

Faculdade de Engenharia da Universidade do Porto



PROJETO - Controlo de Automação de uma Linha de Produção

Francisco Caetano up201705031

Francisco Damas up201604134

Guilherme Pinheiro up201703867

Tomás Araújo up201704738

Relatório do Trabalho Prático realizado no âmbito da Unidade Curricular
Informática Industrial, do 4º ano do ramo de Automação do
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Índice

| | | |
|-------|---|----|
| 1 | INTRODUÇÃO | 3 |
| 2 | ESTRUTURA DO CÓDIGO | 3 |
| 2.1 | MES | 3 |
| 2.2 | PLC | 4 |
| 3 | FUNCIONAMENTO DO CÓDIGO | 4 |
| 3.1 | Interface Gráfica do MES | 5 |
| 3.2 | Arranque Inicial ou Restart | 5 |
| 3.3 | Definição da prioridade das ordens de transformação | 5 |
| 3.4 | Ordem de transformação | 5 |
| 3.4.1 | Ordens difíceis | 6 |
| 3.5 | Controlo sobre as transformações | 6 |
| 3.6 | Request Stores e Request Orders | 6 |
| 3.7 | Estatísticas | 6 |
| 4 | IMPLEMENTAÇÃO | 6 |
| 5 | ESTRUTURA FINAL VS INICIAL | 8 |
| 5.1 | MES | 8 |
| 5.1.1 | Interface Gráfica / MES | 8 |
| 5.1.2 | OPC-UA | 8 |
| 5.1.3 | DB | 8 |
| 5.1.4 | UDP | 8 |
| 5.1.5 | DATA & NEXT | 9 |
| 5.2 | PLC | 9 |
| 6 | COMPARAÇÃO COM PADRÕES EXISTENTES | 9 |
| 6.1 | ISA 95 | 9 |
| 6.2 | RAMI 4.0 | 10 |
| 6.3 | Artigo "Service Granularity in Industrial Automation and Control Systems" | 11 |
| 7 | ORGANIZAÇÃO DA EQUIPA | 11 |
| 8 | TESTES E RESULTADOS | 12 |
| 8.1 | Teste 1 - Rapidez do Sistema | 12 |
| 8.2 | Teste 2 - Cálculo da penalidade | 12 |
| 8.3 | Teste 3 - Manutenção do MESInitTime, após reinicio do MES | 13 |
| 8.4 | Teste 4 - Aplicação Real | 13 |
| 9 | CONCLUSÃO | 14 |

| | |
|-------------------------|----|
| 10 ANEXOS | 15 |
| 10.1 Anexo 1 | 15 |
| 10.2 Anexo 2 | 20 |
| 10.3 Anexo 3 | 21 |
| 10.4 Anexo 4 | 22 |
| 10.5 Anexo 5 | 23 |
| 10.6 Anexo 6 | 24 |
| 10.7 Anexo 7 | 25 |
| 10.8 Anexo 8 | 26 |
| 10.9 Anexo 9 | 27 |
| 10.10Anexo 10 | 28 |
| 10.11Anexo 11 | 29 |
| 10.12Anexo 12 | 30 |

1 INTRODUÇÃO

No âmbito da disciplina de Informática Industrial, foi redigido este relatório que incide sobre o projeto desenvolvido ao longo do semestre. Este tem por base o controlo e otimização de uma linha de produção flexível, que simula um processo industrial, com funções como o armazenamento de peças, a descarga e a transformação das mesmas. As ordens provêm de um ator superior, um ERP.

A linguagem de programação escolhida para o desenvolvimento do MES e das interfaces gráficas foi JAVA. Para o Soft-PLC, usou-se a plataforma *CODESYS*. Para armazenamento dos diversos dados, foi utilizado um sistema de gestão *PostgreSQL*, recorrendo a um servidor externo da *Amazon WebServices*, através da plataforma *Heroku*.

Para além disso, o sistema disponibiliza as estatísticas de diversos componentes operacionais, como as máquinas e os *pushers*, de forma a que seja possível aos utilizadores estarem a par do processo de produção e obterem o máximo de informação em tempo real. Estes valores podem ser observados na interface gráfica projetada fora do MES, uma vez que esta comunica unicamente com a base de dados, de modo a não causar transtornos adicionais. Existe ainda uma outra interface que funciona apenas para inicialização do sistema de supervisão.

A rapidez de processo e a independência de alguns processos são duas das diversas características especiais da solução apresentada. Ao longo do relatório, irá existir uma explicação mais detalhada da abordagem seguida pelo grupo.

2 ESTRUTURA DO CÓDIGO

Antes da implementação do código, foi definida pelo grupo uma divisão de tarefas entre o MES e o PLC. O MES é responsável por receber as ordens, ordená-las em função do atraso máximo e da penalidade, enviar as ordens para o PLC e armazenar toda a informação existente no processo. O PLC é responsável pela organização das máquinas necessárias para executar cada ordem de transformação, bem como pelo controlo de todos os elementos do *Shop Floor Simulator*. No final do processamento de cada ordem, envia informações sobre o processo ao MES.

2.1 MES

Para uma divisão eficiente de tarefas pelo grupo, a estrutura do código do MES foi dividida em várias *"packages"*. Estas são constituídas por várias classes e podem ser visualizadas com maior detalhe no diagrama da Figura 2 (Sub. 10.1).

Existem 3 ficheiros essenciais: *Main.java*, *UI.java*, *MES.java*. O primeiro é responsável pela inicialização, manifestada pela criação da UI. O segundo, a classe UI, é usado para facilitar ao utilizador a inicialização do MES. O terceiro é o centro do MES, responsável pela supervisão e por assegurar as comunicações, criando 5 *threads*.

- *Package DATA* - Engloba as classes *Transform* e *Unload*. Estas são responsáveis por ler e separar as ordens ERP quando recebidas pela *package UDP* .
- *Package DB* - É a package mais importante do sistema, uma vez que permite controlar todo o sistema armazenando e tratando os dados. São, portanto, armazenados diversos valores, desde o momento em que se inicia o MES: as estatísticas do nosso sistema e as ordens de transformação

e descarga que estão por processar, que estão em processamento e as que já foram efetivamente processadas. Com ajuda da DB, temos um controlo claro em tempo real da fábrica.

- *Package NEXT* - Controla o lado esquerdo e o lado direito de forma independente, quer no início quer no fim de cada ordem. As classes *LCTF* e *RCTFUN*, respetivamente, chamam-se *Left Control Transforms* e *Right Control Transforms & Unloads*. Estas classes servem para controlar o envio de transformações e/ou descargas. Já, as classes *LCETF* e *RCETFUN*, servem para receber a confirmação da finalização das ordens e atualizar os valores da DB.
- *Package OPC_UA* - Troca de informações com o PLC, envio e receção de determinadas variáveis necessárias ao controlo dos processos na fábrica.
- *Package UDP* - A classe *receiveUDP* recebe e trata todos os pacotes recebidos através da socket criada pela classe *clientUDP*. A primeira cria e lê o ficheiro "*receiveOrdersXML.xml*", interpretando se foram enviadas novas ordens se devem ser enviados os ficheiros "*sendRequestOrders.xml*" ou "*sendCurrentStores.xml*", para o mesmo endereço de IP e porta.

2.2 PLC

O PLC está dividido em duas células independentes, à semelhança da planta da fábrica: a esquerda, apenas capaz de operações de transformação; e a direita, capaz de transformar, carregar e descarregar peças.

O PLC é responsável pela gestão da parte "física" da planta da fábrica. Para isso, foram criadas várias classes que agregam os diferentes tipos de tapete. Estas codificam os seus comportamentos esperados, nomeadamente as suas respostas a diversos estímulos de sensores da fábrica, estados de tapetes interligados e de variáveis internas. Foram criadas com recurso a *function blocks* do *CODESYS*, através de *Sequential Function Charts*. As ações mais complexas foram programadas em *Structured Text*. As máquinas e os *pushers* são tratados de forma semelhante aos restantes tapetes, uma vez que o acionamento das suas ferramentas está diretamente associado ao comportamento do tapete em que estão incluídos.

Para além do acionamento dos tapetes, o PLC é responsável pelo tratamento das ordens que recebe. A alocação da célula que deve tratar uma determinada transformação é feita pelo MES, no entanto. Isto implica, em primeiro lugar, que ambas as células sejam capazes de retirar as peças necessárias do armazém, bem como de as colocar no fim da operação. Assim, o armazém é o único elo em comum entre as células. A distribuição das peças de uma determinada encomenda pelas diferentes máquinas é implementada com recurso a um esquema de reservas de ferramentas e é regulado pelos tapetes que se encontram antes das máquinas (*pre_machine* na esquerda e *rot_mo* na direita). O diagrama da Figura 7 deve ser consultado como suporte a esta explicação.

3 FUNCIONAMENTO DO CÓDIGO

Em primeiro lugar, cada ordem proveniente do sistema de envio de ordens ERP é tratada pelo MES. Como já foi referido anteriormente, este está encarregue de organizar as ordens por prioridade temporal, guardar na base de dados as informações das mesmas e, posteriormente, enviá-las ao PLC para serem executadas.

Relativamente às diversas interacções entre os elementos do MES e do PLC, estão disponibilizados, em anexo, os diferentes diagramas de atividade e/ou de sequência correspondentes a cada tarefa. Pelo que, nesta secção, apenas são referidos alguns pormenores importantes.

3.1 Interface Gráfica do MES

A primeira *frame* irá pedir ao usuário que coloque o endereço IP do computador que irá correr o PLC. Num ambiente fabril, a interface gráfica fica quase sempre afastada do PLC, pelo que foi considerada importante a implementação desta opção. Desta forma, também se pode abrir a interface num computador diferente daquele que está a ser utilizado para o MES.

A segunda *frame* apenas serve para ligar ou desligar o MES. Ao clicar no botão START, o programa irá iniciar e quando receber a primeira ordem, irá surgir a seguinte mensagem: "Everything is fine!". Além disso, após o arranque do MES, o botão START serve como um botão de *Status*. Clicando nele, aparecerá uma mensagem de erro caso ocorra uma *exception* numa das threads. Ao pressionar o botão STOP, o sistema desliga-se.

3.2 Arranque Inicial ou Restart

O código desenvolvido permite ao MES saber se este está a ser iniciado pela primeira vez ou se está a ser reiniciado, através da informação da base de dados. Isto é, caso não haja ordens por executar ou em execução, assume uma primeira inicialização. Caso contrário, o *MESInitTime* é o valor que ficou, anteriormente, guardado na base de dados.

3.3 Definição da prioridade das ordens de transformação

O MES define a prioridade através da ordenação pelo menor valor do atraso máximo real. Em caso de empate, utiliza-se a penalidade como critério de desempate. Isto é, para o mesmo atraso, selecciona-se primeiro a transformação com penalidade maior. O atraso máximo real, *RealMaxDelay*, é calculado através da seguinte fórmula:

$$RealMaxDelay = InitialMaxDelay - (MESTime - MESInitTime) - TimeExceptedToTransform \quad (1)$$

Ou seja, este atraso máximo real, é, na verdade, o tempo de segurança que ainda se tem para fabricar a transformação. Pelo que, quanto maior esse tempo de segurança for, menor será a sua prioridade.

3.4 Ordem de transformação

Existem três diagramas que ilustram as ordens de transformação. Primeiramente, um diagrama de atividades (Figura 10) para uma melhor explicação das atividades realizadas para completar uma ordem de transformação, sobretudo no caso de uma ordem difícil. Em seguida, existe um diagrama de sequência (Figura 11), onde é exposta a sequência das interações entre os objetos do sistema, desde a receção de uma ordem pelo sistema de envio de ordens ERP até à finalização do processamento da mesma. Por fim, um diagrama de estados para caracterizar a distribuição das peças pelas máquinas que o PLC efetua (Figura 12).

Para minimizar possíveis casos de bloqueio de passagem de peças para descarga, que são prioritárias, não se permite que o lado direito da fábrica realize mais de 2 ordens de transformação em simultâneo.

