Universidade do Minho

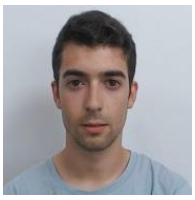
Departamento de Informática

Desenvolvimento de Sistemas de Software

Grupo nº 5

Carlos Preto (a89587)

Maria João Moreira (a89540)



Pedro Veloso (a89557)

Rui Fernandes (a89138)

Índice

1) Introdução

A 3ª fase do trabalho consistiu essencialmente em utilizar todo o trabalho de estruturação e modelação realizado nas fases anteriores para dar vida ao sistema de gestão de armazém.

Este sistema deve implementar funcionalidades que permitam ao funcionário do armazém utilizar o leitor de *QR-Code* presente nas paletes que são lidas antes de ser armazenadas, bem como gerir os *robots* que operam na arrumação e entrega das mesmas no armazém segundo um percurso calculado pelo próprio programa. Deve também permitir que o gestor aceda a informações inerentes ao estabelecimento, como por exemplo a listagem das localizações.

Tendo sido feitas simplificações no que toca a requisitos, decidiu-se implementar um sistema com dois *robots* que operam na recolha e entrega de paletes sob um percurso calculado para ambos individualmente.

Neste relatório final é apresentado o produto final desenvolvido com o auxílio de toda a modelação efetuada anteriormente.

2) Trabalho realizado

Antes de começar a implementar o sistema, realizou-se uma análise da fase 1 e 2, com o objetivo de verificar se o programa idealizado dará suporte a todos os requisitos e se a arquitetura do programa e os diagramas de sequência desenvolvidos permitiam o bom funcionamento do sistema, não originando possíveis *bugs* no futuro.

Ao analisar a fase 1, percebeu-se que existe um passo, no use case "Notificar entrega da palete", referente a mandar o *robot* para a base, se não houvesse paletes para transportar. Este passo pode causar problemas, pois quando o *robot* recebe o percurso a realizar e atendendo a que a sua localização só será alterada no final do percurso, pode ocorrer o caso de o *robot* estar a meio do percurso e receber uma notificação para entrega, contudo a sua posição atual não corresponderia com a do sistema, fazendo com que o percurso calculado tivesse falhas. Tendo este problema identificado, é necessário perceber se o *robot* ir para a base no fim de uma entrega é realmente indispensável. Analisando os requisitos, optou-se por retirar esse passo, deixando o *robot* no local, se não for necessário transportar mais nada. Posteriormente, pode ser acrescentado um use case que permite ao gestor do armazém notificar os *robots* para ir para a base.

Ao analisar a fase 2, percebeu-se que o programa pensado permitiria dar suporte a todos os uses cases previamente identificados, contudo notou-se alguns erros nos diagramas. Assim optou-se por corrigir a fase 2, enviando-a em anexo com esta terceira fase.

Após analisar as fases entregues anteriormente, realizou-se uma análise na arquitetura do sistema de modo a identificar os dados que deviam persistir no programa.

Tendo todos os passos necessários realizados, começou-se a implementação do programa.

3) Identificação das entidades/classes a persistir

Observando o diagrama de classes anteriormente apresentado, percebe que existem classes/entidades que armazenam dados. Assim, determinou-se que era necessário persistir os seguintes dados:

- Todas as paletes armazenadas no armazém;
- Todas as paletes que se encontram na *queue* de entrega;
- Todas as paletes que se encontram na *queue* de receção;
- Todos os produtos que se encontravam na *queue* de produtos em falta;
- O próximo código a utilizar na entrada de uma palete;
- Todas as prateleiras do armazém;
- Os *robots* do armazém;
- A prioridade de entrega, representada por uma *flag*.

4) Mapeamento das classes identificadas em tabelas

Uma vez identificados os dados que devem persistir é necessário criar tabelas, de modo a ser possível a sua representação numa base de dados relacional. Desta forma, obtiveram-se as seguintes tabelas:

- Palete(#codPalete,nomePalete,tamanho,disponibilidade,MateriaPrima,localizacao, codPrateleira);
- PaletesEntrega(#posicaoFila,codPalete);
- PaletesRececao(#posicaoFila,codPalete);
- PaletesFalta(#nomePalete);
- Prateleiras(#codPrateleiras,tamanho,Estado,localizacao,codPalete);
- Percurso(#posicaoPercurso,#codrobot,Nodo)
- robot(#codrobot,disponibilidade,localização,base,codPalete);
- ProxCode(#proxCodigo);
- Prioridade(#prioridade).

Segundo as indicações acima obtêm-se nove tabelas, todas reconhecidas por uma chave primária, identificada por um #, e todas com os seus vários atributos. A sublinhado encontram-se as chaves estrangeiras, que permitem encontrar dados noutras tabelas.

Utilizando o "MySQL Workbench", representou-se as tabelas, que posteriormente servirão para a criação da base de dados, como se pode observar na figura seguinte.

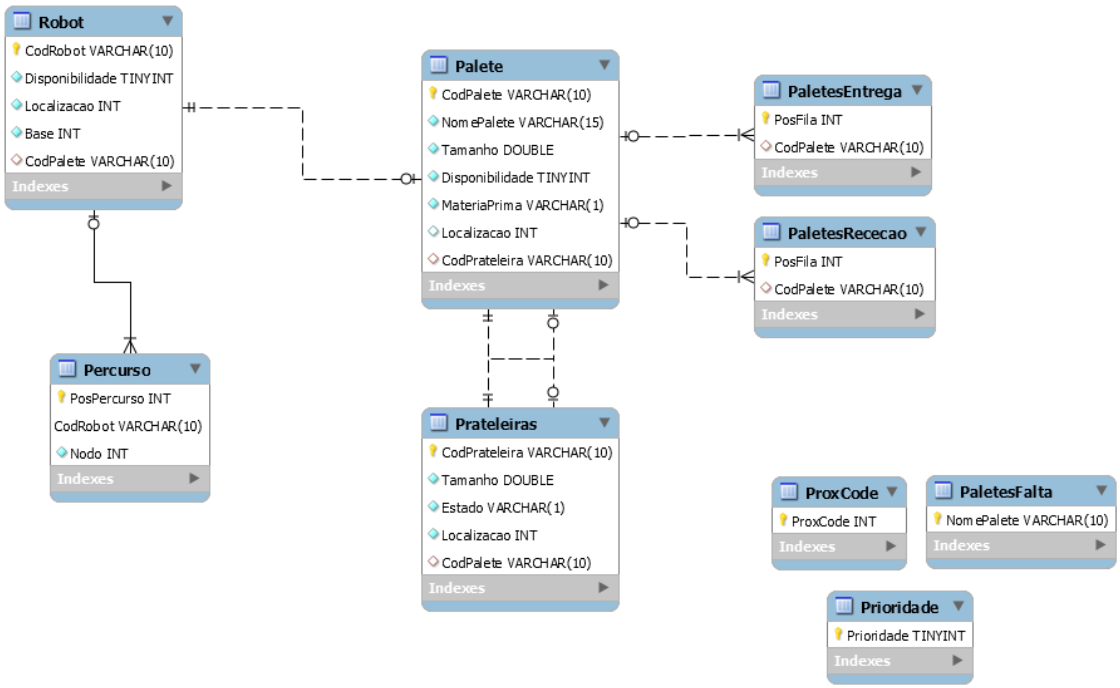


Figura 1: Modelo lógico da base de dados

5) Implementação do Sistema

Para esta terceira fase, realizou-se uma implementação do sistema. Contudo, implementaram-se apenas cinco uses cases (Comunicar código QR, Sistema comunica ordem de transporte, Notificar recolha de paletes, Notificar entrega de paletes, Consultar listagem de localizações), e não se considerou a diferença entre as zonas refrigeradas e as não refrigeradas. Desta forma, adaptou-se a arquitetura do sistema e da base de dados de modo a simplificar a sua implementação.

5.1 Arquitetura adaptada – versão com persistência

Como a arquitetura existente não permite a persistência de dados, foi necessário rearranjá-la de modo a que tal problema fosse ultrapassado. Desta forma, tendo também em conta os requisitos para esta fase do projeto obteve-se a seguinte arquitetura para o subsistema SSArmazenamento_Facade.

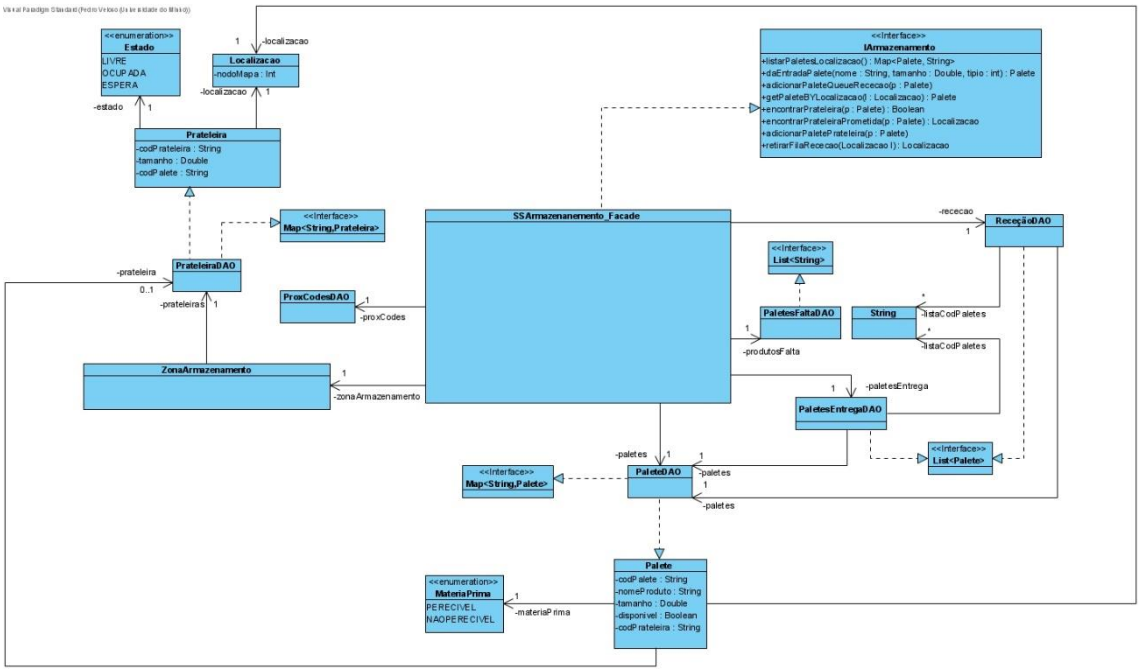


Figura 2: Diagrama de classes do subsistema SSArmazenamento_Facade

Como se observa pela figura acima, optou-se por não trazer as paletes nem as prateleiras para memória, trabalhando diretamente com a base de dados sempre que se precisar de tais dados. Em relação às *queues,* preferiu-se trazer para memória apenas os códigos das paletes e sempre, que se precisar de mais dados ir procurar na base de dados. Para o próximo código a utilizar realiza-se um *update* na base de dados sempre que o mesmo é utilizado.

Para o subsistema SSrobot_Facade, tem-se a seguinte arquitetura.

