Faculdade de Engenharia da Universidade do Porto

Informática Industrial

Automação de uma Linha de Produção

Diogo Oliveira - up201706129@edu.fe.up.pt
João Pinheiro - up201706388@edu.fe.up.pt
Lorena Fernández - up202009944@edu.fe.up.pt
Nuno Pacheco - up201705462@edu.fe.up.pt

21 de junho de 2021

1 Introdução

Este relatório discorre sobre o desenvolvimento de uma solução de automatização de um simulador de uma linha de produção flexível e de uma plataforma de controlo e monitorização desse sistema (i.e. MES, Manufactoring Execution System).

O controlo do SFS, Shop Floor Simulator, foi realizado por um PLC, Programmable Logic Controller, (neste caso o "Codesys Control Win V3 - x64"), programado com as linguagens ST (Structured Text), SFC (Sequential Flow Chart) e CFC (Continuous Flow Chart).

Para o MES, foi desenvolvido um programa utilizando uma linguagem de programação orientada a objetos, Java.

De forma a que a informação gerida pelo MES fosse persistente, isto é, não se perdesse quando o MES fosse desligado, foi ainda utilizada uma base de dados em paralelo com o mesmo, implementada através do sistema de gestão de base de dados MariaDB.

Ainda relativamente ao MES, é de salientar o desenvolvimento de uma interface gráfica para o mesmo que é atualizada em tempo real, ou seja, que está permanentemente a ser atualizada e não depende de uma ação do utilizador para despoletar, por exemplo, um pedido de informação à base de dados.

Por último, embora não seja um requisito do trabalho, uma vez que o MES deverá receber e enviar ficheiros XML de e para um ERP (Enterprise Resource Planning), foi ainda desenvolvido um outro programa com interface gráfica que funcionasse como ERP de modo a facilitar a criação, envio e receção desses ficheiros XML.

A comunicação entre o SFS e o PLC foi realizada através de Modbus/TCP, uma vez que o SFS implementa um servidor desse mesmo protocolo. A comunicação entre o PLC e o MES foi realizada através de OPC-UA. A comuninação entre o MES e o ERP foi realizada através do envio e receção de ficheiros XML sobre UDP.

2 Estrutura do Código

2.1 PLC - Programmable Logic Controller

O programa desenvolvido para o PLC pode ser representado pelo seguinte diagrama de classes:

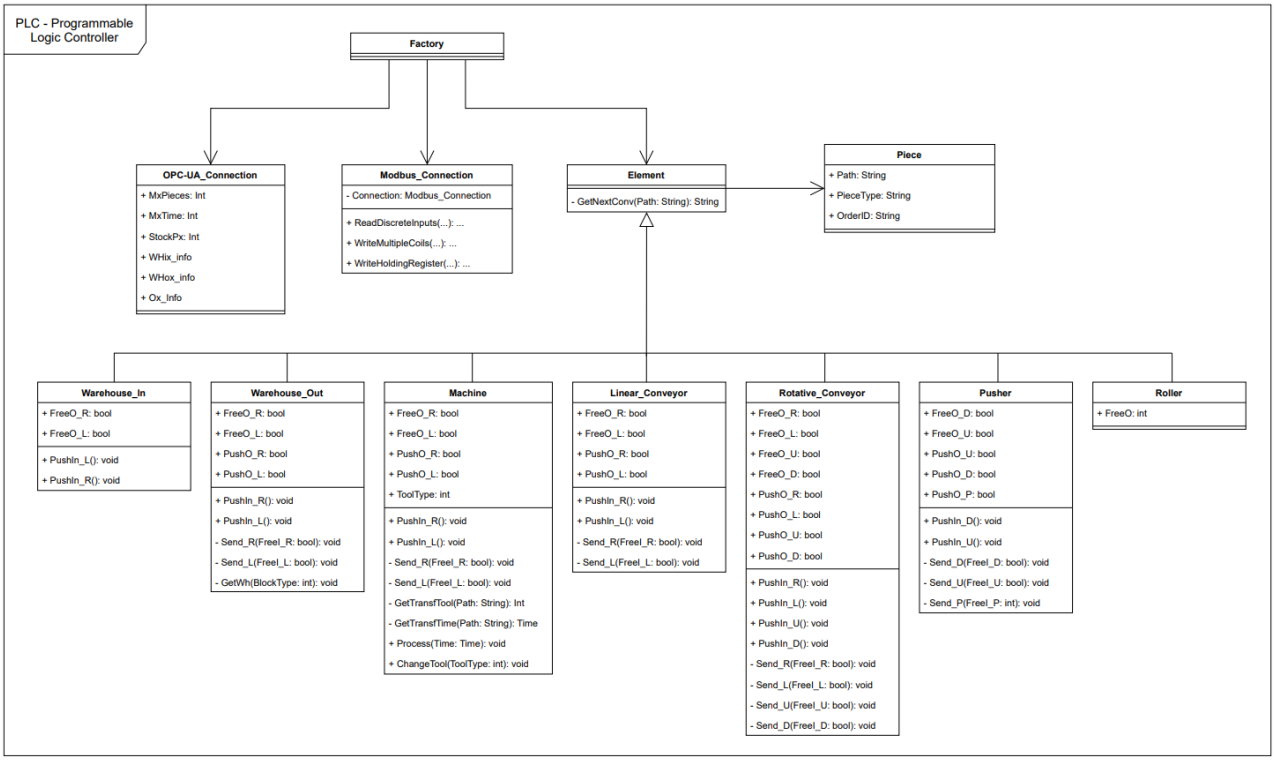

Figura 1: Diagrama de Classes - PLC

As classes “OPC-UA_Connection” e “Modbus_Connection” representam os canais de comunicação $MES \leftrightarrow PLC$ e $PLC \leftrightarrow SFS$, respetivamente.

As classes que extendem a classe “Element” referem-se aos vários tipos de elementos que existem fisicamente na fábrica (ou, neste caso, no SFS).

É de notar que, embora nem todos os elementos do SFS necessitassem de se movimentar em todas as direções e sentidos possíveis, para cada um dos tipos, foi implementado uma única classe com todas as funcionalidades possíveis. Por exemplo, um tapete rotativo que apenas receba peças de “Baixo/Down” e que apenas as envie para a “Direita/Right” tem também a possibilidade de receber e enviar para todas as outras direções. A opção por esta estratégia de desenvolvimento resultou numa maior versatilidade dos blocos e do próprio programa desenvolvido, uma vez que permite realizar facilmente testes mais complexos ao sistema e uma reconfiguração do layout da fábrica sem a necessidade de alteração do código existente.

Por último, a classe “Piece” representa a informação que acompanhará uma peça física na fábrica e que irá circulando pelos atributos dos elementos através das ligações entre estes.