3.4.1 Ordens difíceis

Estas são ordens que têm mais do que 4 transformações (ex.: P1 para P7). Devido a não poderem ser realizadas sequencialmente sem utilizar pelo menos uma máquina mais do que uma vez, desenvolveu-se uma solução para contornar este problema. A transformação será dividida em duas partes: a primeira, com as 4 primeiras transformações; a segunda, com as restantes (1 ou 2, no máximo). Este procedimento também é ilustrado na Figura 12.

3.5 Controlo sobre as transformações

Quando é enviada uma ordem de transformação para o PLC, esta passa automaticamente para a tabela nomeada de *ElapseTransform*, que guarda as transformações que estão a decorrer. Os valores adicionados aí, além dos valores iniciais da ordem de transformação, são: o tempo de envio da ordem, o lado onde vai ser processada e ainda a quantidade de peças por produzir.

No fim da transformação, o MES coloca essa ordem na tabela *EndTransform*, onde armazena toda a informação resultante do processo: valores iniciais; tempo de transformação; penalidade incorrida; tempo (s) em que chegou ao MES; tempo (s) em que foi enviada para processar; tempo (s) em que acabou; e a hora em que acabou.

Desta forma, consegue-se corresponder às necessidades de persistência requeridas pelo caderno de encargos.

3.6 Request Stores e Request Orders

Relativamente aos ficheiros *Request Stores* e *Request Orders*, é possível ver nas Figuras 17 e 18, respetivamente, a sequência de processamento desde o pedido até ao envio dos ficheiros. É ainda importante referir a utilização do método de *parsing* DOM uma vez que facilita o manuseamento dos ficheiros XML.

3.7 Estatísticas

O MES tem duas threads sempre dispostas a atuar, quer no lado esquerdo quer no lado direito, caso se termine uma transformação ou descarga. O PLC envia um número de ordem diferente de 0 com um valor de transformação também diferente de 0. Isto significa que a transformação já foi processada e, portanto, atualiza-se, automaticamente, todos os valores estatísticos nesse patamar. Para as descargas, o PLC envia um booleano caso algum *Pusher* atue, incrementando assim as peças descarregadas. A sequência das interações entre os objetos do sistema pode ser vista com detalhe na Figura 16.

4 IMPLEMENTAÇÃO

Para a implementação do sistema de automatização decidiu-se utilizar o software *CODESYS*, uma vez que este permite implementar os requisitos impostos pelo projeto. Um dos pré-requisitos é o suporte de comunicações *Modbus/TCP*, visto que os atuadores e sensores do simulador fornecido para

o projeto utilizam esse protocolo para comunicarem. Um outro pré-requisito é a comunicação entre o PLC e o MES, através do protocolo OPC-UA. Devido a estes fatores, o *CODESYS*, que suporta estes protocolos e foi aconselhado pelos docentes da unidade curricular, foi escolhido para o desenvolvimento do projeto. Além disso, este software permite a utilização de uma norma de programação com a qual estávamos familiarizados, a norma IEC 61131-3. Esta norma permite implementar classes e objetos (*function blocks*) que são extremamente úteis na implementação de um programa modular.

Relativamente ao MES, utilizou-se a linguagem de programação JAVA, uma vez que, nativamente, já existem bibliotecas que permitem facilmente implementar os protocolos de comunicação que são exigidos para este trabalho. Para além disso, para um projeto deste tipo, é necessária uma linguagem de programação orientada a objetos.

Para a receção das ordens e o envio de ficheiros para o ERP, é necessário utilizar o protocolo UDP. Para a leitura e a análise dos ficheiros XML recebidos, utilizou-se o método de *parsing DOM* (Document Object Model), visto que facilita o manuseamento dos mesmos. Para a escrita destes ficheiros e para o envio de ficheiros XML, utiliza-se o mesmo método. É importante referir que a implementação do *clientUDP* permite enviar os pacotes para o mesmo IP e para a mesma porta que enviou a instrução anterior.

Relativamente à comunicação com o sistema de automatização, é fundamental, como foi mencionado anteriormente, o suporte de comunicações do tipo OPC-UA. Ainda que o JAVA não apresente uma biblioteca deste tipo de comunicações de forma nativa, usou-se como dependência a biblioteca Eclipse MILO OPC-UA. Para o controlo da fábrica ser bem implementado, variáveis como o tempo de cada transformação e ACKs (para um funcionamento seguro das comunicações) devem circular entre o MES e o PLC. Um exemplo disto é o facto de o PLC só dar uma transformação por terminada, quando o MES confirma que foi notificado acerca da sua entrada no armazém. De salientar as ordens são enviadas por vetores com recurso a este protocolo.

Para a implementação da base de dados, utilizou-se uma base de dados externa (não se utilizou uma alojada nos servidores da FEUP) baseada em *PostgreSQL*. O servidor externo está alojado na *Amazon WebServices*, através da plataforma *Heroku*. Optou-se por este processo, visto que não se pretendia que o sistema global ficasse dependente da VPN da FEUP, para evitar a inconsistência dos servidores da faculdade.

Adicionalmente, para as duas interfaces gráficas (a que permite a consulta das estatísticas e informações e a de inicialização do MES), utilizou-se a biblioteca *JAVA SWING*.

É crucial correr 6 elementos em paralelo:

- UDP
- Controlo de Entrada do Lado Esquerdo
- Controlo de Entrada do Lado Direito
- Controlo de Saída do Lado Esquerdo
- Controlo de Saída do Lado Direito
- Interface Gráfica para leitura das Estatísticas

Isto permite corresponder à necessidade existente do controlo do simulador ser um processo contínuo. Para além disso, surgiu a ideia de desenvolver a interface gráfica ser um ficheiro executável

(*ii.ui.jar*) totalmente independente do MES (que engloba as outras 5 threads). Esta interface gráfica apenas comunica com a base de dados.

Para as restantes 5 ações correrem de modo eficaz, o MES, quando iniciado, cria uma thread para cada processo (*thrUDP*, *thrLCTF*, *thrLCETF*, *thrRCTFUN*, *thrRCETFUN*). O protocolo OPC_UA é usado nas últimas 4 threads, daí não fazer sentido criar uma thread só para essa comunicação.

Desta forma, a *thrUDP* estará sempre "à escuta" para receber informações do ERP. Se recebe transformações ou descargas, coloca-as na base de dados nas tabelas **Transform** e **Unload**. Se recebe um pedido de informação, cria o ficheiro necessário com base nas informações que se encontram na base de dados, naquele momento específico.

Já as threads *thrLCTF* e *thrRCTFUN*, mal possam enviar novas ordens para o PLC, removem essa ordem da lista de espera, colocando-a na tabela **ElapseTransform** ou **EndUnload**. As outras duas threads aguardam pela comunicação do PLC acerca de novas entradas no armazém, para atualizarem valores e confirmarem o encerramento de ordens.

5 ESTRUTURA FINAL VS INICIAL

Antes de abordar esta secção, é importante referir que este foi o primeiro projeto onde nos foi pedido para desenhar uma arquitetura antes da implementação de qualquer tipo de função. Desta forma, há um número considerável de diferenças entre a arquitetura projetada inicialmente e a arquitetura final.

5.1 MES

5.1.1 Interface Gráfica / MES

Desde o início, a ideia de ter uma interface para iniciar o projeto estava presente. A única alteração prende-se com o número de threads utilizadas. No início, idealizaram-se 2 threads: uma para o UDP e outra para comunicar e trabalhar com o PLC. A partir do momento em que se começou a interagir com o SFS, sentiu-se a necessidade de otimizar o processo, ou seja, tornar o lado esquerdo da fábrica totalmente independente do lado direito, assim como o controlo de envio e o controlo de receção de transformações e/ou descargas.

5.1.2 OPC-UA

O caderno de encargos era bastante claro no que pretendia com o protocolo OPC-UA, pelo que a única alteração foi criar as variáveis que comunicam entre PLC e MES.

5.1.3 DB

Sabia-se que era necessário um serviço de base de dados para cumprir os requisitos de persistência. Com isso, apenas alguns métodos foram acrescentados à classe *dbConnect*.

5.1.4 UDP

Em relação à estrutura inicial, a única diferença foi o modo como analisávamos e escrevíamos o ficheiro .XML .

5.1.5 DATA & NEXT

Estas duas subsecções na arquitetura inicial encontravam-se na mesma package denominada **"DATA"**. Devido às necessidades de controlo exigidas pelo projeto, foi fundamental dividir esta subsecção em duas. A package **NEXT** veio acrescentar uma melhor dinâmica no controlo da ordenação e envio das ordens de transformação e descarga, enquanto que a package **DATA** passa agora a ter como único propósito trabalhar os dados de cada ordem. Algo muito mais simples em termos de implementação do que o projetado na arquitetura inicial.

5.2 PLC

A arquitetura relativa a este ponto sofreu alterações consideráveis em relação à estrutura projetada inicialmente. Grande parte dessas alterações relaciona-se com o surgimento de limitações inicialmente desconhecidas, tanto no MES como no PLC. As decisões finais foram sempre baseadas no custo-benefício da implementação de uma dada característica.

Inicialmente, todo o controlo do movimento de peças e troca de ferramentas foi estruturado para ser responsabilidade do MES. A passagem constante de parâmetros, bem como a sua quantidade, foi considerada problemática e desnecessária. Optou-se apenas pelo envio da ordem (com o respetivo tipo de peça, operação e quantidade) para o PLC, sendo este responsável por todas as decisões ao nível da reserva das máquinas e respetivas trocas de ferramentas. A utilização de variáveis internas é mais simples, evitando uma série de problemas, como a utilização de variáveis redundantes para confirmar a receção de parâmetros que são comunicados, permitindo obter os mesmos resultados.

Outra das alterações que surgiu está relacionada com o tratamento das chamadas peças "difíceis", isto é, peças que exigem mais de 4 maquinações. Sempre se supôs, tal como é normal em todas as fábricas, um tratamento sequencial das encomendas. Isto permite evitar que peças circulem contra o fluxo natural de progressão (da saída do armazém para a entrada), uma vez que esse movimento implica a reserva de tapetes para evitar a colisão de peças. Assim, o pensamento inicial do grupo consistia na mudança de ferramentas, de modo a poder utilizar as 4 máquinas para 5 ou 6 maquinações. No entanto, o tempo de mudança de máquina não era viável, sendo um *bottleneck* considerável, gerando filas no fim da zona de maquinação. Por fim, optou-se por "partir" estas ordens em duas, assegurando a reposição do *stock* intermédio, bem como a não utilização de mais de 4 máquinas para cada um dos fragmentos da ordem original.

6 COMPARAÇÃO COM PADRÕES EXISTENTES

6.1 ISA 95

Relativamente à norma ISA 95, podem-se fazer comparações em relação ao modelo funcional de controlo e ao fluxo de dados. No que toca ao modelo funcional, não se abordaram as questões de manutenção, pesquisa, qualidade ou recursos, sendo que as funções tratadas foram:

- *Receiving, Order Processing*: podem ser mapeadas pelos receiveUDP por tratarem da interpretação, encaminhamento e criação de ordens;
- *Plant Product Scheduling, Dispatching*: pelas classes (LCTF e RCTF, por escalonarem as ordens e gerirem o seu estado (pendente e em execução));

- *Production Control & Tracking*: é implementado pelo (RCETFUN e LCETF), já que estas classes monitorizam o que decorre na fábrica, analisando a produtividade, verificando sempre as estatísticas e quando uma ordem acaba;
- *Production Inventory Control*: relaciona-se com a package DB, por controlar o stock.
- *Shipping, Production Processes*: tratado no PLC.

No que respeita ao fluxo de dados, dos 31 podem salientar-se:

- *Finished Goods Inventory*, com o registo de peças disponíveis, pelo armazém;
- *Incoming Order*, pela classe *receiveUDP*;
- *Production Schedule*, pelas prioridades atribuídas pelas classes *LCTF*, *RCTFUN*;
- *Product and Process Information Request* pelas estatísticas geradas e pelos ficheiros .XML: *request stores* e *request orders*;
- *Production Performance And Costs*, cada transformação tem um atraso máximo e uma penalidade associada.

