Universidade do Porto
Faculdade de Engenharia da Universidade do Porto
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Relatório Final

Automação de uma linha de produção fléxivel

Integrantes: André Teixeira
Francisco Terra
Paulo Silva
Pedro Castro
Rui Carvalho
Professor: Mário Sousa

Junho 2020

Porto, Portugal

1. Introdução

O presente documento tem como objetivo a demonstração do trabalho desenvolvido no projeto único da Unidade Curricular de Informática Industrial. O projeto consiste numa implementação de uma automatização de uma linha de produção flexível. Para isto, este deve ser capaz de executar diferentes tipos de ordens pré-definidas (transformação, carga e descarga), provenientes de um ERP assim como manter estatísticas das mesmas e dos recursos disponíveis na fábrica.

A linguagem Python foi a eleita para a implementação do controlo de alto nível (MES) e o CODESYS para o controlo de baixo nível (SoftPLC). Para a base de dados utilizou-se o PostgreSQL a correr em Docker que pode ser consultada de forma gráfica através do PgAdmin. O MES lê um ficheiro JSON com as configurações dos IP's de onde corre a base de dados e o PLC.

O trabalho, como já foi referido, está dividido essencialmente em dois níveis (controlo do MES e controlo do PLC), sendo este apresentados mais detalhadamente nos pontos que se seguem, mais concretamente os pontos 2, 3 e 4.

Um aspeto a ter em atenção é que, devido ao grupo ser formado por cinco elementos, os requisitos do projeto incluem também transformações extra que envolvem peças a andar para trás.

2. Estrutura do código

A estrutura pela qual o trabalho se guiou de uma forma geral está focada em dois aspetos essenciais, sendo estes a parte relacionada com o MES e a parte relacionada com o PLC. O objetivo passou por tentar tornar estes o mais distintos entre si nas tarefas a executar, ou seja, tornando a troca de informação o menos frequente possível. O MES ficou encarregue de receber ordens, processá-las e geri-las de forma a poder otimizá-las o melhor possível, assim como comunicar com a base de dados para a gestão das estatísticas. O PLC ficou encarregue da parte atuadora, ou seja executar os caminhos que o MES enviava assim que possível.

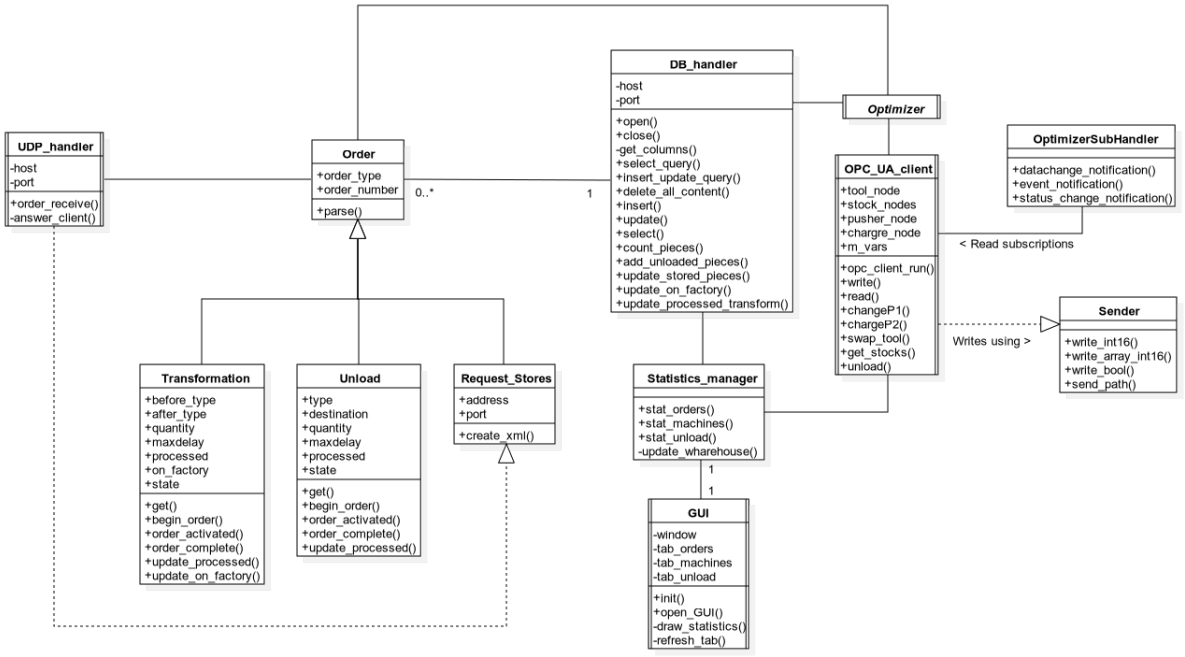

Figura 1: Diagrama de classes referente ao MES

2.1. MES

Como referido anteriormente o MES possui cinco funções principais:

- Recepção e interpretação de ordens com as classes **UDP_handler** e **Order**.
- Comunicação com a base de dados gerida pelo **DB_Handler**: Permite a análise das estatísticas e a persistência do MES.
- Processamento e visualização de estatísticas com o **Statistics_manager** e a **GUI**.
- Supervisão e controlo do estado da fábrica com o OPC-UA.
- Processamento e otimização: A componente principal é a classe **Optimizer**, que é responsável pela decisão das transformações/descargas a realizar para cada **Order**, assim como as máquinas/*pusher* onde as executar e tem como objetivo gerar uma fila de **Pieces** com caminho e transformações definidas a enviar para o PLC. As transformações são agrupadas em receitas na classe **Recipe**. Estas receitas são geradas de forma automática (para cada célula) através das transformações disponíveis representadas num grafo unidireccional, **TransfGraph**. O chão de fábrica é representado também em grafo, **PathGraph**. A

classe **Tracker** permite monitorizar as peças que estão na fabrica.