2.2 MES - Manufactoring Execution System

Tendo sido o programa do MES desenvolvido em Java, uma linguagem de programação orientada a objetos, a estutura do código traduz-se facilmente no seguinte diagrama de classes:

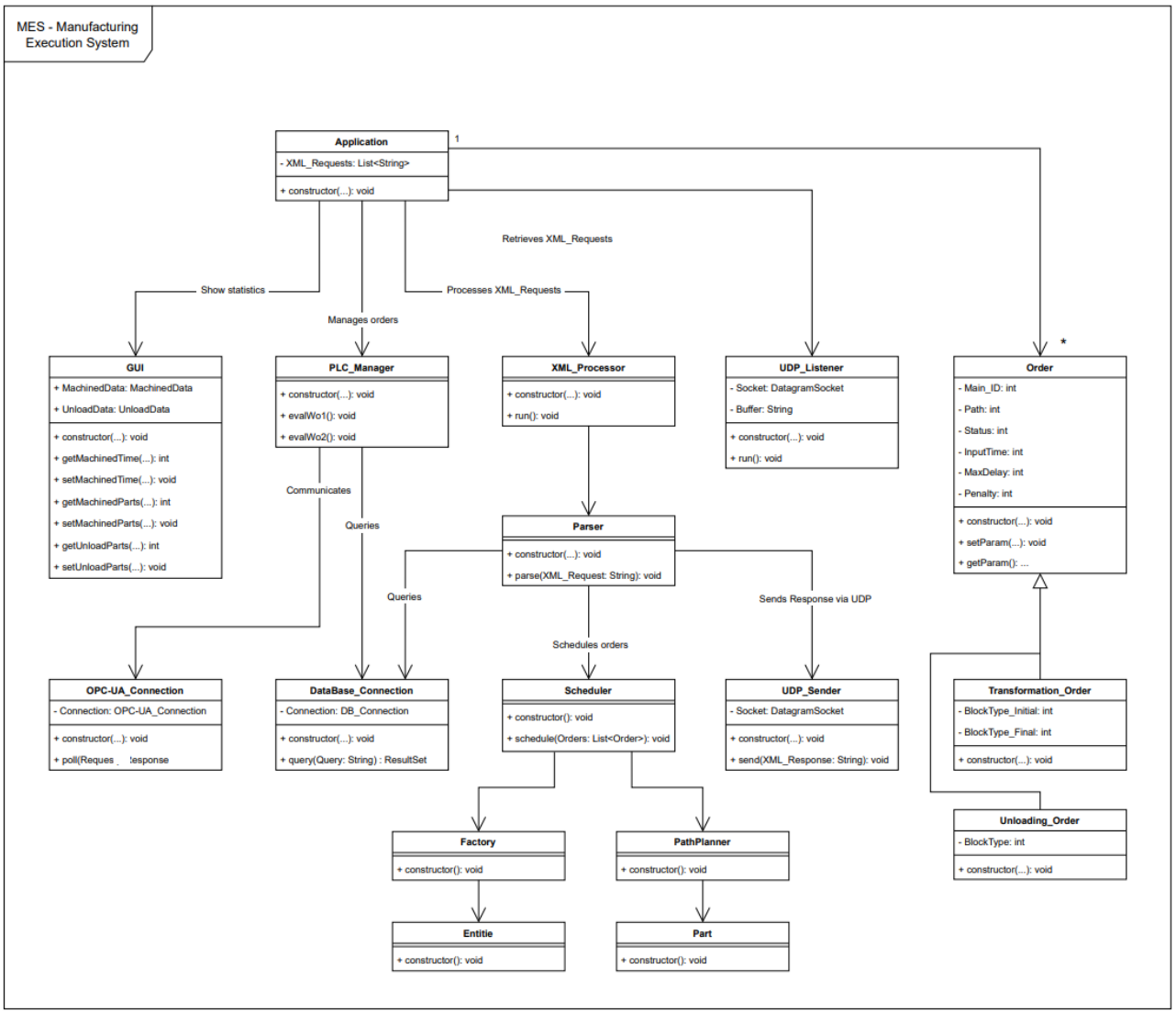

Figura 2: Diagrama de Classes - MES

A classe principal, “Application”, onde está presente a função main(), instancia os objetos das classes de dados (XML_Requests, Transformation_Order e Unloading_Order) e das classes que serão executadas como thread (GUI, UDP_Listener, XML_Processor e PLC_Manager).

Uma vez que em Java todos os objetos passados como parâmetros de métodos são passados por referência e não por valor, as diferentes instâncias das listas dos XML_Requests e Orders nas várias classes apontam, na realidade, para a mesma zona de memória.

Por outro lado, uma vez que existe a necessidade de chamar métodos do mesmo objeto da classe PLC_Manager em várias classes do programa, optou-se por criar um Singleton dessa mesma classe, o que eliminou a necessidade de passar este objeto como parâmetro em cascata através de vários métodos e classes.

A classe “Scheduler” é responsável por organizar as ordens recebidas de forma a minimizar a penalização incorrida após fim do delay máximo de cada ordem. É ainda responsável por atribuir um

caminho na fábrica às ordens de transformação.

A classe “Scheduler” utiliza as classes “Factory” e “PathPlanner” para obter o melhor caminho dentro da fábrica. Dada a natureza do problema de otimização de caminhos quer a “Factory”, quer o “PathPlanner” são gestores de caminhos em grafos, i.e. contêm em si um grafo e são responsáveis por retornar um caminho segundo condições pedidas pelo “Scheduler”.

Para ser possível maior detalhe, encapsulamento e polimorfismo desta secção do MES, o grafo da “Factory” é constituido por ”Entities”, classe estendida depois pelas diferentes máquinas/mecanismos da fábrica (“Linear” Roller, “Rotative” Roller, “Machine”, “Warehouse IN/OUT”, etc). As arestas deste grafo são unidirecionais e permitem definir os caminhos possíveis entre as “Entities”. Em caminhos bidirecionais (e.g. Tapete $\leftrightarrow$ Máquina) existem arestas paralelas no grafo.

O grafo da classe “PathPlanner” é constituído por nós representativos do tipo de peça utilizado (“Part”). As arestas deste grafo são novamente unidirecioniais mas contêm informação sobre a relação entre as peças, representada pela classe “PathEdge”. A classe “PathEdge” armazena a informação do tempo de processamento e da ferramenta necessária para transformar um nó do grafo, no nó seguinte.

Para otimizar o processamento das peças o “Scheduler” contem ainda um calendário de processamento de cada máquina onde são agendadas as passagens de cada peça em cada máquina.

2.3 DB - Database

De modo a tornar o sistema persistente, isto é, a não perder a informação relativa a ordens e estatísticas de maquinação e descarga quando o MES fosse desligado, foi utilizada uma base de dados.

