em riscos (figura 2.20).

	Target Ship Date - Plan		Target Ship Date - Risk Analysis		Target Cost - Plan		Target Cost - Risk Analysis	
Order ID	Value	Status	Expected	OnTime Probability	Value (USD)	Status	Expected (USD)	OnBudget Probability
⊞ Order 14	12/12/2014 9:39:50 AM	OnTime	12/12/2014 10:50:54 PM	93.34%	134,801.1328	Overrun	137,991.6704	6.66%
⊞ Order 15	12/11/2014 11:32:55...	OnTime	12/12/2014 11:23:22 AM	79.47%	126,547.2111	OnBudget	132,261.0780	51.73%

Figura 2.20: Exemplo de resultados da análise de risco no Simio, retirado de [22]

Esta ferramenta é amplamente utilizada com o objetivo de simular o impacto que determinadas alterações teriam na eficiência e nos resultados do processo de produção num contexto real. Por este motivo, é uma ferramenta completa e com um grau de elevado de complexidade.

[2] https://www.3ds.com/edu/education/academics/solutions/3dexperience/delmia-package
[3] https://www.simio.com/index.php

2.5.2 Discussão das soluções encontradas

Apesar das soluções mencionadas poderem ser utilizadas numa vertente didática, não são adequadas ao contexto pretendido para este trabalho por alguns motivos.

O ERPsim é a ferramenta preferencial para a aprendizagem dos conceitos associados aos ERP, baseando-se nos sistemas reais, mas possuindo um grau de simplicidade adequado ao contexto educativo. No entanto, apenas contempla os processos ao nível do ERP, deixando de parte as restantes camadas da pirâmide de automação. Em contrapartida, o 3DEXPERIENCE DELMIA, trata-se do inverso, focando-se somente na simulação do ambiente fabril, no seu controlo e otimização.

Deste modo, a ferramenta Simio é a mais próxima do que se pretende implementar neste trabalho, uma vez que permite o planeamento, controlo das operações e representação física da linha de produção. Apesar disto, possui uma complexidade demasiado elevada para o contexto pretendido, uma vez que é uma ferramenta desenvolvida para a análise dos processos reais num domínio digital.

2.6 Conclusões

Neste capítulo foram apresentados todos os conceitos fundamentais para a compreensão deste trabalho. Tem-se como objetivo a demonstração do funcionamento da pirâmide de automação para fins educativos, pelo que a contextualização detalhada do modelo é fundamental. Além disto, o estudo realizado à ferramenta da dissertação anterior utilizada permitiu perceber o seu funcionamento e de que modo pode ser utilizada, o que é essencial para a definição da arquitetura, em particular, do mini-MES. De seguida, foram apresentadas, as tecnologias que foram utilizadas para a implementação em concreto das aplicações, justificando a sua escolha em detrimento de outras.

Foram discutidos os conceitos principais que permitem perceber o contexto e as funções dos sistemas ERP e MES. O conceito de arquitetura modular, para o caso do ERP, revela ser de extrema utilidade, uma vez que por estes sistemas serem muito complexos, é necessária uma divisão por funcionalidades de modo a permitir a sua implementação. Também ao nível do MES, destaca-se a importância como intermediário entre a camada de topo e as da base. No entanto, alguns aspetos apresentados na literatura referida enquadram procedimentos aplicados em contexto real sendo, por isso, demasiado complexos. Este trabalho visa a vertente educacional, pelo que algumas funcionalidades aqui apresentadas não serão utilizadas na definição da arquitetura das aplicações, que se encontra explicada no capítulo seguinte.

Finalmente, foram apresentadas, em linhas gerais, soluções de *software* que podem ser utilizadas num contexto educativo, tendo sido enunciados os motivos que justificam o desenvolvimento das aplicações pretendidas neste trabalho (mini-ERP e mini-MES), em detrimento do uso de outras já existentes.

Capítulo 3

Arquitetura proposta

Neste capítulo, é apresentada a arquitetura das aplicações desenvolvidas. Inicialmente, é efetuada uma contextualização dos conceitos-base considerados como ponto de partida deste trabalho. De seguida, são analisados os requisitos e apresentada uma arquitetura de alto nível onde são estabelecidas as interações entre as aplicações. Finalmente, é apresentada a arquitetura proposta para o mini-ERP e para mini-MES, descrevendo os algoritmos que permitem implementar as funcionalidades pretendidas.

3.1 Contextualização

3.1.1 Matérias-primas e produtos finais considerados

Inicialmente, é necessário explicitar quais *Items* do *Digital Twin* (apresentados na secção 2.2.1) foram contemplados neste trabalho. Como matérias-primas, consideraram-se os 3 tipos de materiais disponíveis, *BlueRawMaterial* (azul), *GreenRawMaterial* (verde) e *MetalRawMaterial* (metal), representados na figura 3.1.

Figura 3.1: Matérias-primas, retirado de [7]

Estas matérias-primas podem ser transformadas em peças como tampas (*BlueProductLid*, *GreenProductLid* e *MetalProductLid*) ou bases (*BlueProductBase*, *GreenProductBase* e *MetalProductBase*), sendo estes os 6 produtos finais considerados (figura 3.2). A sua montagem, ou seja, as peças *Assembled*, não foi contemplada.

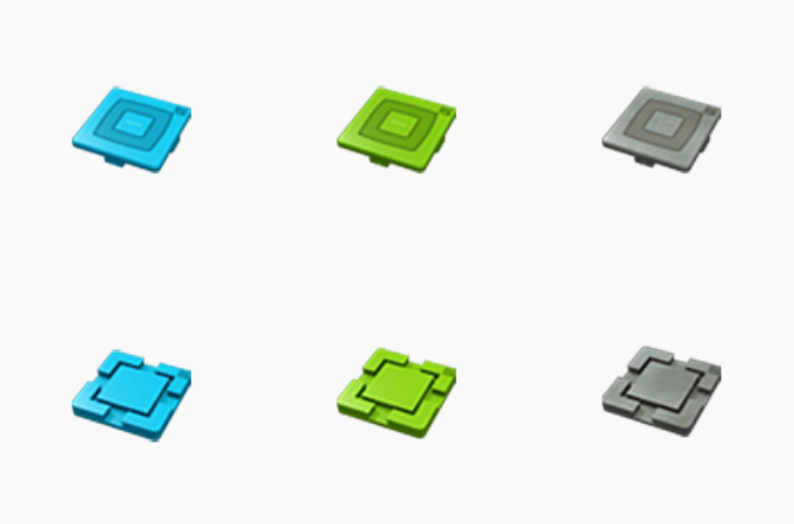


Figura 3.2: Tampas (em cima) e bases (em baixo), retirado de [7]

3.1.2 Divisão do layout e ordens de produção

Como já foi discutido, o *Digital Twin* é uma ferramenta genérica, na medida em que possibilita a montagem de qualquer *layout*, de entre os equipamentos que foram considerados. No contexto do trabalho atual, foi necessário impor algumas restrições aos *layouts* que podem ser montados, para ser possível a demonstração dos conceitos relativos ao funcionamento de um ERP e um MES. Deste modo, como ponto de partida para o desenvolvimento da arquitetura das aplicações, a presença de alguns equipamentos foi considerada obrigatória, uma vez que não faria sentido contemplar um *layout* que não possuísse equipamentos para o processamento das peças, ou que não possuísse um armazém, por exemplo.

Assim, optou-se por dividir a fábrica em 3 zonas diferentes, como representado no diagrama de blocos da figura 3.3, em que os armazéns atuam como um recurso partilhado e delimitam as zonas. A função da zona de entrada é receber as matérias-primas que chegam à fábrica, vindas dos fornecedores, e guardá-las no armazém de matérias-primas. Na zona de produção é necessário que existam máquinas para transformar as matérias-primas em produtos finais, que vão ser guardados no armazém de produtos finais. Finalmente, a zona de expedição é onde os produtos finais vão ser expedidos para os clientes. Isto permite estabelecer um fluxo de produção que garante ao mini-MES que o percurso que cada peça irá efetuar inicia-se na zona de entrada e termina na zona de expedição.

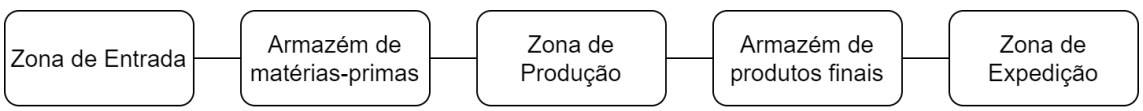


Figura 3.3: Divisão do *layout*

Além disto, foram definidas ordens de produção, que traduzem as operações que vão ser realizadas a um determinado conjunto de peças, numa determinada zona, da seguinte forma:

- *Inbound Orders*: Executadas na zona de entrada. Contemplam o armazenamento das matérias-primas quando estas chegam à linha de produção.
- *Production Orders*: Executadas na zona de produção. Indicam quais as matérias-primas que devem ser retiradas do armazém e processadas para produtos finais.
- *Expedition Orders*: Executadas na zona de expedição. Contemplam a expedição dos produtos finais para os respetivos clientes.

3.1.3 Peças com defeito

O processo de transformação de matérias-primas no *Digital Twin* não contempla a possibilidade de se obterem peças defeituosas. No entanto, considerou-se este aspeto como sendo de interesse didático para este trabalho, uma vez que a existência de produtos defeituosos implica que o algoritmo do mini-ERP tem de ter em conta que nem todas as matérias-primas serão convertidas em produtos finais utilizáveis, obrigando a reescalonamentos.

Assim, pretende-se que, ao nível do mini-MES, seja utilizada uma métrica probabilística de modo a avaliar cada peça processada como sendo ou não defeituosa, simulando as células de inspeção de qualidade existentes nos contextos reais. No final da zona de produção, as peças com defeito devem ser destruídas, enquanto as outras seguem para o armazém de produtos finais.

3.1.4 Clientes e fornecedores

Como no contexto deste trabalho não existem clientes e fornecedores reais, foi necessário introduzir na simulação do comportamento destes intervenientes externos. Efetivamente, seria possível desenvolver módulos totalmente separados da aplicação que simulassem clientes e fornecedores e que, por sua vez, comunicassem com o mini-ERP. No entanto, considerou-se que a simulação poderia ser contemplada pela aplicação do mini-ERP, desde que as interfaces estivessem devidamente separadas, diminuindo a complexidade do desenvolvimento do sistema e facilitando a sua utilização por parte do utilizador.

Assim, optou-se por dividir a interface gráfica do mini-ERP conforme o ponto de vista que o utilizador pretenda assumir em cada interação:

- Interface interna: Permite ao utilizador interagir com o sistema do ponto de vista de um gestor interno da fábrica, como visualizar o planeamento da produção, as encomendas dos clientes e dos fornecedores, e tomar decisões que vão afetar todo o funcionamento.
- Interface do cliente: Permite ao utilizador assumir o papel de um determinado cliente, possibilitando o envio de pedidos de encomenda, denominados *Client Orders*, ou cancelar encomendas em curso.

- Interface do fornecedor. Permite ao utilizador editar os parâmetros de um determinado fornecedor e visualizar as encomendas de matérias-primas em curso, denominadas *Supplier Orders*.

3.2 Requisitos

Os requisitos para cada uma das aplicações foram estabelecidos tendo por base apenas as funcionalidades principais de ERP e MES apresentadas no capítulo 2 e as interações que ocorrem com elementos externos (clientes, fornecedores), uma vez que são estes aspetos que possuem um maior interesse para demonstrar numa vertente educativa. Assim, encontram-se listados na tabela 3.1 os requisitos estabelecidos para o mini-ERP, que podem ser classificados conforme a interface a que dizem respeito. Já os requisitos específicos para o mini-MES, podem ser consultados na tabela 3.2.

Tabela 3.1: Requisitos do mini-ERP

ID	**Descrição**	**Interface**
01	As informações do sistema deverão ser armazenadas numa base de dados.	Interna
02	Deve ser possível definir a capacidade máxima dos armazéns e a capacidade de produção semanal.	Interna
03	O utilizador deve poder registar novos clientes e fornecedores no sistema.	Interna
04	Ao receber uma nova encomenda, deve ser feita uma previsão da entrega.	Interna
05	O ERP armazena o planeamento da produção sobre a forma de ordens na base de dados.	Interna
06	Do ponto de vista interno, deve ser disponibilizado todo o planeamento da produção, bem como o estado das encomendas de clientes e fornecedores.	Interna
07	O planeamento realizado pelo ERP deve ter em consideração a capacidade de produção semanal e a capacidade máxima dos armazéns.	Interna
08	Após apresentado o planeamento realizado, o utilizador deve ter a opção de aceitar ou rejeitar uma encomenda de um cliente.	Interna
09	O ERP deve ser capaz de realizar um reescalonamento da produção na ocorrência dos seguintes eventos: atraso de matérias-primas, cancelamento de encomendas e produção de peças com defeito.	Interna
10	O envio de encomendas de clientes deve ser simulado, no qual o utilizador indica o tipo e a quantidade de produto pretendido.	Cliente
11	Após receber uma resposta com a previsão de entrega de uma encomenda, deve ser possível ao utilizador, no papel de cliente, aceitar ou recusar a proposta.	Cliente
12	Deve ser dada a possibilidade de um cliente cancelar uma encomenda em qualquer momento.	Cliente

13	Os parâmetros de cada fornecedor devem poder ser editados a qualquer momento.	Fornecedor
14	Na interface de fornecedor, deve ser possível o utilizador atrasar a chegada de matérias-primas.	Fornecedor

Inicialmente, ao estabelecer os requisitos, foi necessário definir uma unidade temporal a ser considerada neste trabalho, como é possível verificar pela referência à capacidade de produção mencionada no requisito 2. Este requisito tem como objetivo impor algum realismo ao ambiente simulado, que de outro modo teria um tempo e capacidade de produção infinito. Assim, pretende-se que o mini-ERP realize todo o seu planeamento tendo em conta o limite imposto para cada ciclo de produção (requisito 7), que funcionará como unidade temporal.

Para facilitar a compreensão deste trabalho, foi denominada a semana como sendo este ciclo/unidade. Exemplificando, o utilizador pode definir que o limite sejam 10 produtos. Deste modo, é efetuado um planeamento que restringe a produção semanal à quantidade indicada. Uma semana é dada como terminada quando todas as ordens existentes no seu planeamento estejam finalizadas, sendo que este foi efetuado tendo em conta que a fábrica só poderá produzir, no máximo, 10 produtos em cada ciclo de produção.

O registo de novos clientes e fornecedores no sistema é realizado na interface interna (requisito 3) para que o utilizador possa, posteriormente, efetuar determinadas ações nas interfaces específicas de cada um como, por exemplo, alterar os seus próprios parâmetros, para o caso dos fornecedores (requisito 13). O evento de maior importância consiste na realização de um pedido de uma nova encomenda por parte de um cliente (requisito 10), ficando a aguardar uma resposta. Ao receber um novo pedido, deve ser executado o algoritmo de escalonamento para atualizar o planeamento e assim obter uma previsão de entrega (requisito 4).

Visualizando todo o planeamento realizado (requisito 6), o utilizador deve decidir quais encomendas aceitar ou recusar (requisito 8) do ponto de vista interno. Após isto, também é solicitada uma reposta na interface do cliente, tendo um custo final e uma previsão da semana de entrega, deixando novamente a decisão ao utilizador (requisito 11).

Para além disto, poderão, a qualquer momento, ocorrer eventos que originem uma alteração no planeamento já efetuado (requisito 9) seja devido ao *feedback* da produção do mini-MES, ou por interferência do utilizador (no papel de agente externo). Para este efeito, é possível cancelar encomendas em curso na interface do cliente (requisito 12) ou atrasar a chegada de matérias-primas através da interface do fornecedor (requisito 14).

Tabela 3.2: Requisitos do mini-MES

ID	Descrição
15	Para cada semana, o MES deve obter o respetivo planeamento da produção realizado pelo ERP e executar as ordens para as diferentes zonas da linha de produção (*Inbound*, *Production* e *Expedition*)
16	O estado da execução das ordens deve ser atualizado na base de dados.
17	O MES é o responsável por atualizar os dados de produção de cada peça individualmente (semana de chegada, semana de expedição, tempo de produção) na base de dados.
18	Na interface do MES, deve ser possível visualizar dados do estado da produção em tempo real.
19	O MES deve apresentar o histórico de produção de cada máquina presente no layout.
20	Deve existir uma interface que permita visualizar a localização de cada peça nos armazéns.
21	O MES deve permitir mapear o *layout* da linha de produção que foi construído no *Digital Twin*.
22	O MES deve possuir uma interface de comunicação que permita o envio de comandos para o *Digital Twin*, bem como *feedback* do seu estado.
23	Deve ser possível indicar a probabilidade de uma peça ser considerada defeituosa.

Os requisitos englobando a base de dados e a interação entre as aplicações (requisitos 1, 5, 15, 16, 17) serão abordados na secção referente à arquitetura de alto nível da aplicação (secção 3.3), uma vez estarem diretamente relacionados com a mesma.

Pretende-se que a interface do MES tenha um foco maior em apresentar as informações de cada produto de forma individualizada, como detalhes da produção atualizados em tempo real (requisito 18), o seu histórico (requisito 19) e a ocupação dos armazéns (requisito 20). O controlo da produção dá-se através do controlo do *Digital Twin*, sendo necessário um mapeamento do layout da linha de produção (requisito 21) e uma interface de comunicação (requisito 22). Também é dada a opção ao utilizador de definir o quão frequente serão produzidas produtos com defeito (requisito 23), assim como mencionado na secção 3.1.3.

3.3 Arquitetura de alto nível

A arquitetura de alto nível (figura 3.4) foi definida considerando os requisitos relativos às interações entre as aplicações. Seguiu-se o conceito de uma base de dados centralizada apresentado na secção 2.4.1, embora, neste caso, se trate de um contexto mais simples, com apenas duas aplicações. Optou-se por uma base de dados relacional (requisito 1), uma vez que todas as informações a serem armazenadas se encontram interligadas e podem ser facilmente relacionadas entre si utilizando as chaves primárias e estrangeiras. Além disto, também viabiliza a possibilidade de outras aplicações que sejam desenvolvidas futuramente utilizarem estes dados.

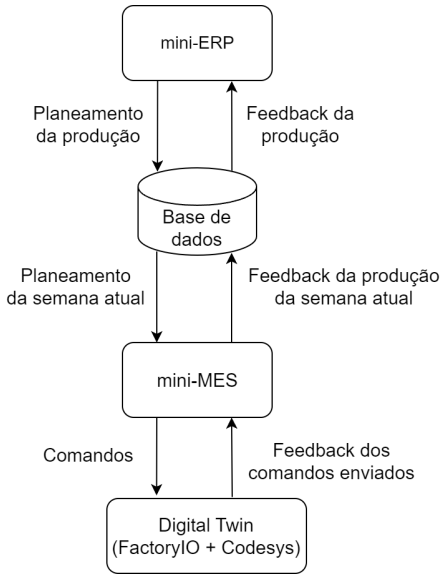


Figura 3.4: Arquitetura de alto nível

A interação entre o mini-ERP e o mini-MES consiste no envio/receção do planeamento da produção e do *feedback* da sua execução (requisitos 5, 15, 16 e 17). O mini-ERP realiza um planeamento sob a forma dos 3 tipos de ordens descritos na secção 3.1.2, *Inbound*, *Production* e *Expedition orders*, que são armazenadas na base de dados.

Posteriormente, o mini-MES consulta a base de dados para obter o planeamento da produção semanal. O mini-MES efetua o controlo da produção enviando comandos para o *Digital Twin* através da interface disponibilizada, permitindo a monitorização do processo através do *feedback* do estado dos comandos enviados. Por exemplo, uma ordem gerada ao nível do mini-ERP (*Inbound*, *Production* ou *Expedition*) é traduzida numa sequência de comandos pelo mini-MES, enviada para o *Digital Twin* e, após ser recebido o *feedback* indicando que todos os comandos foram executados com sucesso, a ordem é considerada como finalizada e o seu estado é atualizado na base de dados. Todo este processo é analisado com mais detalhe na secção 3.5, referente à arquitetura do mini-MES.

Para efeitos de simplificação dos diagramas das próximas secções referentes à arquitetura específica de cada uma das aplicações, é considerada uma comunicação direta entre o mini-ERP e o mini-MES, sem o intermédio da base de dados.

3.4 Arquitetura do mini-ERP

Em termos conceptuais, definiu-se uma arquitetura modular para o mini-ERP consistindo num conjunto de módulos que interagem entre si, representado na figura 3.5. Esta separação facilitou a conceção dos algoritmos para os eventos que podem ocorrer no mini-ERP, dado que estes podem ser traduzidos como um conjunto de interações entre estes módulos.