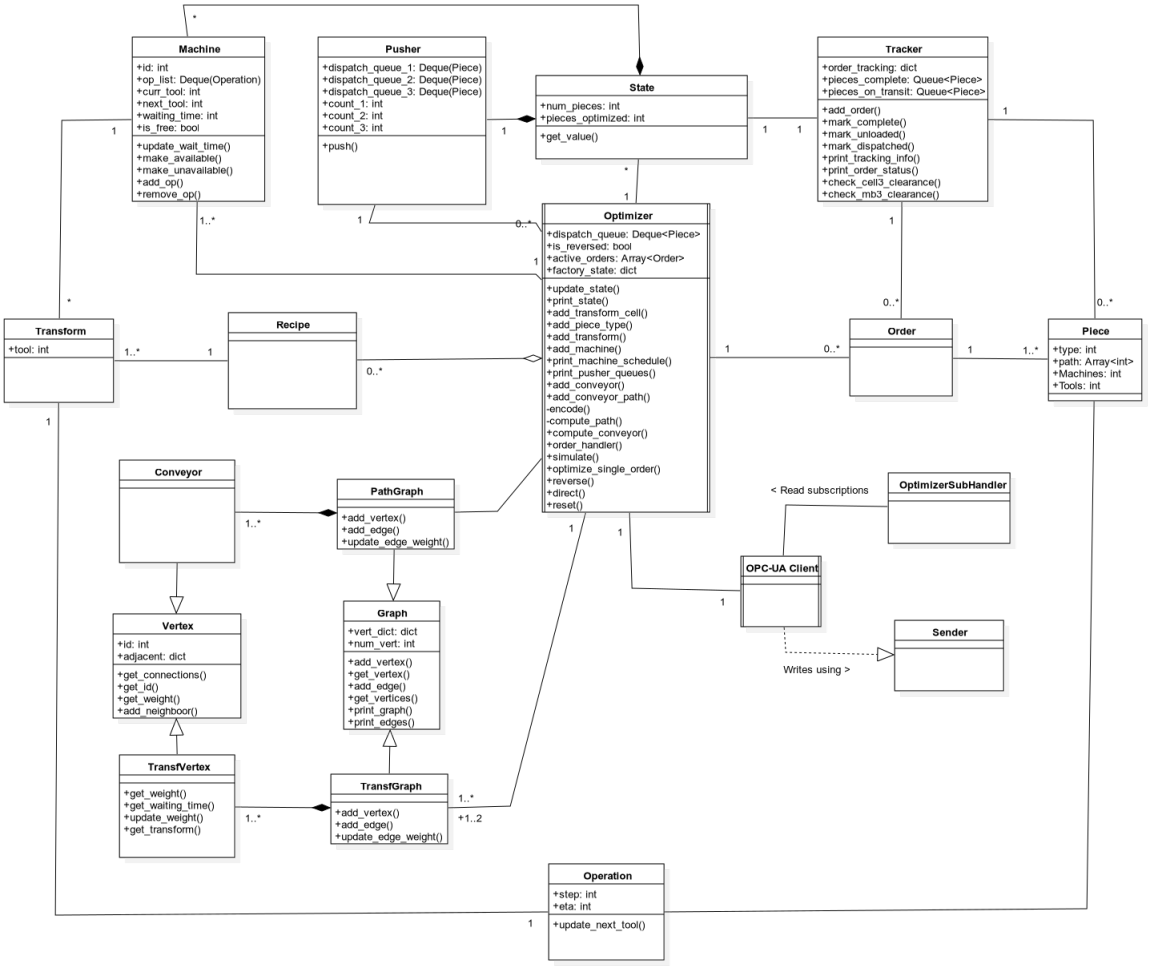

Figura 2: Diagrama de classes referente ao optimizador

2.2. PLC

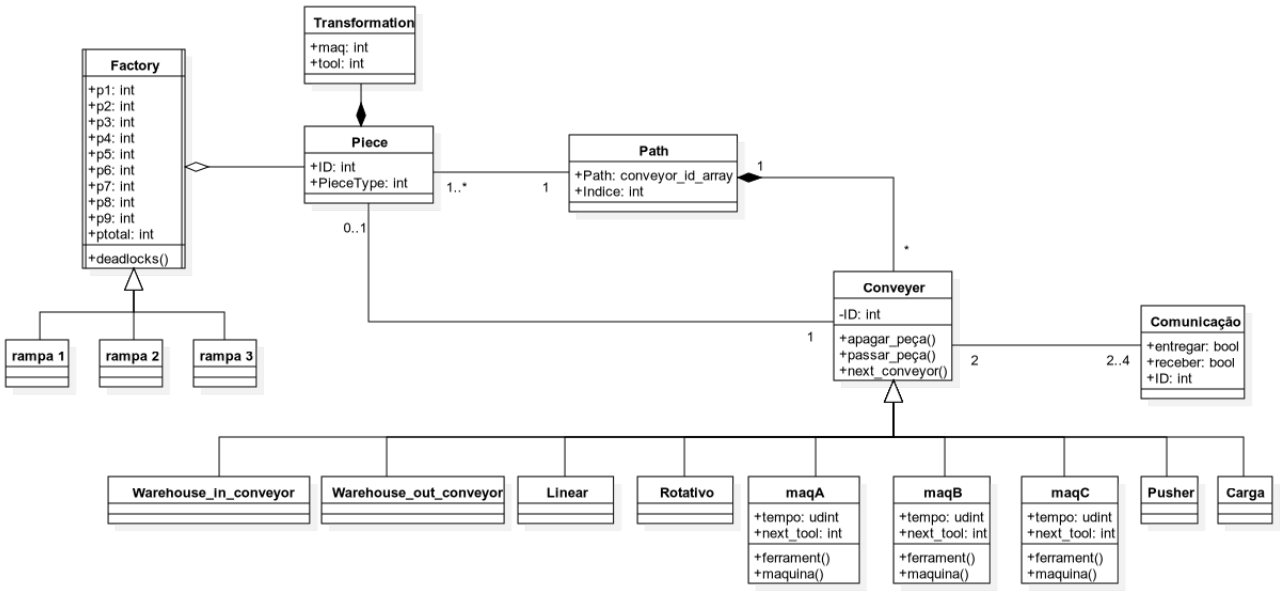

Figura 3: Diagrama de classes do PLC

A *main task* do programa criado no CODESYS apresenta 5 POU's diferentes:

- tapetes - um CFC (*continuous function chart*), que como o nome indica serve para gerir o movimento de todos os tapetes, permitindo a deslocação e maquinação das peças que passam pelo chão de fábrica. Cada tapete é representado por uma instância dos diferentes tipos de FB criados para representar os diferentes tipos de tapetes.
- *Deadlocks*- um segmento de código em ST responsável por ativar flags que permitem impedir o movimento de determinadas peças enquanto estas estiverem ativas de forma a evitar situações de *deadlock*.
- Rampa 1, 2 e 3 - 3 SFC idênticos criados com o objetivo de contabilizar o número de peças presentes em cada *slider*.

A estrutura de dados da peça contem toda a informação necessária para que os tapetes a consigam transportar usando a trajectória presente na peça, bem como a ordem das máquinas e ferramentas usadas pelas operações a serem realizadas.

As peças existentes na fábrica são guardadas num vetor de peças em que cada posição corresponde a um tapete. Não existindo nenhum tapete com ID igual a 0 no simulador, esta posição foi utilizada para receber a informação enviada pelo MES sobre as peças que precisam de ser retiradas do armazém.

3. Funcionamento do código

3.1. MES

3.1.1. *End to End Sequence*

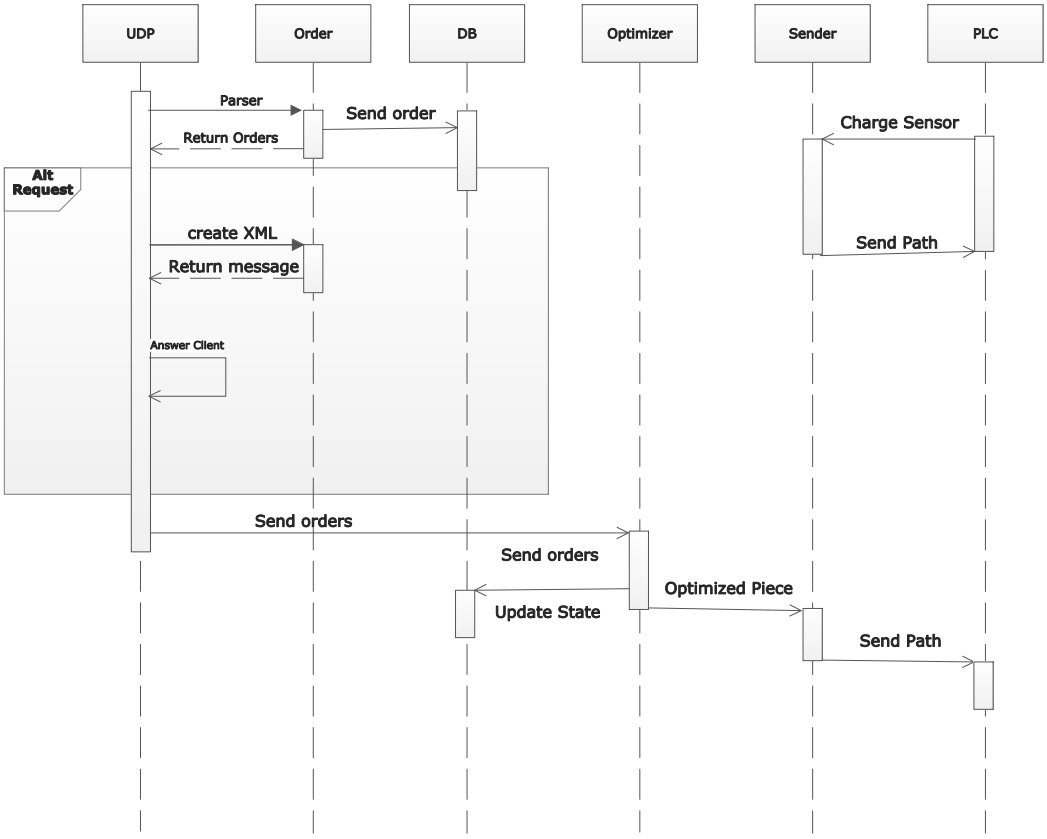

Figura 4: Diagrama de sequência ilustrando desde a comunicação UDP até o PLC por OPC UA