As tabelas criadas nesta base de dados seguem a seguinte arquitetura:

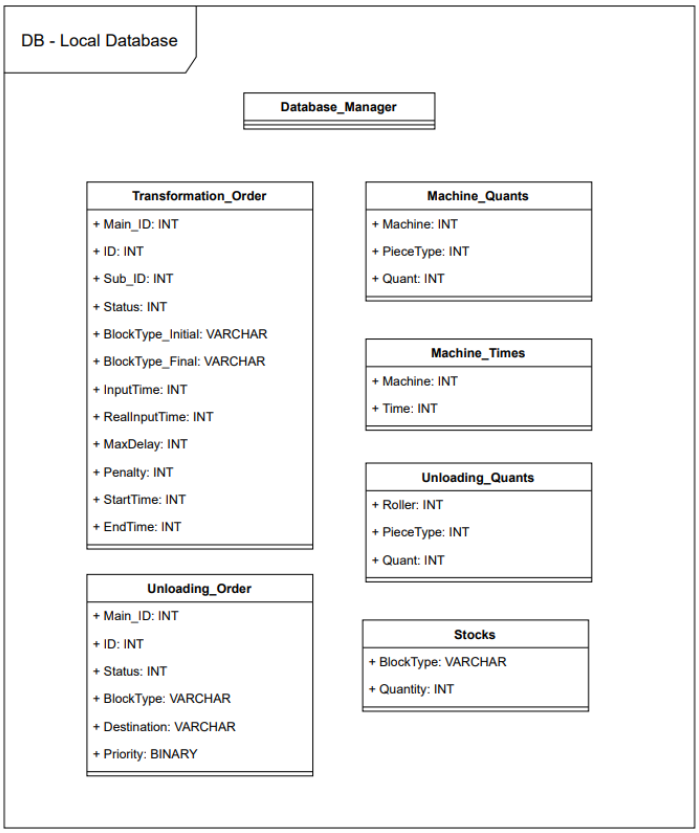

Figura 3: Diagrama de Classes - DB

3 Funcionamento do Código

3.1 PLC - Programmable Logic Controller

No que toca ao funcionamento do PLC, neste estão instanciados os diferentes componentes que compõem a fábrica (Máquinas, Tapetes, etc...).

Aquilo que seria uma interação normal entre os diferentes elementos do PLC pode ser observado através da figura 4 onde se encontram instanciados 3 elementos da fábrica.

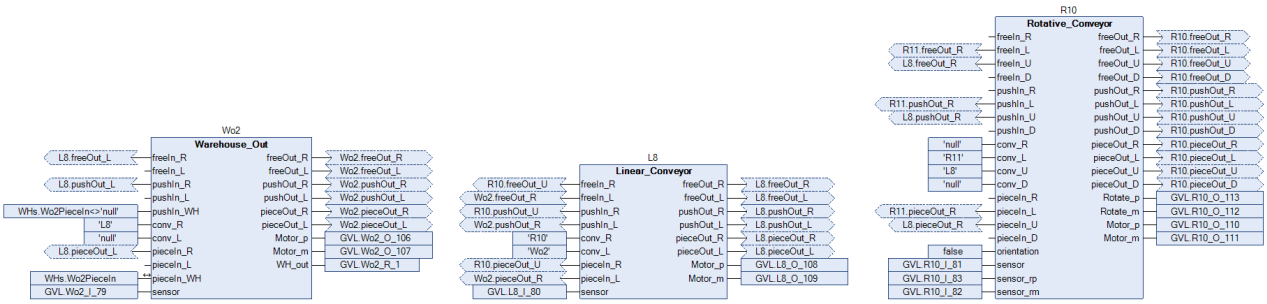

Figura 4: Exemplo de ligação entre 3 tapetes

Assumindo que o $Wo2$ se encontra carregado com uma peça que pretende enviar para $L8$, este deve então acionar a sua saída "PushOut_R" (que está ligada à entrada "PushIn_L" do tapete $L8$). Caso o tapete $L8$ esteja carregado com uma peça nada acontecerá. Porém, logo que o tapete $L8$ esteja livre, detetando que a sua entrada "PushIn_L" está ativa, este aciona a sua saída "FreeOut_L" (conectada a "freeIn_R" de $Wo2$), sinalizando que está pronto a receber peças e iniciando assim a transação.

A passagem de cada peça pelos diferentes tapetes é acompanhada pela passagem de uma string contendo o itinerário da peça. Um exemplo da referida string poderá ser:

```
L8:R10:R11:R12:R13:M6:M#T1|15:R13:R14:R15:R16:Wi2:Wh:?P=1?ID=101_13_1
```

Esta string será enviada ao tapete $Wo2$ pelo MES, via OPC-UA, originando a saída de uma peça do armazém e indicando que o próximo tapete por onde a peça irá passar será o tapete $L8$. O tapete $Wo2$ reconhece o tapete $L8$ como sendo o tapete à sua direita (através da entrada "conv_R") e remove "L8:" da string antes de a passar ao próximo tapete. Sendo assim, cada tapete a montante recebe uma versão truncada da string, contendo sempre em primeiro lugar o próximo tapete a que a peça se deve encaminhar.

Para além do itinerário que a peça deve percorrer, a string é ainda composta pelas maquinações que deve fazer (e.g. "M#T1|15": 15s com ferramenta 1), pelo tipo de peça atual (e.g. "P=1"), que será modificado cada vez que é realizada uma maquinação, e pelo ID da ordem a que a peça se refere (e.g. "ID=101_13_1").

3.2 MES - Manufactoring Execution System

Relativamente ao MES, a sua função “Main” é responsável por inicializar as variáveis locais (ordens de transformação e de descarga não concluídas e estatísticas) a partir da informação guardada na base de dados e por criar e iniciar as classes/threads principais, tal como explicita o seguinte diagrama de sequência:

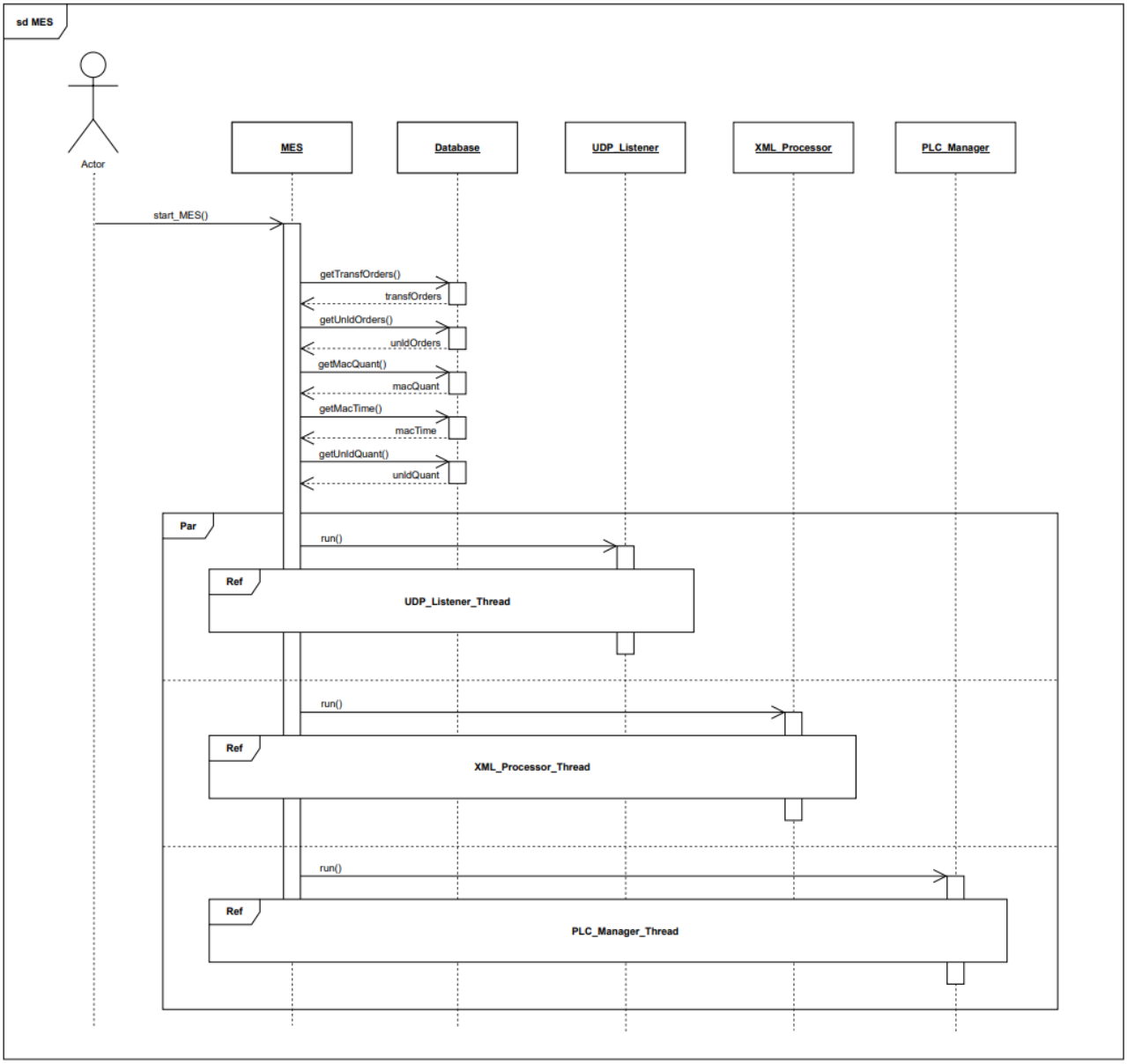

Figura 5: Diagrama de Sequência - MES

Os diagramas de sequência seguintes detalham o funcionamento das classes “UDP_Listener” e “XML_Processor” representadas na figura anterior.

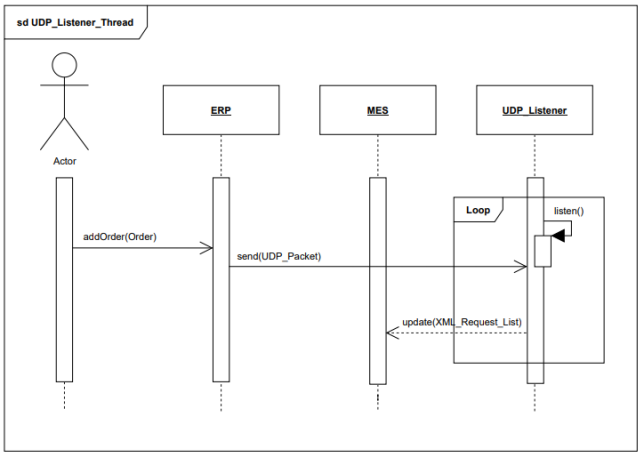

Figura 6: Diagrama de Sequência - UDP_Listener

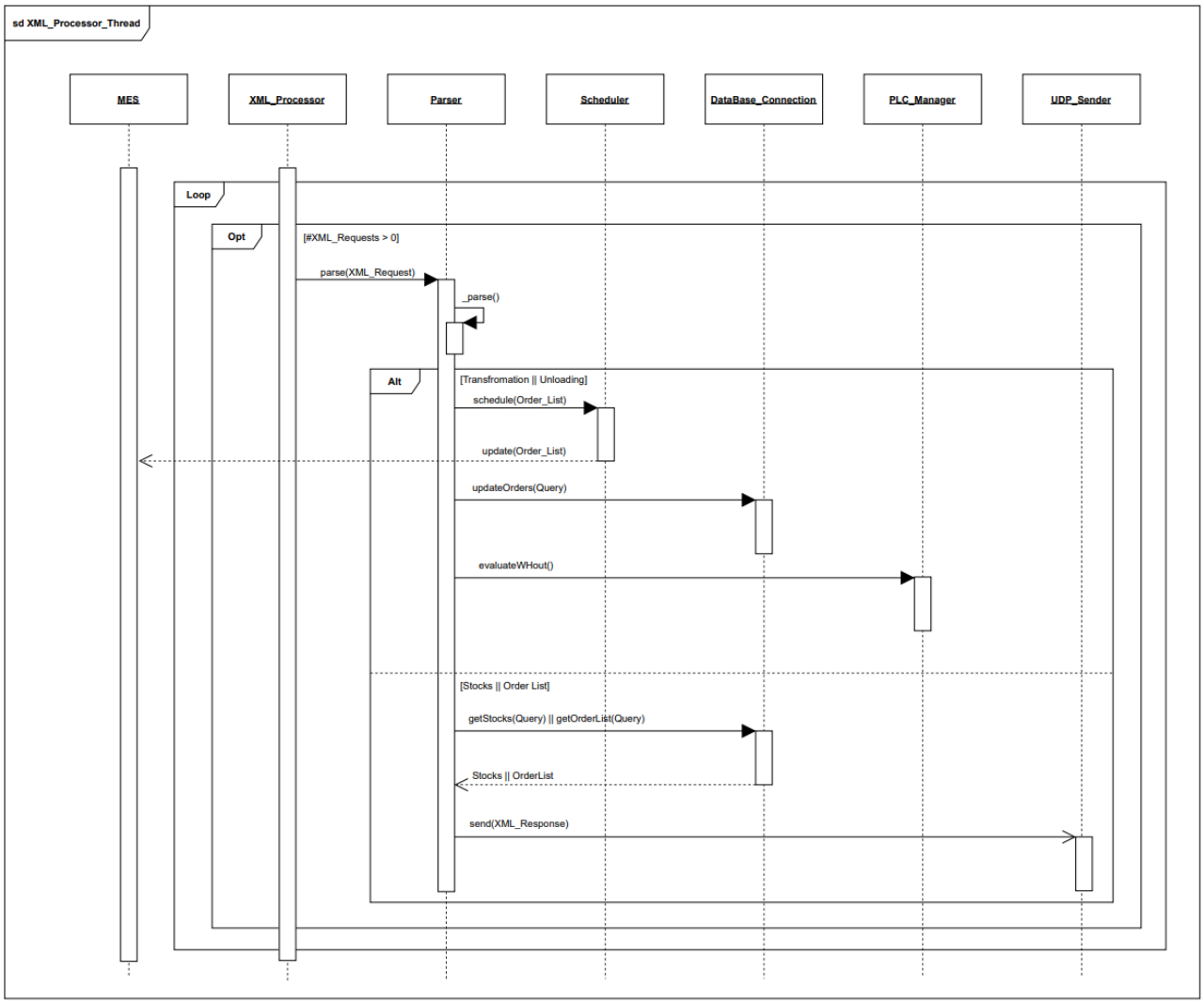

Figura 7: Diagrama de Sequência - XML_Processor

Relativamente à classe “PLC_Manager”, esta tem por base um sistema de subscrições de variáveis do PLC por OPC-UA. Aquando da construção do objeto, são configuradas estas subscrições e adicionados *Listeners* a cada uma delas. Os *Listeners* correspondem a invocações das funções “evaluateWHout()” (que verifica se pode ser enviada nova ordem para o PLC), “updateStocks()” (que atualiza os stocks na base de dados) e “incMacQuant()”, “incMacTime()” e “incUnldQuant” (que incrementam as estatísticas na interface gráfica e na base de dados).