6.2 RAMI 4.0

A norma RAMI 4.0 constitui um modelo para aplicações de soluções de conectividade para projetos aderentes à Indústria 4.0, permitindo um ecossistema descentralizado de toda cadeia produtiva. A arquitetura proposta pode ser comparada com a respetiva norma em 3 aspetos: hierarquia, tempo de vida ou *product life cycle* e arquitetura e camadas constituintes.

Relativamente ao **eixo 1**, a hierarquia, a arquitetura desenvolvida pelo grupo é ainda um pouco centralizada. O sistema de ordem superior, MES, comunica com o Soft-PLC, que escolhe o melhor caminho para as peças de uma dada ordem. Numa sugestão mais descentralizada, as peças (*assets*) poderiam ter um papel ativo nas decisões, trazendo um conhecimento local dos sítios por onde passam e, por exemplo, as máquinas poderiam comunicar atempadamente com as peças para preparar as ferramentas necessárias. Ainda que as peças vão para máquinas com aquela ferramenta, por vezes há necessidade de mudar a ferramenta quando a peça chega ao tapete da máquina. As peças, à medida que são transformadas, teriam de ir definindo os próximos passos, segundo uma arquitetura direcionada para a Indústria 4.0, o que constitui um desafio adicional em termos de previsão e controlo.

Relativamente ao eixo ***product life cycle***, o foco principal foram as combinações possíveis para transformar entre peças (já sabidas à priori, no caderno de encargos).

No **eixo 3**, arquitetura, os dados a trocar são os tipos de transformações a realizar (p.e. P1->P2) e os resultados das mesmas, como o tempo de transformação, entre outras especificações pedidas no caderno de encargos. A integração e a comunicação entre MES e sistema de automatização ficou sob a alcada do protocolo OPC-UA, a informação na base de dados a comunicação com esta foi realizada por JDBC (*Java Database Connectivity*) e por fim, o ERP usa comunicações UDP. Não são abordadas as camadas funcional ou organizacional.

6.3 Artigo "Service Granularity in Industrial Automation and Control Systems"

O artigo em questão discute um dos principais desafios em "*Service oriented systems*", que é encontrar a perfeita granulação para cada serviço no sistema. A granulação destes serviços tem um efeito direto na monitorização, flexibilidade, custos de manutenção, bem como na escalabilidade. A granulação divide-se em *Process Granularity*, *Machine Granularity* e *Functionality Granularity*.

Comparando a nossa arquitetura com os padrões propostos pelo artigo, há uma divisão entre a arquitetura do sistema de automatização (PLC) e o MES.

Relativamente ao MES, a arquitetura e divisão dos blocos assemelha-se à abordagem "*Process Granularity*", uma vez que dividimos os processos de controlo em algumas partes necessárias ao controlo e monitorização da planta. Como já foi referido anteriormente, as threads:

- *thrUDP*, controla a socket UDP, uma vez que é pretendido que se esteja sempre à escuta para receber mais ordens do ERP. Estas ordens podem ser transformações e/ou descargas ou pedidos para enviar, através de um ficheiro .XML, informações relativas ao sistema;
- *thrLCTF* & *thrRCTFUN*, controlam o envio de ordens de transformação e/ou descarga.
- *thrLCETF* & *thrRCETFUN*, verificam a finalização das respetivas ordens e atualizam os valores na base de dados.

Por outro lado, relativamente ao sistema de automatização, a abordagem assemelha-se à "*Machine Granularity*", uma vez que a planta foi mapeada em pequenas "máquinas", tais como tapetes lineares, rotativos, de carga, descarga, máquinas, etc. Posteriormente, o sistema de automatização interage com estes blocos no seu todo, havendo alguns serviços a controlar diferentes partes de cada bloco. Por exemplo, um pedido de transformação envolve serviços ao nível dos tapetes como os sensores e actuadores, envolve o serviço de troca de ferramenta por parte das máquinas e por fim, o serviço de transportar as peças até ao armazém no final das transformações. Em determinados blocos, como as máquinas, é possível haver mais do que um serviço a atuar em simultâneo (utilização de ferramenta e reserva da mesma), o que se considera uma aproximação ao objetivo de obter "*more finegrained services*".

Em suma, neste projeto, a abordagem mais indicada seria optar por granulação híbrida, "*Hybrid Granularity*", visto que apenas um tipo de granularidade poderá não satisfazer todos os requerimentos do projeto. É importante, como mencionado no estudo, encontrar um "*trade-off*" entre granularidade e o custo de operações no sistema.

7 ORGANIZAÇÃO DA EQUIPA

Para o desenvolvimento deste projeto, adotou-se um desenvolvimento contínuo e sempre com interação entre os diversos elementos da equipa.

No inicio do projeto, todos colaboraram de igual forma para estudar o problema. Durante o mesmo processo, a elaboração do MES (incluindo as duas interfaces gráficas) ficou encarregue a 2 elementos fixos, o Guilherme Pinheiro e o Tomás Araújo; o desenvolvimento do sistema de automatização, com recurso ao *CODESYS*, foi responsabilidade do Francisco Caetano. Por fim, o quarto elemento do grupo, Francisco Damas, serviu como elemento móvel do grupo, procurando auxiliar o grupo com maior carga de trabalho, a dado momento. Contribuiu com a sua mais-valia de pensamento na obtenção de melhores resultados de otimização, ajudando também na interface gráfica das estatísticas.

Adotou-se um processo de SCRUM, isto é, definiram-se 4 sprints:

- **1º Sprint:** Desenvolvimento da Ideia - Duração de 3/4 semanas;
- **2º Sprint:** Desenvolvimento das transformações e comunicações - Duração de 3 semanas;
- **3º Sprint:** Desenvolvimento do resto dos pré-requisitos - Duração de 3 semanas;
- **4º Sprint:** Optimização do projeto e algumas melhorias - Duração de 4 semanas;

A partir de uma lista *TO DO* e de alguns *Use Cases* retirados do caderno de encargos, o grupo reuniu-se, semanalmente, para prever, planear e desenvolver cada ideia. Para ajudar a transição entre ficheiros, utilizou-se o *Git*.

Diversas plataformas foram utilizadas: Discord/Zoom (comunicação entre equipa) ; IntelliJ/Eclipse (desenvolvimento do MES); Heroku (base de dados e REST API); *CODESYS* (Soft-PLC); e, por fim, o GitHub.

Por fim, com base no percurso realizado até aqui e após a execução do projeto, o grupo concorda por unanimidade que o esforço de cada membro foi equitativamente dividido, tendo cada elemento dado a sua contribuição possível. Com isto, autoavalia-se de modo uniforme, ou seja, devem ser atribuídos 25 % a cada elemento do grupo.

| Distribuição de Esforço | | | | |
|-------------------------|-------------------|--------------------|-----------------|--|
| Tomás Araújo | Francisco Caetano | Guilherme Pinheiro | Francisco Damas | |
| 25% | 25% | 25% | 25% | |

Table 1: Distribuição da nota pelos elementos do grupo

8 TESTES E RESULTADOS

Para verificar a qualidade da implementação, realizaram-se os seguintes testes:

8.1 Teste 1 - Rapidez do Sistema

| From | To | Quantity | MaxDelay | Penalty | Transformation Time | Penalty Incurred | side |
|------|----|----------|----------|---------|---------------------|------------------|-------|
| 1 | 2 | 16 | 1000 | 500 | 152 | 0 | left |
| 1 | 7 | 4 | 1000 | 500 | 275 | 0 | right |

Table 2: Resultados para o teste de rapidez

Em cerca de 150 s, conseguiu transformar 16 peças P1 em P2. A transformação P1 para P7 é a transformação que mais tempo de transformação exige ao PLC. O sistema desenvolvido pelo grupo consegue realizar a ordem em 275s. Este teste é ilustrativo da mais-valia que a alocação adaptativa de ferramentas, bem como da independência das células, pode trazer ao sistema em termos de rapidez de processamento das encomendas.

8.2 Teste 2 - Cálculo da penalidade

Para a confirmação do cálculo da penalidade, reduziu-se o atraso máximo para 50 e considerou-se uma penalidade de 100 unidades. Das equações demonstradas na Secção 3, resulta:

| From | To | Quantity | MaxDelay | Penalty | TT | PI | side | InitMES | ST | ET |
|------|----|----------|----------|---------|----|-----|------|---------|-----|-----|
| 1 | 3 | 4 | 50 | 100 | 98 | 200 | left | 374 | 448 | 546 |

Table 3: Cálculo da penalidade

$$TotalT = (448 - 374) + 98 = 171(s); PI = (171 - 50)/50 \cdot 100 = 200; \quad (2)$$

O valor resultante será 242, no entanto, apenas deve ser considerado o quociente da divisão inteira, pelo que a penalidade incorrida será de 200. Assim, o valor obtido é o valor esperado.

8.3 Teste 3 - Manutenção do MESInitTime, após reinicio do MES

| From | To | InitMES | ST |
|------|----|---------|-----|
| 2 | 6 | 33 | 107 |

Table 4: Manutenção da coesão temporal

Para confirmar o devido reinicio do processo, através do ERP, enviou-se uma ordem de transformação. Antes que pudesse terminar o seu processamento, o MES foi desligado durante 60s. Após esse período, reiniciou-se o sistema de controlo.

Com base nessa paragem, o processo deveria começar em $33 + 60 = 93$, se o sistema não tivesse atrasos. Como existem alguns atrasos no seu arranque, a transformação apenas começou aos 107 s.

8.4 Teste 4 - Aplicação Real

| From | To | Quantity | MaxDelay | Penalty | TT | TimeMES | ST | ET | side |
|------|----|----------|----------|---------|-----|---------|------|------|-------|
| 3 | 8 | 1 | 50 | 100 | 129 | 144 | 145 | 275 | left |
| 1 | 8 | 1 | 10 | 1000 | 178 | 144 | 149 | 328 | right |
| 1 | 7 | 4 | 1000 | 500 | 275 | 330 | 331 | 606 | right |
| 1 | 3 | 8 | 50 | 100 | 146 | 330 | 331 | 477 | left |
| 1 | 3 | 4 | 50 | 100 | 98 | 374 | 448 | 546 | left |
| 5 | 6 | 6 | 50 | 100 | 108 | 374 | 448 | 557 | right |
| 4 | 5 | 3 | 50 | 100 | 61 | 555 | 560 | 622 | right |
| 5 | 6 | 9 | 50 | 100 | 124 | 555 | 561 | 685 | left |
| 2 | 8 | 4 | 1000 | 500 | 487 | 718 | 897 | 1384 | left |
| 5 | 6 | 8 | 1000 | 500 | 248 | 719 | 1198 | 1446 | left |
| 1 | 9 | 8 | 1000 | 500 | 596 | 718 | 986 | 1583 | right |
| 6 | 8 | 12 | 1000 | 500 | 200 | 719 | 1474 | 1675 | right |

Table 5: Transformações efetuadas no teste

| Type | Destination | Quantity |
|------|-------------|----------|
| 1 | 3 | 4 |
| 1 | 1 | 4 |

Table 6: Peças para descarga

Para o teste final, decidiu-se estudar o comportamento total do sistema. Desde a carga de peças, descargas, transformações, forçar o reinicio do MES e pedido de ficheiros. O tempo total do teste foi de, aproximadamente, 28 minutos.

Em suma, é possível concluir que o sistema total é robusto e seguro. A acrescentar a isso, o sistema tem todas as funcionalidades pedidas no caderno de encargos.

9 CONCLUSÃO

Ao longo do processo documentado neste relatório, foi possível aplicar os conhecimentos adquiridos na UC Informática Industrial, aliando-os a conhecimentos prévios obtidos em unidades curriculares anteriores.

Para além disso, foi possível adquirir mais sensibilidade acerca da implementação de um software de controlo, processamento que permite a automatização e gestão de uma linha de produção.