3.1.2. *Schedulling*

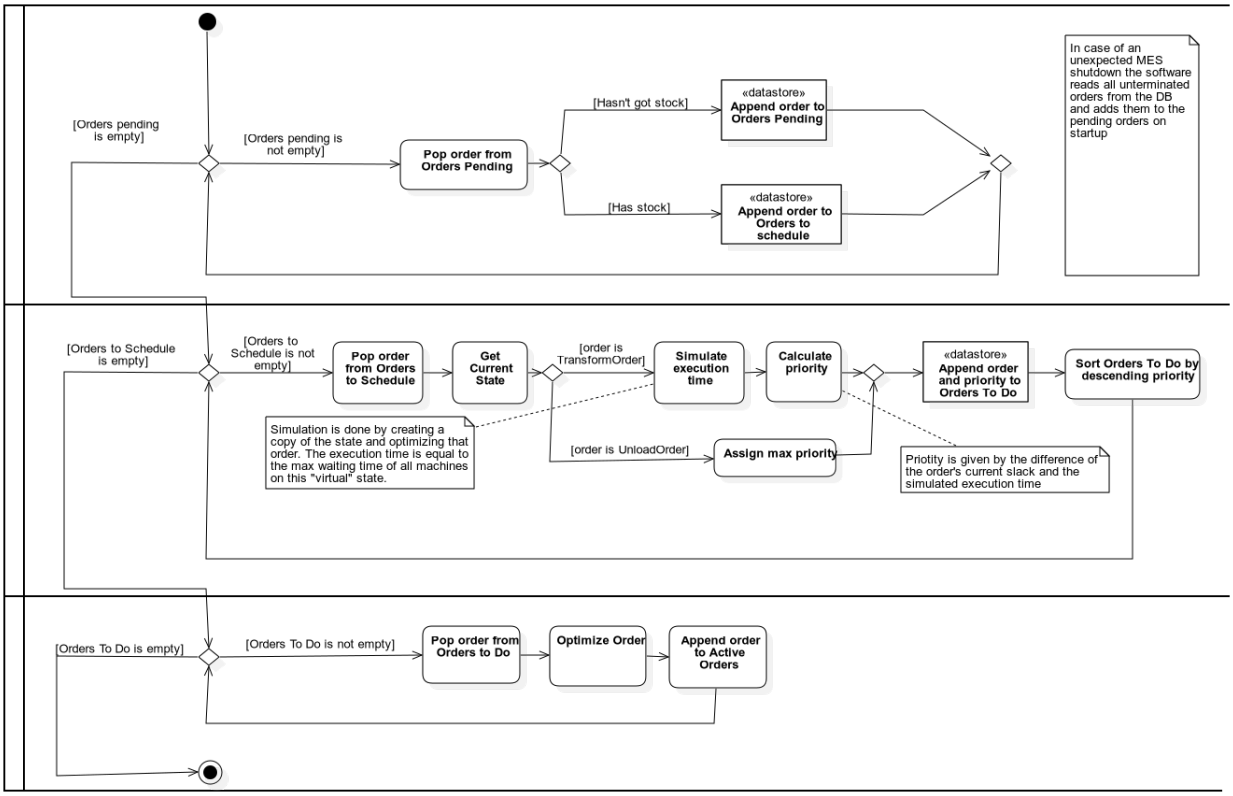

Figura 5: Diagrama de actividades para a optimização de uma ordem

O *schedulling* é feito considerando o parâmetro “max delay” da ordem, a prioridade é definida para as ordens de transformação através de uma simulação do tempo necessário a executar todas as transformações tendo em conta o estado atual da fábrica. As ordens de descarga são consideradas prioritárias.

Para efeitos de persistência, todas as ordens não concluídas são descarregadas da DB no inicio do programa.

3.1.3. Optimização

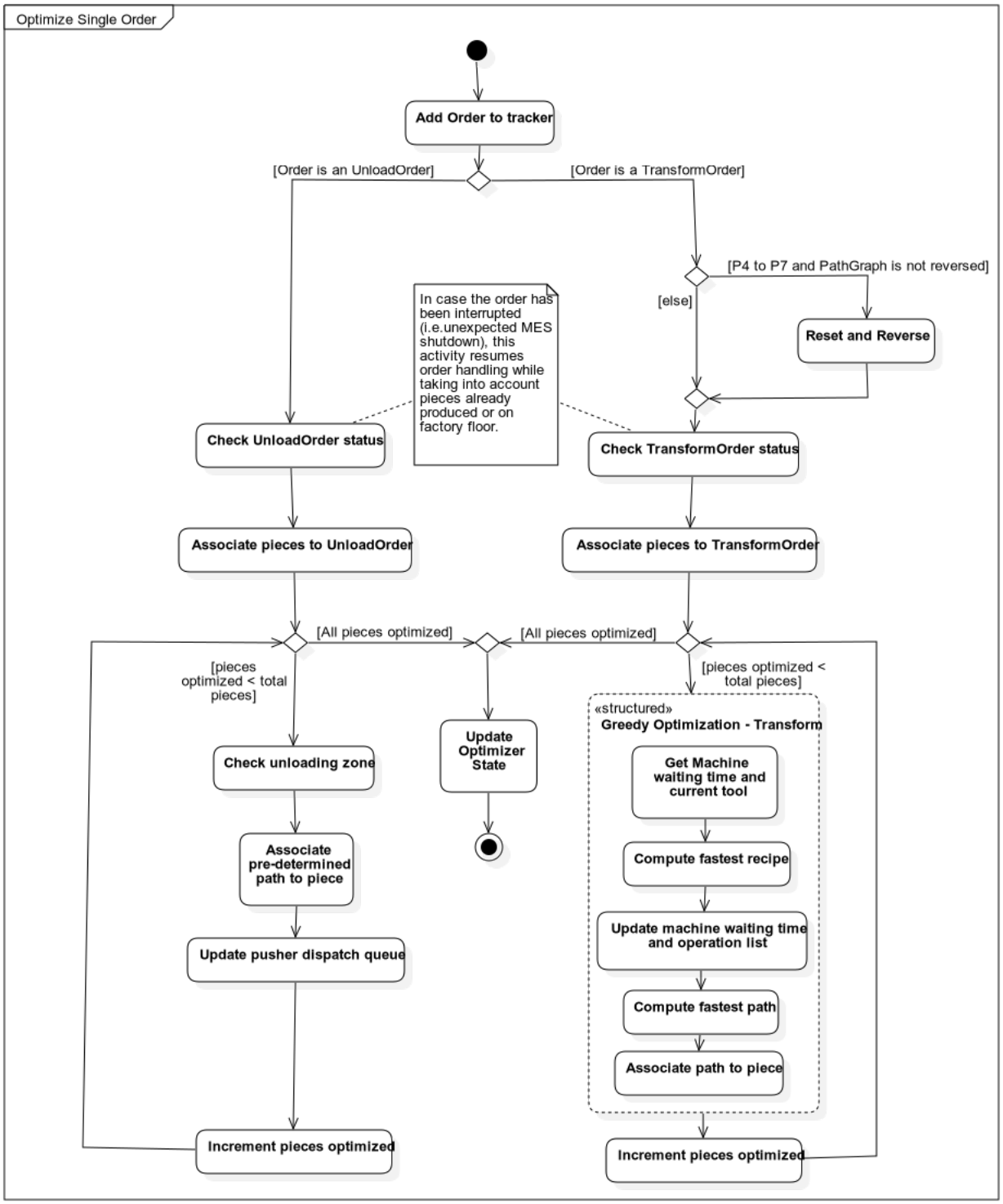

Figura 6: Diagrama de atividades para a optimização de uma ordem

A optimização baseia-se em atribuir a receita mais rápida para cada peça das ordens de transformação de forma sequencial, sendo por isso um processo *greedy*. Em relação às ordens de descarga é dada prioridade máxima portanto são adicionadas à cabeça da fila de saída mal os *pushers* estejam disponiveís. Também foi implementado um algoritmo IDFS[1] que apesar de obter resultados teoricamente melhores é computacionalmente dispendioso devido a possuir uma complexidade temporal exponencial e por isso foi descartado. Cada máquina possui uma fila de operações (peça + transformação) a executar e um tempo de espera que resulta do tempo das transformações, o tempo que a máquina tem que esperar para receber peças de transformações anteriores para cada receita e o tempo de mudança de ferramenta. Assim sendo, as peças são enviadas quando o MES recebe informação através de uma subscrição que a máquina esteja livre (por outras palavras, é a máquina que pede a peça).