Apresenta-se de seguida o funcionamento da função “evaluateWHout()”:

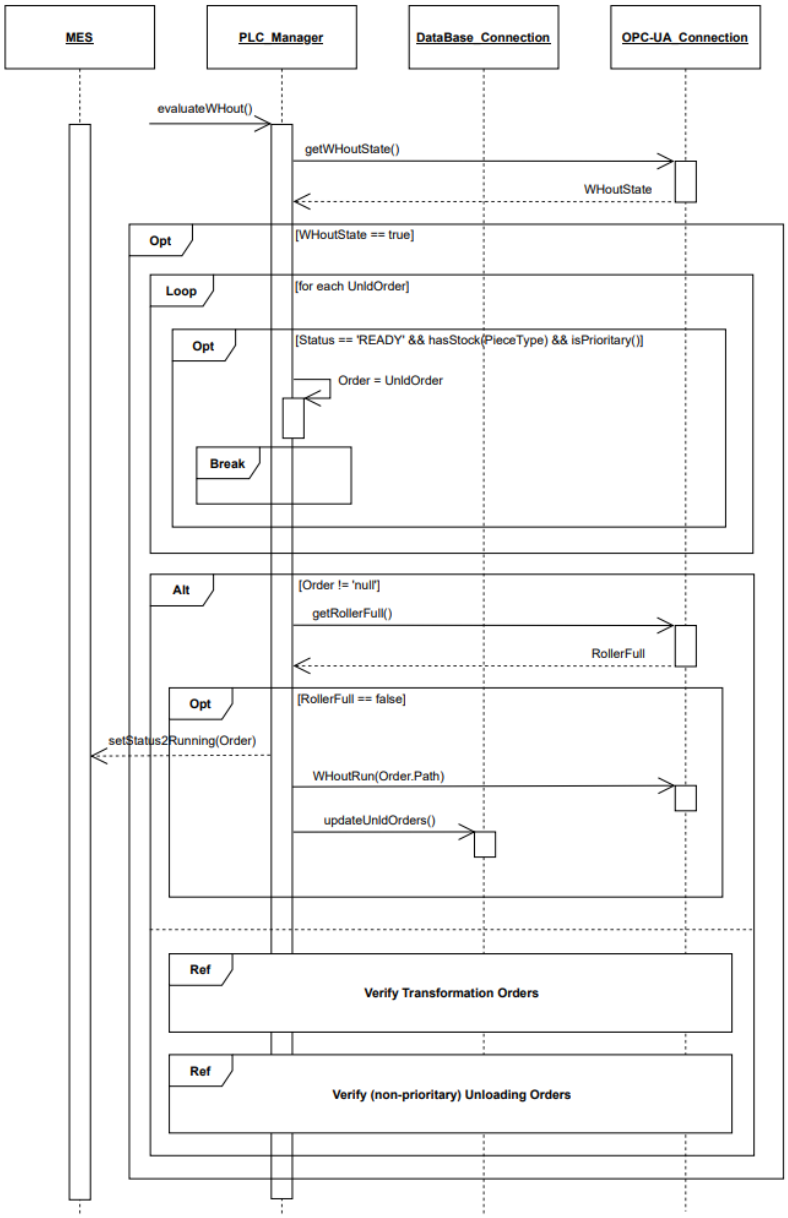

Figura 8: Diagrama de Sequência - PLC_Manager (Evaluate WHout)

Neste diagrama de sequência é apresentado apenas o algoritmo para a avaliação das ordens de descarga prioritárias, uma vez que para as ordens de transformação e de descarga não prioritária será semelhante.

Nas secções seguintes detalha-se o funcionamento do Scheduler, PathPlanner e Factory, invocados no XML_Parser.

3.2.1 Scheduler

O scheduler implementa o processo de escalonamento das ordens às máquinas. O processo é definido segundo o seguinte pseudocódigo:

Para cada ordem:

- Gerar todas as combinações de transformaçãoes em máquinas possíveis;
- Calcular os tempos de ínicio e fim da ordem para cada combinação gerada;
- Escolher a opção que provoque o menor tempo de finalização da ordem para cada um dos lados;
- Escolher o lado que provoque o menor tempo de finalização e em caso de empate, o menor número de troca de ferramentas;
- Guardar a melhor solução atual e repetir recursivamente até não haver ordens restantes.

O algoritmo apresentado é uma variação do algoritmo branch-and-bound, procurando combinatorialmente e de forma local a melhor solução para cada ordem, no momento. Após decidir a melhor solução para a ordem localmente, fixa-a, iterando para a ordem seguinte. Este comportamento, apesar de não ideal foi escolhido devido à complexidade do problema que se tornaria inescalável devido ao comportamento combinatorial de um problema do tipo flow-shop.

3.2.2 PathPlanner e Factory

Ambas as classes são responsáveis por retornar o caminho mínimo entre dois nodos do grafo que contêm. Para isto foi utilizado o algoritmo de Dijkstra para encontrar o caminho com menos arestas possíveis (transformações ou deslocamento entre tapetes, respetivamente).

Para promover a velocidade de operação do programa utilizou-se o método da indexação de todos os caminhos mínimos entre qualquer dois nós do grafo. Desta forma consegue-se reduzir a complexidade das operações de **O(Arestas + Vértices log(Vértices))** para uma complexidade de **O(1)** para obter o caminho indexado.

4 Implementação

Tanto no PLC como no MES, a implementação seguiu a arquitetura definida e apresentada nas secções anteriores.

4.1 PLC - Programmable Logic Controller

O programa do PLC foi implementado usando as liguagens CFC, SFC, e ST para criar 10 "Program Organization Units" (POUs):

- 2 programas (PRG): Factory_Left, Factory_Right (CFC);
- 8 function blocks (FB): Linear_Conveyor, Machine, Pusher, Roller, Rotative_Conveyor, Warehouse_In, Warehouse_Out e DeadlockAvoid (SFC+ST).

As POUs Factory_Left e Factory_Right são os dois programas principais onde se instanciou a parte esquerda e a parte direita da fábrica, respetivamente. Para instanciar os objetos Linear_Conveyor, Machine, Pusher, Roller, Rotative_Conveyor, Warehouse_In e Warehouse_Out em cada um dos lados da fábrica, posicionou-se os mesmos na mesma disposição com que aparecem no Shop Floor Simulator.

É de notar a existência do FB DeadlockAvoid que gere a passagem de peças pelos tapetes de modo a evitar situações em que o tapete que está imediatamente antes de cada máquina seja carregado com uma peça que de seguida irá para a máquina, quando nessa mesma máquina já está uma peça que necessita de ser enviada para esse mesmo tapete no fim da maquinação. Não ter em consideração este pormenor, levaria a situações de Deadlock indesejadas e comprometeriam o correto funcionamento do sistema.

Relativamente à comunicação entre o SFS e o PLC, esta é realizada através do protocolo Modbus TCP, configurado com 3 canais de comunicação com as seguintes funções: "Read Discrete Inputs", "Write Multiple Coins" e "Write Multiple Registers" (com trigger cíclico de período 100ms). Por sua vez, a comunicação entre o PLC e o MES foi realizada através do protocolo OPC-UA através da qual é possível partilhar com o MES informações como o estado dos Warehouse_In/Out, dos Rollers, estatísticas referentes às máquinas, estado dos stocks e das ordens enviadas para o PLC. Alguns destes são implementados utilizando o modelo Publisher-Subscriber. No entanto, é também possível o MES fazer pedidos de leitura/escrita "on-demand" de certas variáveis.