Durante a realização do projeto sentiram-se algumas dificuldades. A maior delas prende-se com a utilização de uma linguagem de programação orientada a objetos. Esta foi a primeira vez que fomos confrontados com a sua utilização numa UC, não tendo, consequentemente, uma formação curricular na sua aplicação. Isto conduziu a um arranque lento no desenvolvimento do MES, bem como a longas sessões de aprendizagem. Para além disso, algumas ambiguidades nos requisitos para o sistema levaram-nos, muitas vezes, a tomar decisões que não coincidiam com o verdadeiro desejo do cliente. Posteiros esclarecimentos conduziram-nos frequentemente a refazer extensas secções de código. Uma adversidade que será bastante comum em projetos reais. Compreende-se, também, que indicações demasiado específicas resultariam numa convergência das soluções propostas pelos diferentes grupos.

Apesar de tudo, foi implementado um sistema robusto e capaz de processar qualquer ordem válida que lhe chegue, de forma correta, segura (mesmo em caso de falha do sistema MES) e interativa (com implementação de uma interface gráfica de dados), produzindo bons resultados a nível temporal.

Considerando todas as especificações pedidas no caderno de encargos, consideramos que a melhor implementação para o projeto reside na descentralização das tarefas e independência suficiente que melhore a resposta às falhas. Estas preocupações vão de encontro aos objetivos da Indústria 4.0 e são requisitos com os quais nos devemos familiarizar.

Por fim, é importante salientar que o projeto foi bem sistematizado e que é possível distinguir as componentes de alto e de baixo nível, estando todas as interfaces bem definidas.

Em jeito de conclusão, resta-nos apenas afirmar que este projeto foi sobretudo uma experiência de aprendizagem. Para além de termos a oportunidade de trabalhar com diversas programas e técnicas que ainda nos eram desconhecidas, conseguimos expandir as nossas capacidades em temas nos quais ainda tínhamos uma parca experiência, nomeadamente no desenho de arquiteturas. A comparação com os padrões existentes a nível industrial é extremamente importante para entender aquilo que será esperado de engenheiros no mundo profissional. Atualmente, a existência de sistemas de controlo de automação para linhas de produção é essencial. Foi a primeira vez, ao longo da nossa experiência universitária, que tivemos a possibilidade de entregar um software funcional, executável em várias máquinas, que um possível cliente pudesse utilizar.