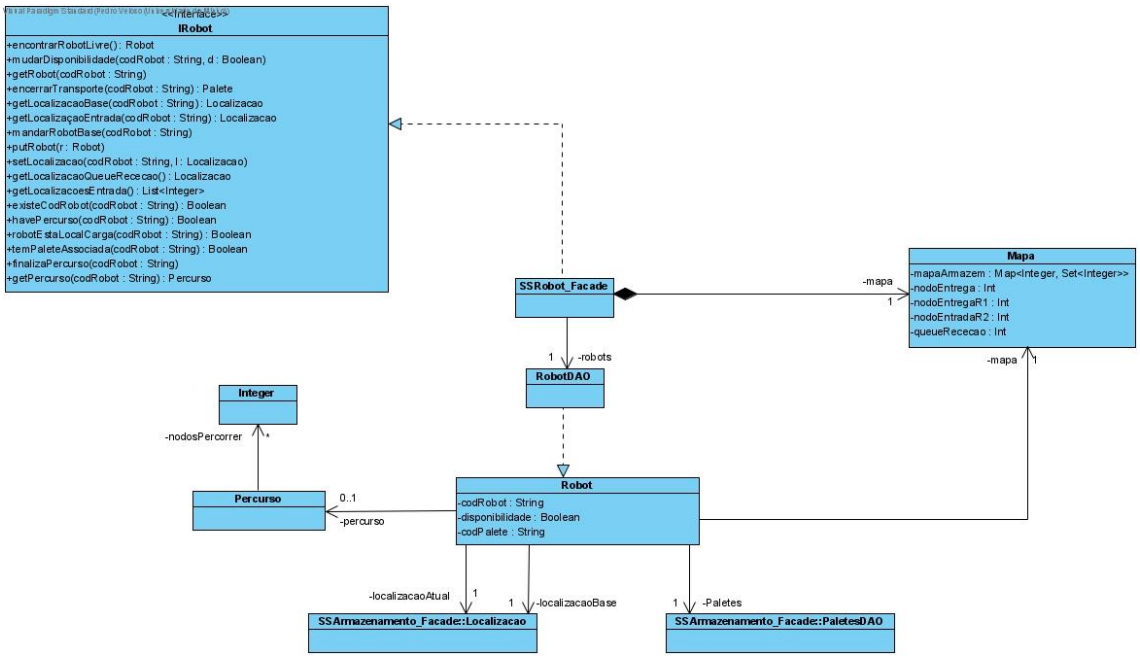


Figura 3: Diagrama de classes do subsistema SSRobot_Facade

Como o mapa do armazém se mantém constante não precisa de ser guardado na base de dados, podendo ser refeito sempre que o programa for iniciado. Já os *robots*, encontram-se na base de dados, sendo necessário retirá-los para memória sempre que utilizados.

5.2 Packages

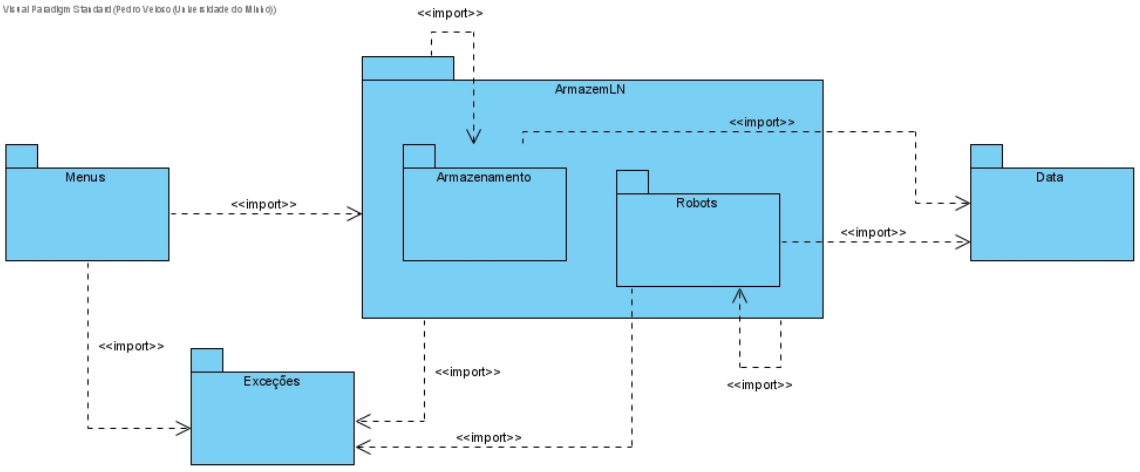


Figura 4: Diagrama de packages

Tendo em vista a evolução do sistema a implementar, adotou-se a utilização de packages, de modo a tornar mais fácil a gestão das classes. Desta forma, a arquitetura do sistema deve seguir a estrutura apresentada acima.

Assim no package do Armazenamento, encontram-se as classes:

- Estado
- IArmazenamento
- Localizacao
- MateriaPrima
- Palete
- Prateleira
- SSArmazenamento_Facade
- ZonaArmazenamento

No package robot encontram-se as seguintes classes:

- Irobot
- Mapa
- Percurso
- robot
- SSrobot_Facade

O package ArmazemLN tem as classes:

- ArmazemLN
- IArmazemLN

Para o package exceções são atribuídas todas classes relativas às exceções que podem ocorrer no programa. Assim temos:

- FinalizarTransporteException
- PrateleirasOcupadasException
- RecolhaPaleteException

No package data estão todas as classes responsáveis por obter dados da base de dados. Assim temos:

- PaleteDAO
- PaleteEntregaDAO
- PaleteFaltaDAO
- PrateleirasDAO
- ProxCodesDAO
- RececaoDAO
- robotsDAO
- DAOconfig

No package menus ficam as interfaces que permitirão aos utilizadores interagir com o sistema. Temos então:

- LocalizacoesUI
- Menu
- TextUI
- View_Localizacoes

5.3 Criação da Base de Dados

Uma base de dados é gerida por um sistema de gestão de bases de dados. Existem diferentes sistemas de gestão de bases de dados diferenciados em função dos requisitos e funcionalidades que proporcionam, como segurança, integridade dos dados e intolerância a falhas.

O sistema de gestão de bases de dados a utilizar deve ser o mais adequado ao problema em causa e o que possibilite a realização de várias operações por vários utilizadores ao mesmo tempo.

Neste trabalho, foi utilizado o sistema de gestão de bases de dados relacional MySQL, para desenvolvimento da base de dados, baseado na linguagem SQL, uma vez que é um sistema simples, rápido e seguro.

Uma vez selecionado o sistema de gestão de bases de dados, criou-se a base de dados a utilizar no trabalho, seguindo o seguinte código SQL:

"CREATE SCHEMA IF NOT EXISTS `ArmazemDSS` DEFAULT CHARACTER SET utf8;"

5.4 Povoamento da base de dados

Tendo a base de dados criada, é necessário realizar o povoamento da mesma consoante a estrutura e componentes do armazém. Assim, admitindo que o armazém está vazio, o povoamento inicial deve ter em conta os *robots* existentes, bem como as prateleiras. O mapa do armazém também é iniciado no programa, através de um povoamento inicial, contudo este não é feito na base de dados, uma vez que o mapa não tem persistência de dados. Analisando a estrutura do armazém que se encontra na figura abaixo, retira-se conclusões sobre o povoamento inicial.

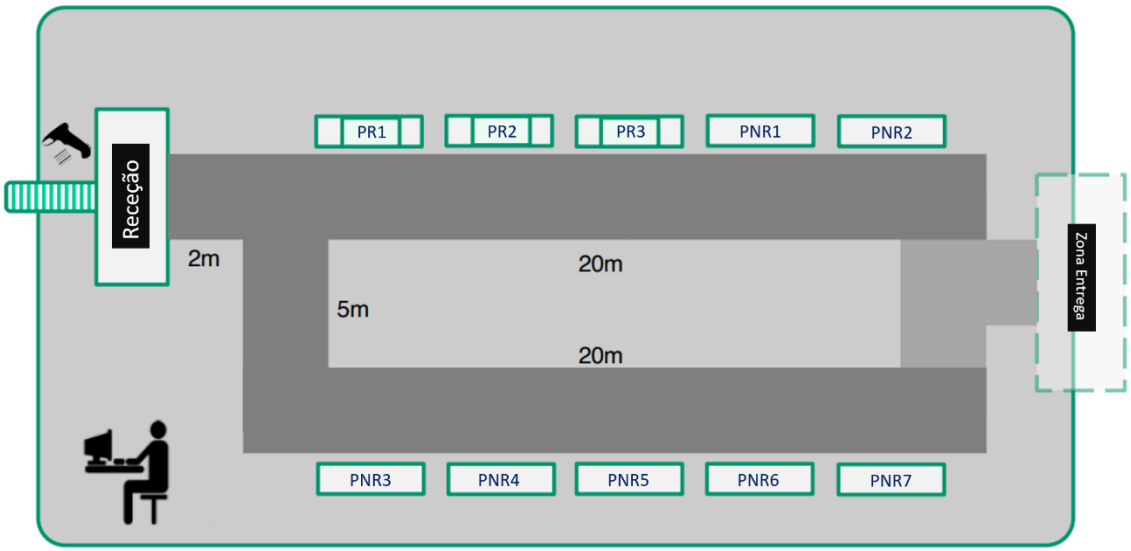


Figura 5: Mapa do armazém

Prateleiras

Como se pode observar pelo mapa do armazém, existem 10 prateleiras, sendo 3 delas refrigeradas. Desta forma, admitindo que o armazém se encontra no seu estado inicial e que todas as prateleiras estão disponíveis, povoou-se a base de dados com 10 prateleiras, tendo todas elas 2 metros de altura e estando todas elas livres. Colocou-se, também, em cada uma a localização correspondente.

O código SQL utilizado para este povoamento foi o seguinte:

```
INSERT INTO prateleiras
    (CodPrateleira,Tamanho,Estado,Localizacao,CodPalete)
    VALUES
        ('PR1',2.0,'L',3,NULL),
        ('PR2',2.0,'L',4,NULL),
        ('PR3',2.0,'L',5,NULL),
        ('PNR1',2.0,'L',6,NULL),
        ('PNR2',2.0,'L',7,NULL),
        ('PNR3',2.0,'L',9,NULL),
        ('PNR4',2.0,'L',10,NULL),
        ('PNR5',2.0,'L',11,NULL),
        ('PNR6',2.0,'L',12,NULL),
        ('PNR7',2.0,'L',13,NULL);
```

Robots

Para o povoamento dos *robots*, foram introduzidos 2 *robots*. Admitindo que os *robots* nesta fase inicial estão na sua base e sem nenhum transporte associado, povoou-se a base de dados com o seguinte código SQL:

```
INSERT INTO robot
    (Codrobot,Disponibilidade,Localizacao,Base,CodPalete)
    VALUES
        ('R1',true,16,16,NULL),
        ('R2',true,16,16,NULL);
```

Mapa

Observando o mapa do armazém percebe-se que pode se representar o mesmo por um grafo, onde cada prateleira representa um nodo do grafo, existindo ainda nodos para a receção e para a zona de entrega. Acrescentou-se nodos nas zonas de mudança de direção.

Assim, obtém-se um grafo como o representado na figura seguinte.