O módulo central é o módulo de planeamento, sendo este que comunica com o MES e com os restantes módulos. É o responsável por recolher toda a informação e tomar decisões, sejam estas de encomendas a fornecedores ou de escalonamento da produção. Possui ligação direta ao módulo de armazém, cuja única função é disponibilizar o estado do inventário dos armazéns do *Digital Twin*, ao nível de matérias-primas e produtos finais. O módulo de vendas funciona como intermediário entre o módulo do cliente e de planeamento, pois transmite os dados das novas encomendas recebidas e também é o responsável por permitir ao cliente tomar decisões que irão interferir com o processo de planeamento.

Como já foi abordado na secção 3.1.4, para o âmbito deste trabalho, é necessário incluir módulos de simulação de entidades externas, como clientes e fornecedores, que podem ser controlados pelo utilizador.

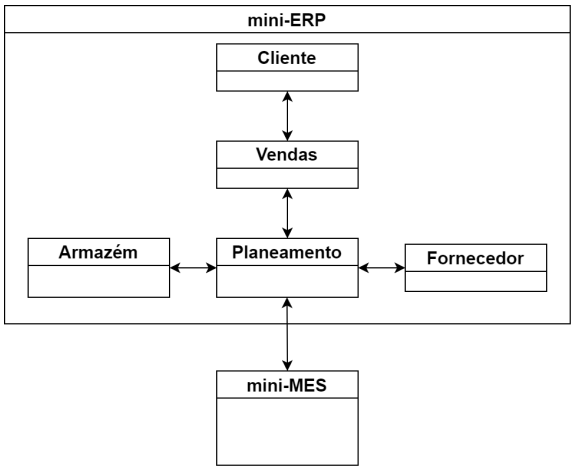


Figura 3.5: Módulos do mini-ERP

3.4.1 Introdução de dados por parte do utilizador

Antes de analisar os diferentes eventos que ocorrem ao nível do mini-ERP, é necessário elencar quais as formas de interação com o utilizador, nomeadamente no quesito de introdução de dados. Como já mencionado anteriormente, pretende-se que o utilizador defina a capacidade de produção semanal e o limite máximo dos armazéns, o que vai afetar diretamente o módulo de planeamento, tendo em consideração que o *FactoryIO* tem uma capacidade máxima por armazém de 54 peças.

Relativamente a fornecedores, cada fornecedor registado pelo utilizador pode fornecer entre 1 a 3 tipos dos materiais que foram apresentados na secção 3.1.1. Para cada material é possível definir o seu preço por unidade, assim como o tempo de entrega em semanas. Ainda na interface de fornecedor, o utilizador pode indicar quantas semanas pretende atrasar uma determinada encomenda de matérias-primas, desde que este procedimento seja feito uma semana antes da previsão inicial da entrega.

Finalmente, em cada pedido de nova encomenda, o utilizador, no papel de cliente, pode escolher entre os 6 tipos de produtos finais apresentados na secção 3.1.1, bem como a sua quantidade. Também é requisitado que seja indicado uma preferência entre tempo de entrega mínimo ou custo mínimo, para servir de critério ao algoritmo na escolha de um fornecedor, uma vez que não existe nenhuma limitação na definição dos seus parâmetros. Após uma encomenda estar aprovada por ambas as partes, é possível o seu cancelamento em qualquer momento, desde que seja anterior à semana de expedição.

3.4.2 Encomendas de clientes

O processamento de uma encomenda é um procedimento complexo que envolve uma interação entre todos os módulos do mini-ERP, tendo sido definido o algoritmo da figura 3.6 que será executado para cada nova encomenda registada. Durante a sua execução, o utilizador pode tomar decisões do ponto de vista de gestor interno da fábrica ou de cliente.

Ao ser gerada uma encomenda, a mesma é registada no módulo de vendas. Para este efeito, o módulo separa as encomendas por 3 tipos conforme o seu estado (por confirmar, confirmadas e finalizadas) sendo que no momento inicial, a encomenda é definida como "por confirmar". De seguida, o módulo de planeamento verifica se existem produtos finais ou matérias-primas em armazém, não reservados, e aloca-os para a encomenda do cliente em questão. Também são consideradas as matérias-primas que já se encontrem encomendadas e não estão reservadas para nenhuma encomenda. Esta alocação é provisória, dado que a encomenda ainda não foi confirmada.

Se o material disponível não for suficiente, é necessário recorrer aos fornecedores. Segue-se um procedimento onde o módulo de planeamento escolhe o fornecedor mais adequado, consoante os seus parâmetros, e recebe uma previsão da chegada das matérias-primas.

Com esta previsão, é iniciado o algoritmo de escalonamento. Primeiro, é necessário consultar o planeamento atual de modo a verificar em qual semana existe espaço suficiente no armazém de matérias-primas, para permitir criar uma *Inbound Order* provisória. Sabendo esta semana (apesar de ainda sujeita a futuros atrasos), a produção é escalonada tendo em conta o número de produtos finais que é possível produzir por semana, as ordens de produção que já se encontram no sistema e o espaço disponível no armazém de produtos finais. Devido a estes fatores, as ordens podem ter de ser divididas por diferentes semanas, originando diferentes *Production Orders*. A semana de envio para o cliente é planeada para a semana exatamente a seguir à produção do último produto dessa encomenda, sendo o planeamento provisório finalizado com uma *Expedition Order*.

Tendo terminado o escalonamento, são somados os custos de todos os produtos associados a esta encomenda, obtendo um custo total que será apresentado na interface interna, com as respetivas ordens de planeamento, para que seja aprovado ou recusado pelo utilizador. No contexto deste trabalho, não foram considerados custos de produção ou lucros, sendo somados apenas o custo individual de cada matéria-prima, dependendo do fornecedor a que foi encomendada.

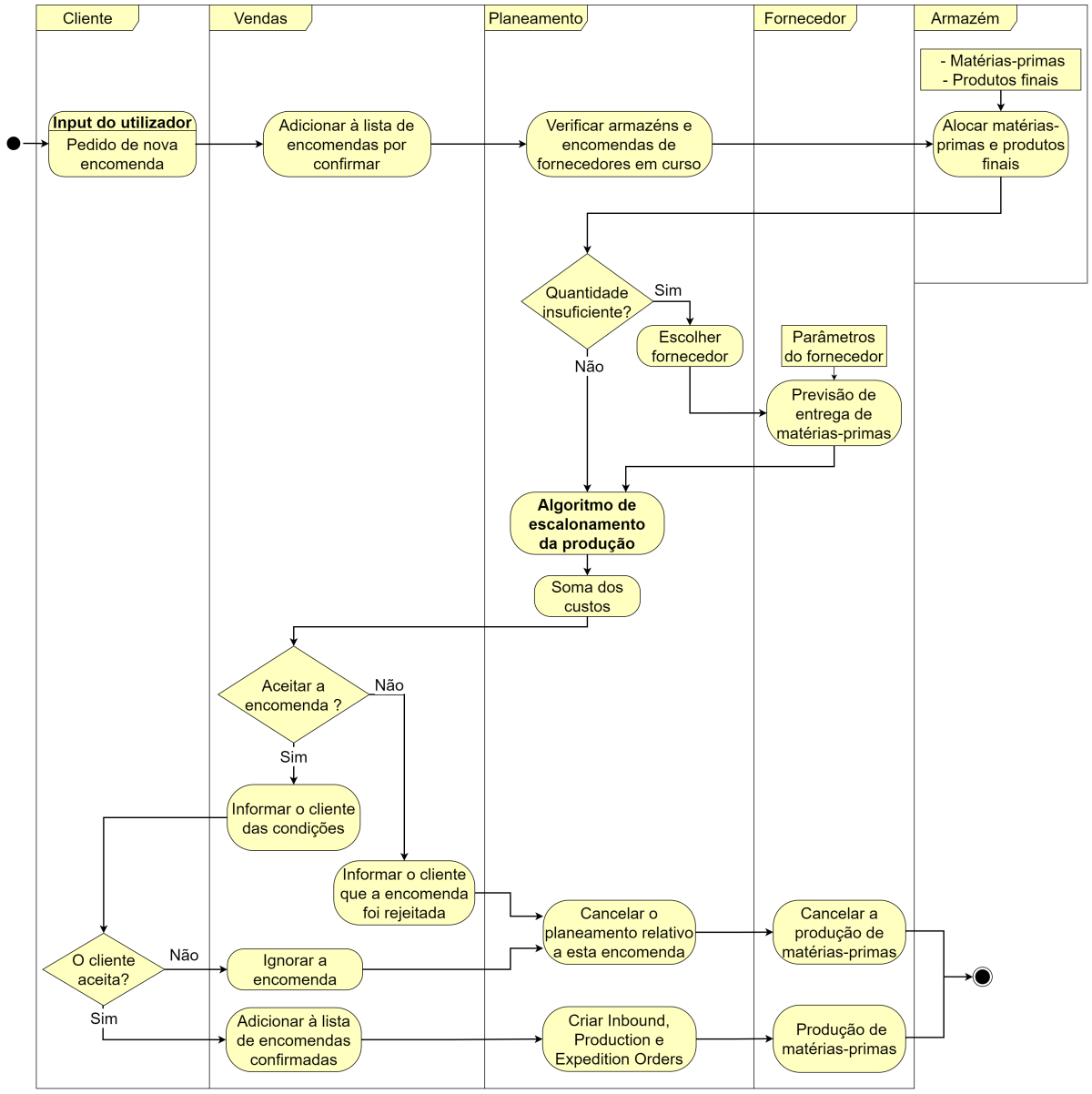


Figura 3.6: Algoritmo para nova encomenda

Caso aprovado, é fornecida uma estimativa de entrega ao cliente, de acordo com o planeamento efetuado. O utilizador, como cliente, toma a decisão de aceitar ou não a data de entrega. Sendo aceite, é confirmada a ordem de compra ao fornecedor escolhido (*Supplier Order*), e as ordens internas relativas a esta encomenda, que antes tinham caráter provisório, são confirmadas.

3.4.3 Eventos que implicam a alteração do planeamento

À medida que vão sendo recebidas novas encomendas, o planeamento vai sendo atualizado com novas ordens, não alterando as já existentes. No entanto, podem ocorrer outro tipo de eventos que, mesmo sendo relacionados a outras encomendas, interfiram com todo o planeamento, seja

ao nível de ordens de armazenamento, produção ou expedição. Estes eventos englobam o não cumprimento dos prazos por parte dos fornecedores, o cancelamento de encomendas por parte dos clientes e a produção de peças com defeito.

3.4.3.1 Chegada de matérias-primas e atrasos

No envio de matérias-primas por parte de um fornecedor, podem ocorrer duas situações distintas: as matérias-primas chegam na semana prevista ou atrasadas, sendo que a última origina um atraso para o cliente final. Uma simplificação do algoritmo para estes casos está representado na figura 3.7. Para dar alguma margem ao planeamento, foi definido que, caso o utilizador pretenda introduzir um atraso através da interface do fornecedor, esta ação pode ser realizada até à semana anterior da previsão inicial da chegada da encomenda.

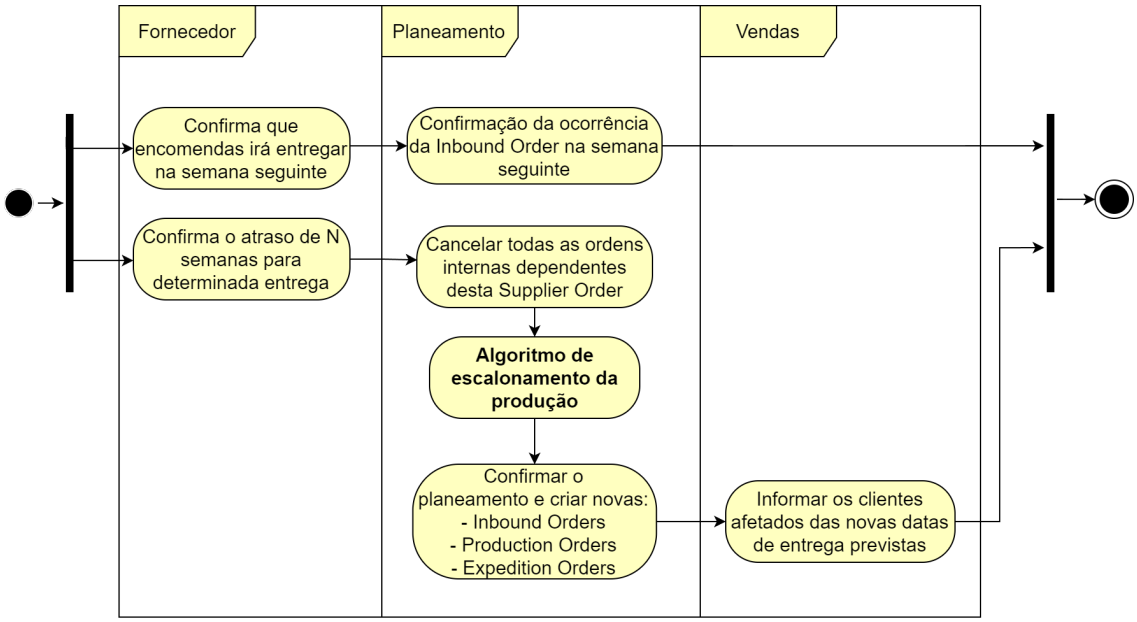


Figura 3.7: Algoritmo para atrasos

A primeira situação é simples, uma vez que segue o planeamento existente. A *Supplier Order* é confirmada, pelo que na semana seguinte, o MES pode executar a *Inbound Order* respetiva.

Para o caso do atraso, é necessário obter todas as ordens internas presentes no planeamento que dependiam da *Supplier Order* em questão e cancelá-las (processo relativamente simples dado todos os dados estarem armazenados numa estrutura relacional na base de dados). Tendo o valor em semanas do atraso introduzido, é realizado o mesmo algoritmo de escalonamento da produção apresentado em 3.4.2 partindo da nova semana de chegada das matérias-primas. Por fim, as encomendas dos clientes afetadas são atualizadas com as novas datas.

Inicialmente, considerou-se contemplar a situação de antecipação, mas esta foi descartada uma vez que o reescalonamento introduzido pela mesma seria de um grau de complexidade muito superior, principalmente devido à gestão do espaço do armazém.

3.4.3.2 Cancelamento de encomendas

Como o cliente pode realizar um pedido de cancelamento em qualquer momento, é necessário a utilização de um algoritmo que implementa diferentes ações dependendo da fase da produção em que essa encomenda se encontra, correspondendo a um modelo composto por várias etapas representado na figura 3.8.

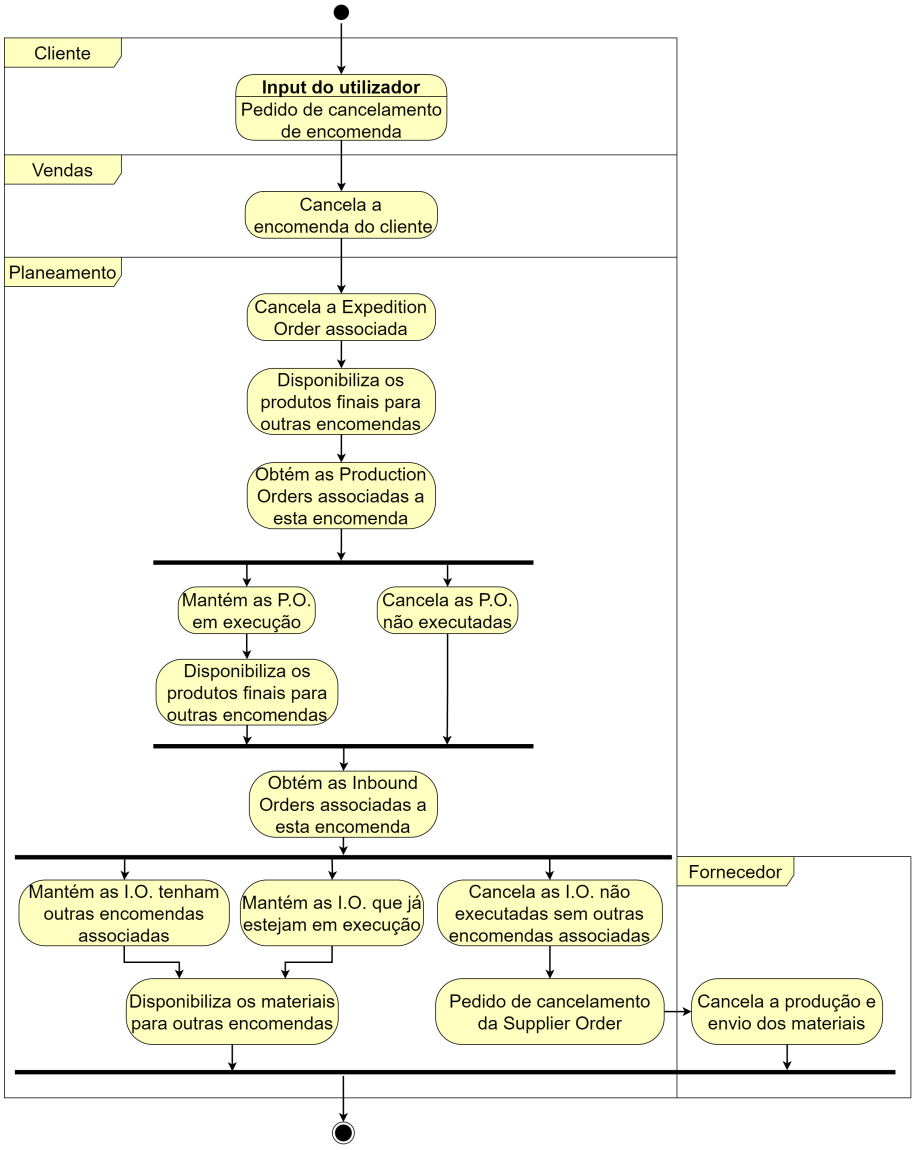


Figura 3.8: Algoritmo para o cancelamento de uma encomenda

A receção deste pedido implica que a encomenda ainda não foi expedida, pelo que se pode cancelar imediatamente a *Expedition Order*.

Devido à existência de várias *Production Orders* para uma mesma encomenda, os produtos correspondentes podem estar em diferentes fases de produção. Aqueles que cuja produção já esteja finalizada são imediatamente libertos para serem alocados a outras encomendas. Relativamente aos produtos que estejam a ser processados durante a semana em questão, optou-se por deixar terminar a sua produção, uma vez que a sua interrupção implicaria o mini-MES receber alterações do planeamento em tempo real e não apenas no fim de cada ciclo de produção. Isto elevaria, inerentemente, a complexidade da interação entre aplicações e do próprio algoritmo do mini-MES, o que não é pretendido para o contexto deste trabalho. Após terminado o ciclo de produção, estes produtos também são disponibilizados para outras encomendas. Por outro lado, as *Production Orders* das semanas seguintes, são canceladas.

Por último, para as *Inbound Orders* existem 3 situações possíveis. Primeiro, seguindo a mesma lógica, as que se encontram em execução são finalizadas e as matérias-primas são disponibilizadas. No segundo caso, se uma *Inbound Order* ainda não foi executada mas também contém matérias-primas que se encontram alocadas a outras encomendas para além da encomenda em questão, então a *Inbound Order* é mantida no planeamento. Finalmente, se não possuir as caraterísticas mencionadas para o primeiro e segundo caso, a *Inbound Order* é cancelada e é realizado um pedido ao fornecedor para cancelar o envio das matérias-primas (*Supplier Order*).

3.4.3.3 Processamento do feedback do mini-MES

Dos 3 tipos de ordens, apenas é necessário analisar o *feedback* do mini-MES relativamente às *Production Orders* (figura 3.9), uma vez que é durante a sua execução que podem surgir peças com defeito (conceito apresentado na secção 3.1.3), comprometendo a quantidade necessária de produtos para uma encomenda.

Assim, no final de cada semana, é necessário verificar a quantidade de peças defeituosas produzidas em cada *Production Order*. Se a quantidade de peças sem defeito não for suficiente para completar a encomenda, é necessário realizar um processo semelhante ao apresentado na secção 3.4.2. O módulo de planeamento tem de verificar o armazém de modo a alocar produtos finais disponíveis ou, caso não existam, utilizar/encomendar matérias-primas e gerar novas ordens de produção, necessitando de um novo reescalonamento e posterior aviso ao cliente de que a sua encomenda irá sofrer um atraso.