[1] IDFS: Iterative Depth First Search, Complexidade temporal O(b^d), b = *branching factor*(numero de receitas) e d =*depth*(número de peças)

3.1.4. Descargas

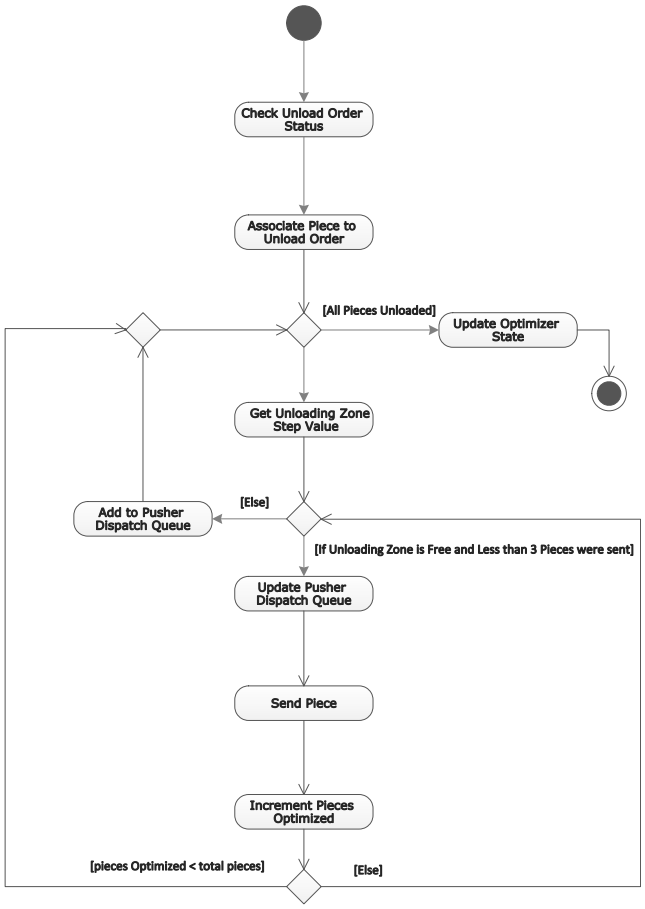

Figura 7: Diagrama de atividade para as descargas

No procedimento de controlo das descargas e, perante a disposição dos sensores nas rampas de descarga, optou-se por uma estratégia de prioridade na fila tendo em conta o pior cenário possível. Na fábrica estão colocados 3 sensores de presença nas 3 primeiras posições deixando a última indefinida. Sendo assim, implementou-se um SFC para cada rampa no CODESYS o qual verifica a ocupação da respetiva rampa e sempre que exista e esteja disponível envia-se até 3 peças do início da fila de despacho.

3.1.5. Rutura de *Stock*

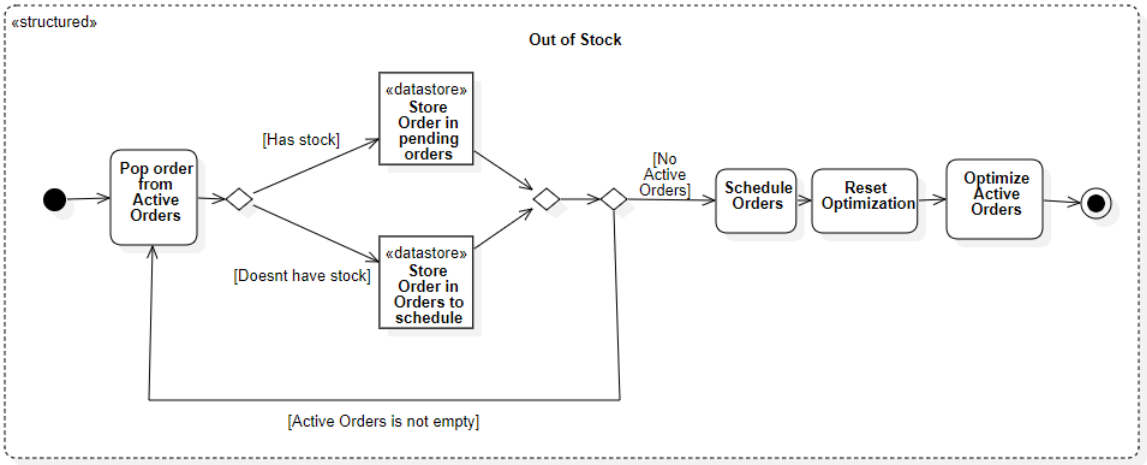

Figura 8: Diagrama de atividade para a rutura de *stock*

Se uma ordem ficar sem *stock* durante a sua execução, a optimização é refeita e a ordem é adicionada às ordens pendentes até existir pelo menos uma peça em stock. Aí, o *scheduller* envia peças à medida que o stock é reposto.