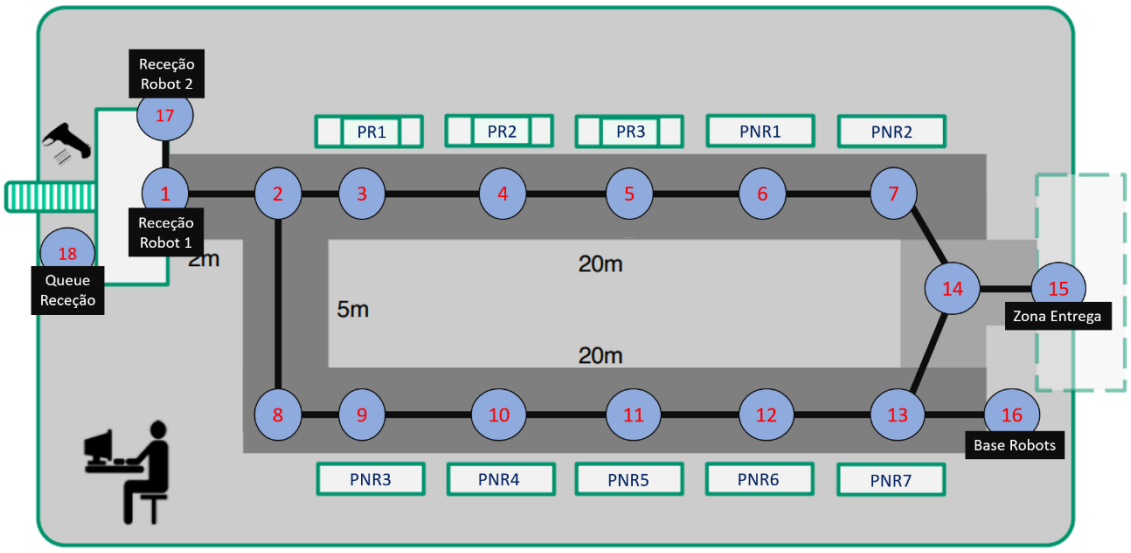


Figura 6: Grafo originado pelo mapa do armazém

Como se optou por utilizar um grafo de listas de adjacências, tem-se um *map* onde a *key* representa o nodo e o *value* é uma lista dos nodos que se consegue atingir a partir da *key*. Assim representa-se o mapa do armazém por:

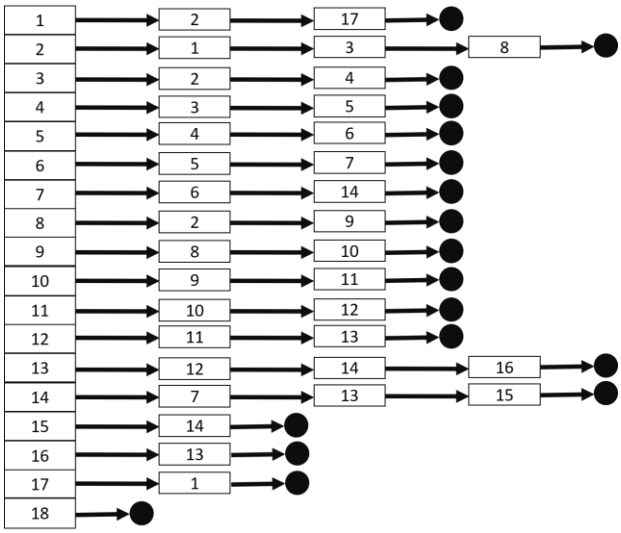


Figura 7: Representação do grafo

Optou-se por não representar as distâncias, uma vez que o mapa do armazém é bastante simétrico e não é muito complexo, sendo que basta apenas considerar que a distância ao mudar de nodo é a distância percorrida mais um.

5.5 Logica de negócio adaptada – versão com persistência

Ao nível da Lógica de Negócio do Armazém, desenvolveram-se as funções referentes a cada um dos use cases apresentados no enunciado do projeto, não distinguindo, as cargas perecíveis das não perecíveis. Essas mesmas funções podem ser representadas em Diagramas de Sequência, que ajudam a perceber a relação entre as diferentes classes, sendo que também indicam o ator associado a cada uma delas.

Comunicar o código QR

Como especificado no enunciado, o LeitorQR deverá ser capaz de "Comunicar o código QR". De maneira a realizar tal ação, o Leitor terá de passar como parâmetros o nome, o tamanho e o tipo da palete associada ao código QR.

Uma vez comunicado o código, será necessário encontrar um *robot* que se encontre livre (Anexo 8.1) e obter o seu código. Sabendo que o armazém terá 2 *robots* funcionais, iremos obter a localização da receção desse mesmo *robot*. Caso não haja *robots* livres, iremos adquirir a localização da *queue* receção, onde será armazenada a palete.

Posteriormente, gera-se um código para a palete, e uma vez que houve uma alteração de dados, temos de atualizar a base de dados de forma a conter a informação mais recente (Anexo 8.2) e verifica-se a existência de alguma prateleira disponível para essa palete (Anexo 8.3), sendo que caso exista se associa a prateleira a essa palete. Por fim, será necessário notificar o *robot* acerca do transporte da palete (ver secção Notificar Robot).

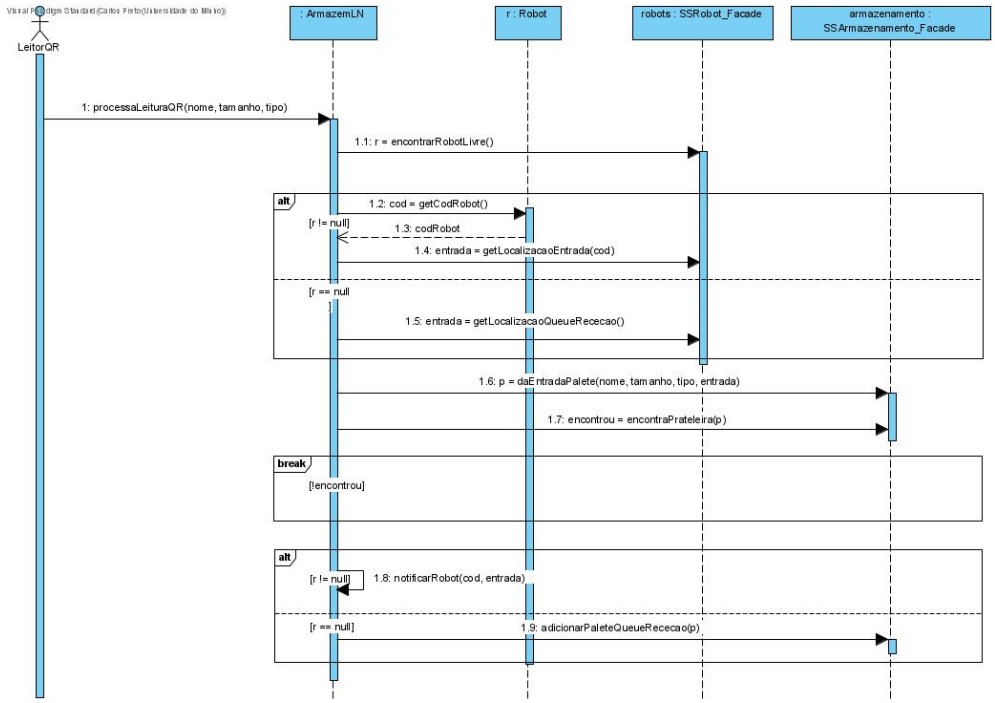


Figura 8: Diagrama de Sequência - ProcessaLeituraQR

Notificar robot

Primeiramente, será necessário definir o *robot* como ocupado e calcular o percurso até à sua receção (Anexo 8.4), onde estará a Palete. De maneira a se calcular o percurso, utilizou-se o algoritmo de Dijkstra (Anexo 8.5), adaptado ao mapa do Armazém. Seguidamente, como houve alteração de dados, deve-se atualizar a informação referente ao *robot* na base de dados.

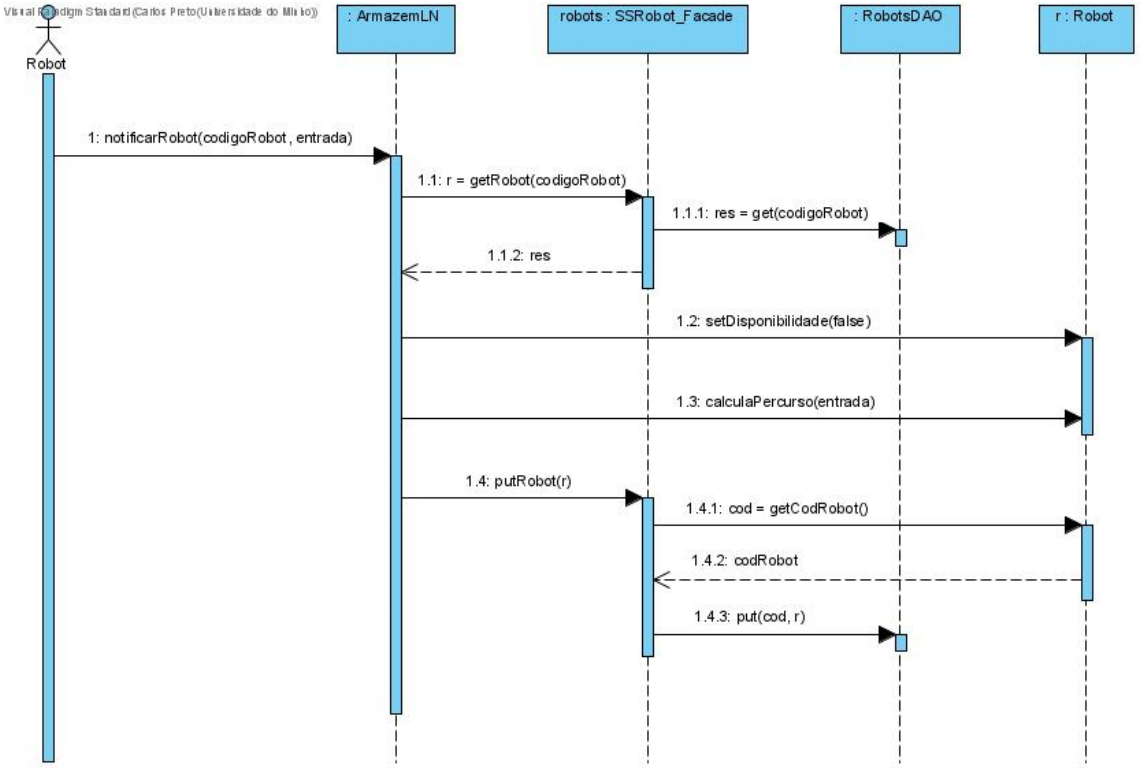


Figura 9: Diagrama de Sequência - NotificarRobot

Notificar Recolha

Após o *robot* chegar à sua receção, irá adquirir a palete, logo será necessário informar o sistema de que a recolha foi efetuada com sucesso, ficando assim o *robot* com a palete que está na receção (Anexo 8.6) e que o percurso até à receção foi concluído (Anexo 8.7). Trata-se então de associar a palete ao seu *robot* e de uma vez mais atualizar a base de dados (Anexo 8.8), sendo que posteriormente se precisa de saber a localização da prateleira que anteriormente se associou à palete (Anexo 8.9). Obtendo essa localização, será possível notificar o *robot* acerca do próximo local até ao qual se tem de deslocar (ver secção Notificar Robot).