10 ANEXOS

10.1 Anexo 1

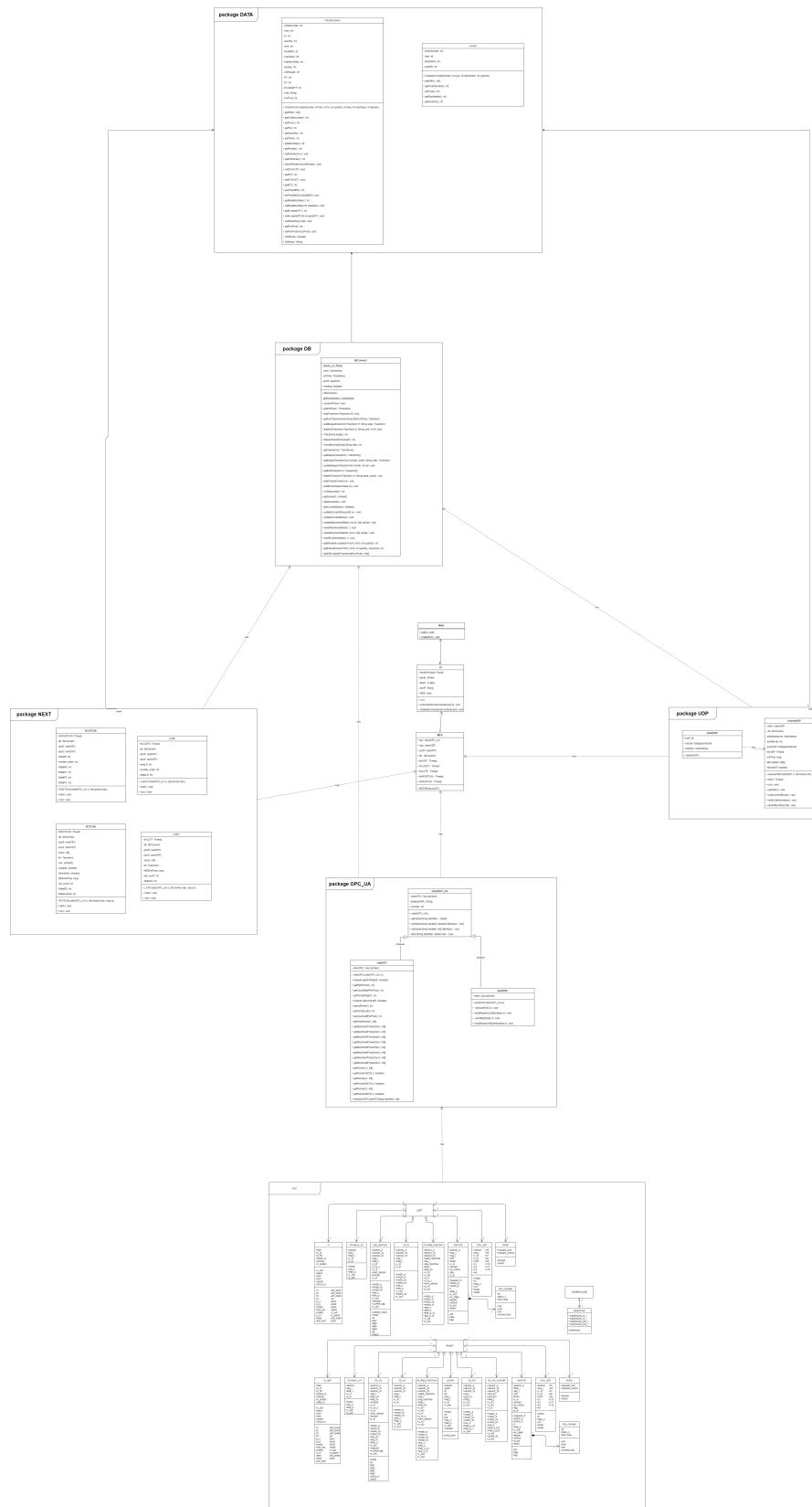


Figure 1: Diagrama de classes completo

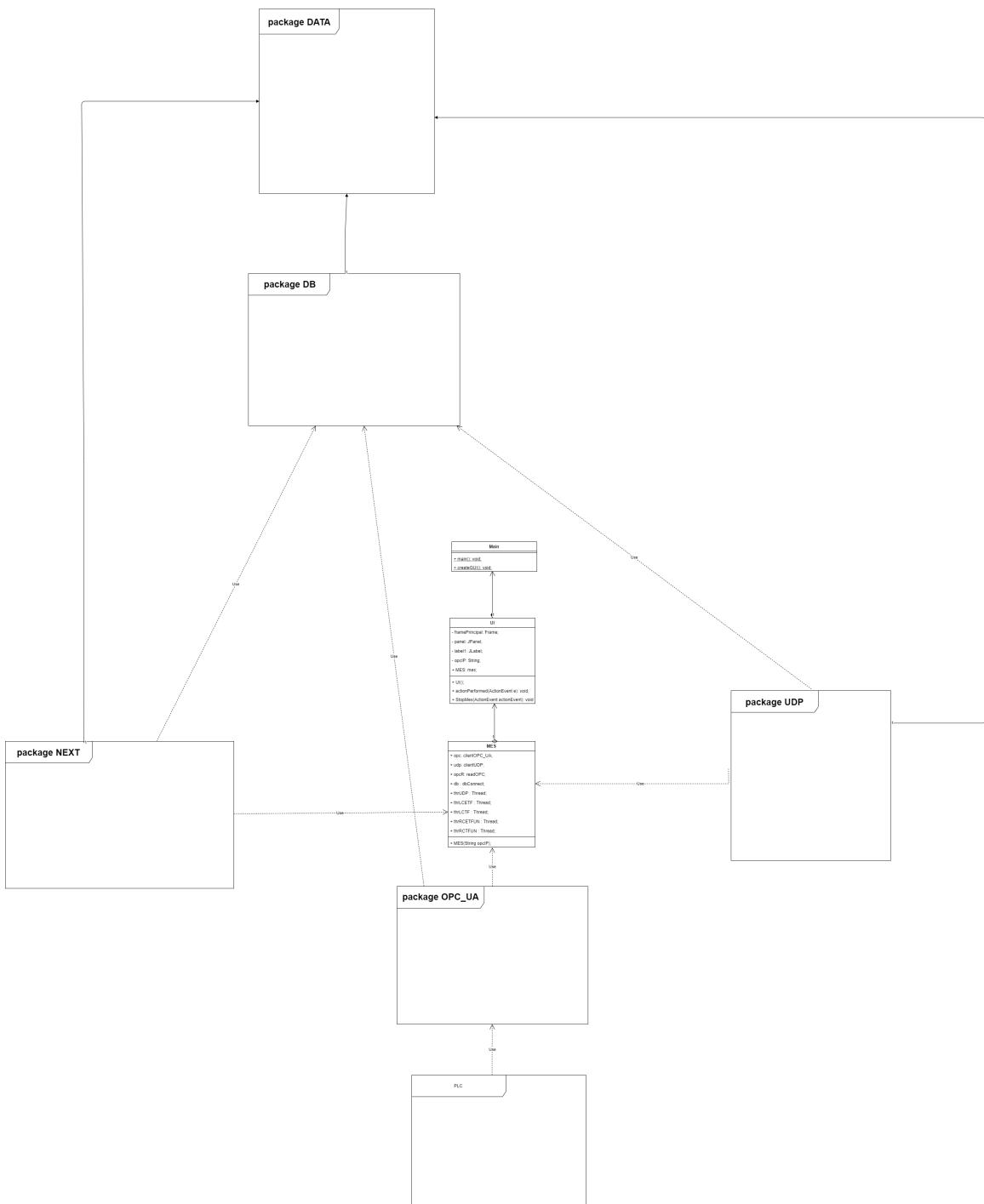


Figure 2: Diagrama de classes composto pelas packages

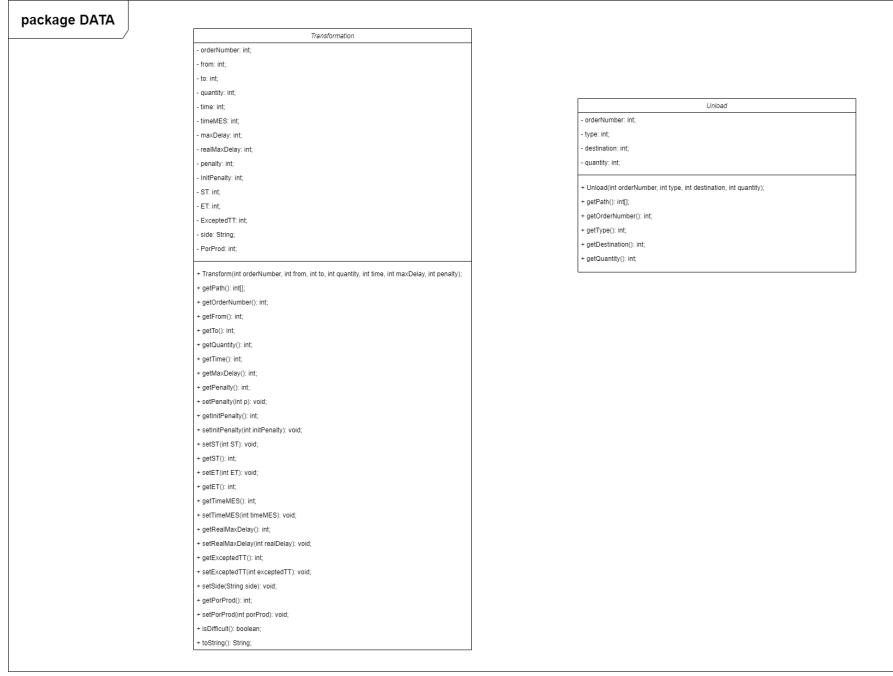


Figure 3: Package - Data



Figure 4: Package - DB

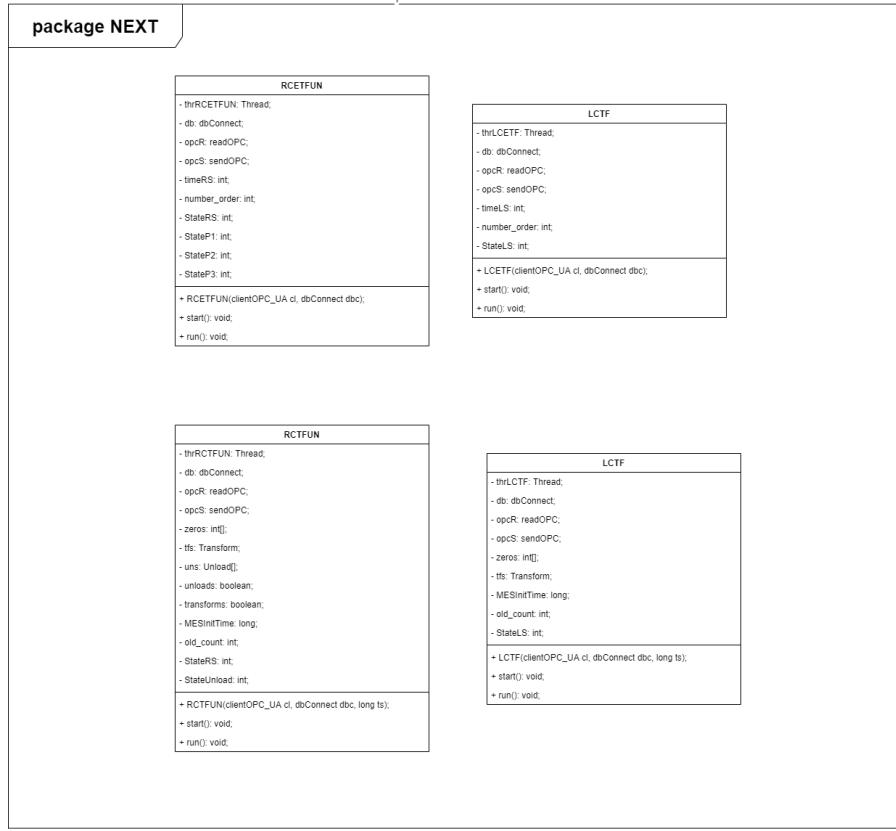


Figure 5: Package - Next

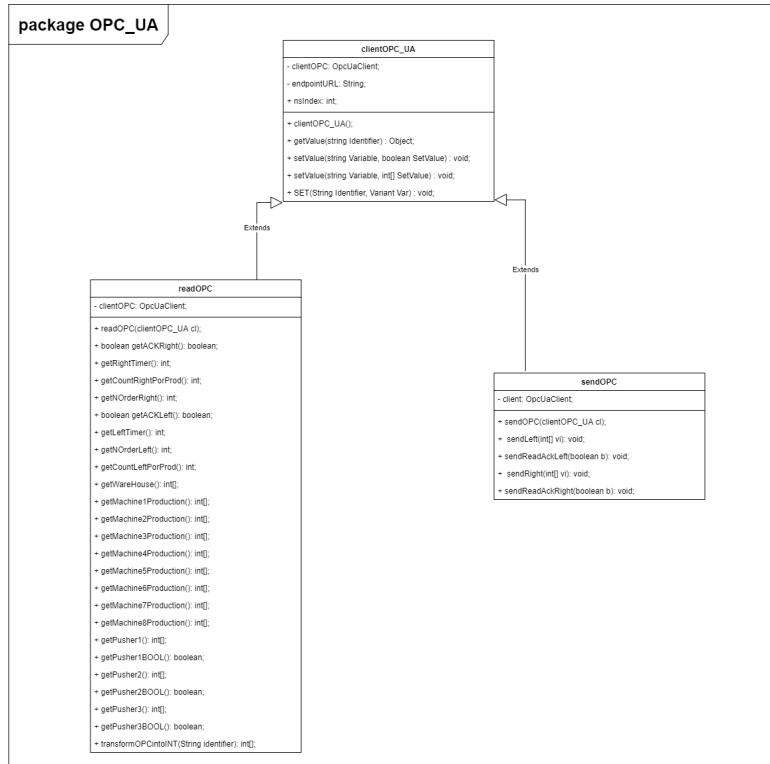


Figure 6: Package - OPC-UA