3.5 Arquitetura do mini-MES

Como já foi referido, a interação entre ERP-MES é simples, dado que o mini-MES limita-se a recolher semanalmente as ordens geradas pelo mini-ERP e atualizar o seu estado durante a sua

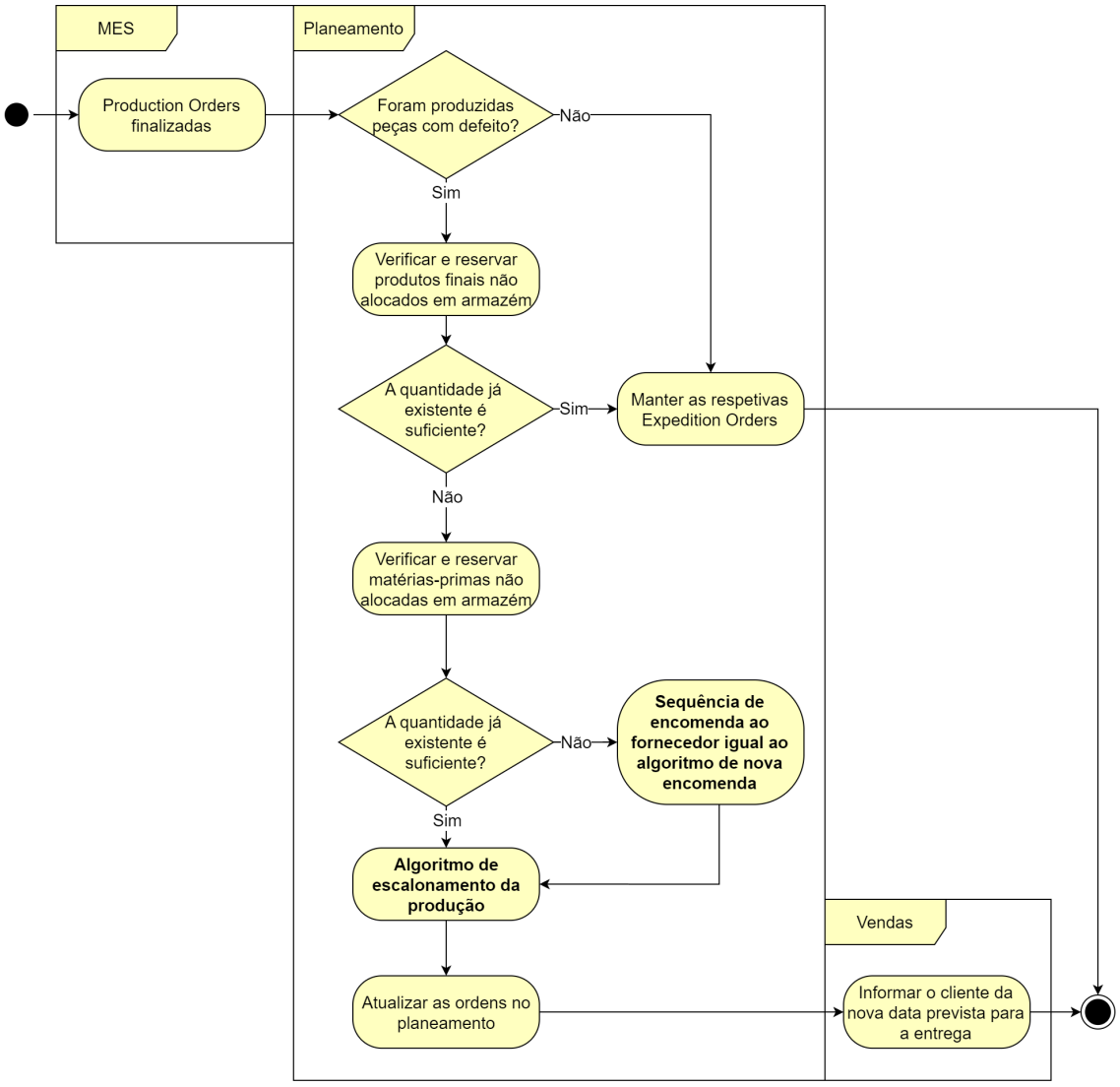


Figura 3.9: Processamento do feedback do MES

execução. A maior dificuldade na estruturação desta aplicação consiste na interação de controlo do *Digital Twin* dado que não existe um *layout* da linha de produção fixo.

Neste contexto, não é possível a implementação de um MES totalmente genérico que controle sempre a linha de produção de modo otimizado, sem nenhuma informação extra. O que é comum nas soluções comerciais é que, após analisado o *layout*, o MES seja configurado de modo a se obter uma otimização ideal cujos parâmetros diferem de caso para caso, conforme referido em 2.4.2.

A otimização da produção a um nível equiparável à existente em ambientes reais não faz parte dos objetivos deste trabalho, pois, com esta aplicação, apenas se pretendem demonstrar os conceitos principais de um MES, num contexto educativo. Tendo em conta este aspeto, e o conceito da divisão do *layout* em zonas apresentado na secção 3.1.2, foram definidos pontos de

interesse, ou seja, determinados equipamentos, que têm de estar presentes na linha de produção e que serão apresentados na secção seguinte. Estas restrições visam equilibrar a liberdade dada ao utilizador na definição de *layouts* e a dificuldade inerente da implementação do seu controlo.

3.5.1 Pontos de interesse

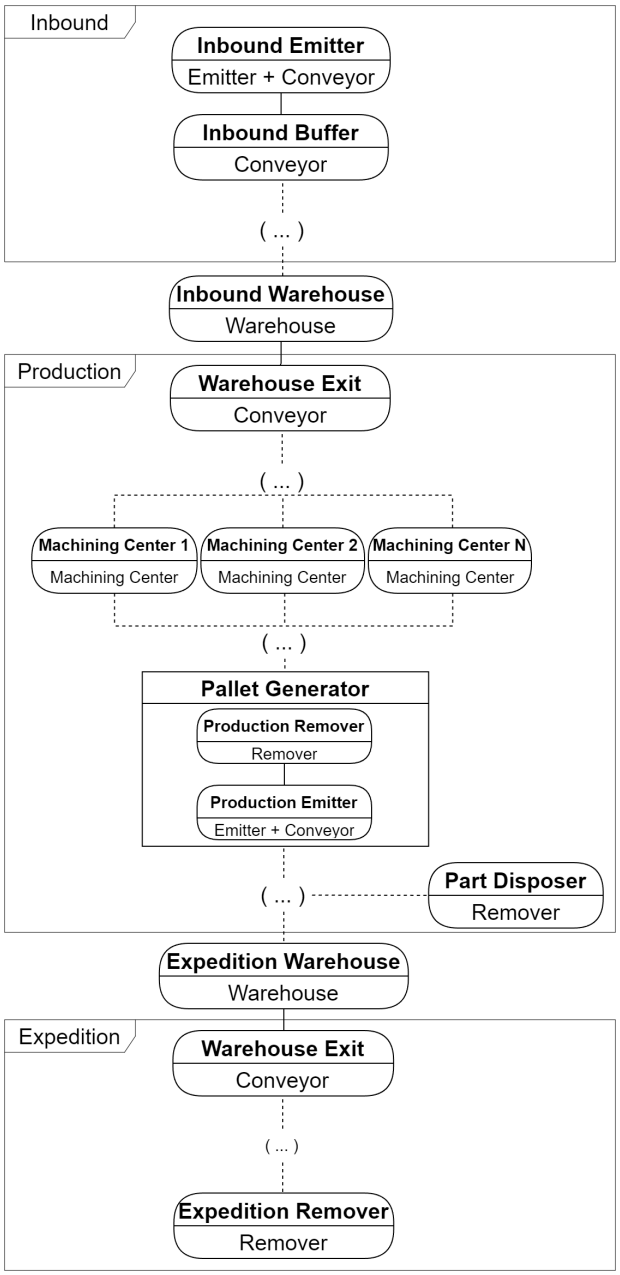


Figura 3.10: Pontos de interesse

O encaminhamento das peças ao longo do chão de fábrica é planeado com o objetivo de que estas percorram um caminho dividido por pontos de interesse. Esta foi a estratégia utilizada que

permite algum grau de liberdade na definição do *layout*, ao mesmo tempo que garante o fluxo da produção pretendido.

Cada uma das zonas possui pontos de interesse específicos de acordo com as suas necessidades, representados no diagrama de blocos da figura 3.10. Cada bloco possui a sua designação ao nível do mini-MES, indicativo da função que desempenha, com uma legenda do seu tipo de equipamento ao nível do *Digital Twin*. Se existirem dois equipamentos que têm de estar necessariamente posicionados um adjacente ao outro, a sua ligação é representada por uma linha fixa. Por outro lado, uma linha tracejada significa que o utilizador pode definir quantos equipamentos de transporte introduzir entre dois pontos.

Nas subsecções seguintes, são discutidos os pontos de interesse de cada zona, respetivamente.

3.5.1.1 Zona de entrada

A única opção para gerar peças contemplada pelo *Digital Twin* é o conjunto formado por um *Emitter* sobre um *Conveyor*. Assim, a zona de entrada é composta por um destes conjuntos, sendo prosseguido de um simples *Conveyor* que atua como *buffer*. Isto garante que existe pelo menos um tapete de transporte entre o equipamento onde são geradas as matérias-primas e o primeiro armazém. O motivo desta necessidade será explicado na secção 3.5.3, referente ao algoritmo de controlo. Tal como mencionado, é colocado o armazém de matérias-primas, delimitando a zona de entrada. Encontra-se representado na figura 3.11 um exemplo para esta zona.

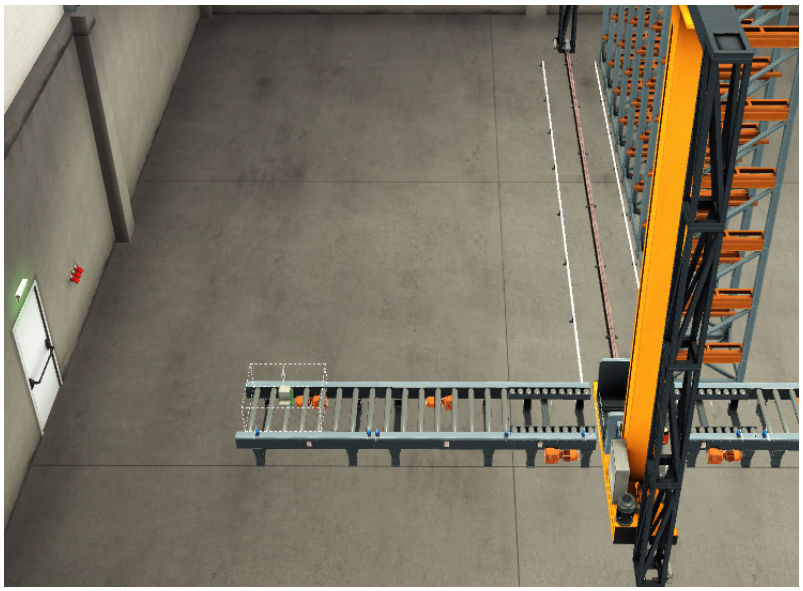


Figura 3.11: Exemplo de uma zona de entrada

3.5.1.2 Zona de produção

É nesta zona onde é permitido um maior grau de personalização do *layout*, dado que a única limitação imposta é a existência de pelo menos um *Machining Center*. Cada peça é retirada do armazém para um tapete de saída. É recomendado que seja colocado sobre um dos tapetes, no simulador FactoryIO, um *Remover* que retire a palete, mantendo somente a peça, para que esta possa ser introduzida nas máquinas.

Após isto, a peça é encaminhada para um dos *Machining Centers* existentes, de acordo com a operação pretendida. No final da zona de produção, é necessário existir uma associação de equipamentos que foi denominada *Pallet Generator*. Para que a peça processada possa ser introduzida no armazém, é necessário que esteja sobre uma palete, como já foi referido em 2.2.1. Assim, a solução encontrada no contexto do *Digital Twin* consiste em passar a peça por um *Remover* e utilizar um *Emitter* para gerar uma peça com exatamente as mesmas caraterísticas, tendo como suporte uma palete. Finalmente, caso a peça seja considerada defeituosa, é encaminhada para um *Remover*, para ser destruída, caso contrário, é armazenada no armazém de produtos finais. Encontra-se representado na figura 3.12 um exemplo para esta zona.

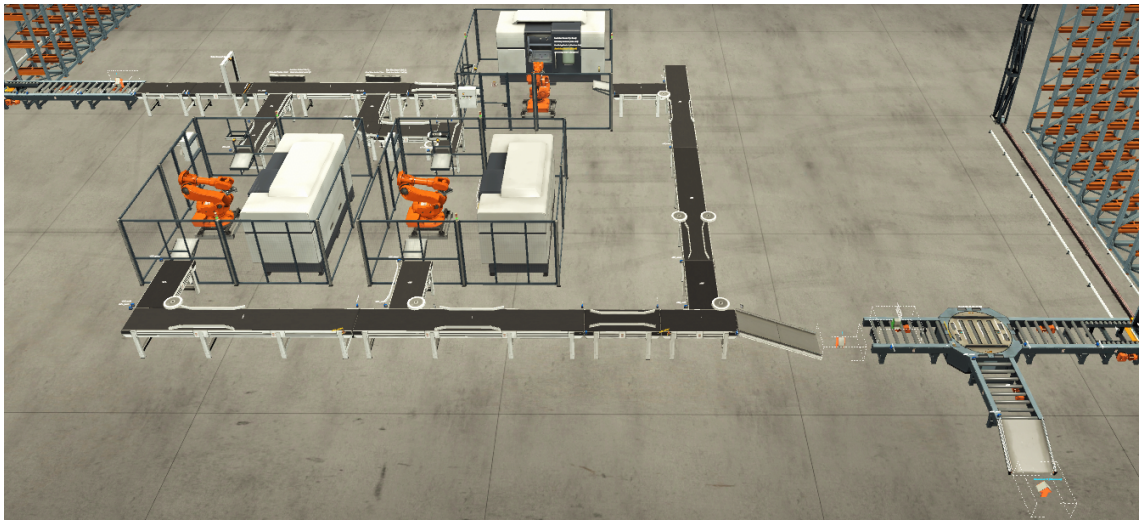


Figura 3.12: Exemplo de uma zona de produção

3.5.1.3 Zona de expedição

A última zona é muito semelhante à zona de entrada, com um *layout* linear, tendo como pontos obrigatórios apenas um *buffer* para saída do armazém e um *Remover* que simula a expedição das peças para os clientes. Encontra-se representado na figura 3.13 um exemplo para esta zona.

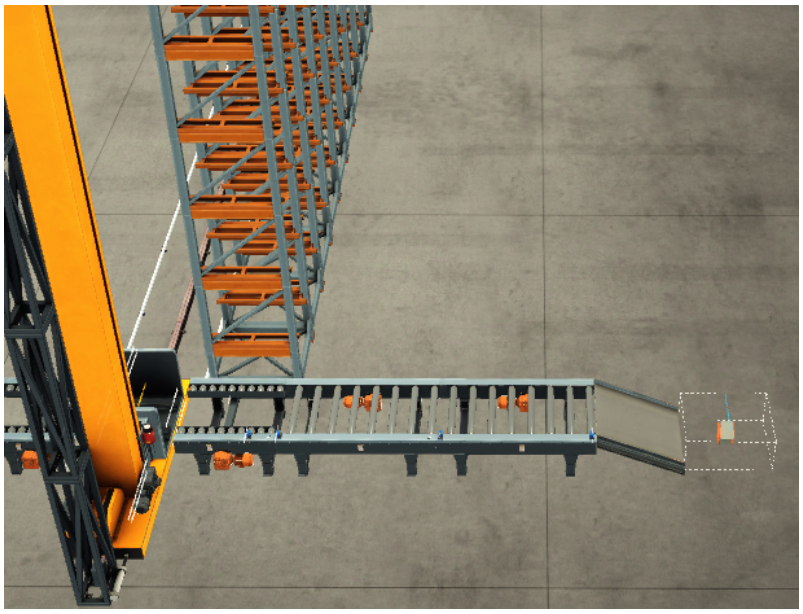


Figura 3.13: Exemplo de uma zona de expedição

3.5.2 Definição dos caminhos

Para movimentar uma peça entre os vários equipamentos presentes no *Digital Twin*, é necessário que o percurso pretendido seja traduzido numa sequência de identificadores dos respetivos equipamentos. Os pontos de interesse mencionados anteriormente têm como objetivo servirem como par origem/destino destas sequências. Tendo todos os identificadores dos pontos de interesse, conclui-se que, por exemplo, o trajeto correspondente a uma *Production Order* terá como origem a saída do armazém de matérias-primas e como destino (primeira paragem) um *Machining Center*. No entanto, será necessário a utilização de um algoritmo que permita obter os identificadores dos equipamentos correspondentes ao caminho mais curto entre uma origem e um destino.

Para esta finalidade, foi escolhido o algoritmo *Dijkstra*. Como foi explicado na secção 2.2.7, este algoritmo já tinha sido implementado no *Digital Twin*, mas apenas era possível a sua utilização uma vez para cada peça emitida, pelo que se optou por integrá-lo diretamente no mini-MES. Além disto, foi incluída a possibilidade dos trajetos entre equipamentos terem diferentes pesos, dependendo do seu tamanho físico no FactoryIO. A sua implementação, bem como a leitura do *layout* sob a forma de um grafo para ser possível a utilização deste algoritmo serão apresentadas no capítulo 4. Desta forma, é possível definir todos os caminhos entre pontos de interesse a serem utilizados pelo algoritmo de controlo do mini-MES.

3.5.3 Controlo da produção

No início de cada semana, o mini-MES obtém as ordens contendo as informações das peças a serem processadas. O controlo da produção é realizado enviando comandos para o *Digital Twin*

e recebendo *feedback* quando a sua execução é finalizada. É por este motivo que, para permitir que estejam a ser movimentadas múltiplas peças tendo como origem o mesmo equipamento, são utilizados *buffers* no início de cada zona.

Para ilustrar a necessidade dos mesmos, tome-se como exemplo a zona de entrada sem o "*Inbound Buffer*". Ao emitir uma peça no "*Inbound Emitter*", com destino ao "*Inbound Warehouse*", não é possível saber o momento exato em que esta deixa de ocupar o "*Inbound Emitter*", uma vez que a estrutura OPC-UA contendo a lista de ordens finalizadas apenas será atualizada quando a peça chegar ao "*Inbound Warehouse*", fazendo com que seja possível apenas uma peça de cada vez para este trajeto.

Seria possível enviar um comando para verificar a ocupação desse equipamento em específico, mas esta opção não se verificou ser uma alternativa viável, dado aumentar o congestionamento da única estrutura existente para o envio de comandos para o *Digital Twin*.

Assim, o trajeto é dividido em duas partes (figura 3.14). É enviado um comando de encaminhamento da peça para o "*Inbound Buffer*" (que é o equipamento adjacente) e, no momento em que esta ordem é finalizada, pode-se concluir que o "*Inbound Emitter*" se encontra livre, permitindo a emissão de uma nova peça neste equipamento, ao mesmo tempo que a primeira é encaminhada para o "*Inbound Warehouse*".

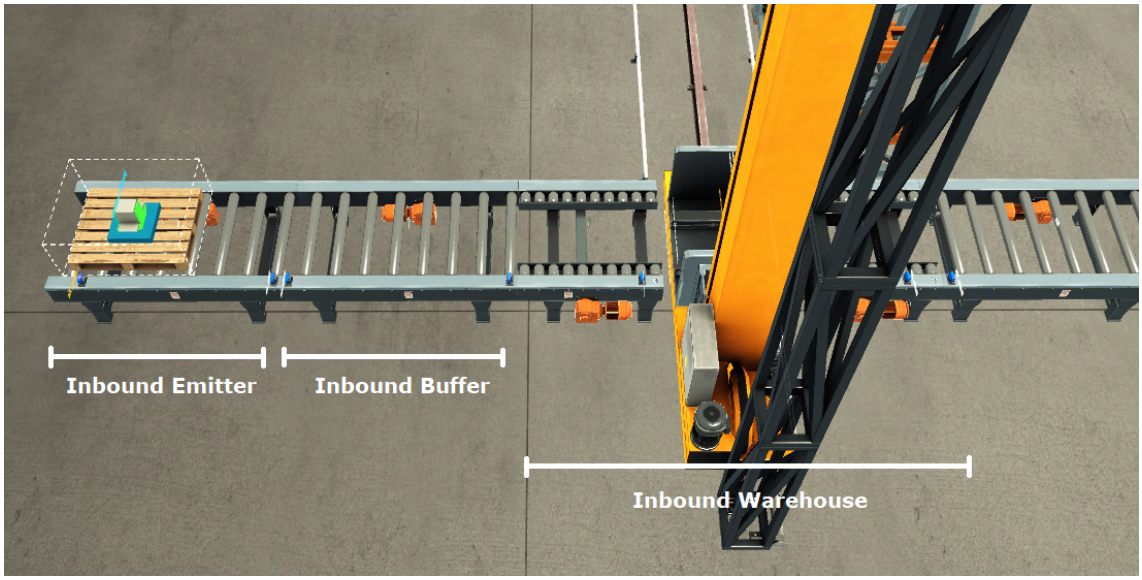


Figura 3.14: *Inbound Emitter*, *Inbound Buffer* e *Inbound Warehouse*

Como a sequência de comandos difere dependendo da ordem em questão, as mesmas serão abordadas separadamente nas subsecções seguintes. É de salientar que os armazéns do *Digital Twin* não permitem introduzir e retirar peças ao mesmo tempo. Por esta razão, são executados em simultâneo os algoritmos para o controlo das *Inbound* e *Expedition Orders*, que atuam em

armazéns distintos. Quando as mesmas forem terminadas (ou caso não existam) o mini-MES inicia a execução das *Production Orders*, que vão utilizar tanto o armazém de matérias-primas, quanto o armazém de produtos finais.