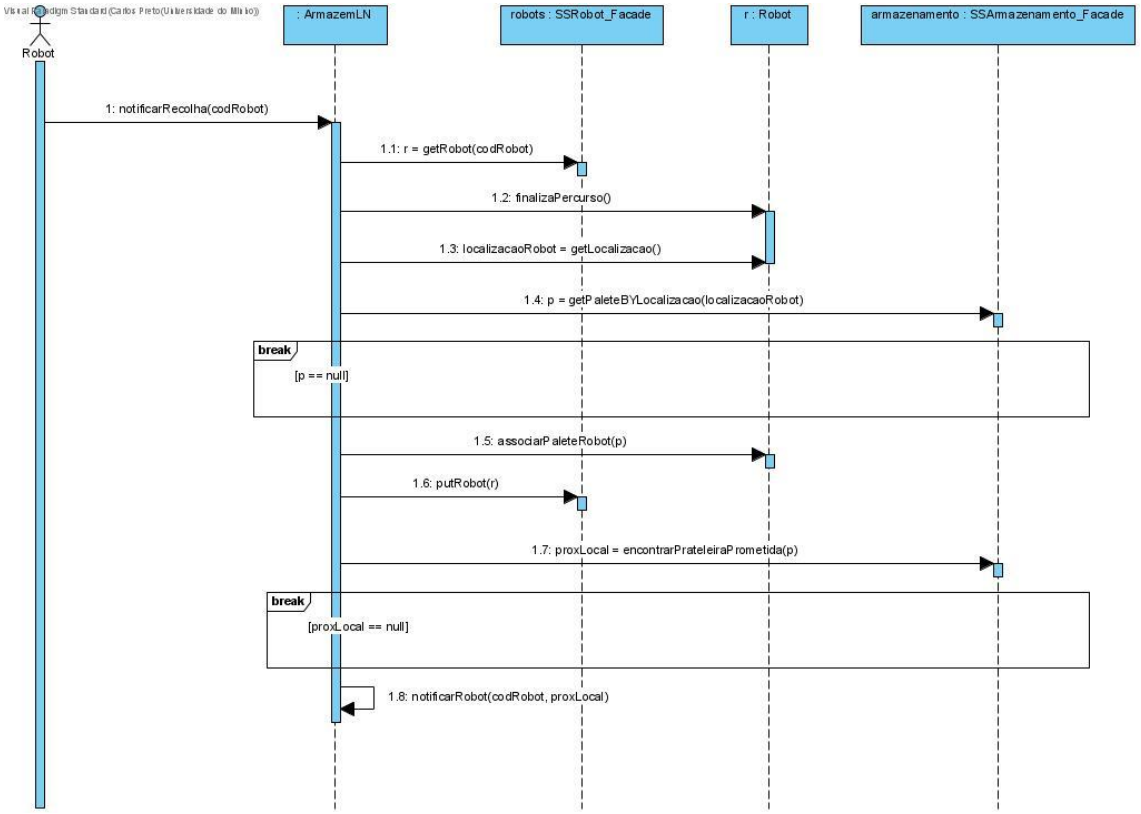


Figura 10: Diagrama de Sequência - NotificarRecolha

Notificar Entrega

Uma vez concluído o transporte da palete até à sua prateleira, por parte do *robot*, será necessário notificar o sistema: desassociar a palete do seu *robot* (Anexo 8.10), para que este possa voltar a estar disponível para transportar mais paletes, e adicionar a palete à prateleira (Anexo 8.11). Porém, o *robot* apenas definirá a sua disponibilidade como *true* caso não haja mais nenhuma palete na receção à espera de transporte (Anexo 8.12), caso contrário será necessário notificar o *robot* do novo pedido de transporte de uma palete (ver secção Notificar Robot).

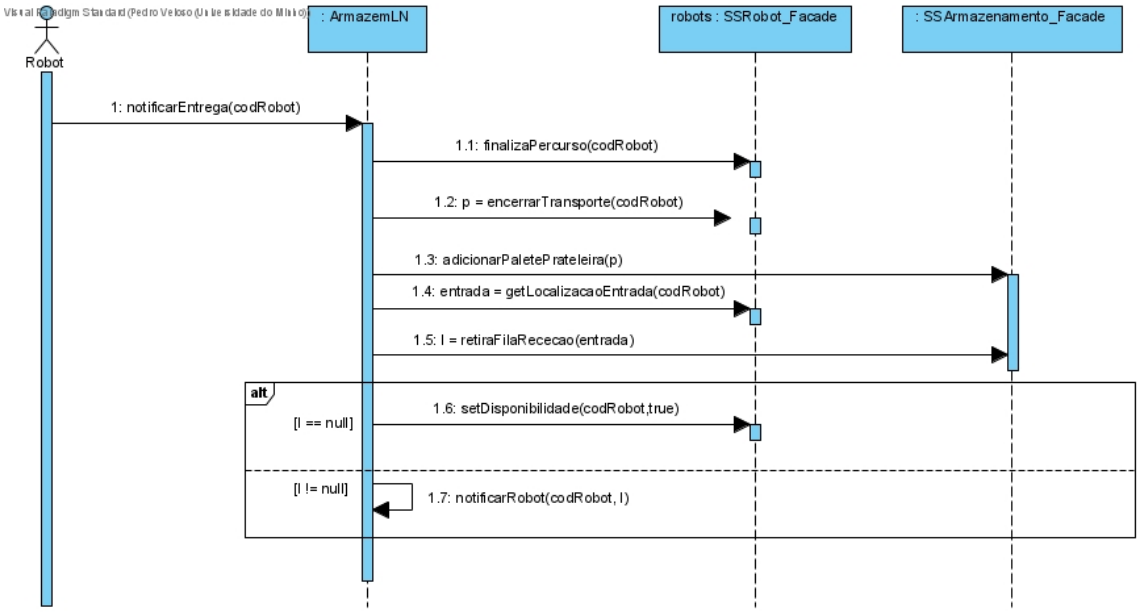


Figura 11: Diagrama de Sequência - NotificarRecolha

Listar Localizações

O gestor terá de consultar a listagem de localizações e, para isso, terá de ter em conta todas as paletes que se encontram quer na localização de entrada (Anexo 8.14), quer na localização da *queue* de receção (Anexo 8.15). Poderá também haver a hipótese de uma palete estar associada a um *robot* ou a uma prateleira. Tendo em conta estas hipóteses, obter-se-á um *map* com as paletes e a informação textual sobre onde se encontram.

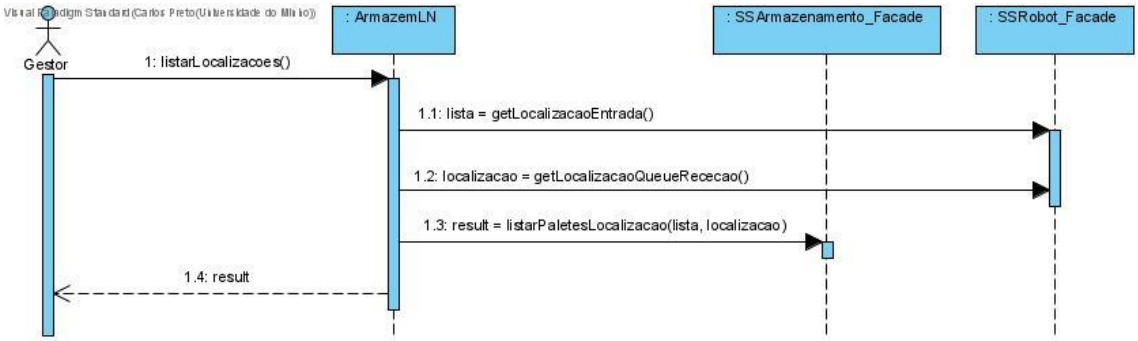


Figura 12: Diagrama de Sequência - ListarLocalizacoes

5.6 Interfaces

Para interagir com o sistema foram criadas interfaces – Menu Principal, Leitor *QR-Code*, *Robot*, Gestor, Localizações – em modo de texto. O comportamento ocorrido ao interagir com a interface pode ser documentado através de uma máquina de estado.

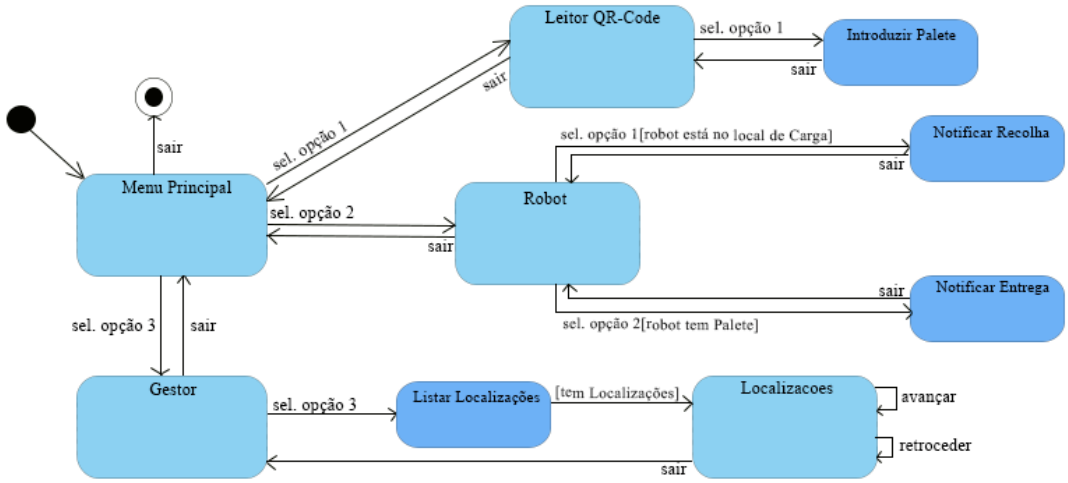


Figura 13: Máquina de estado

Observando a máquina de estado percebe-se que a primeira interação do utilizador com o sistema é através da *view* "Menu Principal" onde é possível escolher o tipo de utilizador – Leitor *QR-Code,* Gestor, *Robot*.

Consoante a opção introduzida, é apresentada ao utilizador a *view* correspondente. Cada *view* apresenta opções de iteração com o sistema diferentes, como se observa na figura acima. No caso da *view* do *robot* é apresentado também o percurso que está a percorrer (no caso de não estar a percorrer nenhum percurso é apresentada essa informação).

Existem pré-condições, para a realização da opção "Notificar Recolha" (*robot* está no local que vai realizar a recolha) e para a realização da opção "Notificar Entrega" (*robot* tem de ter uma palete associada). Para se poder ver todas as localizações das paletes no armazém é necessário existir paletes no armazém.

Refere-se também que, ao selecionar a opção *robot* no menu principal, é necessário introduzir de seguida um código de *robot* válido.

Para todos os menus é utilizada a opção zero como a opção de sair do menu.

O apeto dos menus desenvolvidos, é apresentado na figura seguinte.

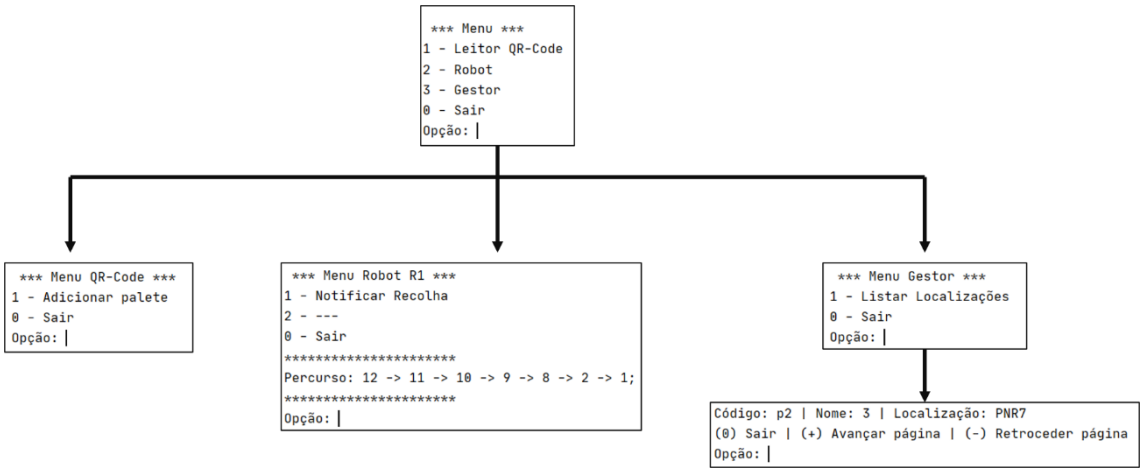


Figura 14: Menus desenvolvidos

6) Análise crítica global dos resultados

Para realizar uma análise do trabalho desenvolvido, utilizou-se o exemplo do use case "Consultar listagem de localizações", numa primeira fase do trabalho também denominado como "Consultar inventário do armazém", representado na figura seguinte.