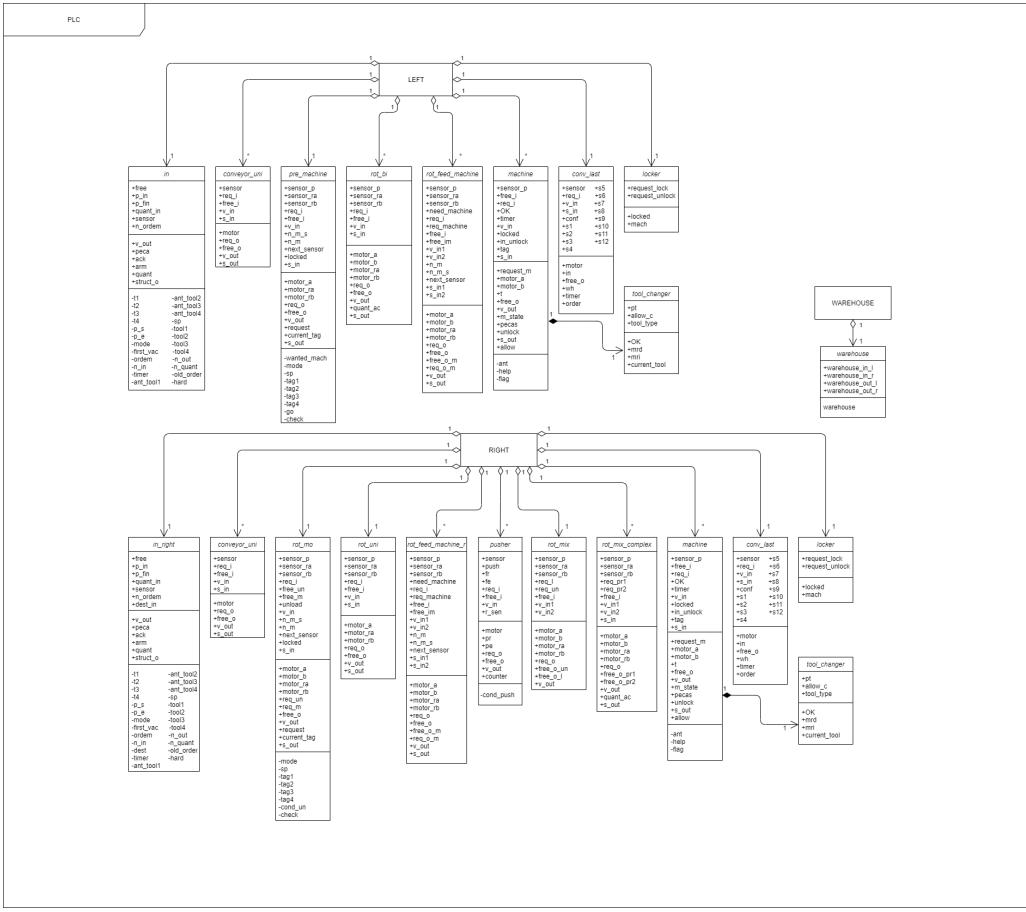


Figure 7: PLC

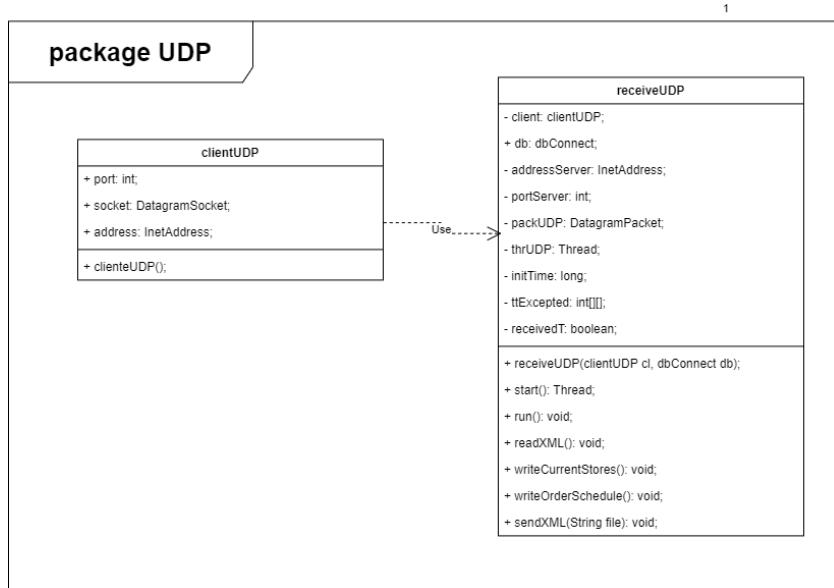


Figure 8: Package - UDP

10.2 Anexo 2

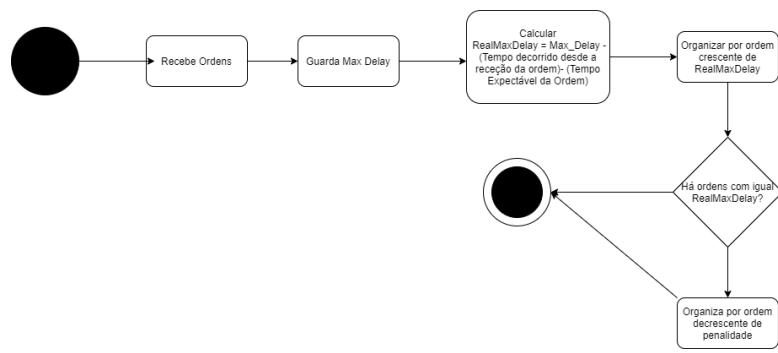


Figure 9: Prioridade das ordens

10.3 Anexo 3

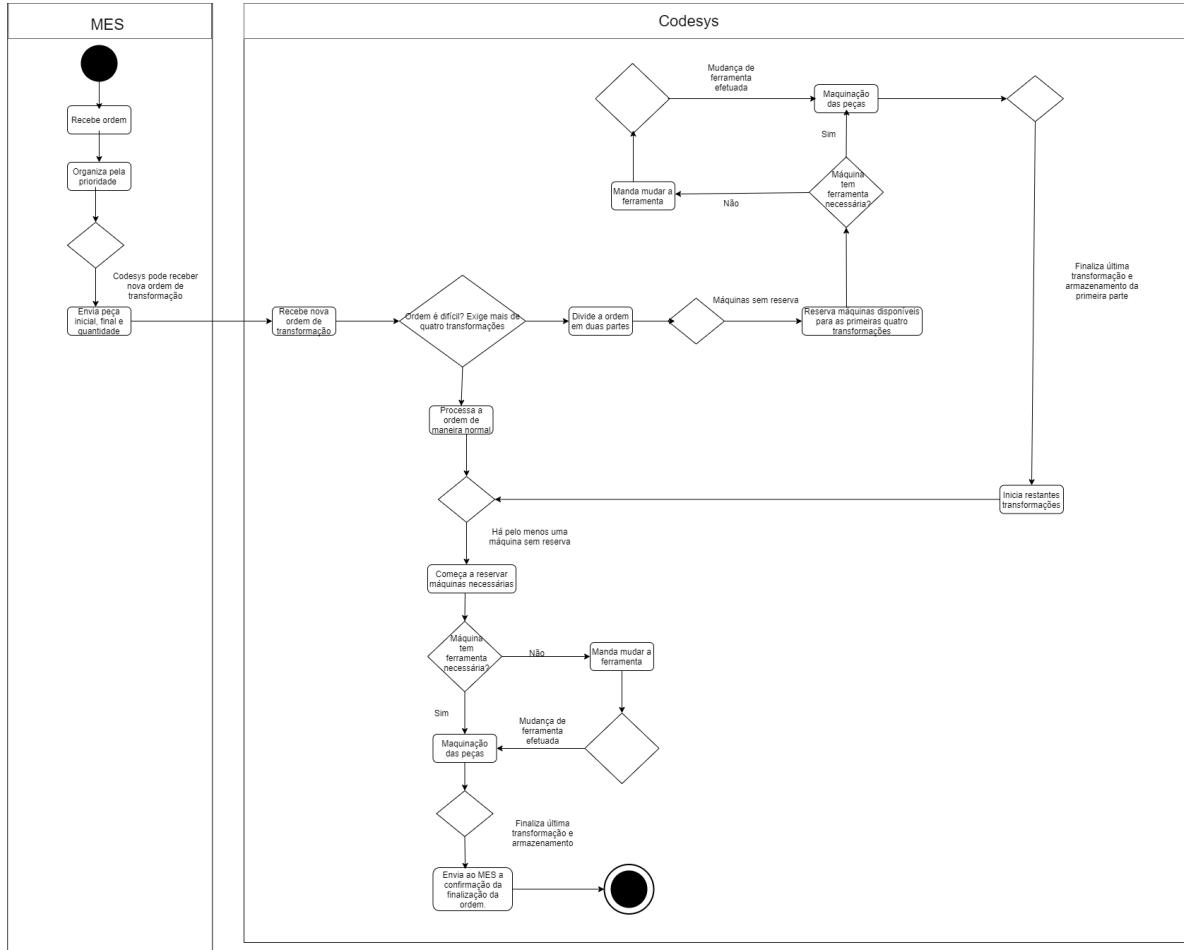


Figure 10: Ordens de transformação

10.4 Anexo 4

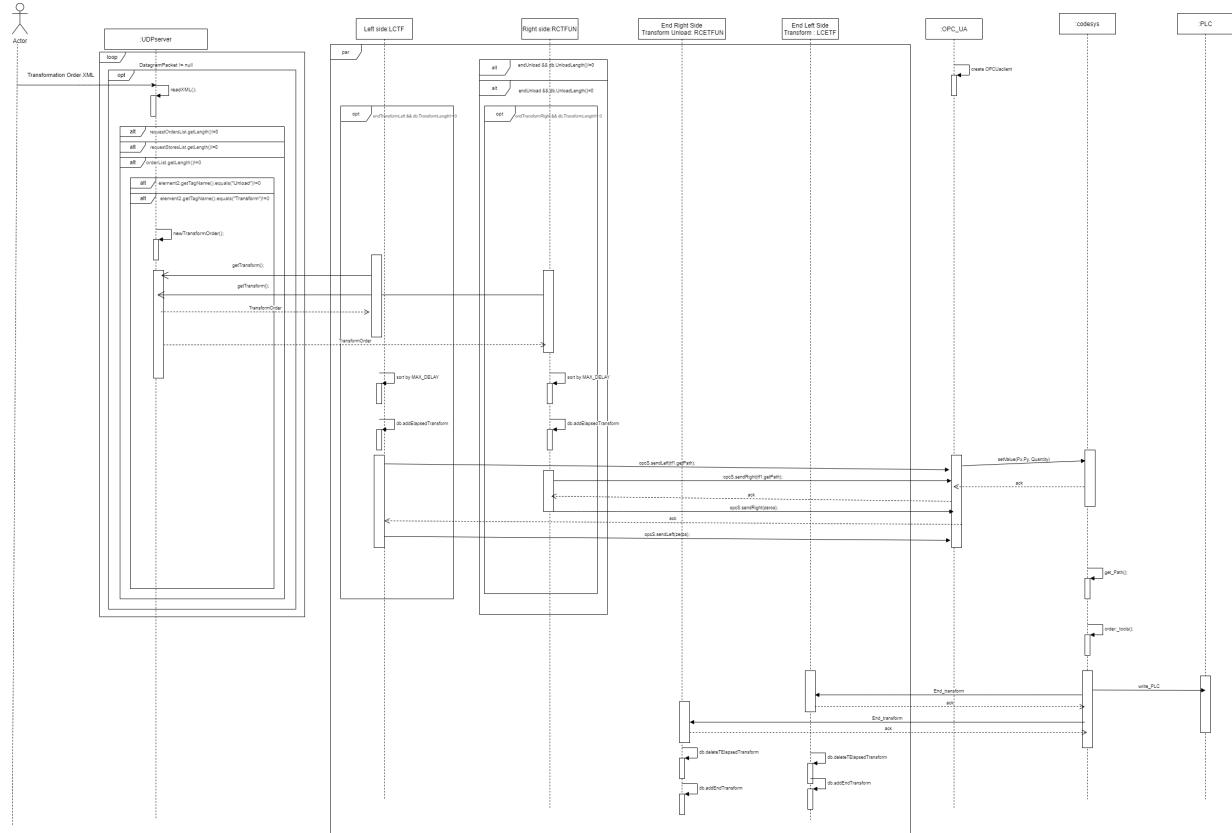


Figure 11: Ordens de transformação

10.5 Anexo 5

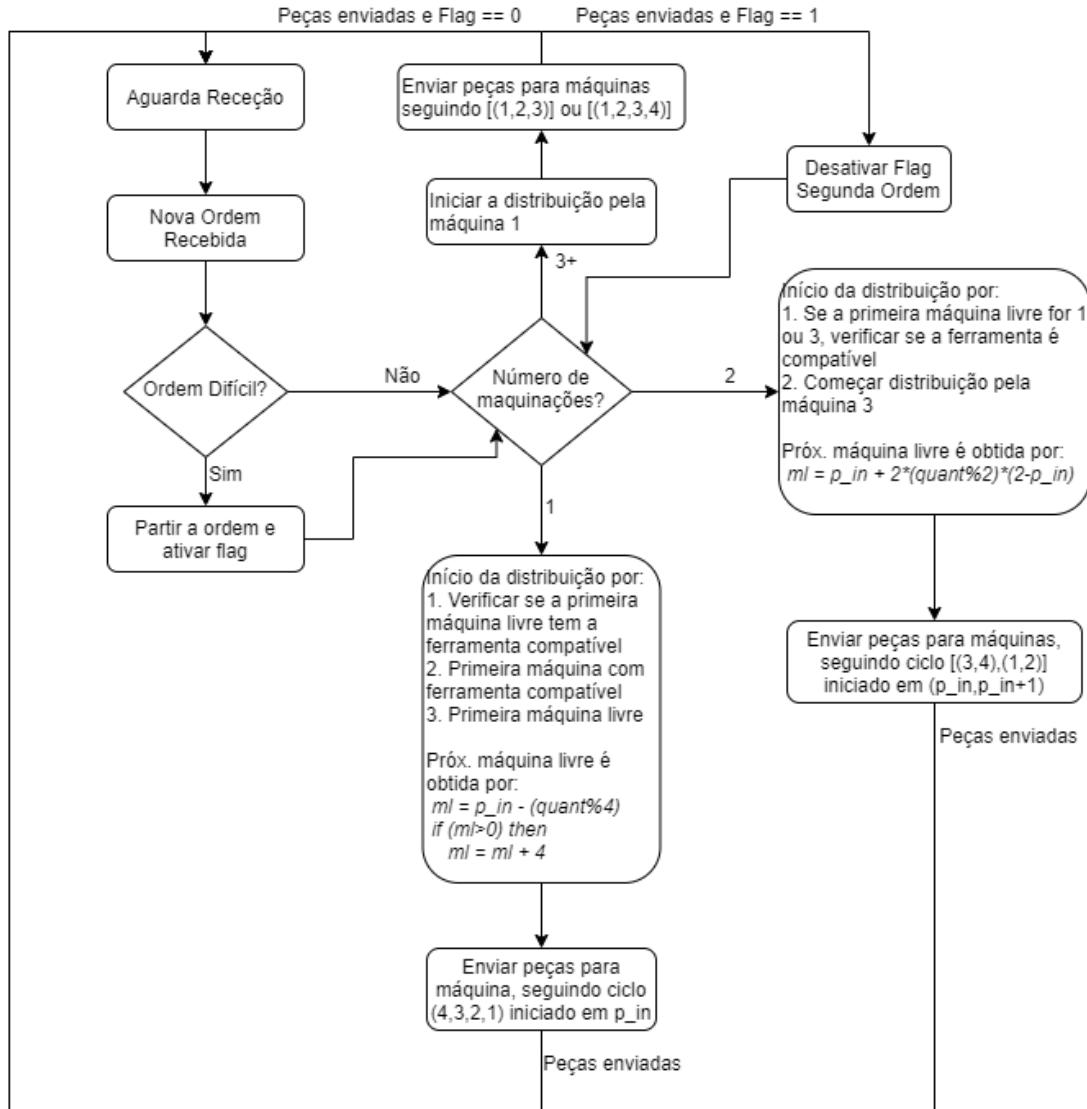