3.5.3.1 Inbound Orders

O controlo da zona de entrada é constituído por 3 algoritmos, representados na figura 3.15, que são executados enquanto existirem *Inbound Orders* não finalizadas. O funcionamento e a necessidade do primeiro algoritmo já foi explicada no exemplo discutido em 3.5.3, permitindo que sejam introduzidas peças na zona de entrada no maior ritmo possível, sem causar sobreposição de comandos para o mesmo equipamento. Já o segundo algoritmo tem como função encaminhar as peças que cheguem ao "*Inbound Buffer*" para o "*Inbound Warehouse*". Á semelhança deste, o terceiro atualiza a localização e o estado das peças quando é detetado que as mesmas chegam ao "*Inbound Warehouse*".

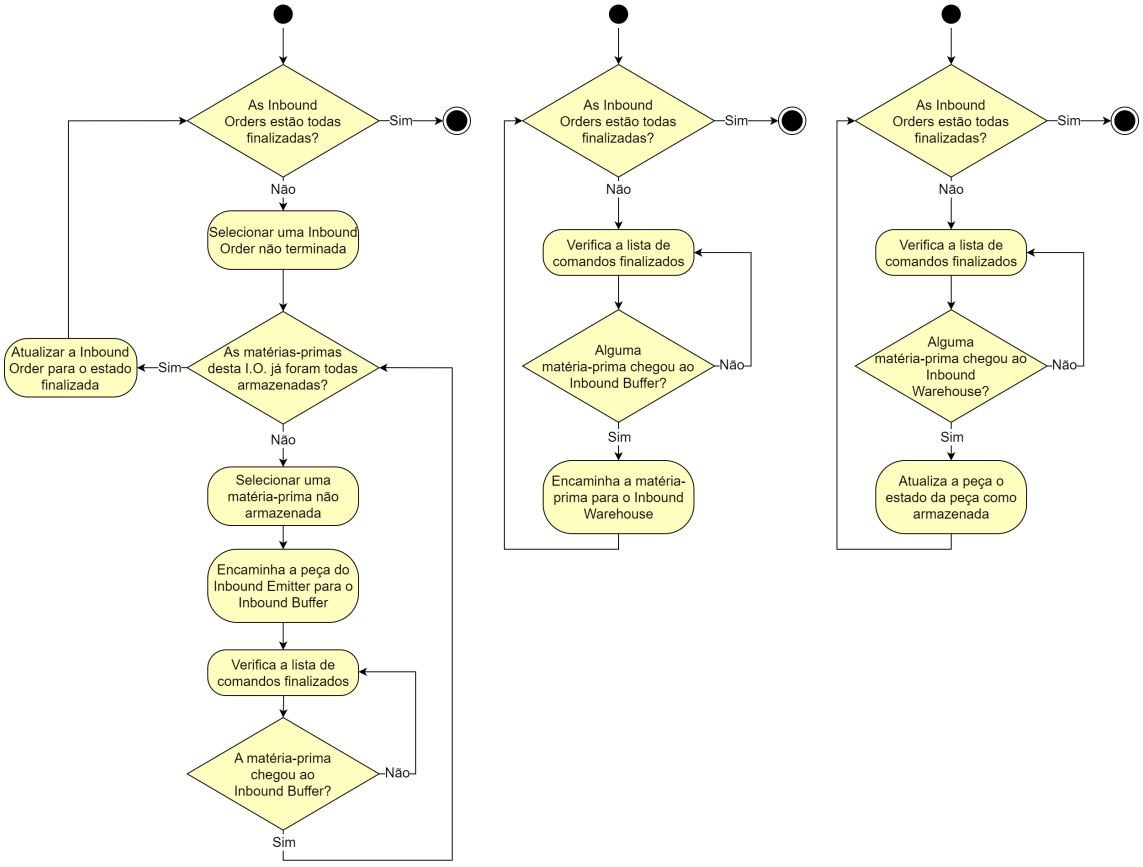


Figura 3.15: Controlo da zona de entrada

3.5.3.2 Production Orders

Antes de iniciar a execução das *Production Orders*, é necessário definir a ordem pelas quais as peças serão retiradas do armazém de matérias-primas. Para este efeito, a métrica utilizada consiste em distribuir igualmente a quantidade de peças que serão alocadas para o processamento em cada *Machining Center*, tendo em consideração o seu tipo. A distribuição é realizada sendo dada prioridade aos Machining Centers que se encontrem mais próximos do *Inbound Warehouse* (conseguido através dos diferentes pesos do algoritmo do Dijkstra, referido em 3.5.2). Com a atribuição concluída, é criada uma lista sequencial das peças a serem processadas, onde se encontram ordenadas de forma a que sejam intercaladas dependendo do *Machining Center* a que foram alocadas. É de salientar que poderiam ter sido utilizados algoritmos que garantissem uma maior otimização. No entanto, dada a natureza deste trabalho, o algoritmo de escalonamento descrito foi desenvolvido com base numa métrica heurística, visando diminuir a complexidade da sua implementação. Com isto, o controlo da produção é efetuado utilizando 4 algoritmos.

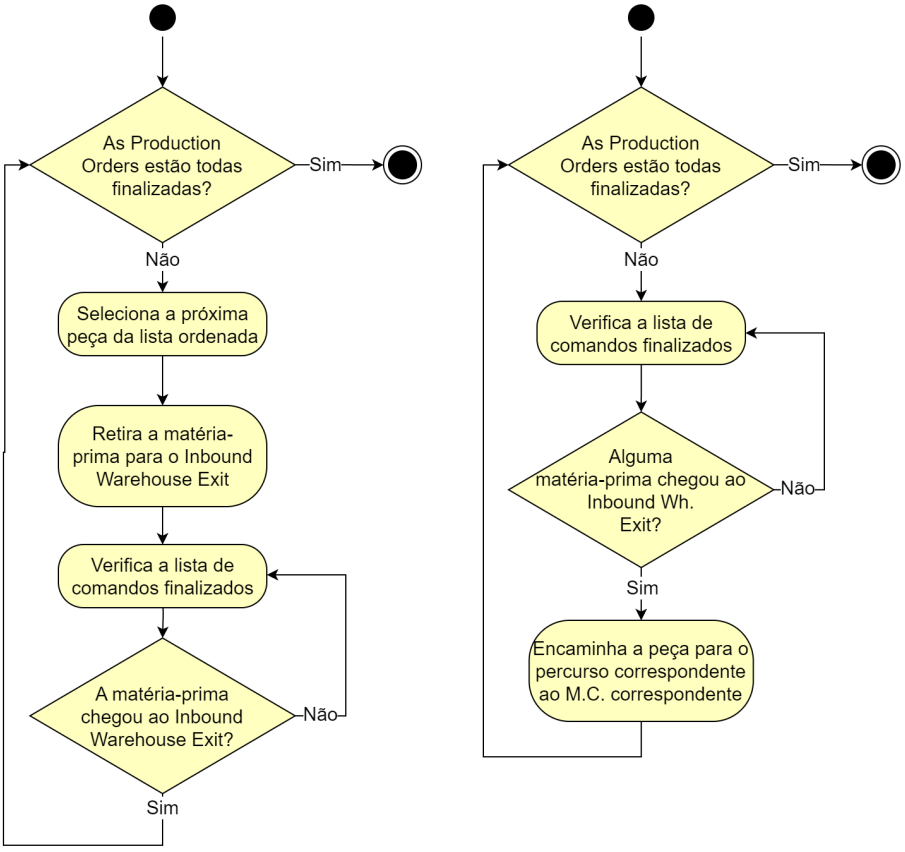


Figura 3.16: Controlo da zona de produção (Parte 1)

Os algoritmos apresentados na figura 3.16 são relativos à transformação das matérias-primas em produtos finais. O primeiro permite gerir a retirada das peças do armazém de matérias-primas

seguindo a sequência que foi definida. O segundo tem como objetivo encaminhar cada matéria-prima para o *Machining Center* que lhe foi atribuído. É importante destacar que o percurso introduzido não termina no *Machining Center*. Na definição dos caminhos (3.5.2), para cada *Machining Center*, é calculado um caminho que atravessa os pontos de interesse, respetivamente, "*Inbound Warehouse Exit*", "*Machining Center*" em questão e "*Pallet Generator*".

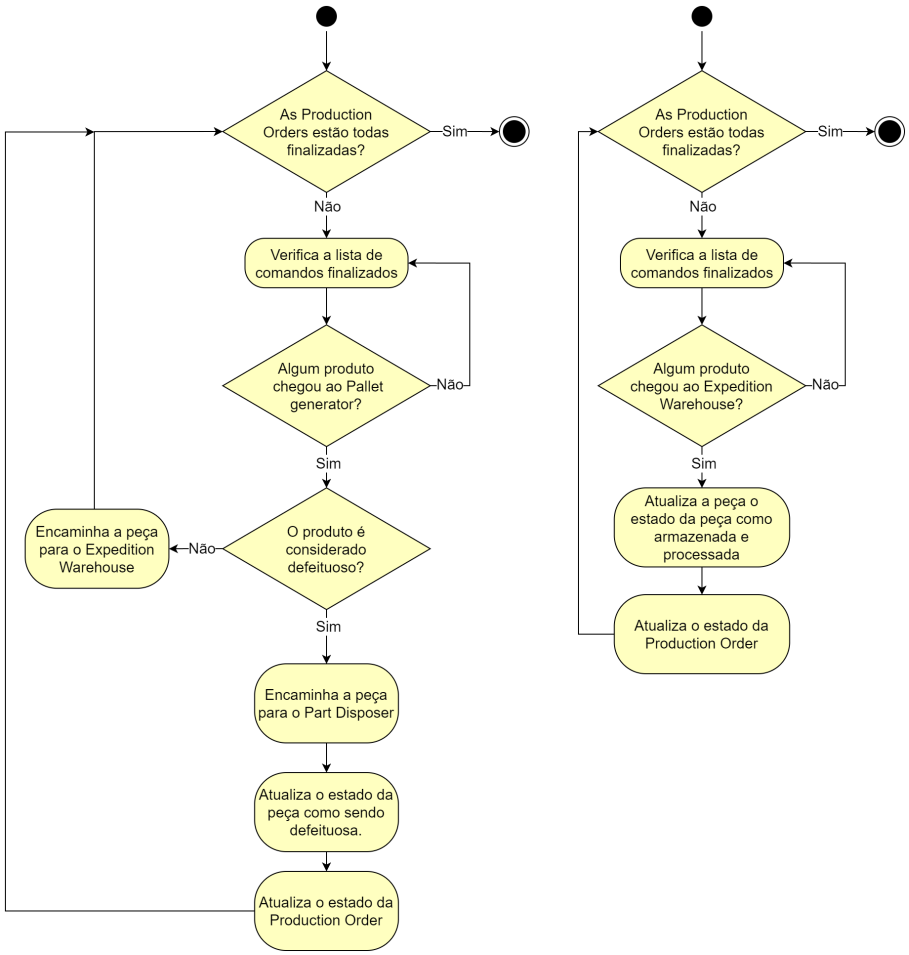


Figura 3.17: Controlo da zona de produção (Parte 2)

O terceiro e quarto algoritmos (figura 3.17) encaminham os produtos já processados. Quando uma peça chega ao "*Pallet Generator*", caso ela seja considerada defeituosa, é encaminhada para o "*Part Disposer*", sendo que o seu estado é atualizado na base de dados. Caso contrário, é armazenada no "*Expedition Warehouse*".

3.5.3.3 Expedition Orders

O controlo da zona de expedição (figura 3.18) segue a mesma lógica apresentada para a zona de entrada, apenas mudando o tipo de ordem e os pontos de interesse para os quais os produtos são encaminhados, pelo que a sua discussão não será necessária.

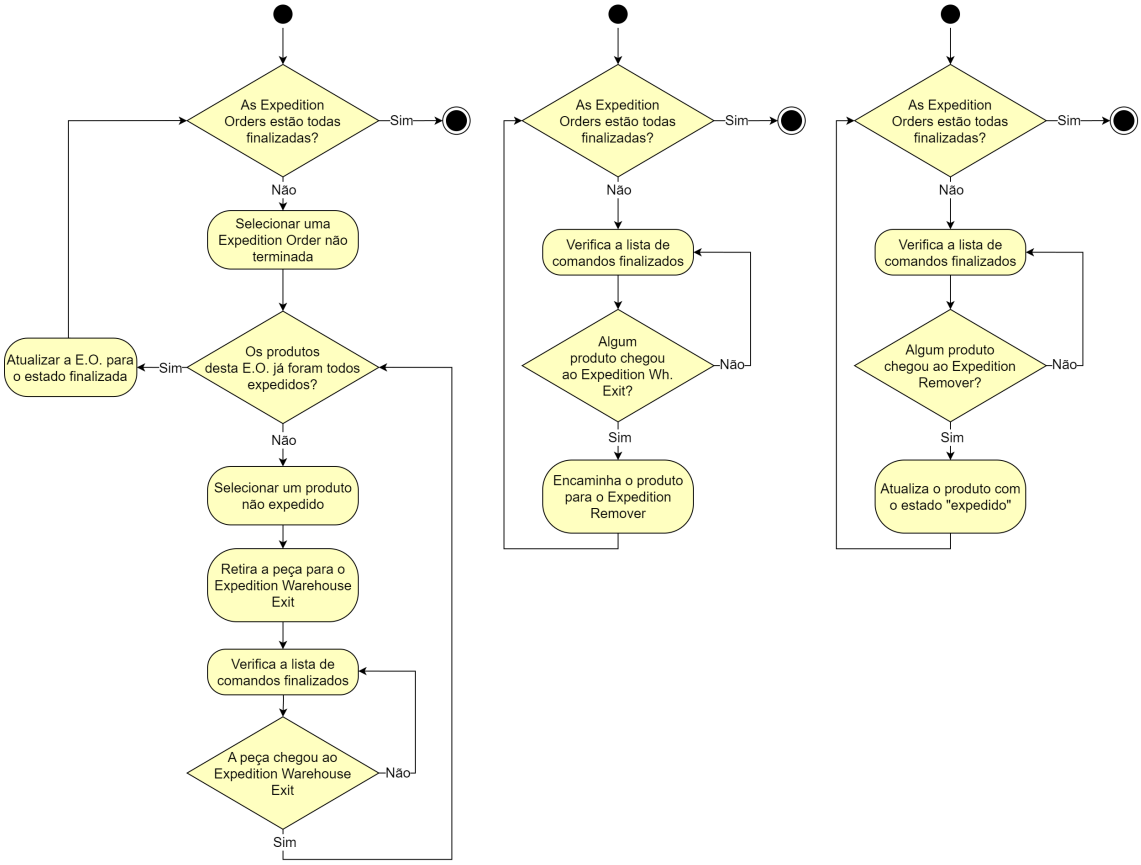


Figura 3.18: Controlo da zona de expedição

3.6 Conclusões

Neste capítulo, foi discutida a arquitetura nos seus diferentes níveis de granularidade. A análise dos requisitos estabelecidos permitiu a definição de uma arquitetura de alto nível que traduz a interação entre as aplicações contempladas neste trabalho. De seguida, foram analisados os algoritmos que compõem a arquitetura das aplicações a desenvolver. Para o caso do mini-ERP, os algoritmos consistem num conjunto de ações a serem executadas para a lista de eventos que se pretendem ser contemplados na aplicação. Por outro lado, o mini-MES possui uma arquitetura direcionada ao controlo em tempo real da linha de produção, tendo sido necessário a definição de algumas restrições para que a sua implementação não possua uma complexidade muito superior à necessária.

Tendo sido apresentada toda a arquitetura proposta, a sua implementação será abordada detalhadamente no próximo capítulo.

Capítulo 4

Implementação e Testes

Neste capítulo é apresentada a implementação da arquitetura proposta no capítulo anterior. Inicialmente são explicados o modelo de dados e os padrões de desenvolvimento de código utilizados, seguindo-se uma análise detalhada da implementação de cada uma das aplicações, bem como das alterações efetuadas ao *Digital Twin*, para ser possível a concretização deste trabalho. No final, são demonstrados os testes efetuados de modo a validar as aplicações.

4.1 Modelo de dados

A implementação da arquitetura descrita anteriormente iniciou-se com o desenvolvimento de um modelo de dados capaz de contemplar todas as necessidades das aplicações em questão (figura 4.1). Este é o modelo relacional que estrutura a base de dados partilhada entre as aplicações, possuindo tabelas que podem ser utilizadas por apenas uma aplicação ou ambas. A utilização das tabelas é a seguinte:

- Utilizadas somente pelo mini-ERP: *SupplierOrder*, *Supplier*, *Supplier_Material*, *Material*, *ClientOrder*, *Client*.
- Utilizadas por ambas as aplicações: *Factory*, *Piece*, *InboundOrder*, *ProductionOrder*, *ExpeditionOrder*.
- Utilizada somente pelo mini-MES: *MachiningCenter*

A tabela *Factory* funciona como uma *lookup table*, ou seja, não possui nenhuma ligação relacional às restantes tabelas, servindo apenas para as aplicações acederem às informações relativas ao estado da simulação. Os diferentes estados serão detalhados na secção seguinte, sendo que os parâmetros da simulação podem assumir os seguintes valores:

- *setup_status*: *waiting_factory_params*, *done*.
- *sim_status*: *waiting_sim_start*, *ongoing_week*, *waiting_week_start*.
- *warehouse_capacity*: valor inteiro entre 1 a 54.

- *weekly_production*: valor inteiro maior que 0.

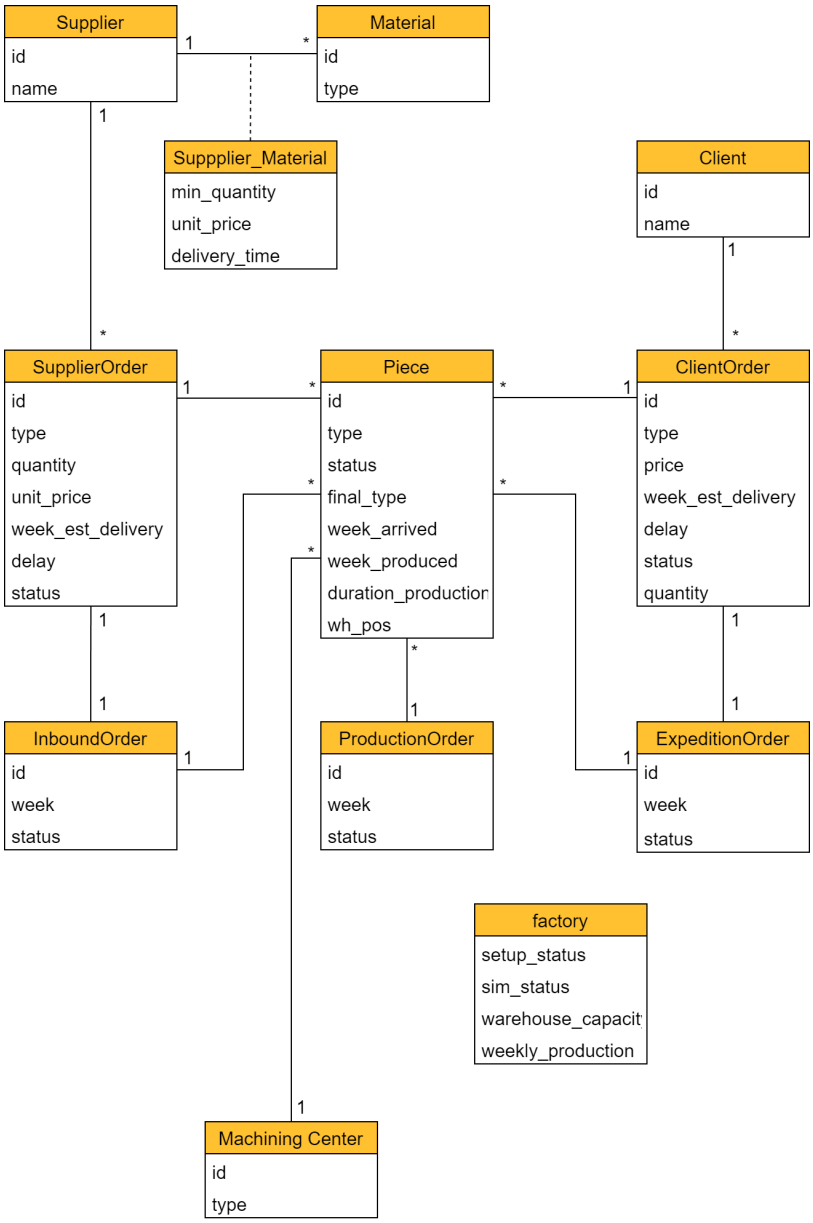


Figura 4.1: Modelo de dados

A semana de entrega prevista inicialmente, para o caso das *SupplierOrder* e *ClientOrder*, é armazenada no parâmetro *week_est_delivery*. Caso ocorram atrasos, o valor correspondente a esse atraso, em semanas, é guardado em *delay*. Isto permite alguma flexibilidade na visualização dos dados na interface, pois é possível manter o registo da primeira previsão independentemente do número de atrasos que sejam introduzidos (*week_est_delivery*), informar na ocorrência de atrasos (se $delay > 0$) e disponibilizar a previsão atual ($week_est_delivery + delay$).