Use Case	**Consultar inventário do armazém**
Descrição:	O Gestor solicita ao sistema o inventário do armazém, isto é, as paletes existentes e respetiva localização.
Pré-condição:	O Gestor está autenticado.
Pós-condição:	O Gestor recebe a lista do inventário do armazém.
Fluxo Normal:	1. O sistema lista as paletes existentes no armazém e respetiva localização.
Fluxo de exceção 1 [O inventário está vazio] (passo 1):	1.1 O sistema informa que não existem paletes no armazém.

Figura 15: Use case "Consultar inventário do armazém"

Como demonstrado através do use case acima, percebe-se que se existirem paletes no armazém é apresentada a informação da palete, bem como a sua localização. Caso não existam paletes, é apresentada a informação da sua inexistência.

Na segunda fase, utilizando o use case anterior definiu-se que a informação seria enviada ao gestor através de um *map*, onde a chave seria um clone da palete e o *value* uma *string* com a localização. A localização é descoberta através das seguintes condições:

- Se o código da prateleira na palete for diferente de *null*, a palete encontra-se localizada numa prateleira;
- Se a localização da palete for diferente de *null* e o código da prateleira for *null*, a palete encontra-se na receção;
- Caso nenhuma das condições anteriores seja verificada, a palete estará num *robot*.

Este processo pode ser verificado na figura abaixo.

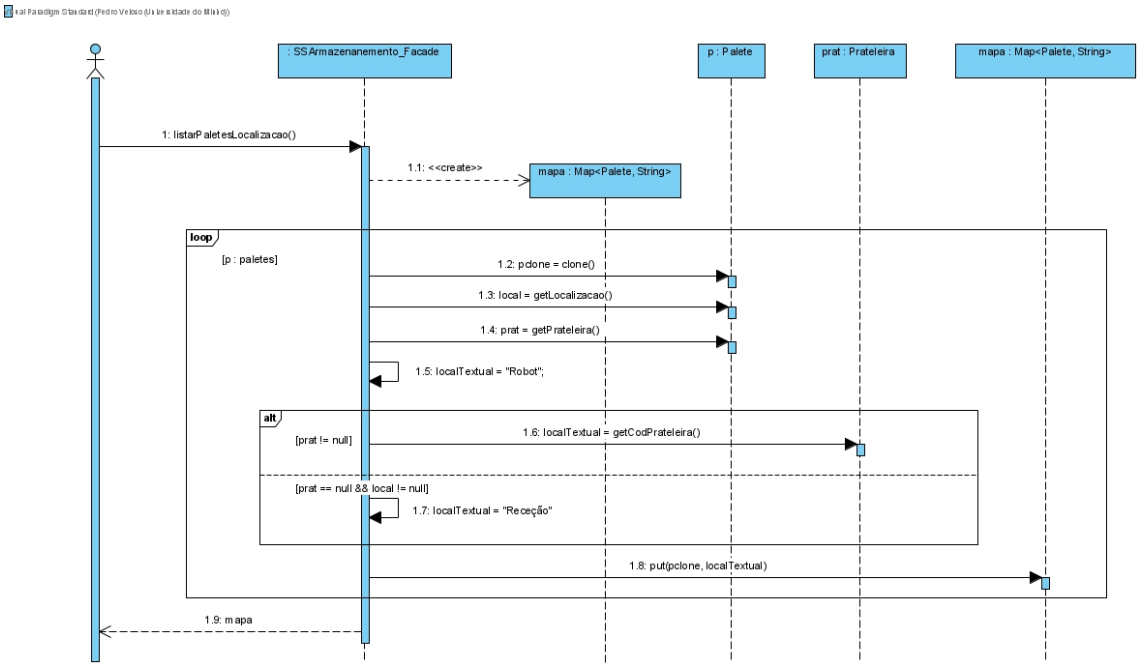


Figura 16: Diagrama de sequência - ListarPaletesLocalização (Fase 2)

Na fase de implementação (fase 3), utilizou-se o raciocínio anterior para o desenvolvimento da funcionalidade no sistema. Contudo, acrescentou-se a distinção entre receção e *queue* de receção. Para tal, passou-se como parâmetros à função as localizações das mesmas.

Como referido, na segunda fase foram cometidos alguns erros nos diagramas, que no início desta fase, foram corrigidos. Contudo, considera-se que o raciocínio das fases anteriores ainda era válido, pelo que foi possível, através deste, desenvolver o sistema.

7) Conclusão

O trabalho realizado nas anteriores fases foi crucial para concluir esta última com objetividade e segurança. Pois após a conclusão deste trabalho verificamos que, segundo os parâmetros que apresentamos anteriormente, tudo aparenta estar a funcionar corretamente, executando as funcionalidades devidas. Salvo algum caso não testado, no geral, a aplicação aparenta estar operacional e consideravelmente robusta.

Dando como concluído o projeto proposto pela Unidade Curricular de Desenvolvimento de Sistemas de Software, obtivemos aptidões a nível de modelação que permitiu estruturar os requisitos e visualizar o problema de forma objetiva, conseguindo prever fluxos de ação que deveriam ser evitados e/ou implementados resultando num programa que julgamos ter alcançado o proposto.