Figure 12: Diagrama de estado das ordens de transformação

10.6 Anexo 6

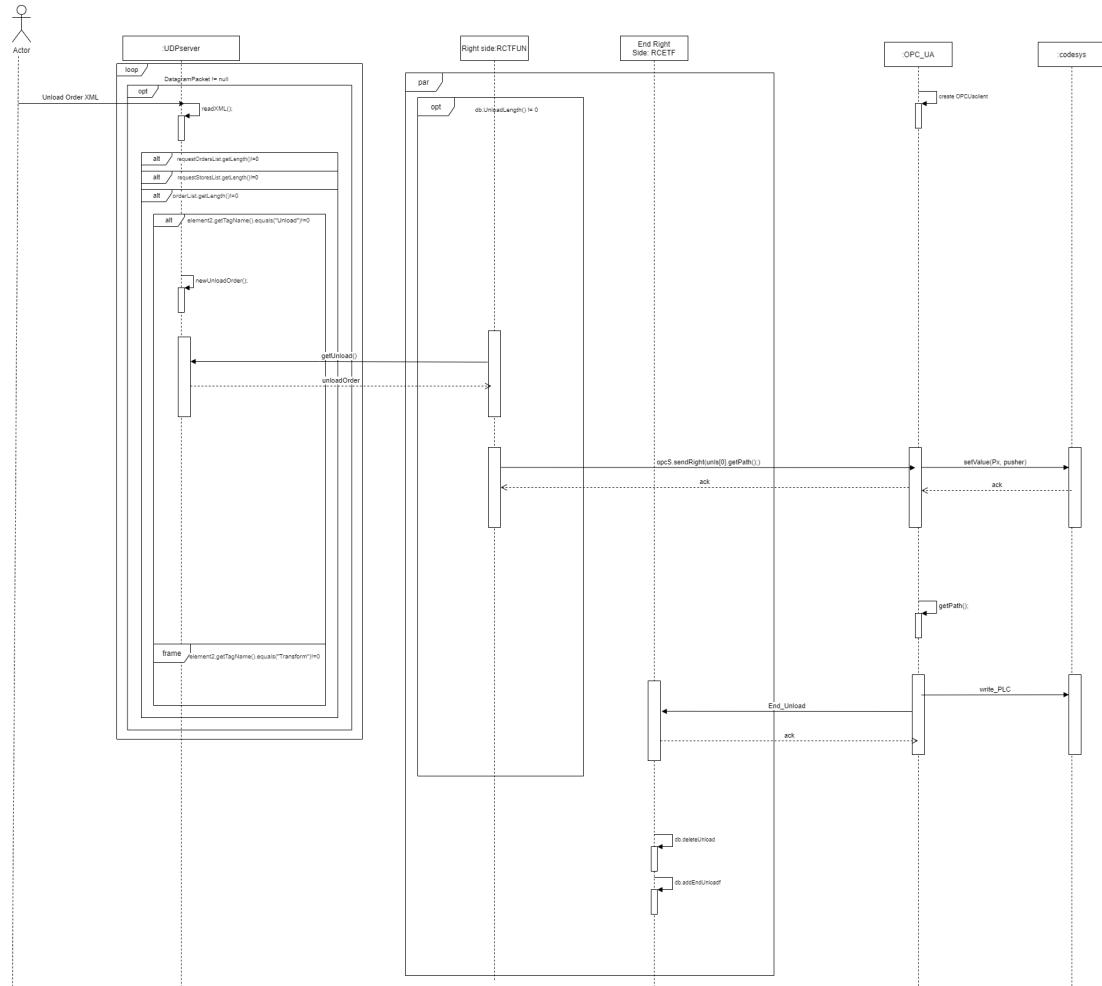


Figure 13: Ordens de descarga

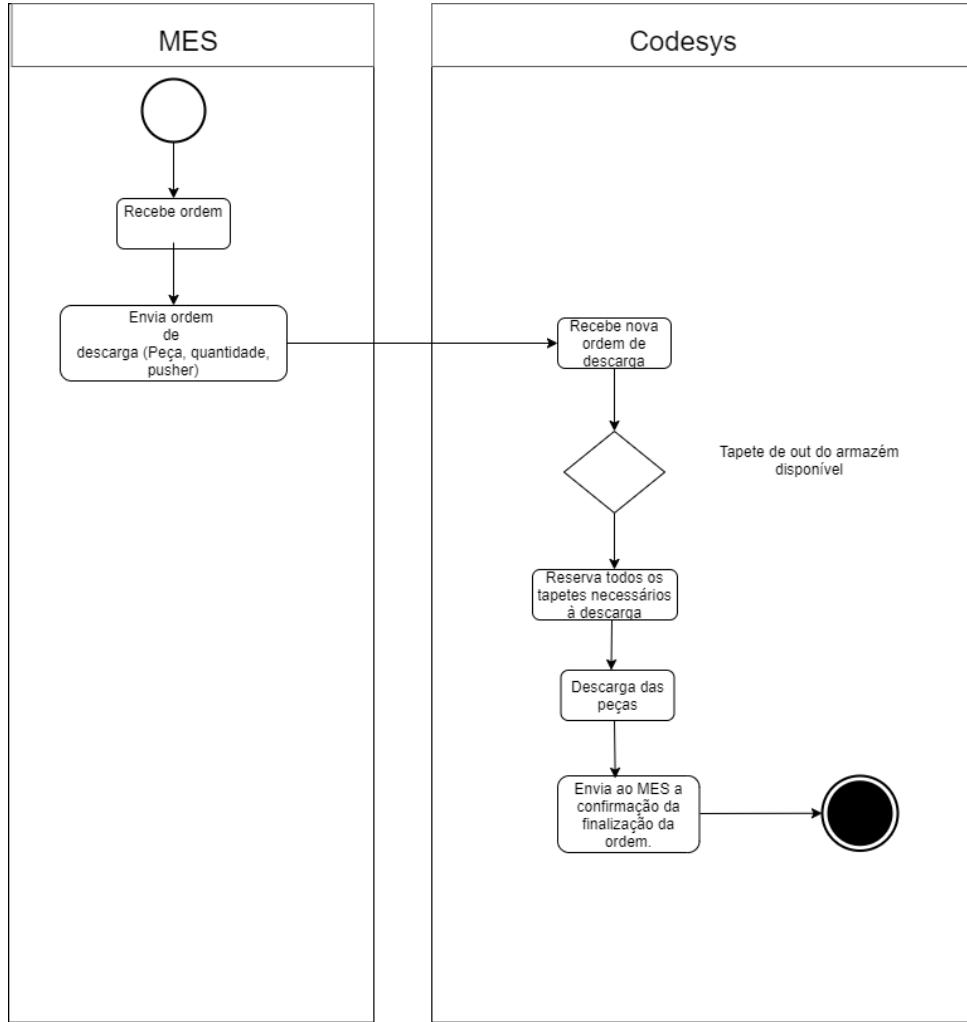


Figure 14: Ordens de descarga

10.8 Anexo 8

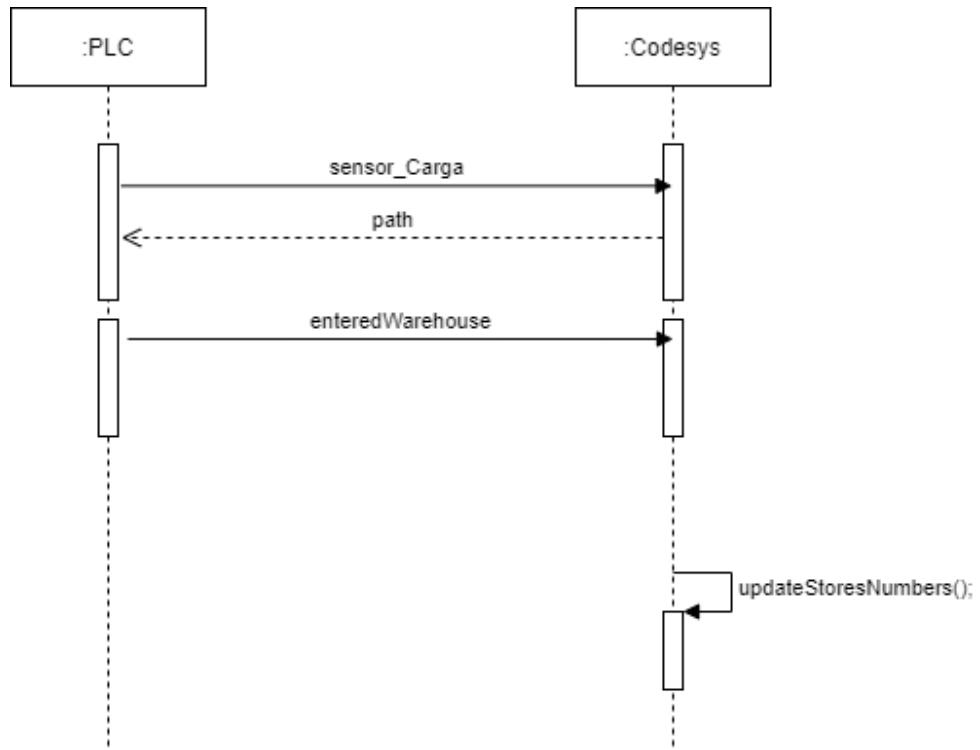


Figure 15: Ordens de carga

10.9 Anexo 9

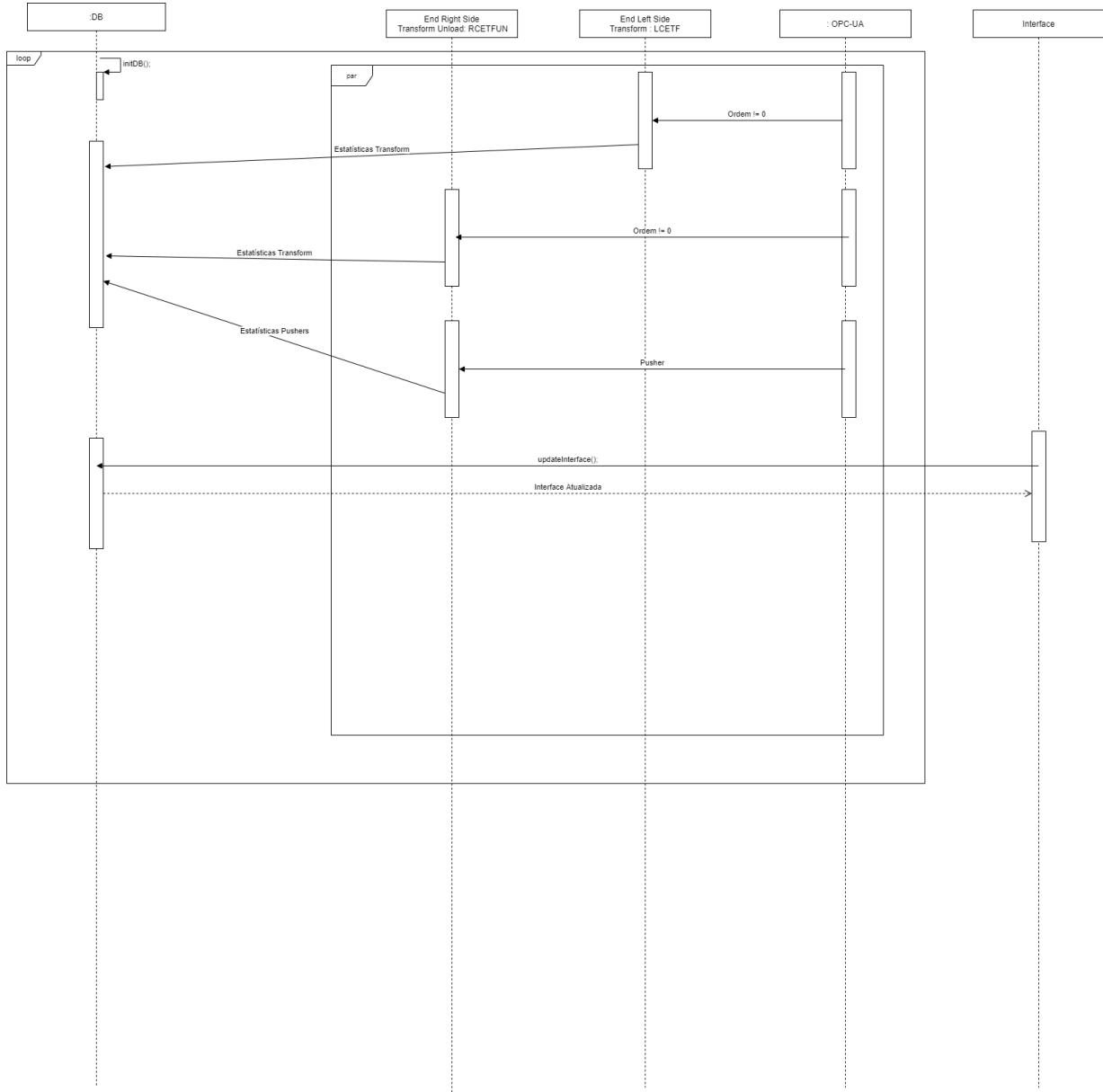


Figure 16: Estatísticas

10.10 Anexo 10

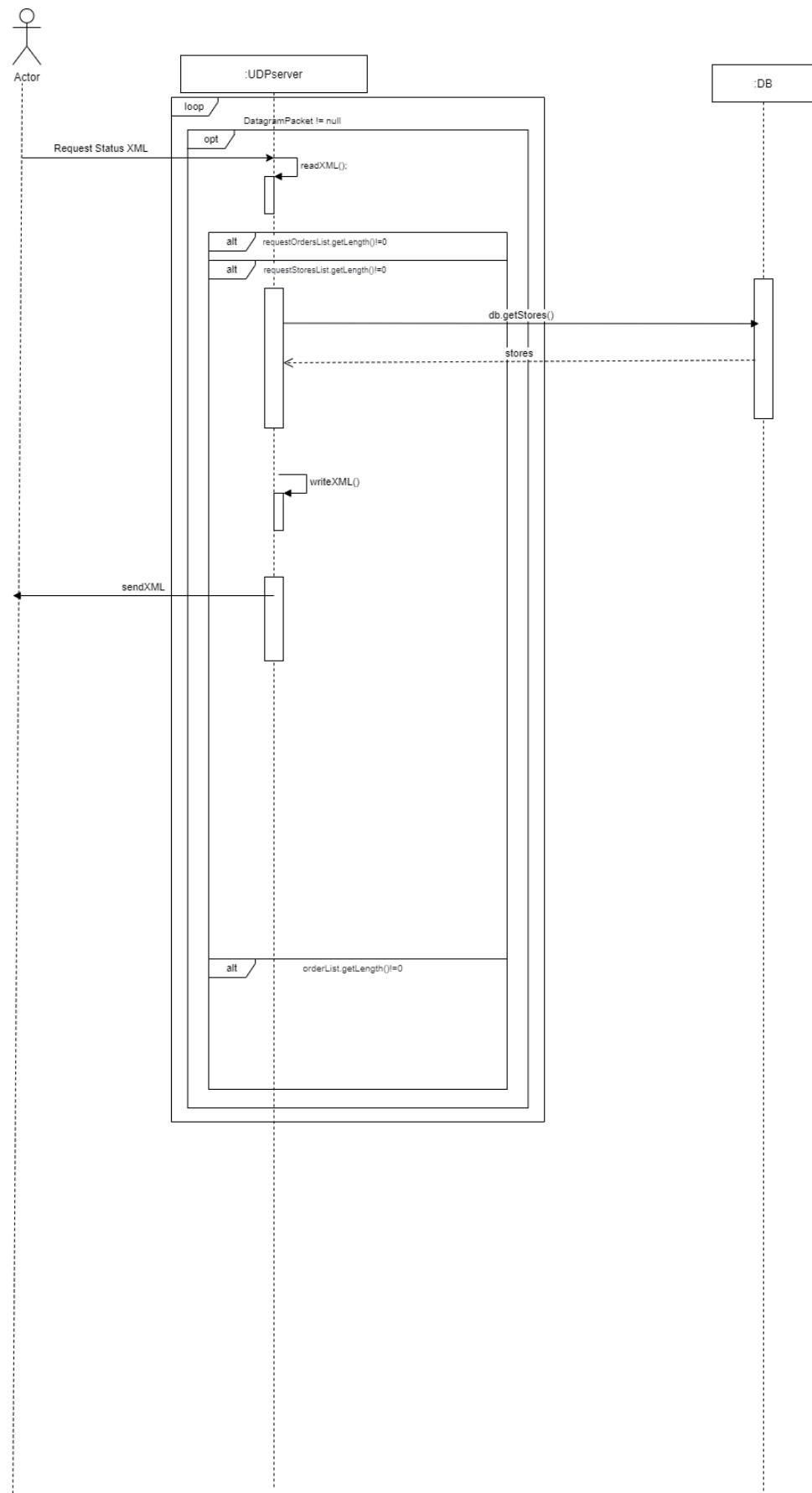


Figure 17: Request Stores