Os programas Factory_Left e Factory_Right executam na "MainTask", sendo esta cíclica com um período de $15ms$.

4.2 MES - Manufactoring Execution System

O MES foi implementado utilizando a linguagem de programação orientada a objetos JAVA e utiliza uma arquitetura baseada em quatro threads que correm sobre a Java Virtual Machine: "Receção e resposta UDP (assíncrona)", "Receção, processamento, gestão, agendamento e armazenamento de ordens","Gestão do Chão de Fábrica com comuniação OPC-UA" e "Interface Gráfica".

A thread de "Receção UDP (assíncrona)" é responsável por verificar a chegada de novas ordens XML através da porta UDP.

A thread de "Receção, processamento, gestão, agendamento e armazenamento de ordens" é responsável por processar as ordens recebidas por UDP, interpretá-las, respondendo via UDP ou agendá-las através do uso do Scheduler e garantir o seu armazenamento na base de dados.

A thread de "Gestão do Chão de Fábrica com comunicação OPC-UA" é responsável por receber os updates fornecidos pela plataforma OPC-UA através do método de subscrição e despoletar o envio de novas ordens para o chão de fábrica através do PLC (e.g. Receiving: Tapete de saída do WareHouse livre $\rightarrow$ Próxima ordem agendada pode iniciar).

A thread de “Interface Gráfica” é responsável por fornecer ao sistema uma interface Humano-Máquina sendo capaz de apresentar em tempo real estatísticas relevantes para efeitos de verificação e manutenção do sistema (e.g. Tempo ativo de cada máquina, Quantidade de peças processadas, etc).

4.2.1 GUI - Graphical User Interface

A interface gráfica do MES foi implementada com o propósito se acompanhar em tempo real as peças maquinadas e descarregadas, sendo que esta informação se encontra discriminada por tipo de Máquina/Tapete. É ainda possível saber o tempo de trabalho de cada máquina (soma dos tempos das maquinações que cada uma realizou).

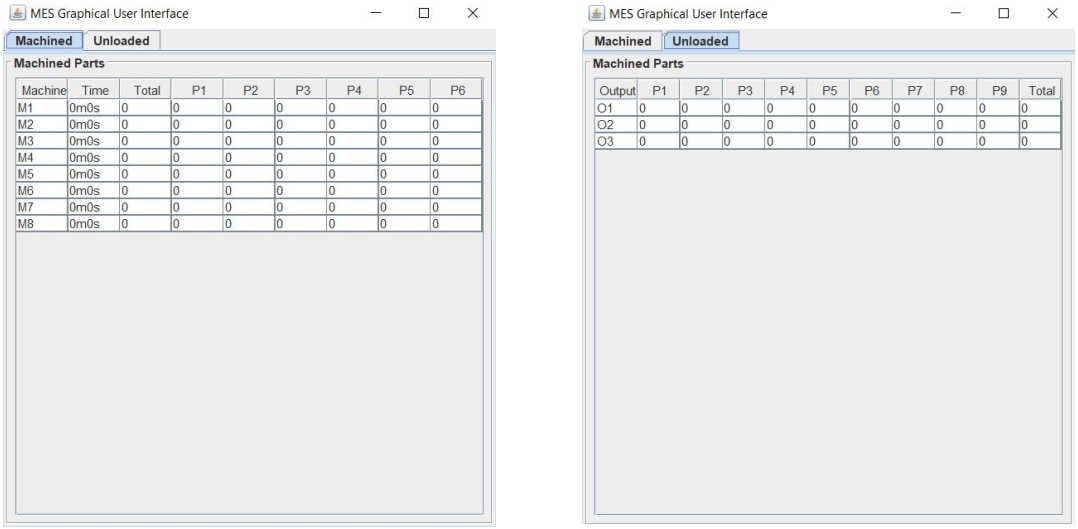

Figura 9: Interface gráfica MES

Para implementar esta GUI foi usada uma framework de design gráfico, comummente designada por *Java Swing*. Esta biblioteca contém componentes gráficos predefinidos como botões, tabelas e agregadores de componentes, facilitando o rápido desenvolvimento de uma interface gráfica.

4.3 DB - Database

Para implementação da Base de Dados decidiu-se utilizar MariaDB.

O sistema de gestão de base de dados MariaDB foi escolhido devido à sua boa reputação enquanto SGBD open source, sendo leve, robusto e estável.

Para efeitos de teste, utilizou-se o cliente de base de dados HeidiSQL.

5 Estrutura Final vs Inicial

A arquitetura final difere da arquitetura inicial principalmente em 2 aspetos:

- Gestão de caminhos de peças no PLC;
- Gestão de ordens no MES e integração com o PLC;

5.1 Gestão de caminhos de peças no PLC

Na arquitetura inicial havia a necessidade de existirem instanciados e pré-definidos todos os caminhos possíveis. Nesta solução, o scheduler do MES decidiria qual o melhor caminho a seguir e ativava o caminho no PLC (transição fonte) atráves de uma variável booleana. No lado do PLC, este caminho era constituído por uma sequência de estados e transições pré-definida que ativavam cada um dos tapetes por onde a peça deveria passar.

Na arquitetura final, optou-se por não utilizar esta solução uma vez que existia a necessidade de prever à partida todos os caminhos possíveis e não era uma solução que fosse facilmente expandida. Ao invés, optou-se por eliminar o conceito de caminho e passou a existir o conceito de peça que percorre os vários tapetes. Sendo esta peça uma string que contém o caminho (formado pelos nomes dos tapetes que tem de percorrer), o seu tipo de peça atual e o ID da ordem que lhe deu origem.

5.2 Gestão de ordens no MES e integração com o PLC

Quanto à gestão de ordens no MES e respetiva integração com o PLC, estavam previstas duas classes, a Order_Manager e a Order_Executer. A classe Order_Manager seria responsável por verificar se existiam recursos disponíveis no PLC para enviar a próxima ordem e, em caso afirmativo, colocava a ordem no estado READY. A classe Order_Executer verificava se havia ordens no estado READY e, em caso afirmativo, enviava para o PLC. Ambas estas classes funcionariam como thread e executavam um ciclo infinito

No entanto, após o começo do desenvolvimento do scheduler e aprendizagem sobre a tecnologia OPC-UA, percebeu-se que a solução necessária era bastante mais simples e só havia a necessidade de utilizar subscrições de diversos objetos do PLC, por exemplo, do estado dos tapetes WarehouseOut, WarehouseIn e Rollers, e, pelo contrário, não seria necessário implmentar um ciclo infinito. Tendo o scheduler ordenado a lista de ordens por prioridade, assim que o WarehouseOut esteja livre é lida a próxima ordem dessa lista e enviada para o PLC.

6 Comparação com Padrões Existentes

6.1 ISA 95 - High Level Hierarchy

O standard ISA 95 define 5 níveis hierárquicos cooperativos responsáveis pelo planeamento, gestão e execução das operações relativas à produção de bens. O projeto apresentado integra-se no standard ISA 95 com a seguinte relação a cada um dos níveis.