3.1.6. Envio de peças

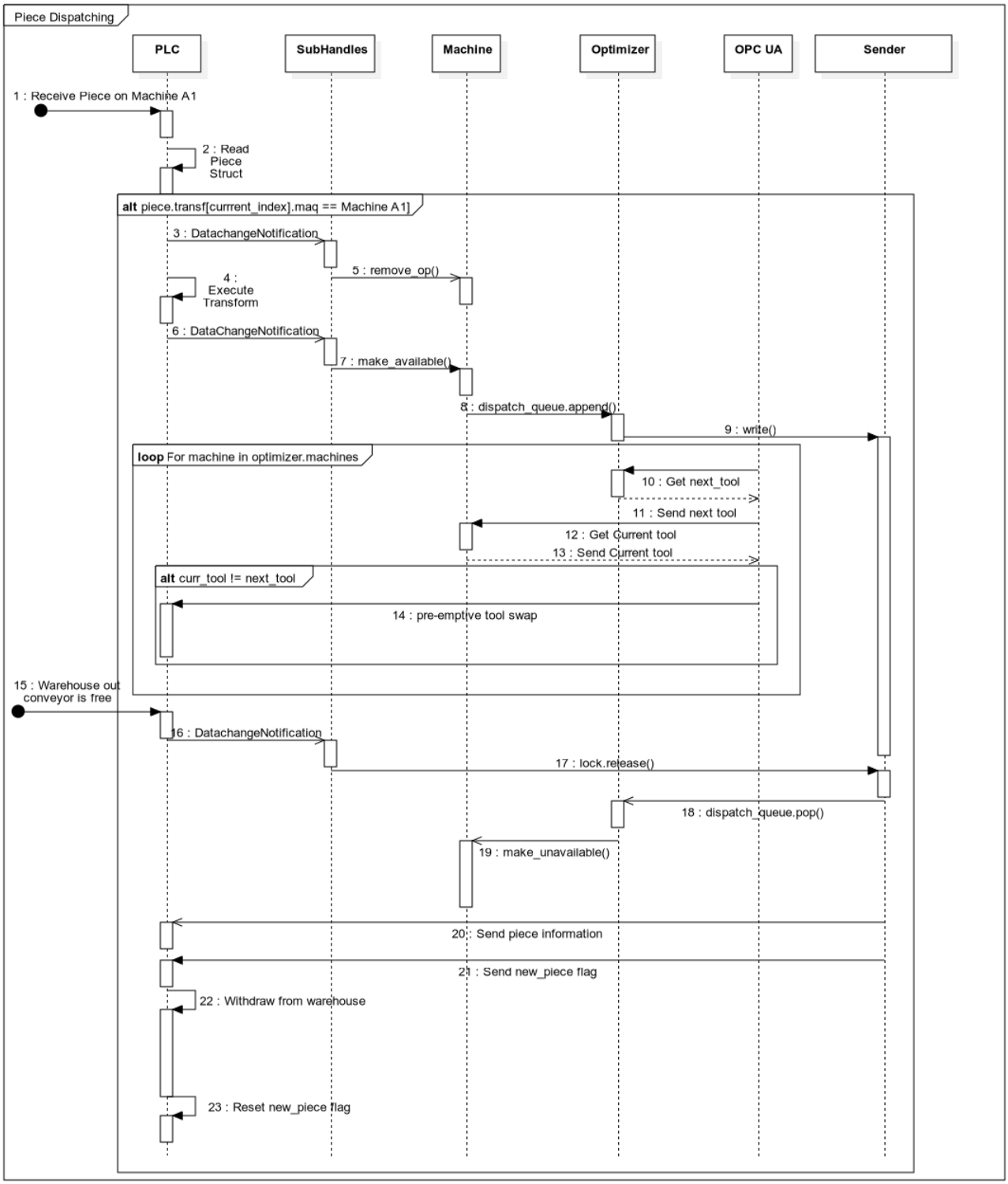

Figura 9: Diagrama de sequência do envio de uma peça

3.1.7. Requisito extra

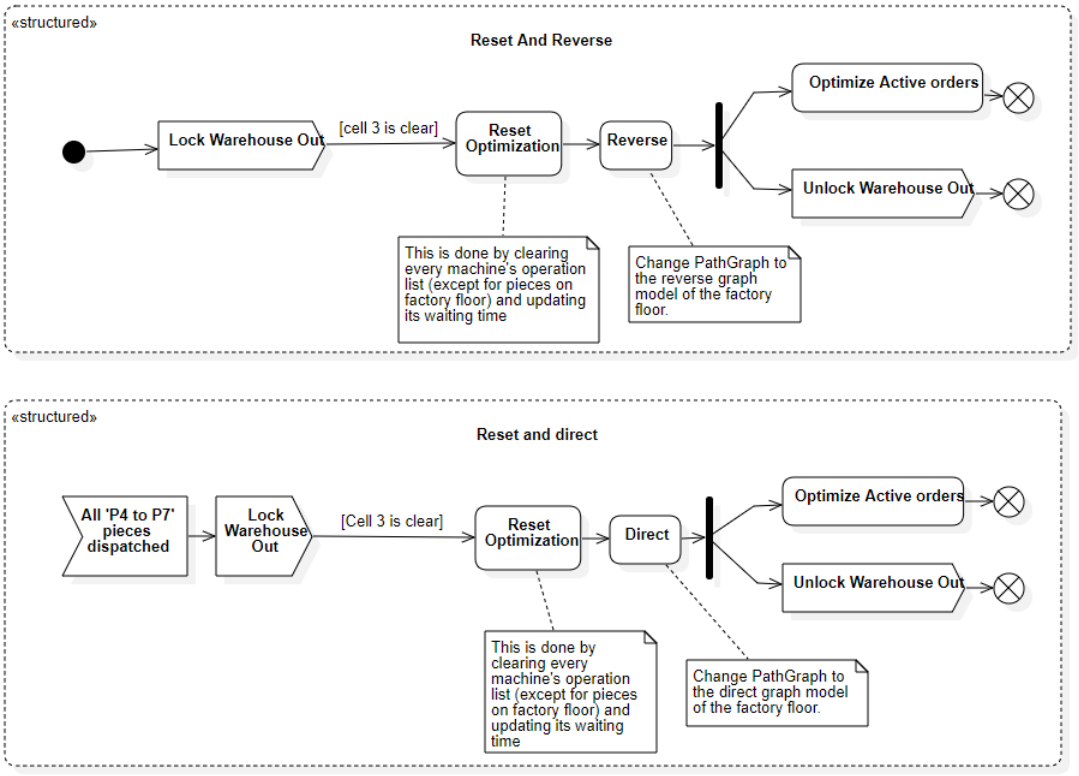

Figura 10: Diagrama de atividade da troca entre os modos direto e reverso

O requisito extra consiste em adicionar uma transformação adicional que inclui a possibilidade de peças moverem-se para cima no sentido contrário às restantes. De forma a evitar *deadlocks* implementou-se um modo “inverso” em que a célula 3 dedica-se apenas a realizar transformações “P4->P8->P7”, esta decisão foi feita devido a esta ordem criar um *bottleneck* (a segunda transformação demora o triplo da primeira). A troca de modo é feita, quando o MES recebe uma ordem que necessite de movimentar-se para cima bloqueando a entrada até que todas as peças em regiões criticas sejam processadas, o retorno à normalidade é feito de forma análoga quando todas as ordens ’P4->P7’ que o MES tenha recebido sejam completas. Nota: Neste modo não é possível fazer movimentos inter-células.

3.2. PLC

Apesar de haver diferentes tipos de tapetes (linear, rotativo, máquinas...), todos eles apresentam o mesmo modelo de funcionamento. Como se pode observar na figura 11 é possível dividir o SFC em 4 etapas que são percorridas de forma cíclica, sendo estas: o estado livre, a receber, ocupado e a entregar.

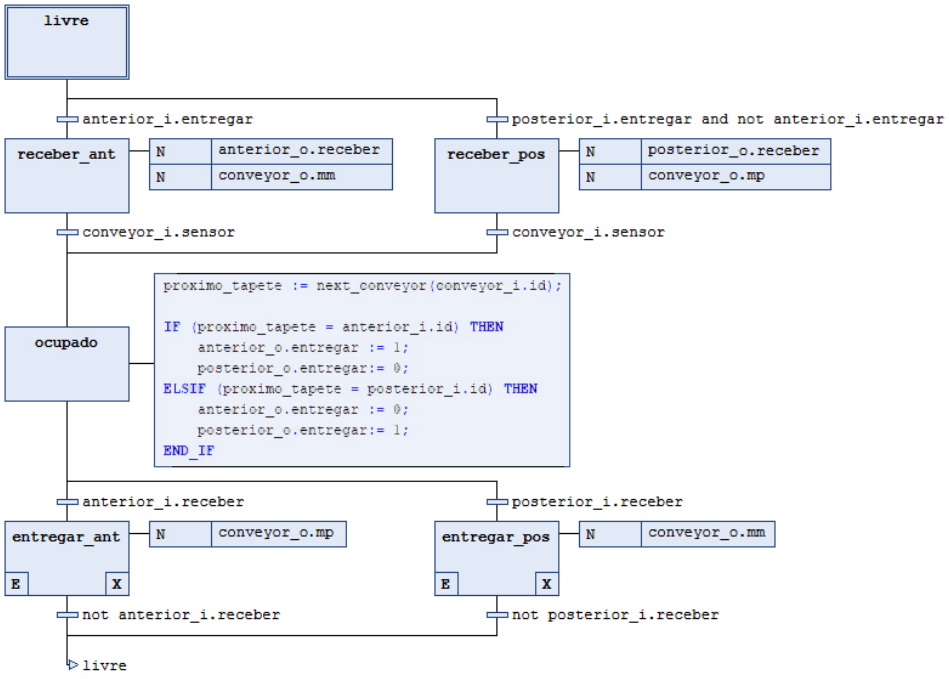

Figura 11: Versão simplificada do SFC criado para o controlo dos tapetes

No estado livre o tapete encontra-se em repouso até receber um pedido de entrega de outro tapete. No caso do tapete rotativo foi acrescentado a este estado um conjunto de ações que possibilitam a rotação do mesmo de forma a ficar alinhado com tapetes que tenham peças cujo próximo destino seja o tapete em questão.

O estado ocupado utiliza o método **next_conveyor()** de forma a saber o ID do próximo tapete e compara-o com o ID dos tapetes à sua volta, enviado um pedido de entrega para o tapete correspondente. Quando este recebe confirmação que pode enviar a peça este passa ao estado entregar, para além de entregar a peça ao próximo tapete, também atualiza a localização da peça no vetor piece_array. Após receber confirmação que a peça foi entregue, regressa-se ao estado

livre. Este processo pode ser visualizado de uma forma geral na Figura 12.

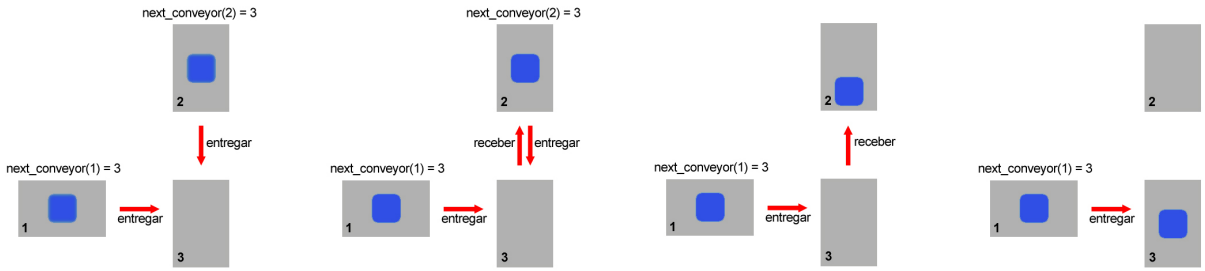

Figura 12: Exemplo de comunicação entre tapetes

O SFC responsável pelos tapetes onde existem as máquinas apresenta um estado adicional entre o ocupado e o entregar que corresponde à maquinação das peças. Foram também acrescentados dois métodos a este FB que permitem saber o ID da próxima máquina e da próxima ferramenta a ser utilizada. As instruções presentes no estado ocupado também são ligeiramente diferentes, dando prioridade ao processo de maquinação em detrimento de o de entrega. O método de **next_conveyor()** só é chamado quando o **maquina()** retornar um ID diferente do da máquina atual, sinalizando que não existem mais transformações a serem realizadas neste tapete. Este estado extra é responsável por trocar a ferramenta se esta não for igual à ferramenta retornada pelo método **ferramenta()** e pelo processo de ativação da ferramenta durante o tempo necessário para cada uma das diferentes transformações. Após a transformação ter sido concluída, regressa-se ao estado ocupado. Foi criado um SFC adicional de forma a ter um controlo independente da ferramenta, visto que esta só era utilizada no estado de maquinação. Desta forma era possível receber informação do MES sobre a próxima ferramenta e em paralelo realizar as operações acima descritas e simultaneamente trocar a ferramenta. Estes dois SFC apesar de serem independentes partilham um recurso que é a ferramenta, sendo o uso da ferramenta mutuamente exclusivo, dando prioridade ao estado de maquinação quando os dois querem utilizar a ferramenta no mesmo instante.