10.11 Anexo 11

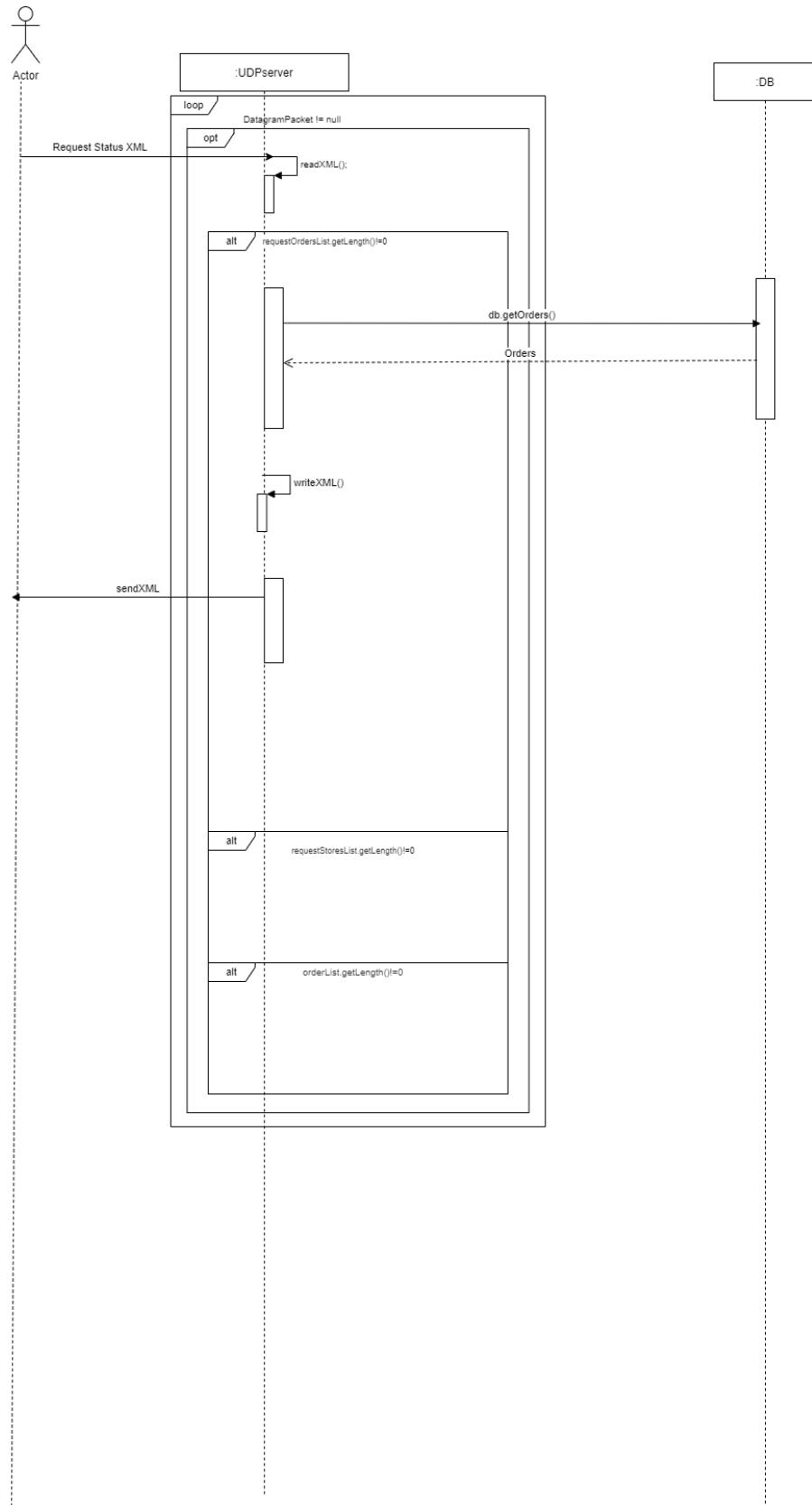


Figure 18: Request Orders

10.12 Anexo 12

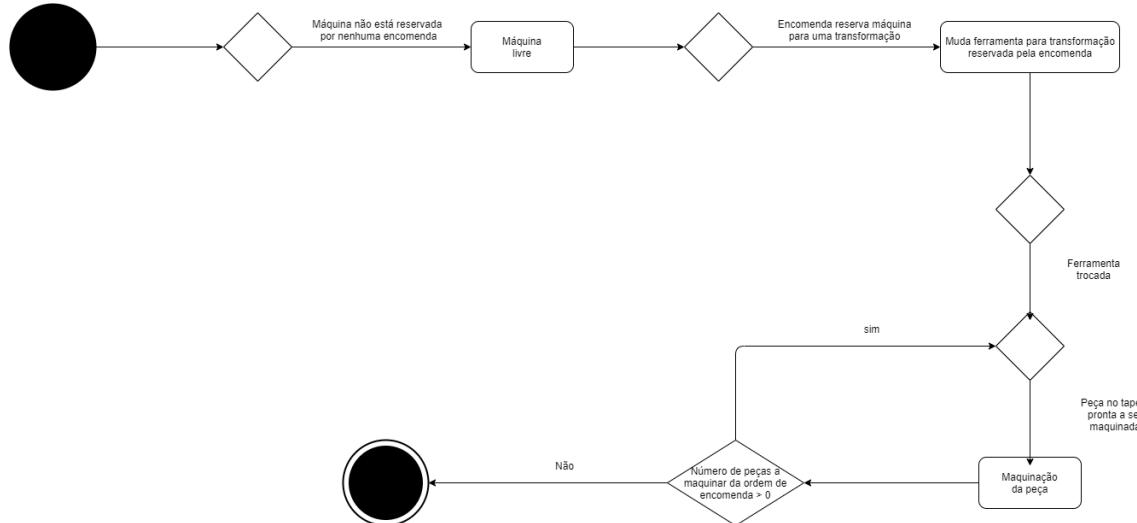


Figure 19: Máquina

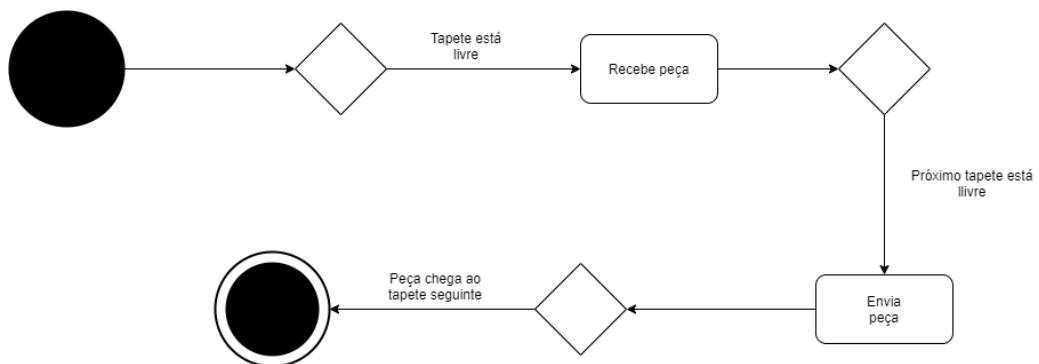


Figure 20: Tapete Linear

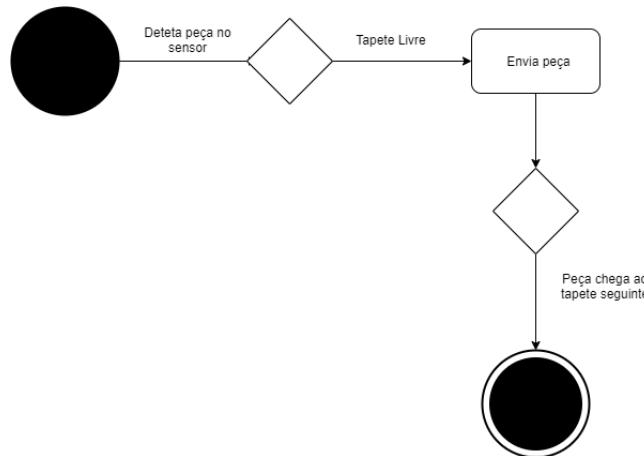


Figure 21: Tapete Carga

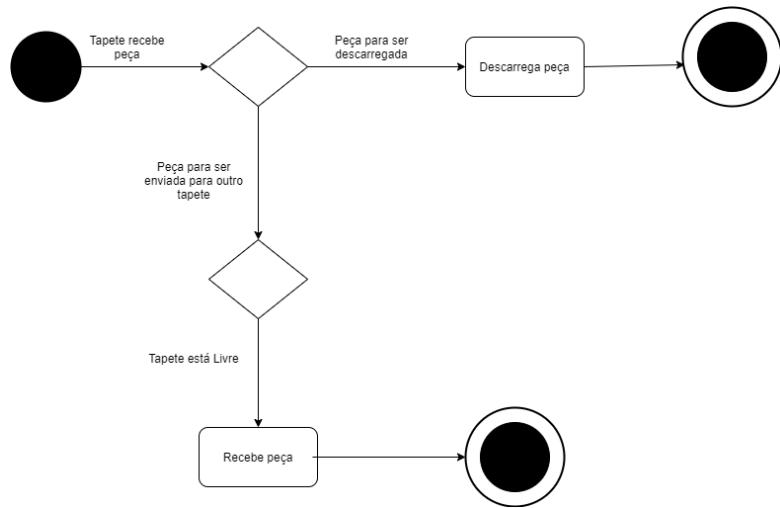


Figure 22: Tapete Descarga

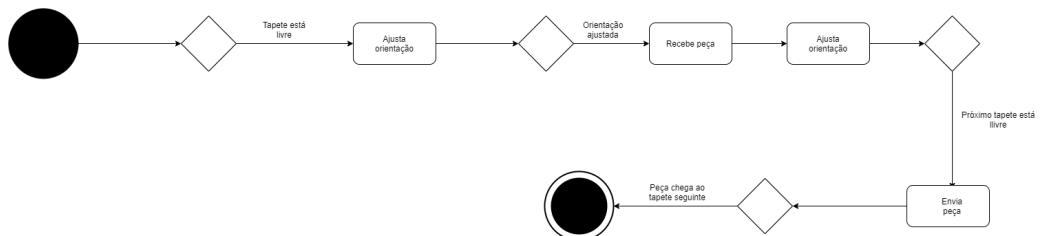


Figure 23: Tapete Rotativo