8) Anexos

8.1 Encontrar *robot* livre

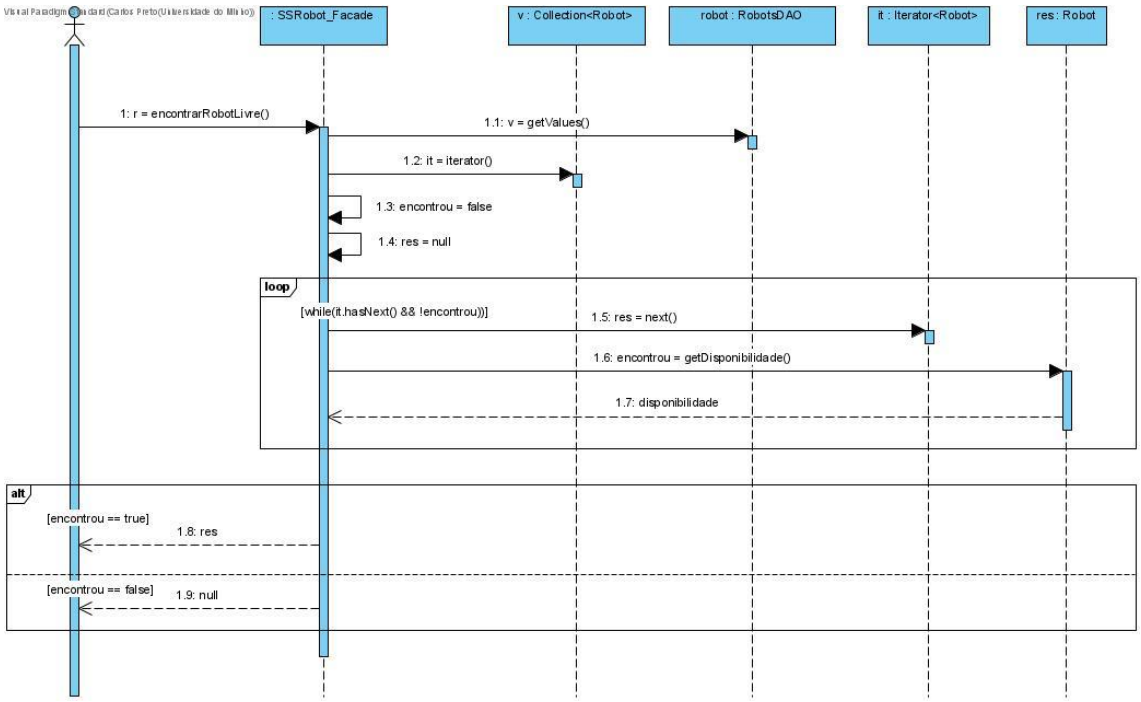


Figura 17: Diagrama de Sequência - EncontrarRobotLivre

8.2 Gera código palete e atualiza base dados

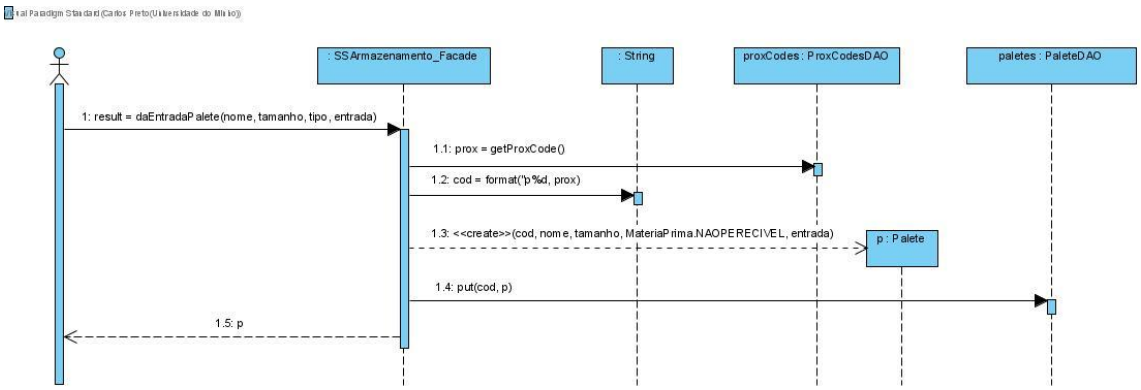


Figura 18: Diagrama de Sequência - DaEntradaPalete

8.3 Encontrar prateleira disponível

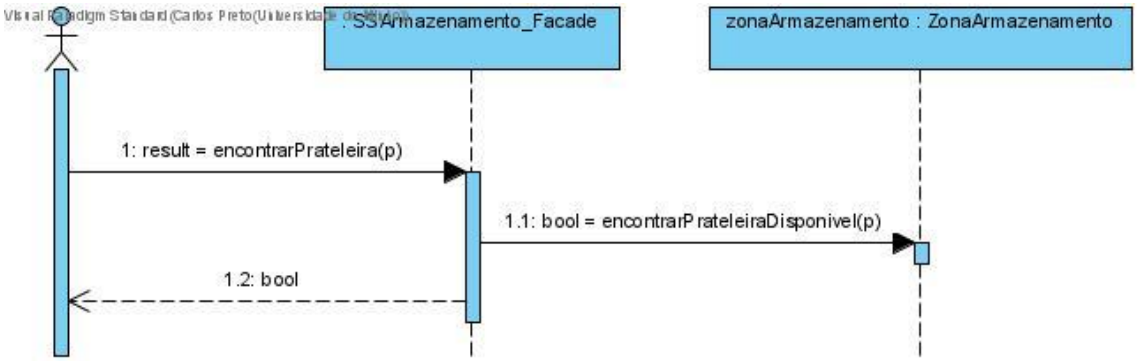


Figura 19: Diagrama de Sequência - EncontrarPrateleira

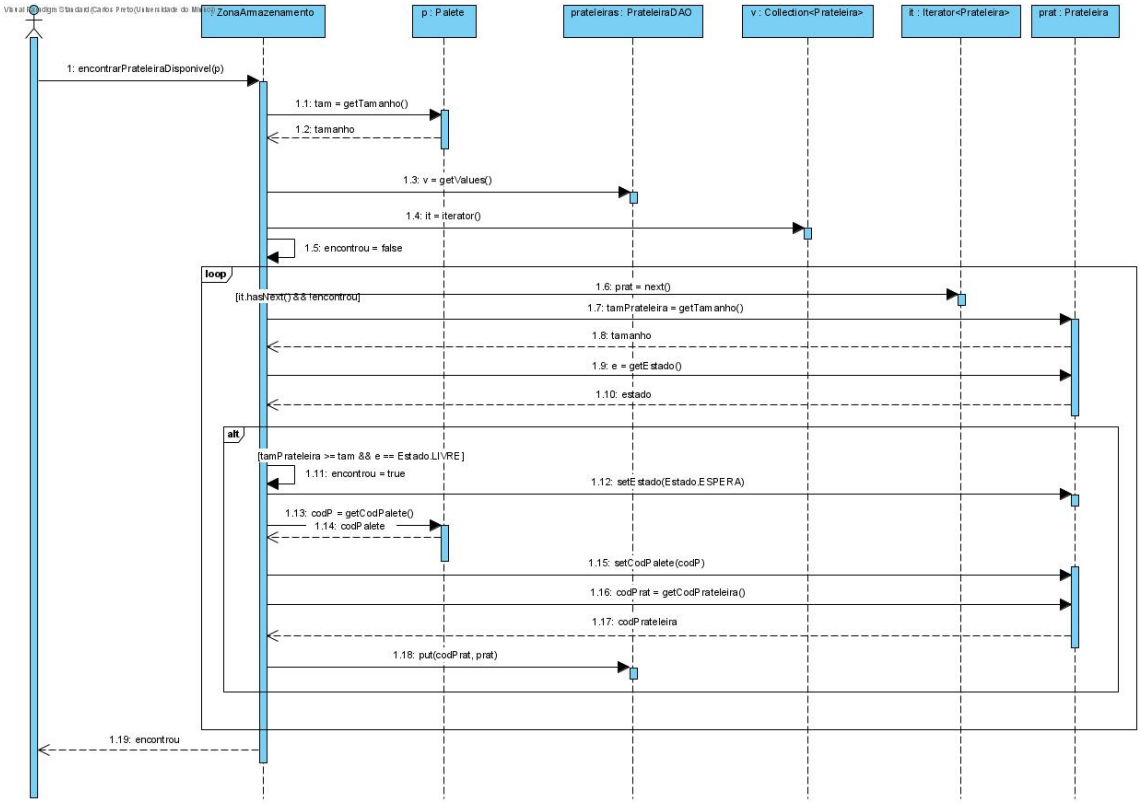


Figura 20: Diagrama de Sequência - EncontrarPrateleiraDisponivel

8.4 Calcular percurso

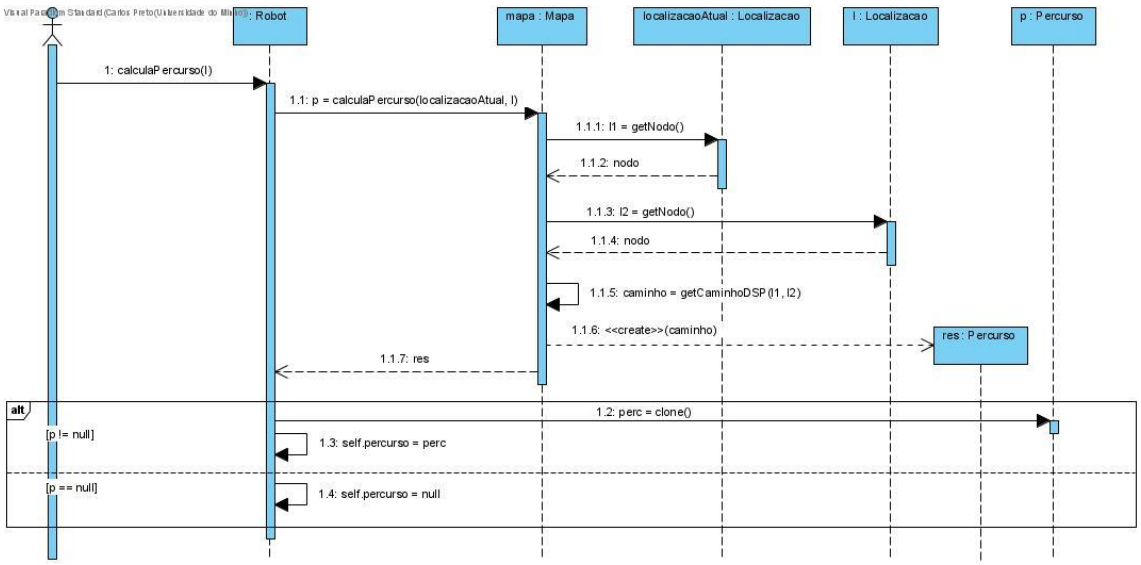


Figura 21: Diagrama de Sequência - CalculaPercurso

8.5 Algoritmo de *Dijkstra*

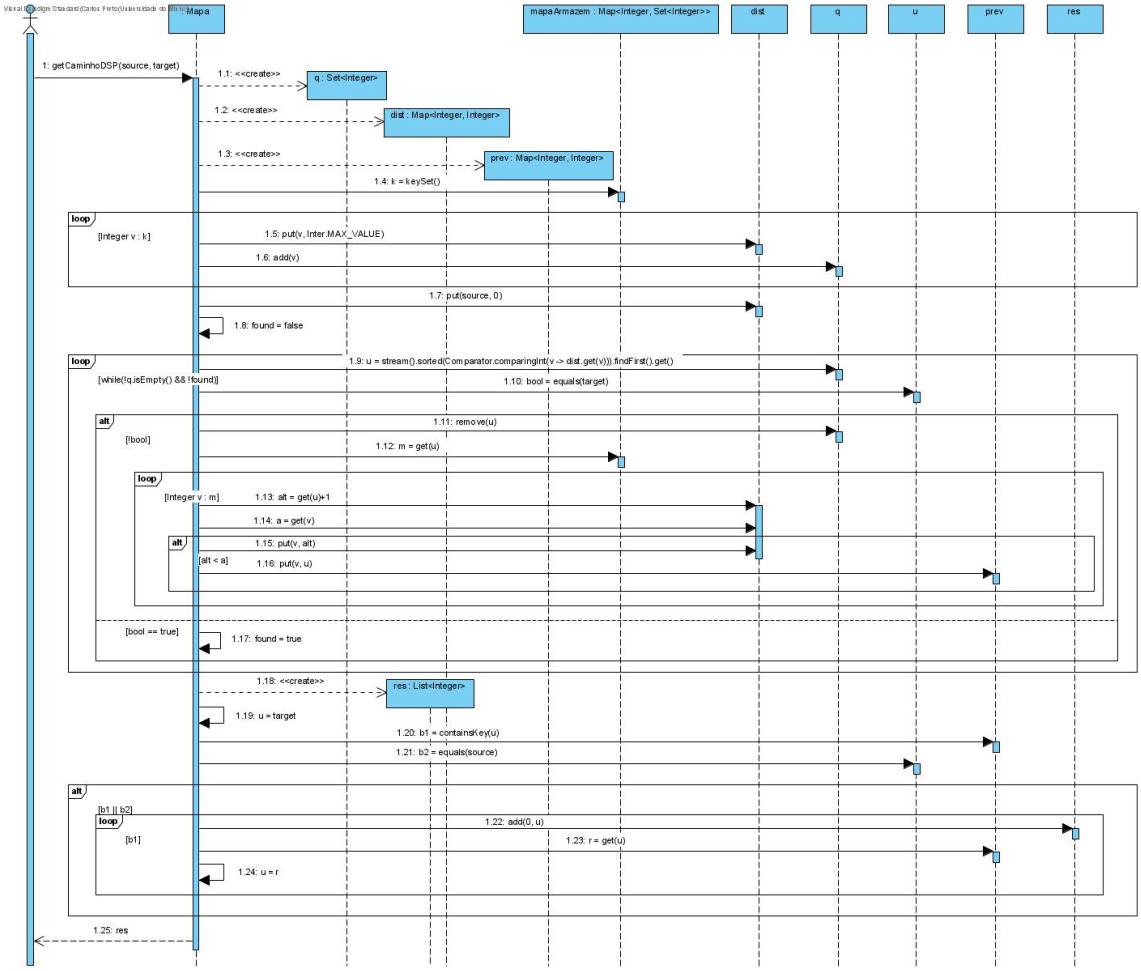


Figura 22: Diagrama de Sequência - GetCaminhoDSP

8.6 Descobrir a palete, pela localização