4. Implementação

Para o funcionamento intencionado do MES iriam ser necessárias uma boa quantidade de ferramentas por forma a facilitar a sua implementação. Para isso, Python é uma linguagem com uma imensa quantidade de recursos disponíveis para uso livre. Dessa forma, o processo de desenvolvimento de software é mais acelerado, com a acrescente que é uma linguagem de um alto nível, podendo-se assemelhar quase a pseudo-código.

No entanto, esta linguagem é muito lenta em comparação às linguagens de baixo nível. Num projeto desta dimensão, isto torna-se evidente. Para além disso, ao longo do desenvolvimento do programa, conseguiu-se reparar que esta ferramenta tem tendência para a falta de escalabilidade. Um dos fatores para isto é por ser uma linguagem interpretada, em vez de compilada. Isto torna o seu uso arriscado para grandes aplicações.

Dado que o problema usa muitas comunicações assíncronas entre tipos de comunicações, foi necessário o uso de threads a correr em ciclos infinitos para as mesmas. Assim, não haveria risco de perder ordens dos clientes ou dados do PLC. No entanto, o problema de tempo de execução, discutido anteriormente, torna-se ainda mais agravado. Cada tarefa fica consideravelmente mais lenta.

O transporte de recursos dentro das threads do MES foi implementado com o uso de filas. Dessa forma, mal fossem recebidas ordens do cliente, estas eram descodificadas numa estrutura compreensível e colocadas numa fila.

Para o MES enviar caminhos para novas peças a colocar no chão de fábrica, recorreu-se à biblioteca *asyncua*[2] que implementa um cliente de OPC-UA através de programação assíncrona, o que permite correr várias tarefas de forma assíncrona como o envio de peças, a carga e a troca de ferramentas de uma forma mais eficiente e possui uma *framework* de alto nível para subscrever mudanças de variáveis implementada em uma thread dedicada.

[2] Disponivel em: https://github.com/FreeOpcUa/opcua-asyncio

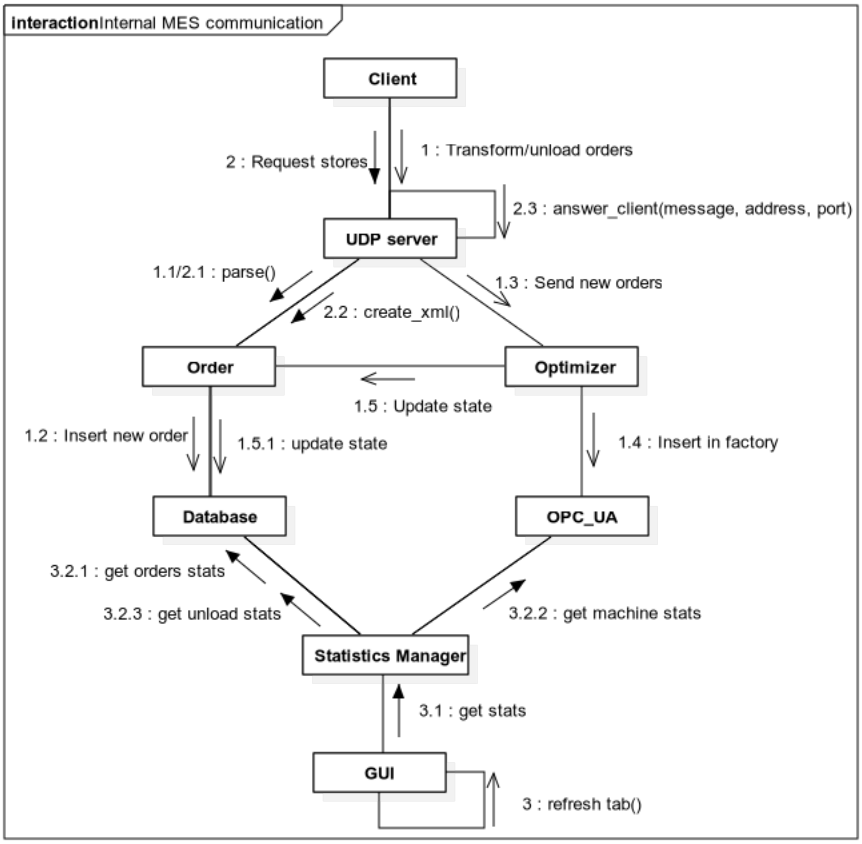

Figura 13: Diagrama de comunicações no interior do MES

Foi colocada uma *Graphical User Interface* (GUI) por forma a exibir as estatísticas ao utilizador. Porém, observando a Figura 13 é possível constatar que este não tem conecção direta com o estado interno do MES. Isto aconteceu porque o ambiente no qual foi inserida não estava preparada para o suportar. Significa que há uma grande dependência da base de dados para aceder às estatísticas, o que pode resultar em informações desatualizadas. Ademais, o MES usa muitos recursos do computador, por estar a correr em python, o que significa uma taxa de atualização de informação muito lenta para o utilizador.