Como cada fornecedor pode fornecer entre 1 a 3 materiais, com diferentes quantidades mínimas, preços e tempos de entrega, as tabelas *Supplier*, *Supplier_Material* e *Material* estão representadas por uma relação de associação. A tabela *Material* é inicializada com um conjunto fixo de materiais (os que são contemplados pelo *Digital Twin*). Ao registar um novo fornecedor, é introduzida uma nova entrada na tabela de *Supplier* e várias na de *Supplier_Material*, dependendo da quantidade de materiais fornecidos por esse fornecedor.

Para possibilitar a edição dos parâmetros dos fornecedores a qualquer momento, independentemente do estado das encomendas em curso, são registados em cada *SupplierOrder* o tipo e o preço que esse fornecedor oferecia no momento da realização da encomenda.

Todas as ordens são compostas por matérias-primas/produtos, daí todas estarem relacionadas à tabela de *Piece*, sendo os parâmetros de cada peça os seguintes:

- *id*: Identificador individual de cada peça.
- *type*: Tipo atual da peça. No momento em que uma peça chegue de um fornecedor, o seu tipo pode ser uma das matérias-primas, por exemplo, *GreenRawMaterial*. Se esta for processada como uma tampa, o seu tipo será atualizado para *GreenProductLid*.
- *status*: O estado da peça é atualizado durante o seu processamento. Enquanto espera no armazém de entrada, possui o valor *stored*. Durante o processo de transformação possui o valor *producing*. Quando o processo for finalizado, possui o valor *produced* ou *defective*, dependendo da avaliação realizada. O estado *shipped* indica que a mesma já foi expedida e já não se encontra na linha de produção.
- *final_type*: No momento em que é agendada uma *ProductioOrder*, o tipo final da peça é registado neste parâmetro, para que seja possível identificar a transformação a ser aplicada (para base ou para tampa).
- *week_arrived* e *week_produced*: São parâmetros preenchidos pelo mini-MES indicando a semana em que chegou à fábrica e a semana em que foi produzida, respetivamente.
- *duration_production*: Parâmetro registado pelo mini-MES que indica o tempo, em segundos, que a peça esteve na zona de produção. Este parâmetro pode ser visualizado pelo utilizador na interface do mini-MES, nas informações específicas de cada peça.
- *wh_pos*: Posição do armazém em que a peça se encontra. Caso a peça esteja no estado de matéria-prima, esta posição diz respeito ao armazém de matérias-primas. Se já tiver sido processada, diz respeito ao armazém de produtos finais.

A tabela *MachiningCenter* é exclusiva para o mini-MES pois serve apenas para alocar o processamento de cada peça a uma determinada máquina, tal como explicado na secção 3.5.3.2. O tipo de cada máquina corresponde à sua capacidade de processamento das peças. Se produzir somente tampas ou bases possui o tipo *LidProducer* ou *BaseProducer*, respetivamente. As máquinas capazes de produzir ambos os tipos são as *BothProducer*.

4.2 Ciclo de simulação

Pelas razões enunciadas no capítulo anterior, foi definida uma unidade temporal que está diretamente relacionada com a capacidade de produção, fazendo com que não exista um tempo fixo para cada ciclo de produção. Por este motivo, seria possível que uma semana fosse finalizada demasiado rápido, impedindo que o utilizador pudesse realizar determinadas ações ao nível do mini-ERP na semana em questão.

Para solucionar este problema, foram implementados diferentes estados para o ciclo de simulação, sendo que o utilizador terá controlo da evolução temporal em alguns pontos, como representado no diagrama da figura 4.2.

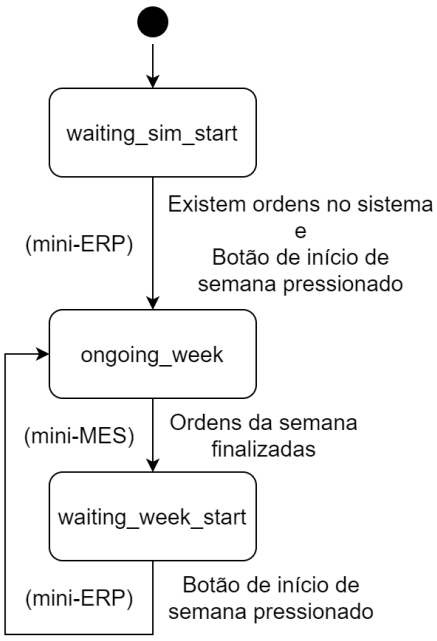


Figura 4.2: Ciclo de simulação

Enquanto a simulação não é iniciada (*waiting_sim_start*), o utilizador pode criar os primeiros fornecedores e clientes, bem realizar os primeiros pedidos de encomendas. Quando as ordens internas do planeamento estiverem confirmadas, é desbloqueada no mini-ERP a possibilidade de iniciar a primeira semana (pressionando um botão, por exemplo). O estado *ongoing_week* indica ao mini-MES que pode começar a executar as ordens presentes no planeamento para a semana em questão. Quando o mini-MES finalizar todas as ordens, o estado passa para *waiting_week_start*. Neste ponto, quando o utilizador assim o bem entender, pode iniciar a segunda semana, repetindo-se o ciclo.

4.3 Padrões de desenvolvimento

Tal como foi referido na secção 2.3, foi utilizado o JavaFX com a ferramenta *Scene Builder* para construir as aplicações. Existem vários padrões de desenvolvimento de código (*Design Patterns*) que podem ser utilizados em linguagens de programação orientadas a objeto como o Java. Nesta secção são apresentados quais os padrões utilizados neste trabalho e qual o seu enquadramento nas aplicações em questão.

4.3.1 Model-View-Controller

O principal padrão utilizado em ambas aplicações é o Model-View-Controller. Este padrão consiste em separar o código da aplicação em 3 camadas:

- *Model*: É a camada de dados que contém as informações de forma encapsulada [23], ou seja, os objetos correspondentes devem servir apenas para armazenar a informação e não devem conter funções relacionadas com o funcionamento lógico da aplicação [23].
- *View*: Esta camada é a responsável por exibir a informação presente nos objetos da camada *Model* [23]. Os objetos desta camada também são os responsáveis por receber e transmitir os *inputs* do utilizador para os objetos da camada *Controller*.
- *Controller*: Os objetos desta camada recebem a informação vinda do *View* e atualizam o *Model* com base na lógica definida para cada ação.

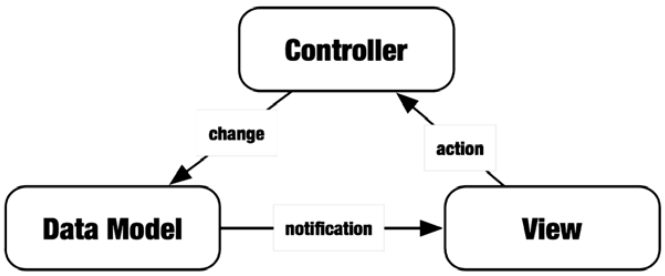


Figura 4.3: Model-View-Controller, retirado de [23]

No contexto deste trabalho, a camada *Model* corresponde ao modelo da base de dados apresentado na secção anterior, sendo apenas necessário o seu mapeamento para classes Java. No JavaFX Scene Builder, por sua vez, pode ser feita uma correspondência direta às camadas *View* e *Controller*. A camada *View* corresponde às interfaces gráficas codificadas através da linguagem FXML que, tal como descrito anteriormente na secção 2.3, já se encontram desta forma separadas do resto do código. Finalmente, a camada *Controller* corresponde às classes que controlam as interfaces FXML referidas. Nestas últimas serão implementadas em Java os algoritmos descritos na arquitetura proposta.

Tome-se, como exemplo, a realização de uma nova encomenda. É construída uma página com vários campos e um botão (*View*). Ao pressionar esse botão, o controlador dessa página (*Controller*) recebe os *inputs* e executa o algoritmo descrito na secção 3.4.2. Quando terminado, as tabelas da base de dados (*Model*) são atualizada e a informação é exibida (*View*).

4.3.2 Singleton

O *Singleton* consiste numa classe que possui apenas uma instância [23]. Por esta razão, são utilizados quando é necessário que um determinado objeto seja único e acessível globalmente [23].

No contexto deste trabalho, cada página tem a sua classe *Controller* correspondente, em que é necessário realizar determinadas operações na base de dados. Para este efeito, foi criado um Singleton denominado *DatabaseHandler*, que é instanciado quando o utilizador introduz os dados de conexão à base de dados. Nesta classe, foram implementadas todas as funções que realizam as operações CRUD na base de dados (*Create*, *Read*, *Update* e *Delete*), podendo ser utilizadas por todas as outras classes existentes. Também foram implementados outros Singleton que serão explicados nas secções correspondentes à aplicação a que dizem respeito.

4.4 Implementação do mini-ERP

4.4.1 Interface inicial

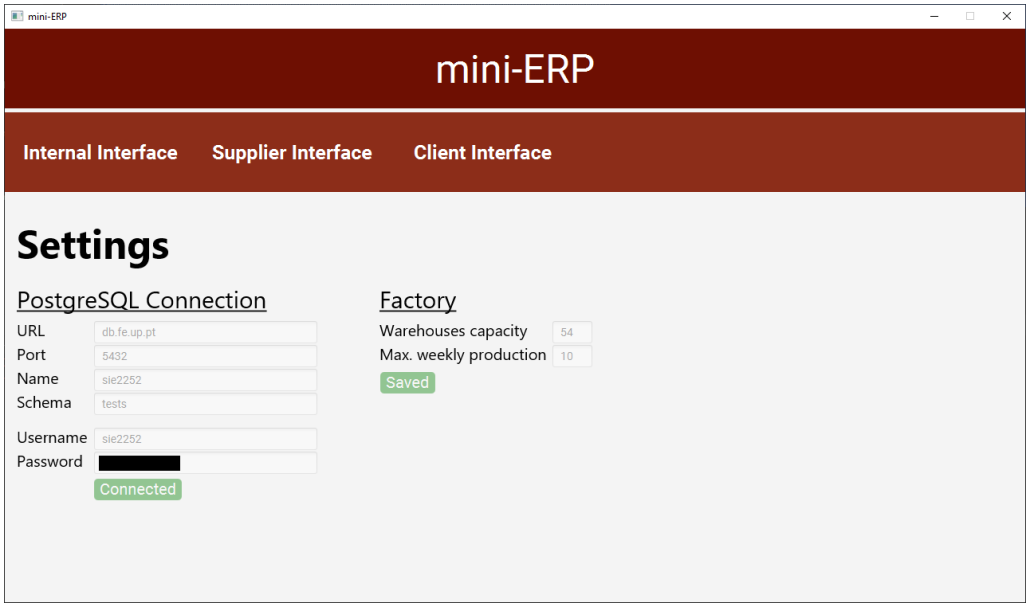


Figura 4.4: Interface inicial

Implementou-se uma interface inicial (figura 4.4) que permite ao utilizador conectar-se à base de dados. Antes de realizar a conexão, é necessário que o utilizador apenas introduza manualmente

a *lookup table* da *Factory* (disponível no anexo A), uma vez que ao ser iniciada, a aplicação vai sempre recorrer a esta tabela para apresentar configurações anteriores. Isto permite evitar que seja necessário que o utilizador preencha os parâmetros todas as vezes que inicia a aplicação. A restante estrutura do modelo de dados encontra-se embebida na aplicação e é introduzida na base de dados no momento em que é efetuada a conexão com sucesso. Com as configurações definidas, o utilizador pode abrir as 3 interfaces através da barra de navegação principal.

4.4.2 Interface interna

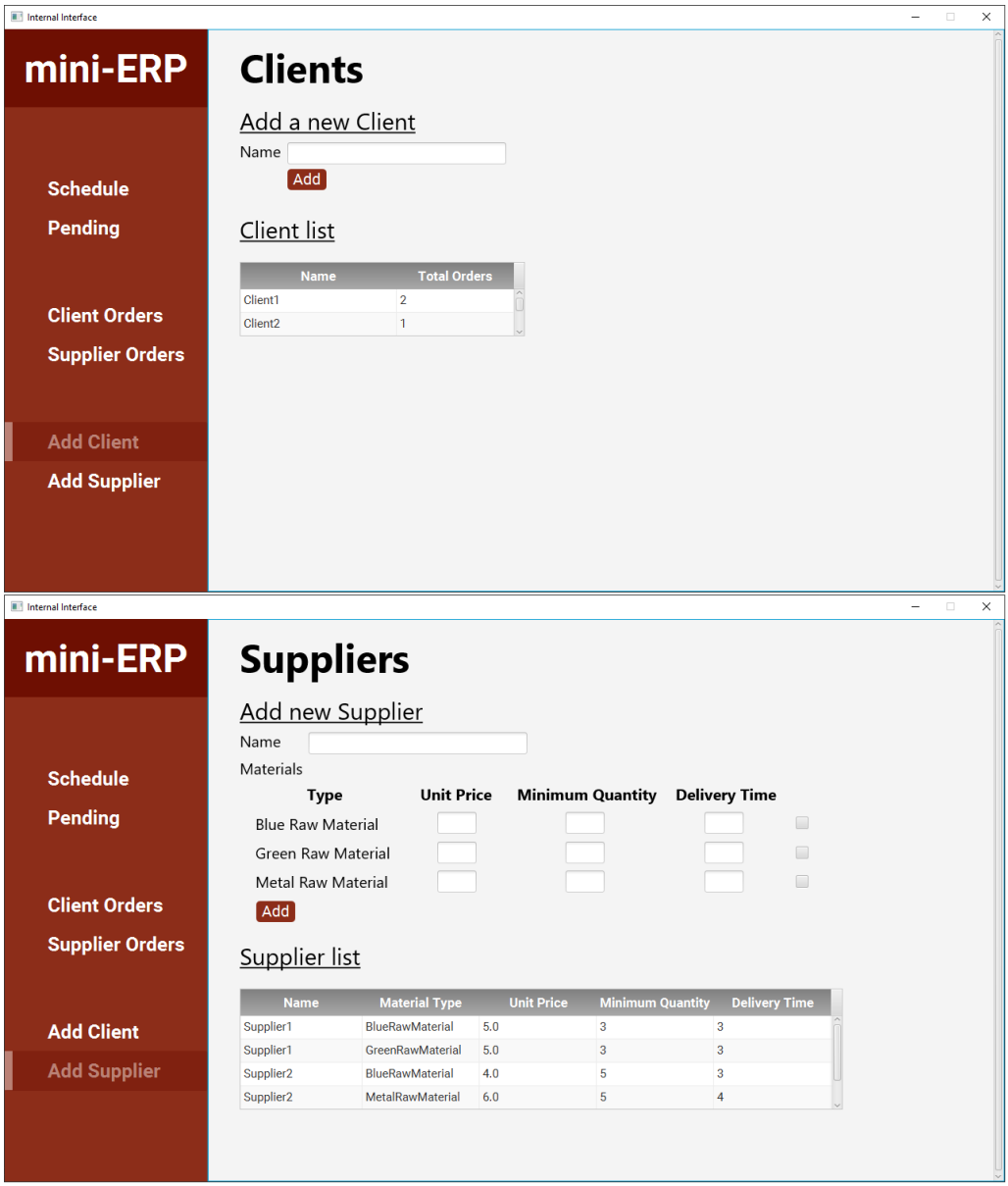


Figura 4.5: Páginas *Add Client* e *Add Supplier*

A interface interna é a mais completa, pois permite ao utilizador visualizar a maioria das informações presentes no modelo de dados e atuar na simulação do ponto de vista interno. No quesito de introdução de dados, existem duas páginas (figura 4.5) em que é possível registar novos clientes e fornecedores, encontrando-se listados os já existentes.

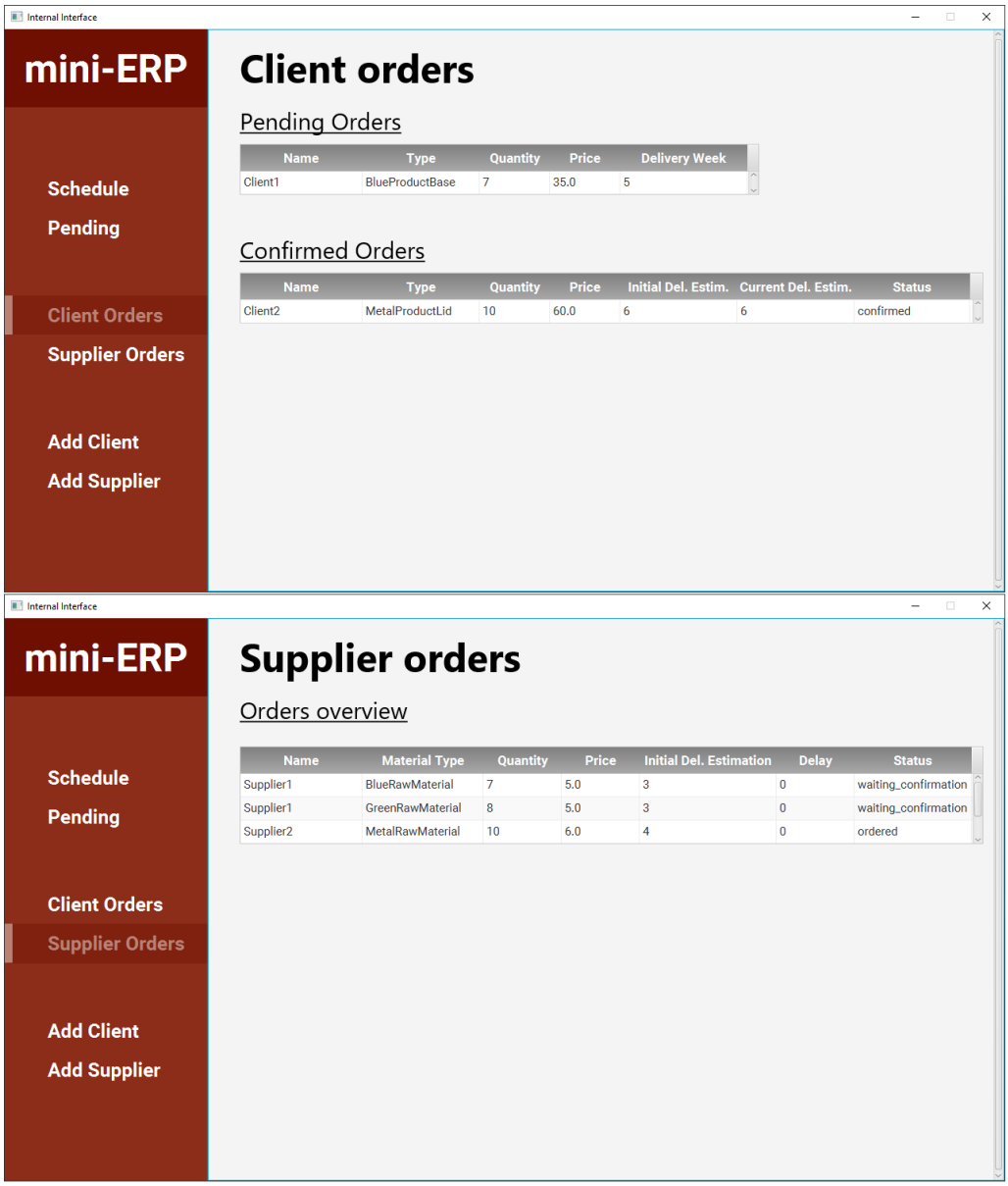


Figura 4.6: Páginas *Client Orders* e *Supplier Orders*

Para a visualização das encomendas aos agentes externos, foram implementadas duas páginas (figura 4.6). Todos os detalhes das encomendas realizadas aos fornecedores podem ser consultadas na página *Supplier Orders*. Já as encomendas de clientes, encontram-se divididas pela fase em que se encontram. Assim como foi descrito no algoritmo 3.6, para uma encomenda ser considerada confirmada, é necessário haver uma confirmação tanto por parte do cliente como pela parte interna

da fábrica. Na página *Client Orders*, é possível visualizar as encomendas já confirmadas, independentemente do seu estado (em curso, completas, canceladas), e as que se encontram pendentes de uma resposta do cliente.