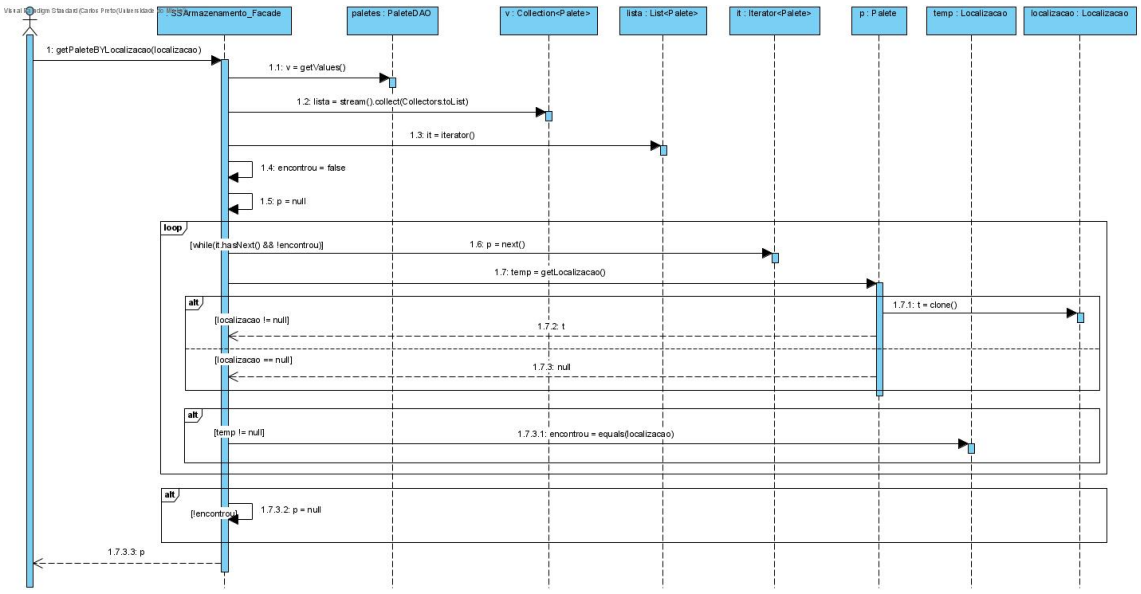


Figura 23: Diagrama de Sequência - GetPaleteBYLocalizacao

8.7 Finalizar percurso

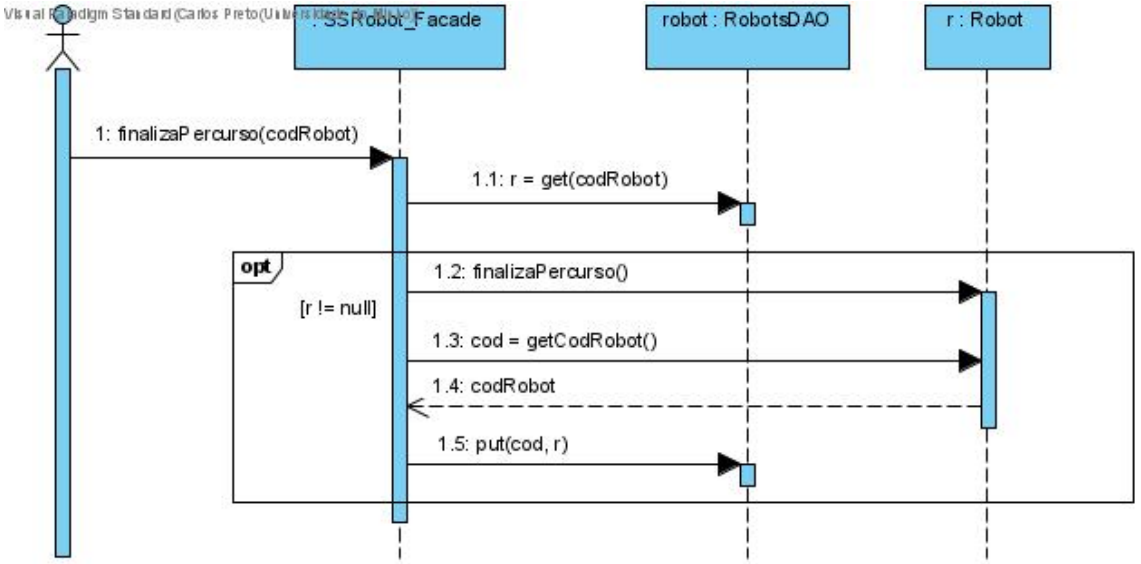


Figura 24: Diagrama de Sequência - FinalizaPercurso

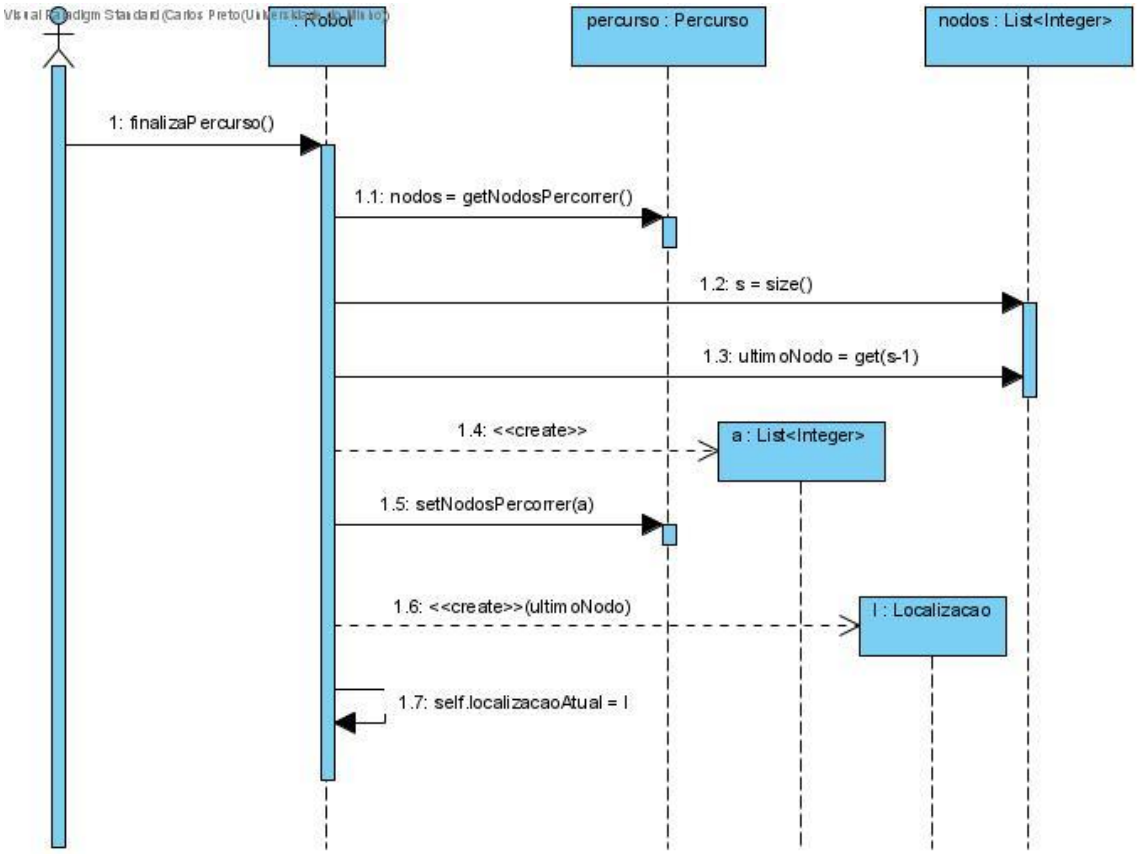


Figura 25: Diagrama de Sequência - FinalizaPercurso

8.8 Associar palete ao *robot*

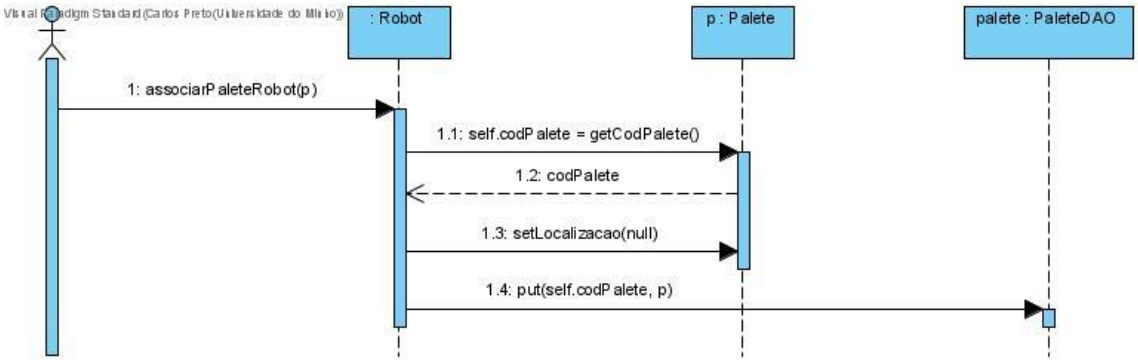


Figura 26: Diagrama de Sequência - AssociarPaleteRobot

8.9 Encontrar prateleira prometida

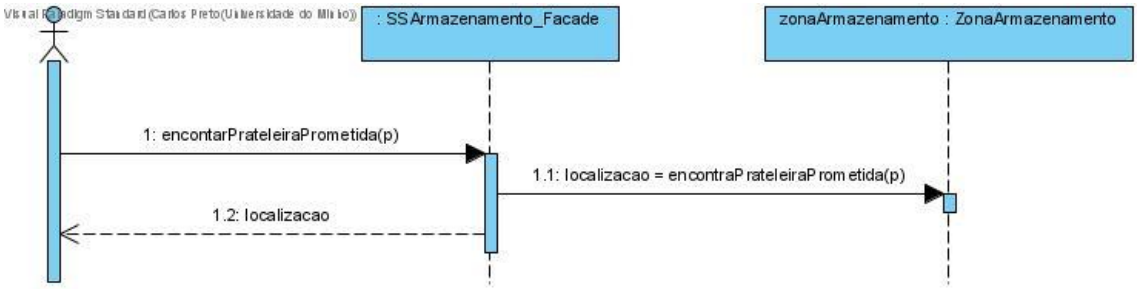


Figura 27: Diagrama de Sequência – EncontrarPrateleiraPrometida I

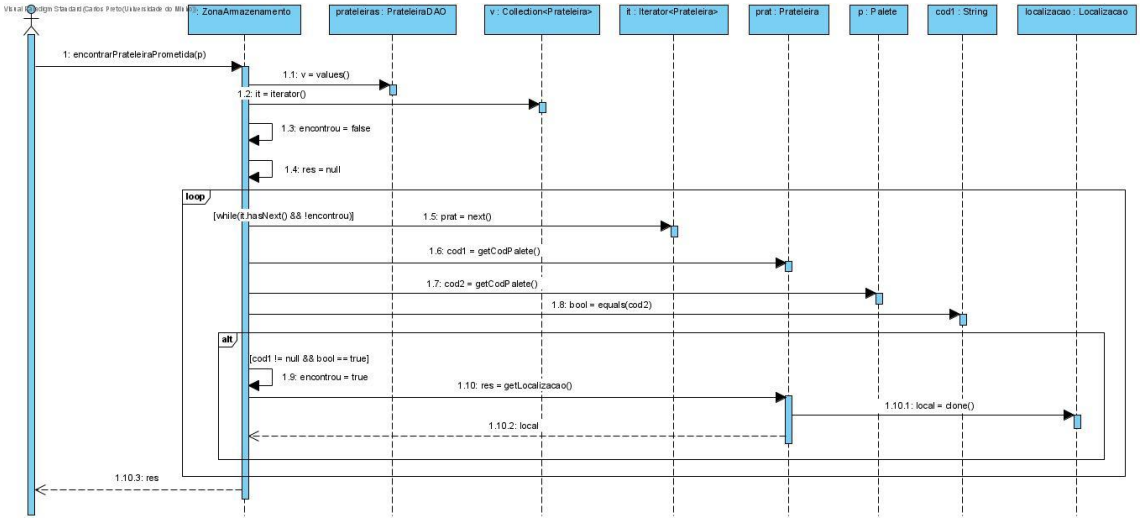


Figura 28: Diagrama de Sequência – EncontrarPrateleiraPrometida II

8.10 Desassociar palete do *robot*

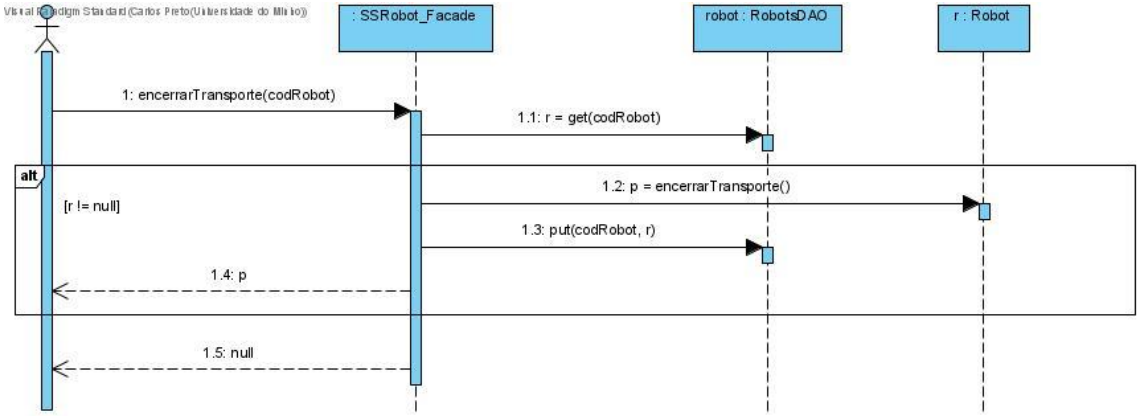


Figura 29: Diagrama de Sequência – EncerrarTransporte I