5. Estrutura final vs inicial

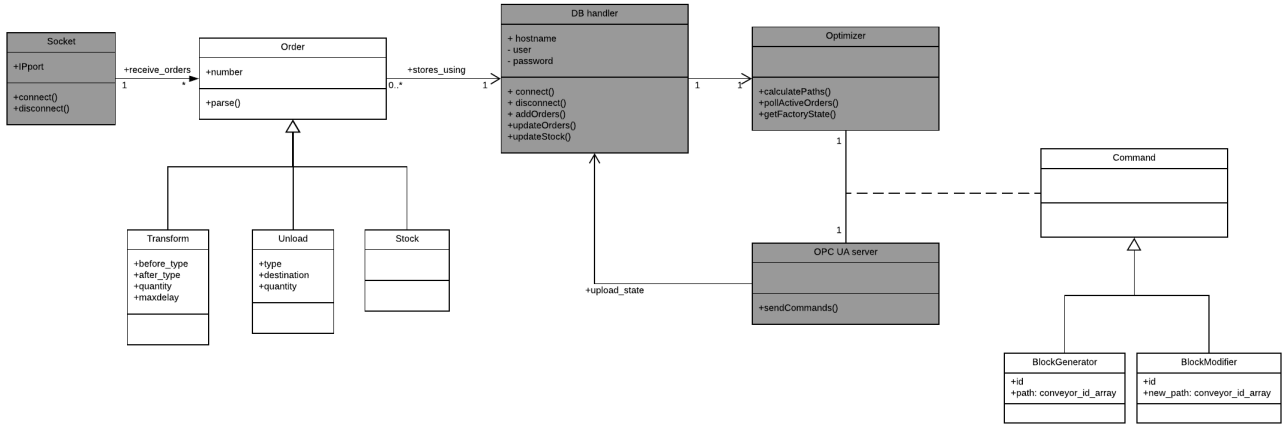

Figura 14: Diagrama de classes inicial do MES

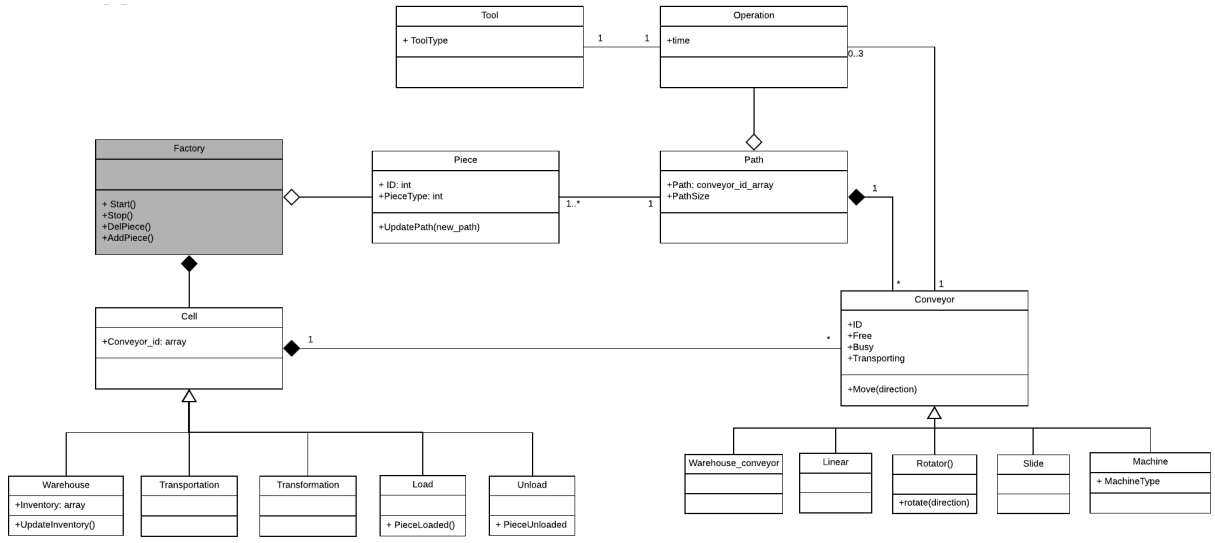

Figura 15: Diagrama de classes inicial do PLC

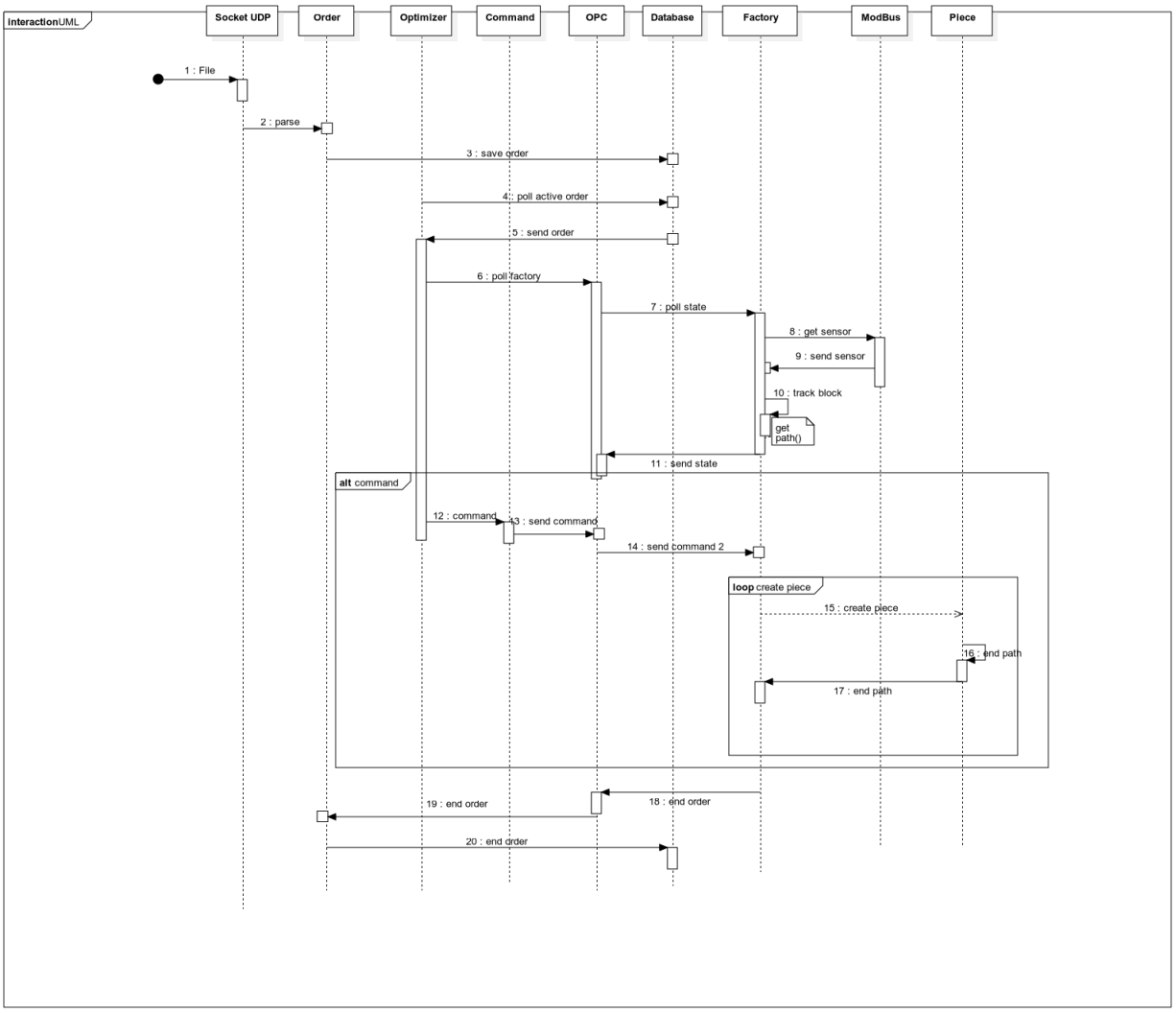

Figura 16: Diagrama de sequência inicial da interpretação de uma ordem

A arquitetura inicialmente proposta foi sofrendo pequenas alterações e melhorias ao longo do desenvolvimento do projeto. Foi mantida a ideia inicial sobre a estrutura que contem o caminho da peça como sendo um vetor de inteiros representando os ID's dos tapetes por onde a peça deve passar. No entanto os mecanismos que permitiriam passar esta informação entre tapetes bem como a interação entre os mesmos não estava bem definida. De forma a colmatar esta falha foi adicionado à classe *Conveyor* um conjunto de métodos que permitem lidar com a estrutura *piece* e ainda uma estrutura de comunicação de forma a coordenar e sincronizar a passagem de peças.

A estrutura que contém a informação relativa às transformações também foi ligeiramente al-

terada, sendo necessária a informação do id da máquina e a ferramenta. O atributo tempo foi retirado, o FB da máquina sabendo apenas a ferramenta e o tipo atual da peça é capaz de realizar a operação corretamente. Foi ainda criada uma função que analisa a posição das peças em alguns tapetes da fábrica e o seu percurso com o objetivo de prevenir o surgimento de *deadlocks*. A noção de células foi eliminada por não ser necessário, alguns tipos de tapetes foram adicionados e criaram-se as rampas, que têm como objetivo contabilizar as peças existentes nos *sliders*.

Relativamente ao MES, foram adicionados métodos (nomeadamente nas classes Tracker e SubHandles) para contabilizar as peças que estão na fábrica sendo necessário para a persistência e para conseguir inverter o sentido do movimento na célula 3. Devido a alguma falta de conhecimento e experiência no que diz respeito ao OPC-UA, a arquitectura foi consideravelmente alterada a nível de comunicação sendo os *commands* iniciais que foram substituídos por um sistema de subscrições que permite de certa forma descentralizar o controlo no Optimizer e correr processos como a carga e o envio de peças para a fábrica de forma independente. Nesse contexto, também foram acrescidas filas de operações para cada máquina de forma a gerir cada recurso de forma mais modular.

6. Comparação com padrões existentes

A arquitetura implementada neste projeto tem algumas semelhanças com a norma ISA 95, no entanto, não se enquadra completamente na visão da industria 4.0. Para a conseguir alcançar, falta uma melhor conectividade entre camadas. De facto, o modelo implementado é muito centralizado no MES. Este é o ponto de distribuição e processamento da informação principal.