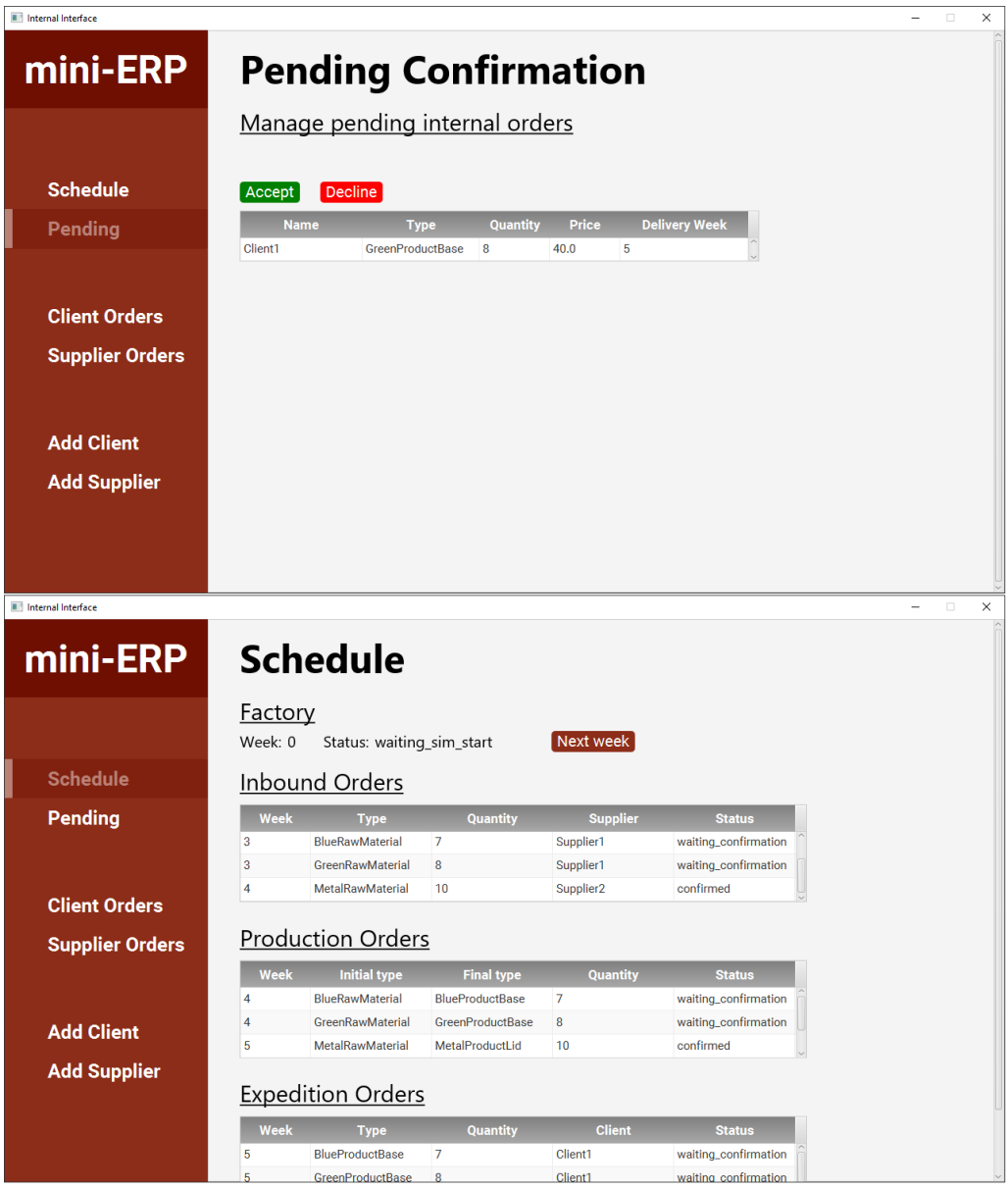


Figura 4.7: Páginas *Pending Orders* e *Schedule*

As encomendas que estão pendentes da confirmação interna encontram-se na página *Pending Orders* (figura 4.7), onde o utilizador pode selecionar cada encomenda individualmente e confirmá-la ou cancelá-la.

Finalmente, o planeamento da produção, ou seja, as *Inbound*, *Production* e *Expedition orders* podem ser visualizados na página Schedule (figura 4.7). Além disto, nesta página o utilizador

pode ver o estado da simulação, a semana em que se encontra e avançar para a semana seguinte, de acordo com a lógica apresentada em 4.2. Ao avançar para a semana seguinte é despoletada, também, a execução do algoritmo de processamento do *feedback* apresentado anteriormente na secção 3.4.3.3.

4.4.3 Interface do fornecedor

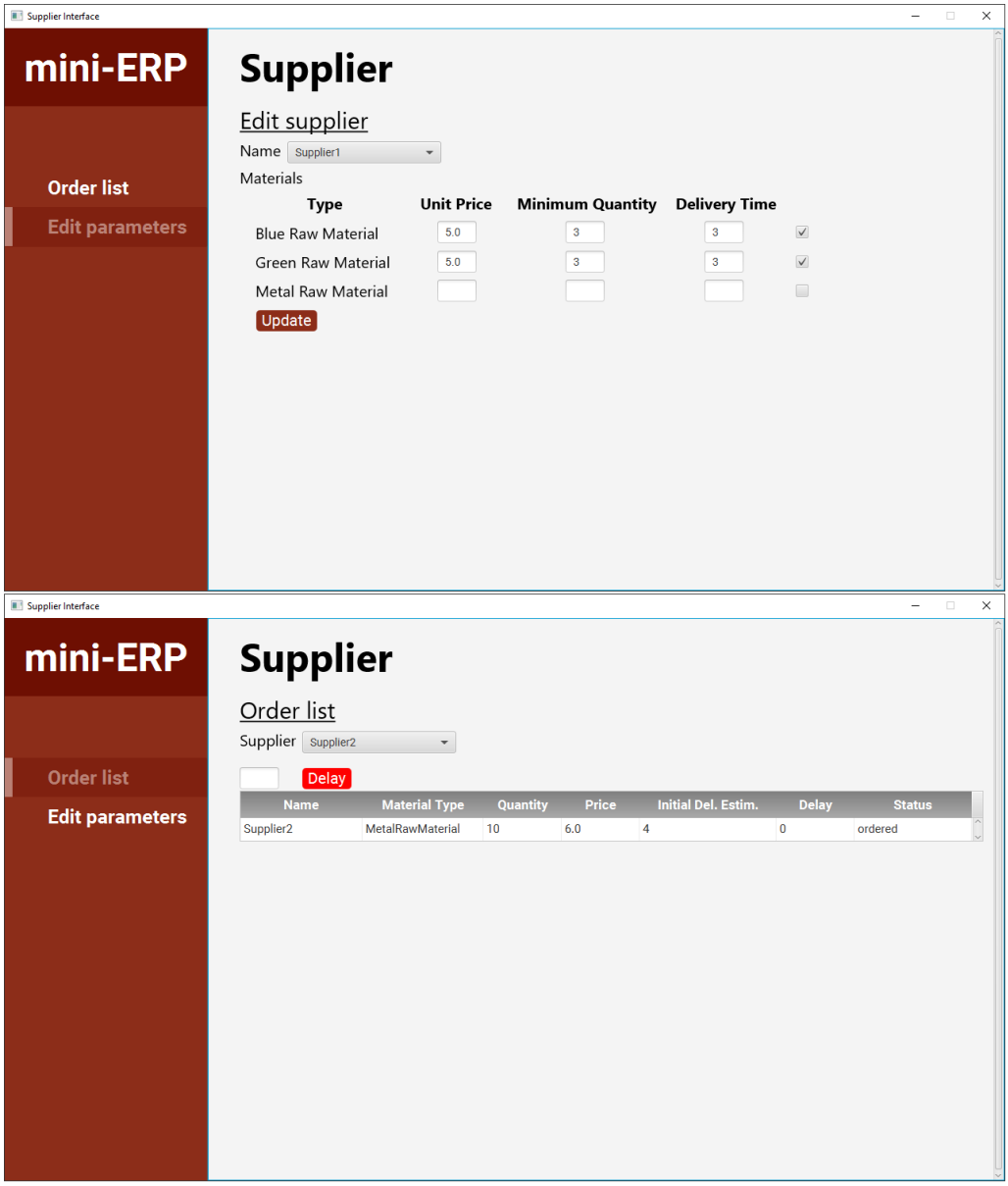


Figura 4.8: Páginas *Edit parameters* e *Order List*

Na interface do fornecedor foram implementadas apenas duas páginas (figura 4.8). Em ambas, o utilizador possui uma *combo box* que permite a seleção do fornecedor de entre os existentes.

Tal como o nome indica, a página *Edit parameters* permite a edição dos parâmetros do fornecedor selecionado, independentemente das suas encomendas em curso. A página *Order List* lista as encomendas existentes para o fornecedor em questão e permite atrasá-las, introduzindo o valor em semanas do atraso pretendido. Ao realizar este procedimento, é executado o algoritmo apresentado na secção 3.4.3.1, de modo a efetuar um reescalonamento.

4.4.4 Interface do cliente

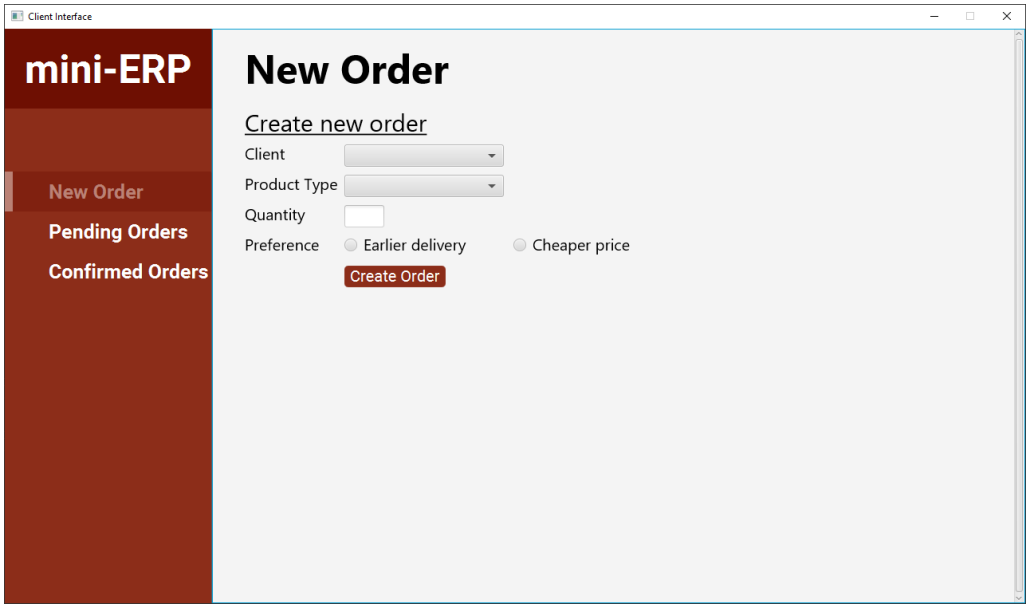


Figura 4.9: Página *New Order*

A principal função da interface do cliente consiste em realizar novas encomendas. Na página *New Order* (figura 4.9), o utilizador pode selecionar o nome do cliente para o qual pretende realizar uma encomenda e preencher os parâmetros disponíveis. A opção *Earlier delivery* ou *Cheaper price* tem como objetivo dar prioridade à escolha de um fornecedor com menor preço ou menor tempo de entrega, caso seja necessário encomendar matérias-primas. É, desta forma, executado o algoritmo para novas encomendas descrito na secção 3.4.2.

À semelhança da interface interna, na interface do cliente as encomendas também se encontram divididas por páginas diferentes. A página *Pending Orders* (figura 4.10) disponibiliza a resposta interna que é dada ao cliente após o pedido de encomenda. Finalmente, na página *Confirmed Orders* (figura 4.10), são listadas todas as encomendas de um determinado cliente, sendo possível o seu cancelamento. Ao confirmar um cancelamento, é executado o algoritmo descrito na secção 3.4.3.2.

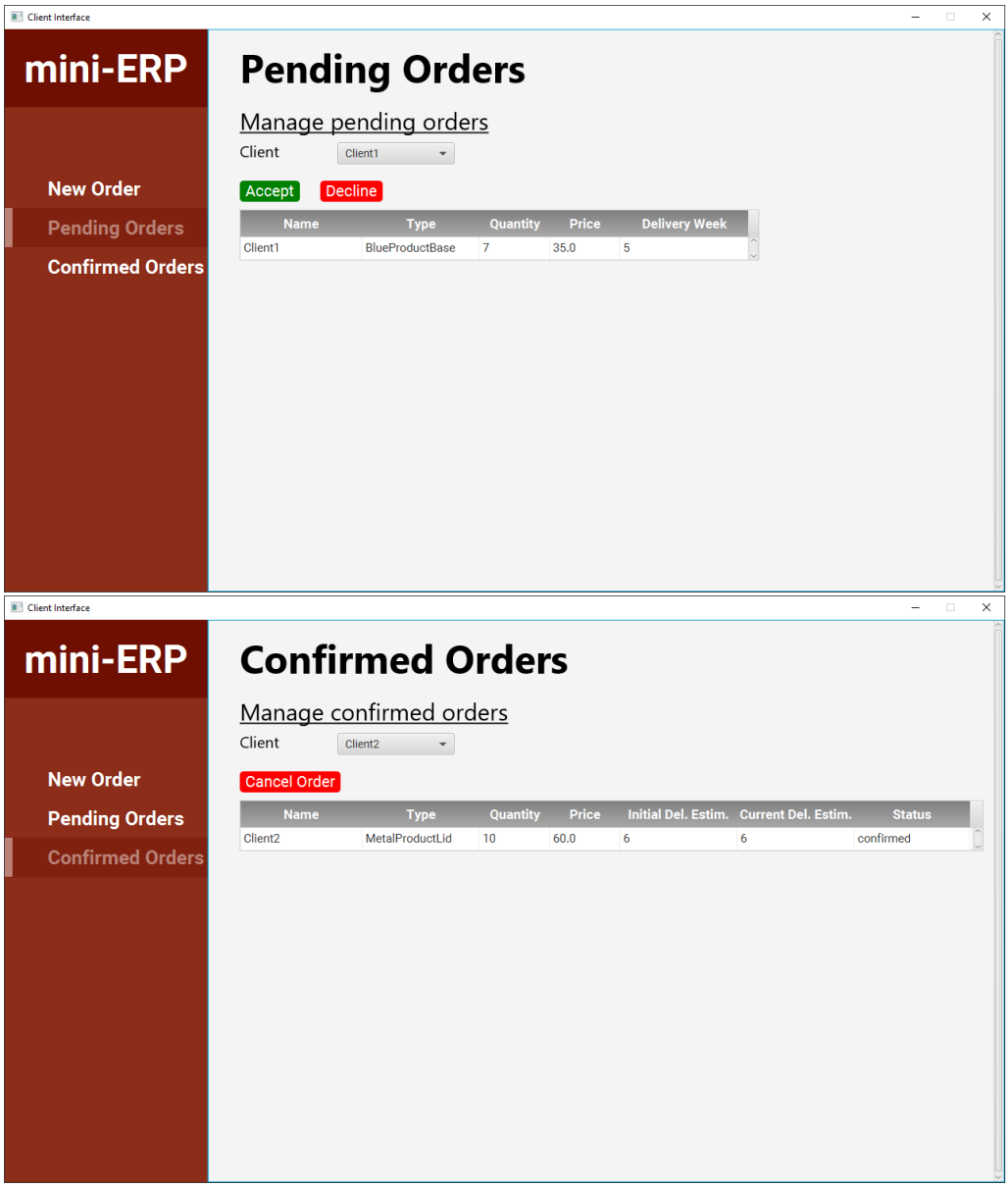


Figura 4.10: Páginas *Pending Orders* e *Confirmed Orders*

4.5 Alterações ao *Digital Twin*

Antes da implementação do mini-MES, foram realizados vários testes ao *Digital Twin*, enviando sequências de comandos semelhantes às que o mini-MES teria de ser capaz de enviar. Com isto, foram detetadas falhas no funcionamento do *Digital Twin*. Nesta secção são apresentadas as correções efetuadas ao *Digital Twin*, que permitiram a posterior implementação do mini-MES.

4.5.1 *Feedback* dos comandos

A interface de comunicação disponibilizada no *Digital Twin* possui uma estrutura OPC-UA para o envio de comandos, *order*, detalhada na secção 2.2.6. Cada *order* possui, de entre vários outros parâmetros, um identificador único e uma estrutura representativa da peça, *part*. Por sua vez, é importante destacar que a estrutura *part* possui como um dos seus parâmetros uma variável denominada *order_id*. Por este motivo, é necessário que a aplicação que envia comandos para o *Digital Twin* coloque o mesmo valor no *order.id* e no *order.part.order_id* para cada comando enviado.

Quando uma *order* é finalizada, todos os seus parâmetros são copiados para a lista *orders_status_out*, possibilitando que a aplicação hierarquicamente superior verifique quais comandos foram finalizados com sucesso, pesquisando pelo identificador atribuído para cada um. Após testes, verificou-se que, na lista *orders_status_out*, o *order.id* não coincidia com o *order.id* enviado, enquanto o *order.part.order_id* encontrava-se correto. Apesar de ter sido realizada uma análise ao código dos *Function Blocks* com o objetivo de identificar a origem do problema, a sua resolução revelou-se um processo muito complexo dado estar relacionada com a comunicação entre *Function Blocks*. Por este motivo, as conclusões obtidas sobre esta questão podem ser consultadas no anexo B. Com isto, definiu-se que quando a aplicação superior, neste caso, o mini-MES, for verificar a lista de comandos completos, terá de pesquisar pelos identificadores pretendidos na variável *order.part.order_id*.

De um modo geral, esta alternativa é viável, pois o *Digital Twin* permite que as aplicações superiores alterem os parâmetros das peças, possibilitando, consequentemente, alterar o valor do *part.order_id* no envio de novos comandos. No entanto, foi necessário realizar uma alteração ao código dos *Function Blocks* da classe *Warehouse* (armazéns). Como são o único *Function Block* que guarda várias peças, possui uma memória interna para cada uma (estrutura *part*) que é armazenada. Isto origina a que, quando é enviado um comando para retirar determinada peça, a estrutura *part* que é transmitida para o *Function Block* adjacente possui os mesmos parâmetros da *part* que foi armazenada, contendo o *order_id* referente ao comando de armazenamento e não ao de retirada.

Para solucionar este problema, foi adicionado um bloco de código ao *Function Block* da classe *Warehouse* que faz com que, ao ser lido um comando de remoção de peça, o *part.order_id* antigo é substituído pelo identificador do comando em questão. Isto permite à aplicação saber o momento em que o comando de remoção da peça do armazém foi finalizado.

4.5.2 Limite no número de comandos enviados

Como abordado na secção 2.2.6, a lista *orders_status_out* consiste num *buffer* que permite armazenar 100 comandos finalizados. Com isto, é necessário que a aplicação que envia comandos para a *Digital Twin* envie periodicamente, também, um sinal *read_sig*, para a limpeza deste *buffer*, de modo a que não seja ultrapassado o limite de armazenamento estabelecido. Apesar disto, nos

teste realizados, verificou-se que após o envio de apenas 51 comandos, a *Digital Twin* interrompia o funcionamento, independentemente do envio do sinal de limpeza do *buffer*.