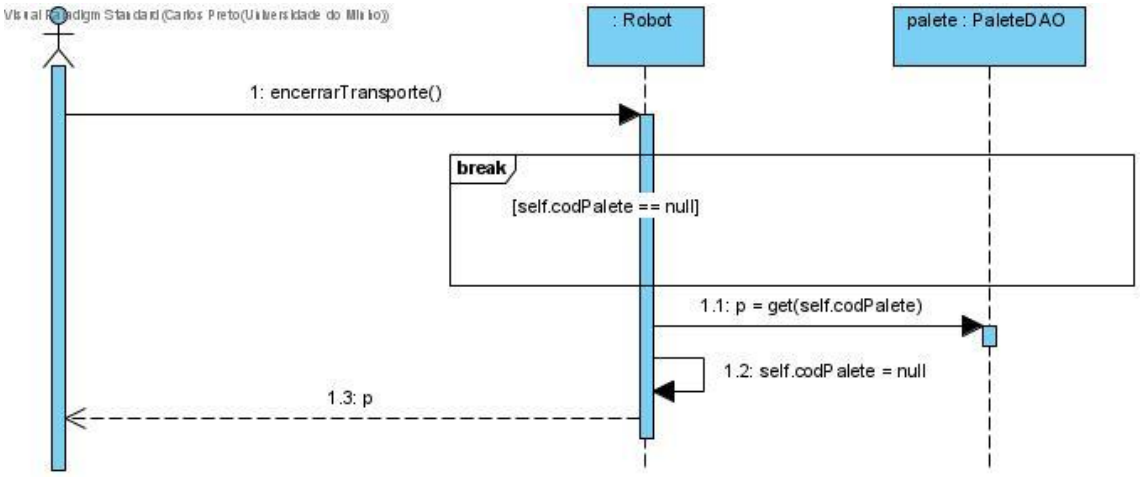


Figura 30: Diagrama de Sequência – EncerrarTransporte II

8.11 Adicionar palete à prateleira

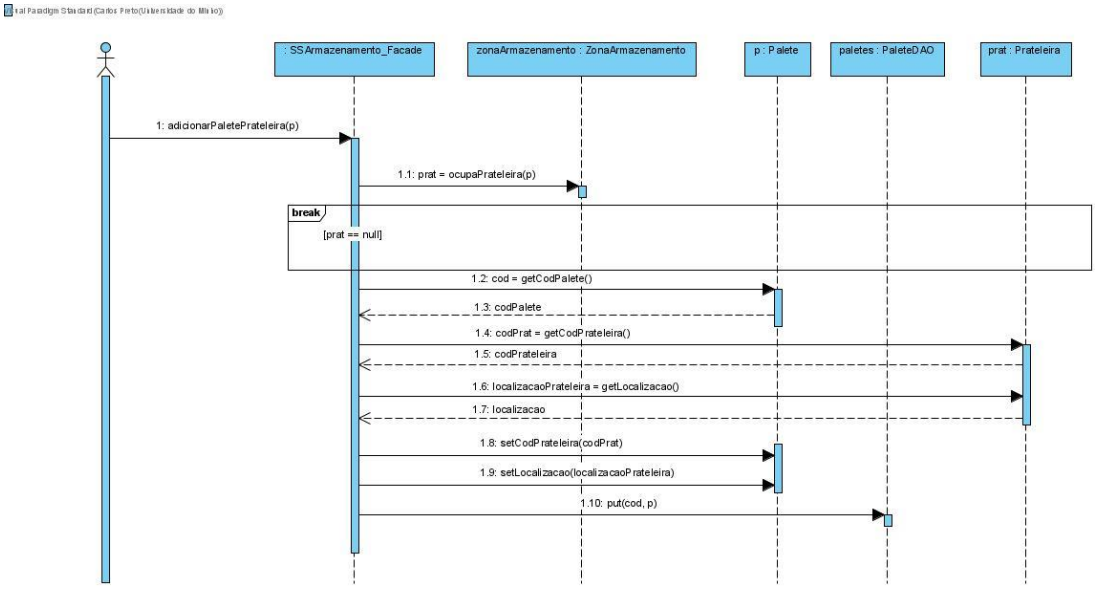


Figura 31: Diagrama de Sequência – AdicionarPaletePrateleira

8.12 Saber se há paletes à espera de transporte

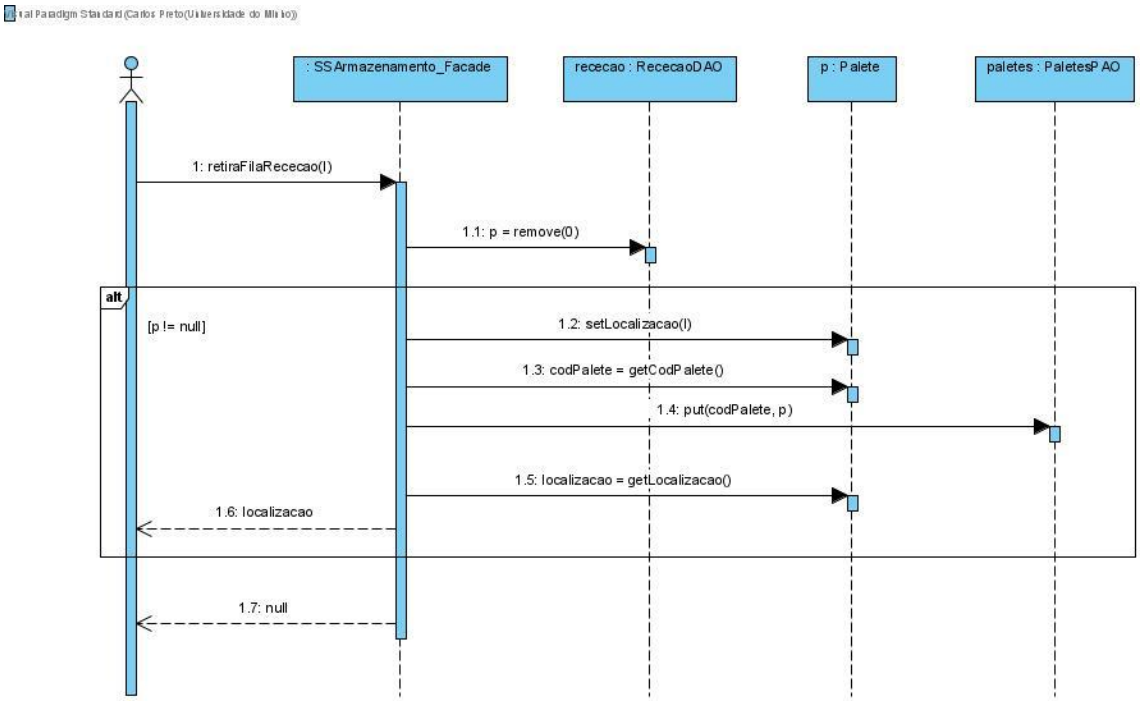


Figura 32: Diagrama de Sequência – RetirarFilaRececao

8.14 Localização da entrada

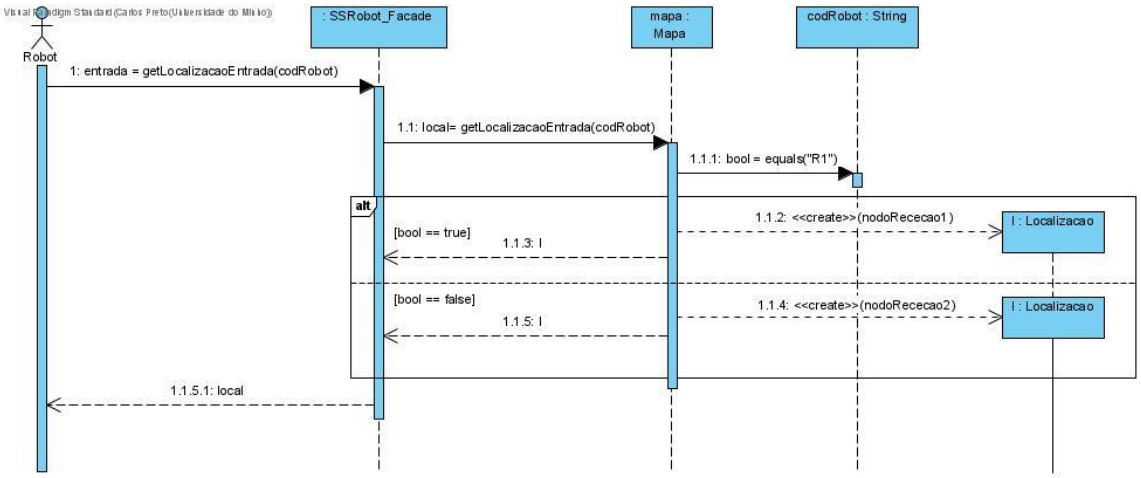


Figura 33: Diagrama de Sequência – GetLocalizacaoEntrada

8.15 Localização da *queue* receção

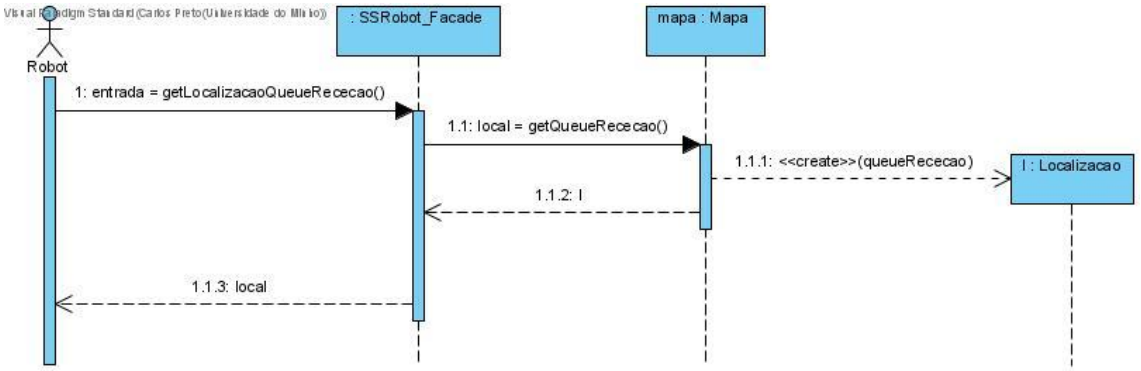


Figura 34: Diagrama de Sequência – GetLocalizacaoQueueRececao

8.16 Paletes e suas localizações

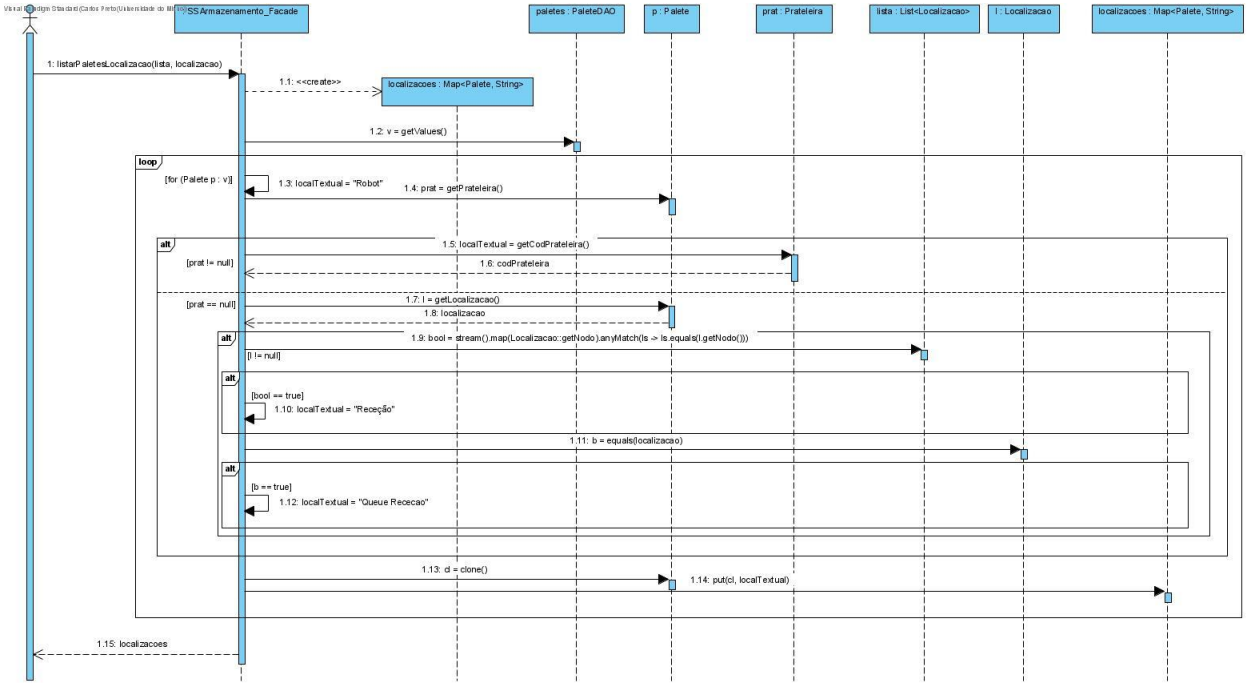


Figura 35: Diagrama de Sequência – ListarPaletesLocalizacoes