Isto significa que existe algum lapso no acesso livre à informação. Neste caso, o problema estava bem definido e por isso muita da informação relativa aos processos, às transformações e ao *layout* da fábrica foram *hard-coded*. Ou seja, caso fosse necessário generalizar o problema ou a arquitetura para outros problemas, esta implementação teria falta de flexibilidade a adaptar-se a tal.

Por outro lado, esta implementação partilha um pouco a ideia de *administrative shell* com o RAMI 4.0. Com programação orientada a objetos a interface entre os vários módulos é feito através dos atributos e dos métodos de cada uma das instâncias. Como por exemplo, as peças

nos tapetes são inspecionadas pelas máquinas e pelos mesmos para saber qual o serviço que esta necessita. Contudo, o inconveniente anteriormente visto, da falta de conectividade, mantém-se.

No ISA 95 são generalizados estruturas para objetos de diversas categorias. Para este projeto, essa metodoligia não foi usada. Dada a quantidade de objetos distintos, optou-se por usar estruturas customizadas. É de reconhecer, no entanto, que o uso deste tipo de modelos generalizados e standardizados como vantajoso para uma implementação com menos problemas de integração e coerência.

Relativamente aos modelos de atividades do ISA 95, para este caso será apenas abordado o controlo da produção. O modelo de atividades de operações de produção no ISA 95 é ilustrado na seguinte figura.

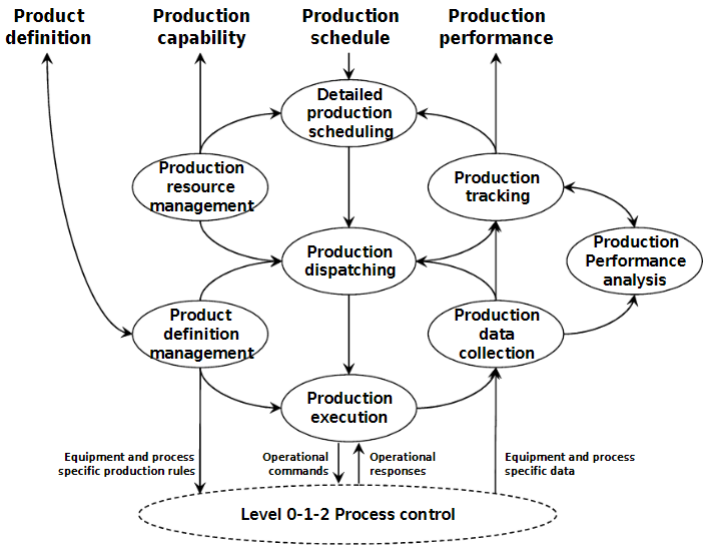

Figura 17: Modelo de atividades de operações de produção

No caso implementado, o próprio ERP tem limitações, visto que não está preparado para modificar a definição do produto e avaliar as capacidades de produção do MES, apesar de simular a receção de informação desta última. Então, o MES revolve mais entorno do agendamento da produção e da avaliação da performance do nível inferior, apesar desta ultima não ser retornada para o ERP.

6.1. Comparação com o artigo de serviço de granularidade industrial

No trabalho foram usados dois métodos de granularidade:

- Granularidade de processos: Por forma a diminuir as interações entre os módulos adjacentes, nomeadamente os de maior nível, caso do otimizador, foi usado uma granularidade maior. Os serviços deste são de grande dimensão, dessa maneira as comunicações são diminuídas com o exterior.
- Granularidade de funcionalidade: Para conseguir disponibilizar às classes conectadas uma maior flexibilidade e alcance de ferramentas, de grão mais fino, à disposição. Implementado principalmente na área das estatísticas e do armazenamento de dados, isto permite alterar de forma menos complexa o código a implementar.

7. Organização da equipa

Foi utilizado como software de controlo de versões o git com um repositório online no Github. A atribuição geral das tarefas manteve-se e foi adotado um modelo de desenvolvimento incremental com base na arquitectura de alto nível desenvolvida no inicio do projecto, com modificações ao longo da evolução.

7.1. Repartição do trabalho

Tabela 1: Distribuição do trabalho

André Teixeira	Optimização, ordens de descarga e carga e escrita por OPC-UA
Francisco Terra	PLC
Paulo Silva	PLC
Pedro Castro	GUI, Estatísticas, base de dados e comunicação UDP
Rui Carvalho	Optimização, ordens de transformação e subscrições OPC-UA

Considerou-se que todos os elementos trabalharam de uma forma equivalente. Cada mem-

bro, mesmo que tenha desenvolvido a sua própria parte do projeto, esforçou-se em comunicar, compreender e integrar todas as partes que constituem o trabalho num todo. Portanto propõe-se uma distribuição igual da nota por todos.

Tabela 2: Proposta de distribuição da nota

André Teixeira	20 %
Francisco Terra	20 %
Paulo Silva	20 %
Pedro Castro	20 %
Rui Carvalho	20 %

8. Testes e resultados

O software correu com sucesso a sequência A fornecida com um tempo de 10m21s, além disso também foram criadas ordens adicionais de forma a testar a persistência, a ruptura de stock, inversão de sentido e combinação destas.

A implementação da troca antecipada de ferramentas e movimentação dos tapetes, melhorou consideravelmente a performance do sistema. Obteve-se uma redução de tempo de aproximadamente 1m30s para a sequência A. Notou-se que uma possível optimização seria a pré-alocação das peças seguintes nos tapetes de entrada da célula, no entanto seria necessário uma mudança considerável na arquitectura desenvolvida (cf. Figura 9).

De forma a testar a situação de falta de *stock* foi utilizada uma versão modificada do simulador com um número reduzido de peças. Verificou-se que o envio de peças à medida que o *stock* é reposto impacta de forma negativa as ordens em execução (especialmente no caso de uma transformação que necessite trocar uma das ferramentas) no entanto esta foi a maneira implementada conforme os requisitos do caderno de encargos.

No modo “inverso”, como a célula 3 está dedicada a realizar a transformação “P4->P7” apresenta uma performance claramente inferior ao modo “direto”. Isto deve-se a não ser possível realizar movimentos entre células, no entanto facilmente se verifica para ordens de grande dimensão que a célula 3 fica congestionada devido à diferença de tempo entre a transformação

“P4->P8” e “P8->P7”. Por essa razão, optou-se por não permitir movimentos ascendentes em mais nenhuma célula e a considerar dois modos de funcionamentos distintos e intercambiáveis.

O facto de a escolha das máquinas ser feita no momento da otimização (aquando da receção das ordens) e o tempo de movimentação das peças ser desprezado resultou em situações que devido a congestionamentos nas células, outra máquina do mesmo tipo fique livre e peças poderiam ser maquinadas nessa máquina em vez de esperarem pela máquina pré-determinada obtendo resultados melhores. Por esta razão, um planeamento das máquinas mais dinâmico poderia ser uma opção mais viável.

O *Schedulling* podia ser melhorado para suportar a paragem de ordens activas caso seja recebida uma ordem cuja a urgência o justifique.

9. Conclusões

O projeto tem os requisitos do caderno de encargos cumpridos. Contudo, a otimização poderia ser ainda mais melhorada, para além de que problemas na implementação poderiam ter sido evitados. Para isso, teria de se ter recorrido mais vezes ao redesenhar e o repensar da arquitetura. Dessa maneira, o desenvolvimento seria mais limpo e compreensível, dada a dimensão do projeto.

Devido à complexidade do trabalho, utilização de diversas bibliotecas e de uma linguagem interpretada, não foi possível resolver problemas de compatibilidade para empacotar num executável. A execução do MES é, portanto, meramente uma aplicação de consola.

Foram feitas várias alterações, que demonstra a evolução da compreensão do tema. Mesmo para o caso da programação orientada a objetos, foi possível ganhar entendimento de como se pode modular o código e o interligar usando esta metodologia. Apesar da falta de prática dos elementos nesta área, ainda foram conseguidos resultados satisfatórios.

Em suma, apesar de o resultado final não ter sido na totalidade aquilo que se pretendia, de uma maneira geral podemos afirmar que os objetivos estabelecidos foram cumpridos com sucesso. Além disso este trabalho permitiu ficar com uma boa ideia do tipo de ferramentas utilizadas na indústria, como é o caso do CODESYS e assim ter uma noção do que nos poderá ser pedido um dia como futuros engenheiros.

Referências

[1] Alois Zoitl, Aydin Homay, Mário de Sousa, Martin Wollschlaeger, *Service Granularity in Industrial Automation and Control Systems*