Após depuração do erro, constatou-se que no *Function Block Orders Manager* existia uma estrutura denominada *all_orders*, consistindo num *buffer* de 100 posições, semelhante ao *orders_status_out*. Quando um novo comando era detetado na estrutura OPC-UA *order*, ou quando a execução desse comando era finalizada, era realizada uma cópia dos seus parâmetros para o *all_orders*. Ao contrário do *orders_status_out,* este *buffer* nunca era limpo. Com a análise realizada, não foi encontrada uma necessidade para a existência deste *buffer*, pelo que se optou pela sua remoção. O código relativo ao mesmo encontra-se devidamente comentado e sinalizado dentro do *Function Block Orders Manager*.

Além disto, de forma a garantir que não ocorrem este tipo de problemas, o mini-MES irá enviar o sinal *read_sig* várias vezes durante a simulação, dependendo da quantidade de comandos enviados.

4.6 Implementação do mini-MES

4.6.1 Mapeamento do *layout* da fábrica

Para que o controlo da fábrica, em concreto, o encaminhamento das peças, seja possível, é necessário o mapeamento do layout da fábrica existente, ou seja, dos identificadores de cada equipamento ao nível do *Digital Twin*. Este procedimento permite ao mini-MES verificar se o *layout* em questão cumpre as restrições impostas e definir os percursos a serem utilizados entre os pontos de interesse.

Assim, optou-se por descrever o layout utilizado através de um ficheiro XML, uma vez que este tipo de ficheiros pode ser facilmente convertido para classes Java. Desta forma, para cada equipamento presente no *Digital Twin*, devem constar no ficheiro XML os seguintes parâmetros:

- *dt_id*: valor inteiro, contendo o identificador deste equipamento ao nível da Digital Twin.
- *equipment_type*: *String*, contendo o tipo do equipamento ao nível do mini-MES.
- *neighbourIdList*: lista de *dt_id* dos equipamentos adjacentes para os quais uma peça pode ser encaminhada.

Os equipment_type que devem estar obrigatoriamente presentes correspondem aos pontos de interesse descritos no capítulo anterior, presentes na tabela 4.1. Relativamente aos *Machining Center*, foram criados 3 tipos, de acordo com a capacidade de produção que o utilizador pretenda para cada uma, listados na tabela 4.2. Finalmente, na tabela 4.3, encontram-se os equipamentos de transporte a serem colocados entre os pontos de interesse. Os *Conveyor* possuem 3 tipos dependendo do seu tamanho, uma vez que o mesmo será considerado no cálculo dos caminhos.

Tabela 4.1: *equipment_type* obrigatórios

equipment_type
InboundEmitter
InboundBuffer
InboundWarehouse
InboundWarehouseExit
ProductionRemover
ProductionEmitter
ExpeditionWarehouse
ExpeditionWarehouseExit
ExpeditionRemover
PartDisposer

Tabela 4.2: *equipment_type* dos *Machining Center*

equipment_type
BothProducer
LidProducer
BaseProducer

Tabela 4.3: *equipment_type* dos equipamentos de transporte

equipment_type
Conveyor2m
Conveyor4m
Conveyor6m
Turntable

Exemplificando, um *Conveyor* de 6 metros com um identificador 9, cujas saídas estejam conectadas (através de um *Sorter*) a três outros equipamentos, pode ser representado pelo seguinte código XML:

```
<equipment>
    <id_dt>9</id_dt>
    <equipment_type>Conveyor6m</equipment_type>
    <neighbourIdList>
        <id>10</id>
        <id>16</id>
        <id>21</id>
    </neighbourIdList>
</equipment>
```

Código 4.1: Exemplo XML para um equipamento

Com isto, para cada equipamento presente no ficheiro XML é instanciado um objeto da classe *Equipment* em Java (composta pelos mesmos parâmetros), para posterior processamento.

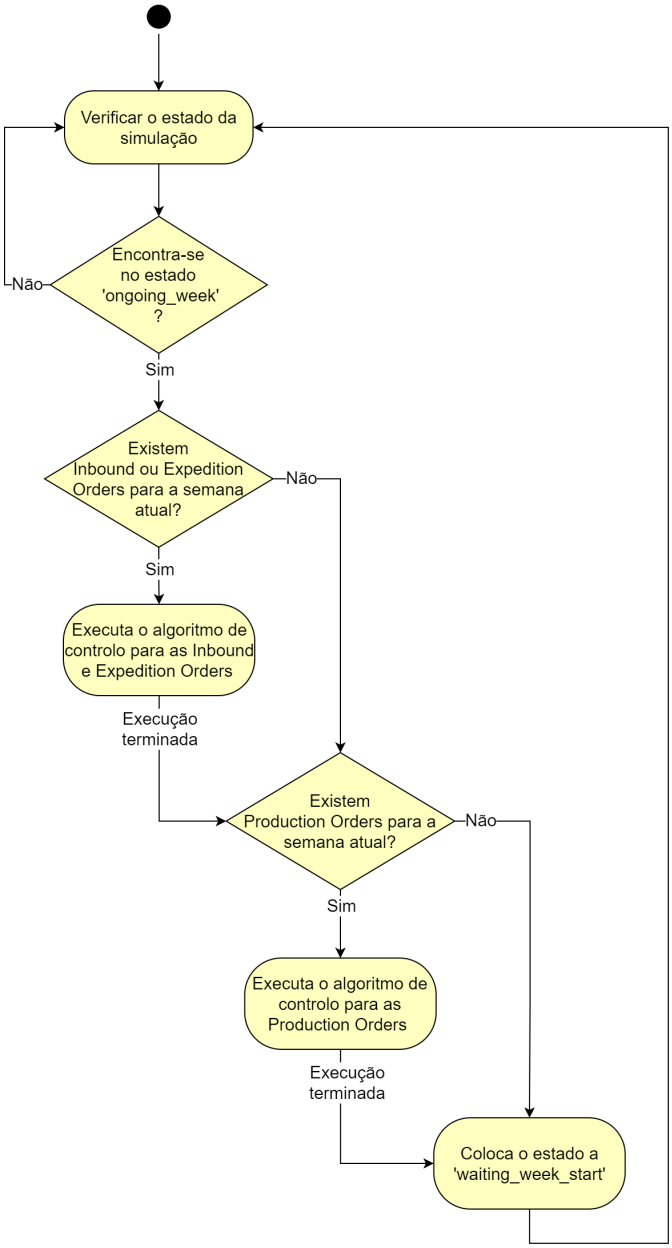


Figura 4.11: *Thread* de controlo

4.6.2 Gestor de percursos

Para a gestão de todo o processo relativo à definição dos caminhos, foi implementado um Singleton denominado *PathManager*. Nesta classe, o método *generatePaths* recebe como entrada a lista de *Equipment* e com ela, calcula e armazena as sequências de identificadores que correspondem aos caminhos entre os pontos de interesse.

O cálculo dos caminhos é efetuado recorrendo ao algoritmo Dijkstra (cuja utilização já foi justificada no capítulo anterior), uma vez que cada *Equipment* contém o identificador do equipamento adjacente, ou seja, segue a estrutura de um grafo. São atribuídos aos equipamentos de transporte um custo equivalente ao seu tamanho em metros (daí a diferente *equipment_type* dos *Conveyors*). Utilizando esta informação, é gerada uma matriz de custos utilizada pelo algoritmo, sendo particularmente útil para a definição de caminhos na zona de produção, uma vez que vão existir várias possibilidades de chegar a cada máquina. Deste modo, para cada caminho existente entre pontos de interesse, o *PathManager* utiliza este algoritmo, fornecendo-lhe o identificador do equipamento de origem e de destino, obtendo como retorno a sequência de identificadores correspondentes que serão utilizados pelos algoritmos de controlo das zonas explicados no capítulo anterior.

4.6.3 Thread de controlo

O mini-MES, tal como o mini-ERP, foi desenvolvido utilizando o JavaFX Scene Builder, uma vez que também se pretende que possua interfaces gráficas. No entanto, como é necessário que o algoritmo de controlo esteja permanentemente em execução, implementou-se uma *thread* secundária, para que o tempo de resposta interfaces não fosse comprometido. Nesta *thread*, foram implementados os algoritmos de controlo para as diferentes zonas (abordados na secção 3.5.3), sendo estes executados como representado na figura 4.11.

Para a interação com o *Digital Twin*, a *thread* necessita de uma interface de comunicação OPC-UA. Para este efeito, foi implementado um Singleton, semelhante ao *DatabaseHandler* já referido, denominado *OpcUaHandler*. Nesta classe foram criados métodos que preenchem os parâmetros da variável OPC-UA *order*, conforme o comando do *Digital Twin* que se pretende enviar (Emit, NewPath, NewProps, OutWH). A leitura e escrita dos valores nas variáveis OPC-UA é realizada através da biblioteca Eclipse Milo[24], que implementa um cliente OPC-UA em Java.

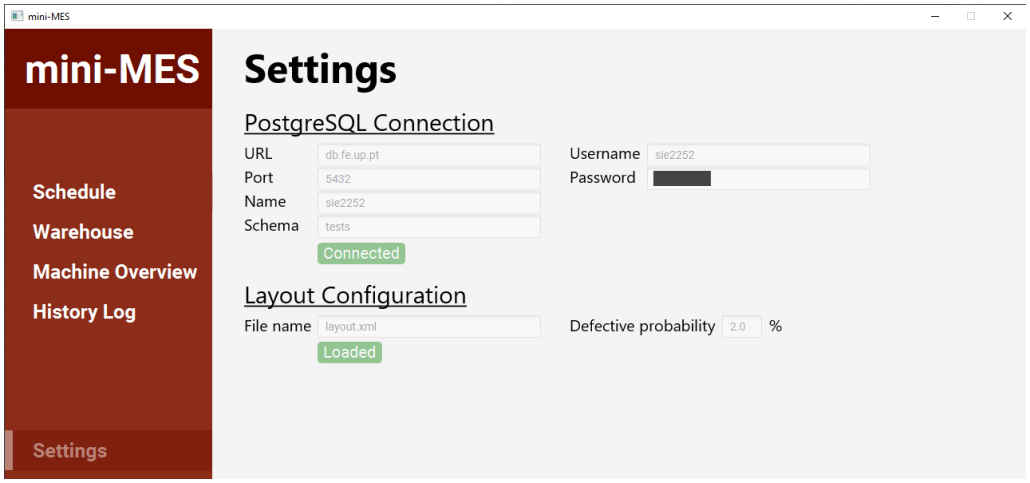


Figura 4.12: Página *Settings*

4.6.4 Interfaces

O mini-MES possui uma interface inicial (figura 4.12) que permite a conexão à base de dados e a leitura do ficheiro XML que será usado para mapear o *layout*. Adicionalmente, encontra-se também disponível um campo em que utilizador define qual a percentagem de peças que serão consideradas com defeito. Após o mini-MES ser configurado corretamente, é iniciada a *thread* de controlo referida na secção anterior.

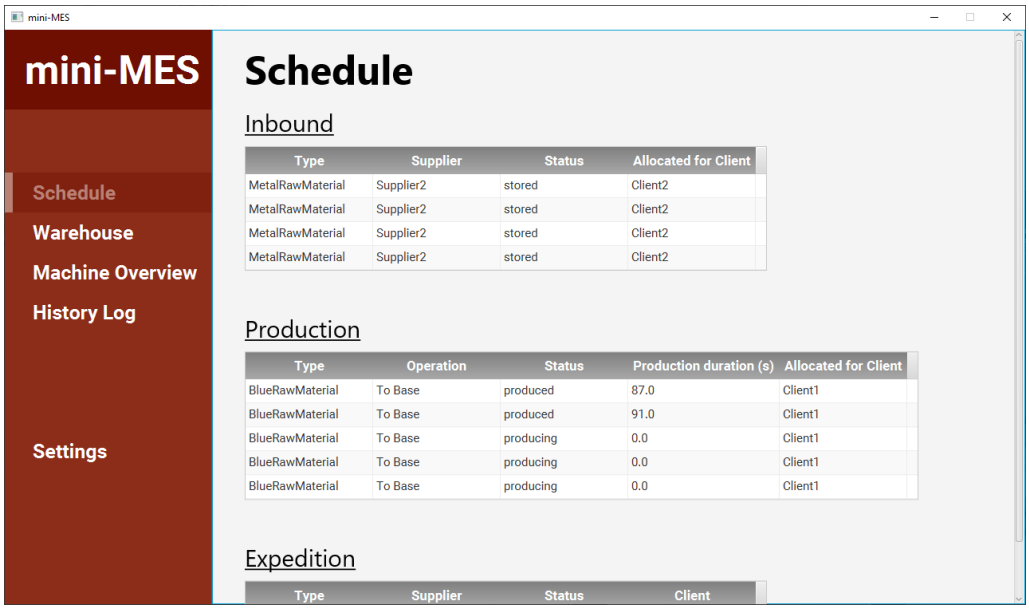


Figura 4.13: Página *Current Schedule*

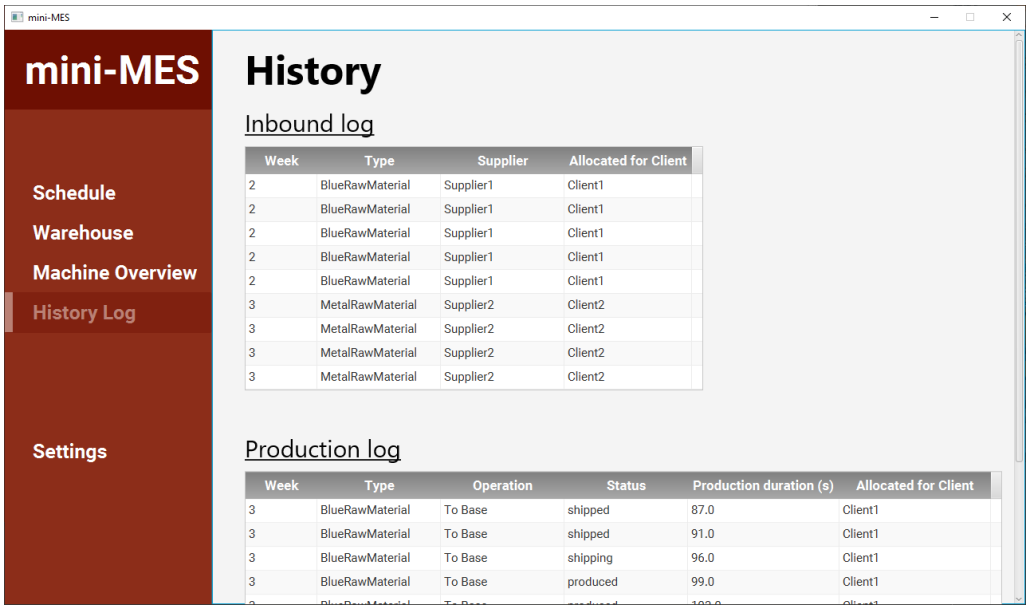


Figura 4.14: Página *History Log*

Ao contrário do mini-ERP, em que as informações apresentadas têm um escopo mais abrangente, no nível de ordens que afetam várias peças, as interfaces do mini-MES foram desenvolvidas com o foco nas caraterísticas individuais de cada peça.

A página *Schedule* do mini-MES (figura 4.13) permite ao utilizador acompanhar em tempo real o estado de cada uma das peças que estão contempladas nas ordens da semana atual. As informações relativas ao *Schedule* das semanas anteriores à atual podem ser consultadas na página *History Log* (figura 4.14).

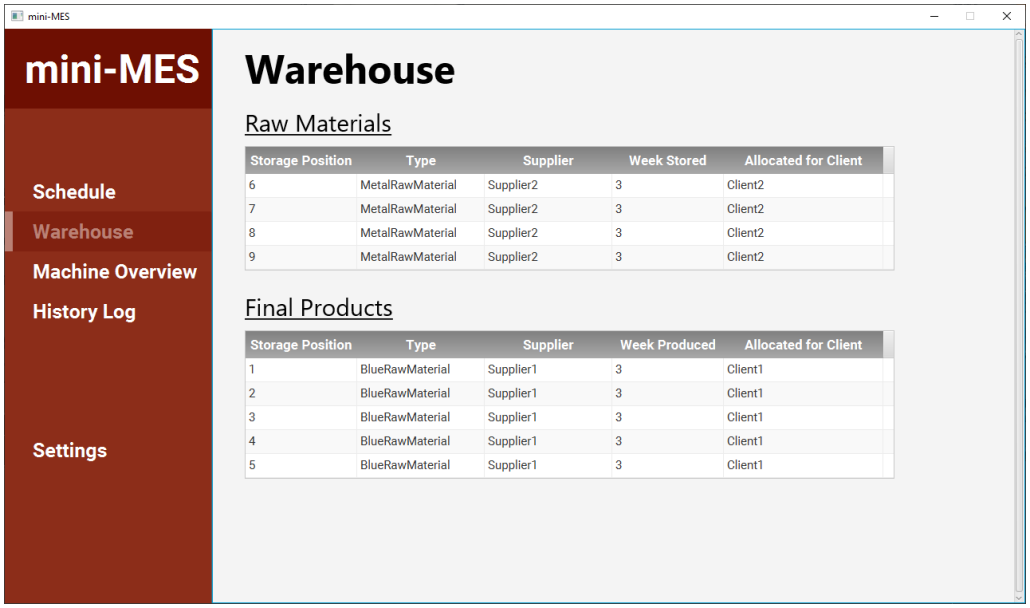


Figura 4.15: Página *Warehouse*

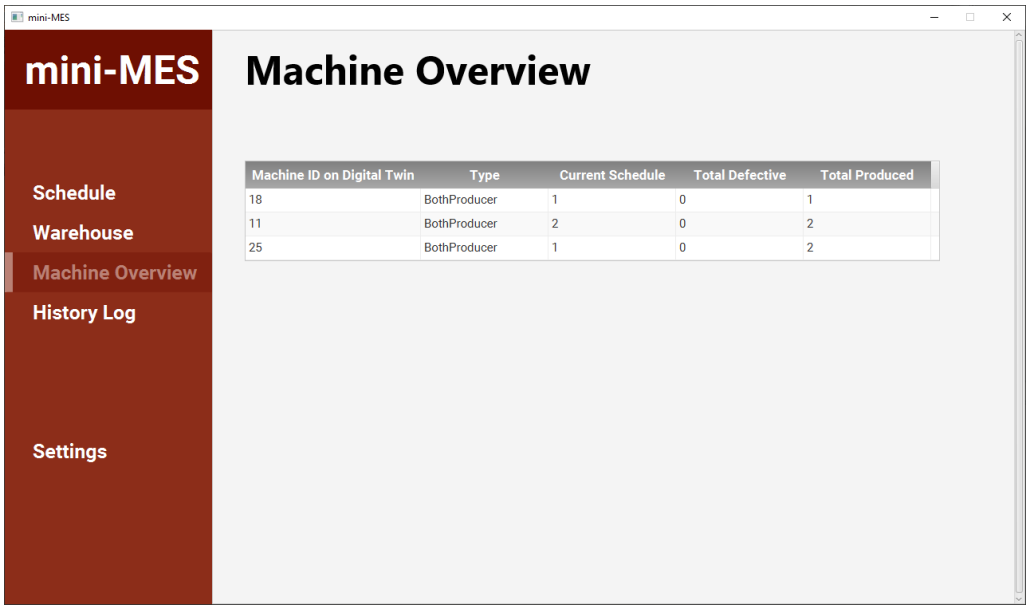


Figura 4.16: Página *Machine Overview*

A ocupação dos dois armazéns, de matérias-primas e produtos finais, pode ser consultada na página *Warehouse* 4.15.

Por fim, foi implementada a página *Machine Overview* (figura 4.16), que permite ao utilizador verificar as informações relativas a cada *Machining Center*. Para cada uma, apenas foram considerados o número de peças no escalonamento da semana atual, o número de peças defeituosas e o número total de peças já produzidas.

4.7 Testes e Validação

Nesta secção é apresentado um exemplo de utilização das aplicações que permite testar e validar o funcionamento de todo o conjunto. Inicialmente, é descrito o *layout* da linha de produção que foi construído no *Digital Twin*. A seguir, foi estabelecida uma sequência de eventos a serem executados, do ponto de vista do utilizador, permitindo uma validação das funcionalidades implementadas.

4.7.1 Layout utilizado no *Digital Twin*

Para ser possível a realização dos testes do conjunto, foi construído um *layout* na *Digital Twin* que possui as divisões e restrições descritas nos capítulos anteriores. Uma vez que o objetivo deste trabalho é a vertente educacional, não foi implementado um *layout* de teste com um grau de complexidade muito elevado.