Level 4: Business Logistics: A interface gráfica ERP GUI permite gerar ordens de transformação e ordens de descarga consoante as encomendas feitas à fábrica. Segundo o guião proposto esta operação também poderia ser feita via netcat com ficheiros XML pré-programados.

Level 3: Manufacturing Operations Management: O MES aqui implementado recebe do nível superior ordens XML geradas pelo ERP via UDP e é capaz de as organizar, escalonar e definir detalhadamente o seu método de produção (caminhos, transformações, etc). É também responsável por enviar via OPC-UA ordens detalhadas para o nivel 2 e acompanhar a evolução destas ordens. Através destes processos é ainda capaz de gerar respostas com estatísticas e dados sobre o processo a ser realizado.

Level 2 + 1: Discrete Production Control: O controlo direto da produção é realizado por um PLC, independente, responsável por converter as ordens detalhadas do MES, recebidas por OPC-UA, em transações MODBUS contendo a informação necessária para a operação das máquinas e tapetes, sob fim de correta execução das ordens.

Level 0: Production Process: Chão de fábrica emulado pelo simulador SFS.

Focando na implementação do nível 3:

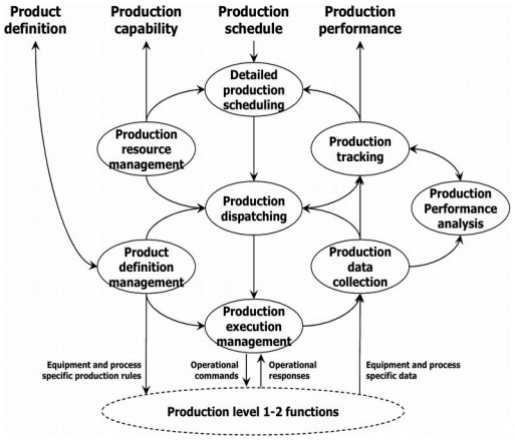

Figura 10: Production Operations Management Activity Model

Todas as funções definidas na figura 10 são implementadas pelo MES do projeto apresentado, à exceção do ponto *Production Performance analysis*. A *Product definition* foi realizada *a priori* através da matriz de transformações fornecida, e é imutável ao longo da vida do projeto. A *Production capability* é comunicada ao ERP através da ordem de *Request Stores*. O *Production Schedule* é gerado internamente pelo MES, mas é influenciado pelo delay máximo e pela ordem com que são entregues as ordens. A *Production performance* pode ser medida através do GUI do MES ou através das informações apresentadas através da ordem *Request Orders*.

6.2 Service Granularity in Industrial Automation and Control Systems

Na implementação relativa ao nível de *Discrete Production Control* utilizou-se o conceito de *Machine Granularity*. Todas as entidades relativas ao chão de fábrica são mapeadas para o PLC através

de Function Blocks. Desta forma é possível garantir um nível de abstração sobre o hardware. Deste modo, um tapete e uma máquina podem ser ambos encapsulados sob este conceito apenas sendo diferenciados pelas capacidades de cada um (e.g. uma máquina é capaz de: mover a peça e maquinar e um tapete é capaz de apenas mover a peça).

Esta arquitetura permite que os caminhos planeados pelo MES sejam definidos como conjuntos ordenados de "máquinas" pelas quais as peças tem de passar, podendo definir operações de maquinação necessárias nas máquinas que as permitam.

Esta granularidade também permite que, em caso de reorganização/remodelação da fábrica, seja fácil atualizar o software para o novo modelo. Apenas seria necessário ligar os novos blocos e atualizar as ligações entre os existentes. Do lado do MES apenas seria necessário atualizar os nós no grafo da "Factory" e facilmente teriamos todo o sistema *online* de novo.

6.3 RAMI 4.0 - Reference Architectural Model Industrie 4.0

Se se observar de um ponto de vista abstrato o funcionamento do MES, pode-se assumir que este funciona como um serviço: recebe como entrada um pedido e diligencia de forma a cumprir este pedido. Assim, o MES pode ser encapsulado como um prestador de serviços ao ERP. Ao realizar este tipo de abstração pode-se encontrar uma semelhança entre o encapsulamento de todos os níveis inferiores à camada de FUNCTIONS da arquitetura RAMI e o sistema MES + PLC implementado.

Ainda assim, não é possível definir uma cadeia de prestação de serviços clara e concisa entre todos os níveis implementados pelo MES e PLC. Para isso seria necessário definir estruturas de dados e interfaces entre os sistemas que os tornassem iguais de um ponto de vista externo. Contudo, pode-se considerar a existência deste isolamento se se assumir o PLC como implementador de um Digital Twin do chão de fábrica. O PLC recebe ordens claras e concisas do MES e, tal como acima, diligencia para as cumprir.

Entre o PLC e o MES (sentido inverso ao parágrafo anterior) esta barreira também não é clara pelo que seria necessário implementar uma camada de COMMUNICATION melhor definida, abstraindo esta ligação das implementações do PLC e do MES, unindo numa só implementação toda a gestão relacionada com estas duas unidades (ADMINISTRATION SHELL).

Em suma, é possível encontrar traços da arquitetura RAMI 4.0 nesta implementação, mas não é possível definir este trabalho como enquadrado nesta.

7 Organização da Equipa

De forma a acompanhar e distribuir o trabalho internamente na equipa foi utilizada uma mecânica baseada na criação de Issues na plataforma GitHub. De forma a iniciar o desenvolvimento começou-se por definir uma linha comum de pensamento através da definição da arquitetura base do software utilizando para isso diagramas de classes, objetos e sequência. Estes diagramas iniciais foram objeto de entrega para avaliação provisória pela equipa docente a 18/mar/2021.

Após esta entrega foram definidos conjuntos de Issues relativos ao trabalho:

- Gestão do chão de fábrica (PLC);
- Comunicações, Gestão de Ordens e Persistência;
- Algoritmo de Escalonamento e Caminhos;
- Interface Gráfica.

Cada um destes conjuntos continha múltiplos Issues mais detalhados de forma a isolar concretamente unidades de trabalho facilmente atribuíveis a cada um dos intervenientes no trabalho.

A alocação do trabalho foi feita à base da auto-atribuição. Cada Issue podia ser alocado a uma ou mais pessoas que quisessem colaborar nesse mesmo Issue.

Desta forma quando um desenvolvedor terminasse um Issue poderia começar num novo Issue. Semanalmente, a equipa reunia no mínimo uma vez para fazer um ponto de situação e fazer um *brainstorming* sobre ideias a implementar e problemas a resolver. Desta forma, definiam-se objetivos semanais a obter, numa ótica equiparável a um *sprint* semanal, segundo a ideologia de desenvolvimento *SCRUM*.

Após reunião final e por unanimidade, pôde-se designar a afetação ao trabalho como igual entre os colegas Diogo Oliveira, João Pinheiro e Nuno Pacheco, tendo-se a lamentar um não maior envolvimento por parte da colega Lorena Fernández. Desta forma alocaram-se as seguintes frações de participação:

- Diogo Oliveira: $\frac{1}{3}$
- João Pinheiro: $\frac{1}{3}$
- Lorena Fernández: 0
- Nuno Pacheco: $\frac{1}{3}$

8 Testes e Resultados

Uma vez que foi desenvolvido um programa que emula o funcionamento do ERP, o envio e receção de pedidos XML tornou-se bastante ágil.

Assim, ao longo do desenvolvimento do MES e do PLC, estes puderam ser testados frequentemente com diferentes tipos e sequência de ordens.

Dos testes realizados, são de destacar os seguintes:

- Envio de ordens de transformação com até 4 subtransformações;
- Envio de ordens de transformação com mais de 4 subtransformações;
- Envio de ordens de descarga;
- Envio de ordens de transformação com peças inicais cujo stock é zero - ordem fica pendente;
- Verificação que as ordens de transformação pendentes por falta de stock são iniciadas após armazém receber peça desse tipo, resultado de uma transformação;
- Verificação que as ordens de transformação pendentes por falta de stock são iniciadas após armazém receber peça desse tipo, resultado de uma carga;
- Envio de ordens de descarga cujo stock é zero - ordem fica pendente;
- Verificação que as ordens de descarga pendentes por falta de stock são iniciadas após armazém receber peça desse tipo, resultado de uma transformação;
- Verificação que as ordens de descarga pendentes por falta de stock são iniciadas após armazém receber peça desse tipo, resultado de uma carga;
- Envio de ordens de descarga para uma saída que está cheia - ordem fica pendente;
- Verificação que as ordens de descarga pendentes por saída cheia são iniciadas após a saída ficar livre;
- Envio de ordens de transformação impossíveis - MES descarta essa ordem;
- Verificação que, após o envio de uma ordem de descarga, a próxima peça a sair do armazém (caso exista stock disponível) é relativa a essa ordem, mesmo que existam outras ordens a ser executadas.
- Desligar e voltar a iniciar o MES e verificar o correto funcionamento de:
  - Ordens (transformação e descarga) pendentes - não são perdidas e serão executadas logo que possível;
  - Ordens (transformação e descarga) em execução - são concluídas com sucesso e MES tem conhecimento dessa conclusão;
  - Ordens (carga) em execução - são concluídas com sucesso e MES tem conhecimento dessa conclusão;
  - Alteração de stocks após conclusão de ordem - é corretamente lido após reinício do MES;
  - Estatísticas de maquinação e descarga - são corretamente apresentadas,

Em todos os testes foram consideradas as situações da ordem ter uma quantidade igual a 1 ou superior a 1, tendo todos eles passado com sucesso.

9 Conclusão

Como gesto de conclusão não poderia ser dado como terminado o relatório deste trabalho prático sem uma análise dos objetivos conseguidos com a sua realização.

Consideram-se concluídos todos os objetivos propostos para a realização do trabalho tendo-se a sublinhar os seguintes resultados:

- Paralelização total da carga de trabalho, conseguindo-se obter uma distribuição bilateral balanceada garantindo quer a utilização de todas as máquinas, quer a otimização dos fluxos de peças interno do sistema;
- Garantia da persistência de todas as ordens no sistema no rearranque do MES;
- Escalonamento planeado, mas com capacidade de reorganizamento *just-in-time* para satisfazer ordens urgentes;
- Interface gráfica atualizada em tempo real, com reflexão exata e instantânea da informação transmitida ao MES;
- Facilidade da reorganização do programa para aceitar novos modelos de fábricas e novas peças e transformações possíveis;
- Facilidade de teste devido à implementação da interface gráfica que permite emular o ERP.

Como trabalho futuro refletiu-se sobre o seguinte:

- Estudo mais aprofundado da utilização de programação genética para encontrar soluções ótimas localmente, continuando a evolução do algoritmo durante a execução de ordens, permitindo remover a limitação temporal que provocou a necessidade de minimizar o tempo de escalonamento;
- Criar endpoints para interfaces gráficas do tipo HMI para permitir a instalação de painéis do tipo SCADA para visualização das estatísticas fornecidas pelo MES;
- Criar um logbook de todas as transações de e para o MES de forma a permitir o traceback de falhas, caso existam.

Rematando o trabalho, é consenso do grupo a sensação de *fulfillment* dos requisitos do trabalho bem como da aprendizagem dos conteúdos da UC Informática Industrial.

Anexo 1 - Interface Gráfica para o ERP

Tal como referido ao longo deste documento, foi implementada uma interface gráfica para o ERP de forma a substituir o envio de comandos pelo terminal e respetiva personalização dos ficheiros XML manualmente.

Esta interface permite definir ordens de transformação de peças, ordens de descarga de peças, pedir informação sobre o stock de peças no armazém e pedir informação sobre todas as ordens concluídas e por concluir de forma bastante intuitiva.

Uma vez formulada uma ordem na interface gráfica do ERP, este cria o ficheiro XML correspondente e envia-o para o MES por UDP.

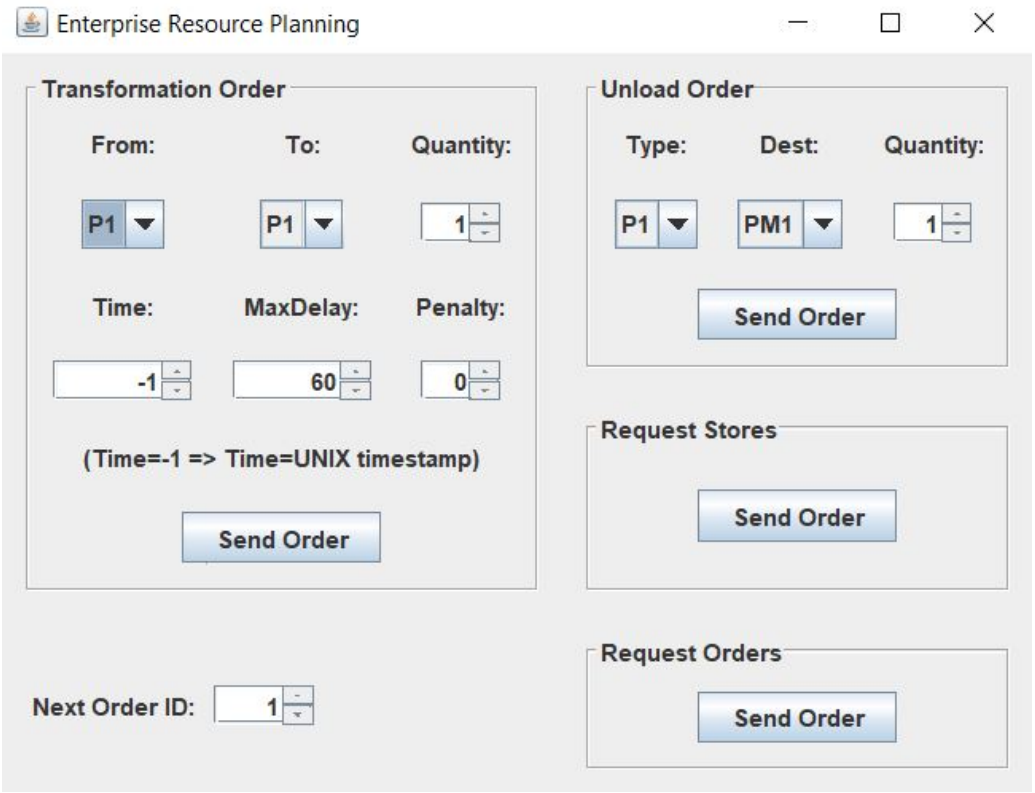

Figura 11: Interface gráfica ERP