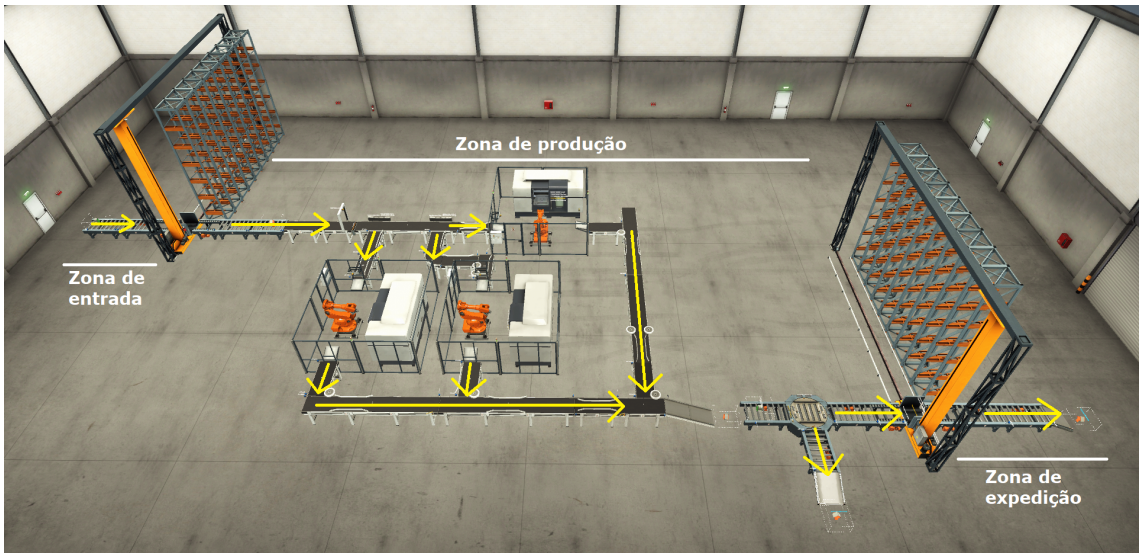


Figura 4.17: *Layout* de teste no FactoryIO

Optou-se por manter uma zona de entrada e de expedição simples, que consistem maioritariamente nos pontos de interesse obrigatórios. Como a zona de produção permite um maior grau de

liberdade na construção do *layout*, foram colocados 3 percursos alternativos, um para cada *Machining Center*. Deste modo, vão existir recursos partilhados entre os percursos (*Sorter* e *Conveyors* partilhados) e também será possível testar a funcionalidade do mini-MES que limita a capacidade de produção dos *Machining Centers* de acordo com o tipo definido pelo utilizador (*LidProducer*, *BaseProducer*, *BothProducer*).

O layout pretendido foi construído nas duas aplicações que compõem o *Digital Twin*, ou seja, no FactoryIO e no Codesys. A montagem efetuada no FactoryIO encontra-se na figura 4.17, com as respetivas zonas e percursos demarcadas. Para efeitos de elucidação do leitor, a montagem dos FB no Codesys encontra-se representada no anexo D.

4.7.2 Teste de utilização

É importante salientar que, enquanto decorria o desenvolvimento das aplicações, cada funcionalidade era testada após a sua implementação. Tendo isto em consideração, foi efetuado um teste final, ou seja, elaborada uma sequência de ações a serem executadas em diferentes momentos da simulação, que permitem demonstrar algumas funcionalidades do mini-ERP e do mini-MES.

Antes de se iniciar a simulação, são necessárias as configurações das aplicações. Ao nível do mini-MES, este requer um ficheiro XML com o mapeamento do *layout* de teste que será utilizado, descrito na secção anterior. O código XML respetivo pode ser consultado no anexo C. Relativamente ao mini-ERP, optou-se por definir uma capacidade semanal de 10 peças para que esta simulação de exemplo não se torne demasiado longa. A capacidade dos armazéns utilizada foi de 30 peças.

Foram registados dois fornecedores com as caraterísticas descritas na tabela 4.4. Deste modo, é possível que os clientes encomendem todos os produtos possíveis. Além disto, para o caso da matéria-prima azul, a opção entre priorizar o tempo de entrega ou o menor custo vai influenciar a escolha de fornecedor, uma vez que o primeiro fornecedor possui um custo maior e um tempo de entrega menor, e o segundo, o contrário.

Tabela 4.4: Fornecedores registados

Nome	**Tipo de material**	**Preço unitário**	**Quantidade mínima**	**Tempo de entrega**
Fornecedor1	Azul	5.0	5	2
Fornecedor1	Verde	5.0	5	3
Fornecedor2	Azul	3.0	5	4
Fornecedor2	Metal	6.0	5	5

A sequência de eventos que foram executados durante a simulação pode ser consultada na tabela 4.5, com a semana em que deverão ocorrer e com uma pequena descrição do evento em questão. A semana 0 corresponde à semana antes de ser iniciada a simulação. Para ilustração

deste processo, foi elaborado um vídeo[1] de demonstração do funcionamento das aplicações no decorrer da execução da sequência de eventos descritos.

Tabela 4.5: Eventos

Semana	**Descrição do evento**
0	O Cliente1 encomenda 5 *BlueProductBase* com preferência de menor preço.
0	O Cliente2 encomenda 3 *MetalProductLid*.
1	O Cliente3 encomenda 5 *GreenProductBase*.
1	O Cliente4 encomenda 5 *MetalProductBase*.
2	O Fornecedor1 informa o atraso da encomenda de *GreenRawMaterial*.
5	O Cliente2 cancela a sua encomenda.

Ao longo da execução desta simulação, é possível verificar o planeamento realizado para cada nova encomenda, as alterações efetuadas em caso da ocorrência dos eventos de cancelamento e atraso e o controlo da linha de produção presente no *Digital Twin*. Deste modo, este exemplo de utilização permite a validação do funcionamento de todo o conjunto.

4.8 Conclusões

Neste capítulo foi explicada a implementação em concreto das aplicações. A partir da arquitetura proposta, foi desenvolvido um modelo de dados a ser utilizado na base de dados partilhada. Para o mini-ERP foi dado ênfase na conciliação entre as diferentes interfaces e os algoritmos para os eventos pretendidos (já descritos no capítulo anterior). Foram enunciadas e explicadas as alterações realizadas ao *Digital Twin* de modo a ser possível a concretização deste trabalho. No mini-MES, foram detalhados os procedimentos que permitem à aplicação abranger diferentes *layouts*, tendo em consideração as restrições impostas.

A última secção consiste na descrição de um exemplo demonstrativo do funcionamento das aplicações. Primeiro, é apresentado de que forma o *Digital Twin* desenvolvido no trabalho anterior pode ser utilizada no contexto do trabalho atual. Com o layout construído e as configurações realizadas, foi iniciada a simulação. Como o objetivo final deste trabalho é a vertente educativa, as aplicações foram validadas recorrendo a um exemplo de utilização, ilustrando os conceitos educacionais que se pretendem demonstrar sobre a pirâmide de automação, com o foco nos ERP e MES.

[1] https://youtu.be/49uUoJ7FxFI

Capítulo 5

Conclusões

Este trabalho, em conjunto com a ferramenta desenvolvida numa dissertação anterior, disponibiliza um ambiente de simulação que permite apresentar os conceitos da pirâmide de automação e das suas camadas. Os sistemas ERP e MES comerciais, que implementam as camadas superiores, não são adequados para o meio educacional, por serem soluções complexas desenvolvidas para a indústria. Assim, com este trabalho, pretendeu-se o desenvolvimento de duas aplicações, o mini-ERP e o mini-MES, que ilustrem os conceitos principais destes sistemas, mas que sejam mais simples, para serem utilizados num contexto educacional.

Em primeiro lugar, foi efetuado um estudo detalhado da ferramenta de simulação desenvolvida na dissertação anterior, permitindo perceber não só de que forma esta poderia ser utilizada, mas também todo o seu funcionamento. As duas vertentes do estudo foram essenciais uma vez que, num determinado ponto, foi necessário efetuar algumas alterações ao *Digital Twin*, para que o trabalho da dissertação atual fosse realizado de forma mais adequada.

Ao nível do mini-ERP, a sua função principal consiste no planeamento da produção. De modo a atender os pedidos enviados pelos clientes, são utilizados os materiais já existentes em armazém e, caso seja necessário, procede-se à escolha de um fornecedor, baseada nos seus parâmetros, para efetuar uma encomenda de matérias-primas. Partindo da previsão inicial da sua entrega, o algoritmo de escalonamento organiza as ordens de produção, que afetam as diferentes zonas da fábrica, pelas várias semanas, conforme a disponibilidade existente. Além disto, foi contemplado um conjunto de eventos que podem ocorrer a qualquer momento, por parte dos clientes, fornecedores ou devido à produção de peças com defeito, e que alteram o planeamento existente.

Na aplicação do mini-MES, tiveram de ser consideradas algumas restrições. Num contexto real, após analisado o *layout* existente, os MES são configurados individualmente para efetuarem o controlo da produção de forma otimizada. Neste caso, o *Digital Twin* permite construir uma grande variedade de *layouts*, pelo que foi necessário impor restrições, como a divisão da linha de produção em zonas (de Entrada, Produção e Expedição) e a obrigatoriedade de estarem presentes alguns equipamentos. Com isto, e recebendo o planeamento efetuado pelo mini-ERP, o mini-MES

procede ao controlo da execução das ordens de produção, interagindo com o *Digital Twin* pelo envio de comandos e receção do seu *feedback*.

Finalmente, o conjunto de aplicações desenvolvidas foi validado recorrendo a um exemplo de utilização que permite demonstrar o funcionamento e a interação entre as várias camadas da pirâmide de automação, permitindo concluir que os objetivos principais foram concluídos.

5.1 Trabalhos futuros

Como trabalho futuro, existem alguns aspetos já existentes que podem ser melhorados, relativamente aos algoritmos de planeamento e controlo da produção. Todos os algoritmos deste trabalho foram desenvolvidos com base em métrica heurísticas, ou seja, apresentam um desempenho satisfatório para o contexto deste trabalho, mas não são a solução ótima. Deste modo, sugere-se que sejam implementados algoritmos, de base científica, que permitam avaliar quantitativamente a gestão das operações e efetuá-la de forma otimizada.

Além disto, no contexto do planeamento efetuado pelo mini-ERP, poder-se-ia considerar a implementação de um reescalonamento total na ocorrência dos diferentes eventos contemplados neste trabalho. Apesar da complexidade mais elevada, poderia evitar situações em que uma sequência excessiva de cancelamentos, por exemplo, leve a uma situação de *deadlock* devido à sobrelotação dos armazéns.

Numa vertente de acrescentar novas funcionalidades, sugere-se a possibilidade de dois mini-ERP interagirem entre si, como clientes e fornecedores, contemplando dois ambientes de simulação.

Anexo A

Setup da base de dados

Caso seja a primeira vez que a base de dados vai ser utilizada para o mini-ERP e o mini-MES, é necessário realizar duas operações, correspondentes ao seguinte código:

```
/*** CREATE Tables (copy to pgadmin) ***/
/** LOOKUP TABLE **/
CREATE TABLE factory (
    setup_status        VARCHAR(50) NOT NULL,
    sim_status          VARCHAR(50) NOT NULL,
    warehouse_capacity  INT         NOT NULL,
    weekly_production   INT         NOT NULL
);
INSERT INTO factory (   setup_status,
                        sim_status,
                        warehouse_capacity,
                        weekly_production)
       VALUES ('waiting_db_conn','waiting_sim_start',0,0);
```

Anexo B

Identificadores no *feedback* do *Digital Twin*

Nos testes realizados ao *Digital Twin*, verificou-se uma inconsistência entre o valor dos identificadores dos comandos enviados e o valor que era colocado na lista de comandos finalizados, *orders_status_out*.

Ao analisar o código dos *Function Blocks* do *Digital Twin*, foi identificada a origem do problema como estando na estrutura de dados (*equip_info*) que é enviada na comunicação entre *Function Blocks*.

Quando um comando é enviado, o seu identificador deve ser introduzido no *order.id* e no *order.part.order_id*. Durante a execução deste comando, a estrutura *order.part* (representativa da peça em questão) é transmitida pelos Function Blocks que a peça percorre. No entanto, o valor *order.id* é perdido neste processo..

Para facilitar a compreensão do problema, tome-se como exemplo, o envio de um comando para deslocar uma peça de um equipamento com identificador 5 para o seu adjacente, com o identificador 6. Quando a peça for enviada para o equipamento 6, a estrutura *equip_info* enviada pelo equipamento 5 terá, de entre vários parâmetros, o identificador do equipamento de origem (5) e uma estrutura de dados, *part* que contém as informações da peça em questão.

Com isto, conclui-se que a estrutura *equip_info* não contempla uma variável reservada para o identificador do comando que está, efetivamente, a ser executado. A única maneira de aceder a este valor é recorrendo à informação das propriedades da peça, *equip_info.part.order_id*.

Por fim, no momento em que é finalizado o percurso da peça, era transmitido para o *Orders Manager* o identificador presente na *equip_info*, que era, na verdade, o identificador do último equipamento pelo qual a peça passou, como sendo o identificador do comando executado.

Efetivamente, em algumas partes do código existente, é utilizado o identificador do comando presente em *equip_info.part.order_id*. No entanto, verificou-se que isto não foi implementado

para todas os cenários existentes, impossibilitando a obtenção de um *feedback* fidedigno. A correção deste problema é muito complexa, por estar diretamente relacionado com os mecanismos de funcionamento interno do *Digital Twin*, não sendo possível a sua realização no âmbito desta dissertação. Por este motivo, para a realização deste trabalho foi utilizada a alternativa descrita na secção 4.5.1.

Anexo C

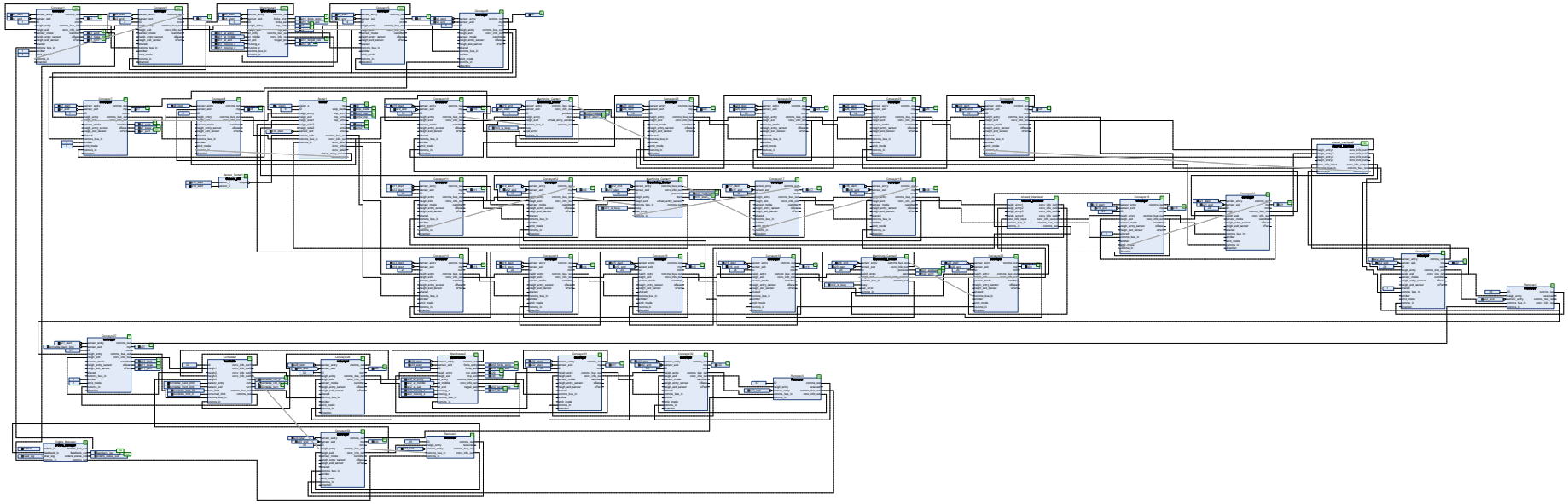
Repositórios de código

- Repositório GitHub da aplicação mini-ERP: github.com/pedrosilva2703/mini_erp
- Repositório GitHub da aplicação mini-MES: github.com/pedrosilva2703/mini_mes
- Ficheiro XML para o layout de teste: layout.xml

Anexo D

Montagem do layout de teste no Codesys

(Montagem na próxima página)



Referências

[1] Daisy Valle Enrique, Érico Marcon, Fernando Charrua-Santos, e Alejandro G. Frank. Industry 4.0 enabling manufacturing flexibility: technology contributions to individual resource and shop floor flexibility. *Journal of Manufacturing Technology Management*, 33:853–875, 7 2022. doi:10.1108/JMTM-08-2021-0312.

[2] Edwin Mauricio Martinez, Pedro Ponce, Israel Macias, e Arturo Molina. Automation pyramid as constructor for a complete digital twin, case study: A didactic manufacturing system. *Sensors*, 21, 7 2021. doi:10.3390/S21144656.

[3] ISA95, Enterprise-Control System Integration. https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa95. Acedido a: 2022-06-23.

[4] André Sousa. A Digital Twin for Education with FACTORY IO, 2022. Dissertação de Mestrado, Faculdade de Engenharia, Universidade do Porto, https://hdl.handle.net/10216/142420.

[5] Daniel Cortes, Jose Ramirez, Luis Villagomez, Rafael Batres, Virgilio Vasquez-Lopez, e Arturo Molina. Digital pyramid: An approach to relate industrial automation and digital twin concepts. *Proceedings - 2020 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2020*, 6 2020. doi:10.1109/ICE/ITMC49519.2020.9198643.

[6] Laurence C. Magalhães, Luciano C. Magalhães, Jhonatan B. Ramos, Luciano R. Moura, Renato E. N. de Moraes, João B. Gonçalves, Wilian H. Hisatugu, Marcelo T. Souza, Luis N. L. de Lacalle, e João C. E. Ferreira. Conceiving a digital twin for a flexible manufacturing system. *Applied Sciences (Switzerland)*, 12(19), 2022. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85139958384&doi=10.3390%2fapp12199864&partnerID=40&md5=c72c35b0f7f15672466c8218a2f4fcd3, doi:10.3390/app12199864.

[7] About - FACTORY I/O. https://docs.factoryio.com/. Acedido a: 2022-12-28.

[8] FACTORY I/O. https://factoryio.com/. Acedido a: 2022-12-28.

[9] Codesys. https://www.codesys.com/. Acedido a: 2022-12-28.

[10] *Programmable Logic Controllers: A Practical Approach to IEC 61131-3 using CODESYS*. John Wiley Sons, Ltd, 2015. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118949214, doi:https://doi.org/10.1002/9781118949214.

[11] OPC UA. https://opcfoundation.org/about/opc-technologies/opc-ua/. Acedido a: 2022-06-23.

[12] Dijkstra Algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. Acedido a: 2022-05-01.

[13] Java. https://www.java.com/pt-BR/. Acedido a: 2022-02-13.

[14] JavaFX Scene Builder. https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html. Acedido a: 2022-02-13.

[15] PostgreSQL. https://www.postgresql.org/. Acedido a: 2022-02-13.

[16] JDBC. https://jdbc.postgresql.org/. Acedido a: 2022-02-13.

[17] Kenneth C. Laudon e Jane Price Laudon. *Management Information Systems: Managing the Digital Firm*. Pearson, Boston, 13 edição, 2013.

[18] Marianne Bradford. *Modern ERP: Select, Implement, and Use Today's Advanced Business Systems*. Lulu Enterprises, London, 3 edição, 2014.

[19] System Applications and Products – What is a manufacturing execution system (MES)? https://www.sap.com/products/scm/execution-mes/what-is-mes-manufacturing-execution-system.html. Acedido a: 2022-12-28.

[20] H. Meyer, F. Fuchs, e K. Thiel. *Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment*. McGraw Hill LLC, 2009. URL: https://books.google.pt/books?id=M4hK1c9VibQC.

[21] Teaching virtual manufacturing with DELMIA. https://www.3ds.com/edu/education/academics/solutions/3dexperience/delmia-package. Acedido a: 2022-12-28.

[22] The Application of Simio Scheduling in Industry 4.0. https://www.simio.com/resources/papers/WinterSim2019/The-Application-of-Simio-Scheduling-in-Industry-4.0.php. Acedido a: 2022-12-28.

[23] *Learn Objective-C for Java Developers*. Apress, Berkeley, CA, 2009. URL: https://doi.org/10.1007/978-1-4302-2370-2_20.

[24] Eclipse Milo. https://github.com/eclipse/milo. Acedido a: 2022-04-24.