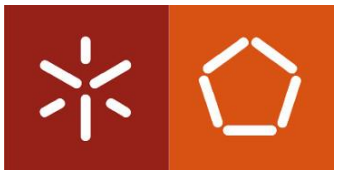


UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Projeto Final – 3ª Fase

Sistema de Gestão de um Armazém

Desenvolvimento de Sistemas de Software
21 de dezembro de 2020

(a) Ana Filipa Pereira
A89589

(b) Carolina Santejo
A89500

(c) Raquel Costa
A89464

Índice

Índice de Figuras

Introdução

Ao longo deste semestre, fomos desenvolvendo através de várias fases um sistema de gestão de um armazém de uma fábrica. Este projeto evoluiu de forma iterativa e incremental, sendo que o seu processo baseia-se resumidamente em 3 fases. O desenvolvimento faseado é uma metodologia que prossupõe que o programador deve analisar o desafio previamente antes de começar a escrever código. Assim é possível estruturar primeiro uma solução sólida e só depois implementá-la. Sendo esta a metodologia que foi utilizada para este projeto.

A primeira fase, tal como vimos, consistiu na análise dos requisitos e no escrutínio dos cenários apresentados, onde definimos o Modelo de Domínio e o Modelo de Use Cases. Já na segunda, reavaliou-se o que foi feito na primeira, fazendo-se as alterações que o grupo considerou necessárias. Além disto, definiu-se os subsistemas e os correspondentes Diagrama de Componentes, o Diagrama de Packages, os Diagramas de Classes e os de Sequência. A terceira e última fase será a que vamos abordar neste relatório.

É importante realçar que o grupo teve o cuidado de desenvolver o projeto, de forma a que este respeitasse os requisitos exigidos pela equipa docente além de ser o mais realista possível e que funcionasse independentemente da estrutura do armazém ou do número de robots que nele trabalham.

Principais Objetivos

Nesta terceira fase, a equipa docente selecionou um grupo de Use Cases para implementarmos em Java, tendo em conta todo o processo que temos vindo a desenvolver até agora, de modo a mostrar-nos como este se tratou de um processo sequencial, iterativo e incremental, e, além disso para conseguirmos termos uma noção do produto final no qual trabalhamos ao longo do semestre.

Tendo em conta os requisitos propostos, o nosso “Sistema de Gestão” terá de ser capaz de atribuir a todas as paletes que entram num certo armazém, um sítio destinado à sua arrumação, tal como também é capaz de receber notificações dos robots que transportam as mesmas. Além disso, ainda deve listar todas as paletes que estão no armazém e a sua localização, podendo estas estar numa das zonas existentes (zona de descarga ou entrega), numa prateleira, ou então num robot.

Além disso, visto que a persistência de dados foi um dos requisitos, foi necessário criar uma base de dados que conseguisse armazenar todas as informações precisas para o funcionamento do nosso software. Para tal, recorremos a classes próprias (DAOs) que estabelecem uma conexão com o “MySql”, e ainda tivemos o auxílio da ferramenta “MySql Workbench”. Desta forma, toda a informação relativa a paletes, robots e topologia do armazém fica permanentemente guardada na base de dados, não sendo apagada quando o programa termina.

Descrição da abordagem realizada

Nesta fase começamos a implementar uma parte do software, seguindo a metodologia que temos vindo a seguir. Primeiramente, houve uma seleção de quais os Use Cases a tratar, sendo esta informação fornecida pela equipa docente. De seguida fizemos uma modelação detalhada de cada um deles, fazendo a manutenção dos diagramas e corrigindo situações que não tínhamos previsto ou não tínhamos dado tanta atenção. Referimos ainda que é muito importante este tipo de estruturação pois permite corrigir e encontrar problemas que mais á frente tornar-se-iam muito dispendiosos de concertar. Finalmente, após todas as análises e decisões efetuadas passamos para a implementação dos mesmos Use Cases. Futuramente, teríamos de repetir este processo, selecionando outros Use Cases dentro daqueles que foram feitos inicialmente.

Alterações dos Modelos da Fase 1 e Fase 2

Como nesta fase optamos por armazenar as informações necessárias ao funcionamento do programa através de uma Base de Dados, e não nos limitarmos ao armazenamento em memória, foi necessário criar novas classes e, consequentemente, reformular os modelos previamente feitos. Além disso, com a modelação e especificação dos Use Cases de uma forma mais detalhada foi-nos possível desmascarar qualquer comportamento que não tínhamos previsto anteriormente como também antecipar possíveis problemas de insatisfação de requisitos.

Iremos agora abordar os vários modelos e diagramas que necessitaram de sofrer algum tipo de alteração com os novos requisitos e com a fase de Análise e Modelação Detalhada de cada Use Case selecionado.

Especificações de Use Cases

Após uma reavaliação dos requisitos e uma análise detalhada do trabalho que temos vindo a desenvolver, decidimos reformular os seguintes Use Cases selecionados para esta fase:

Leitura de códigos QR

• Ator: Leitor.

• Pré-Condição: Todas as paletes passam pelo leitor.

• Pós-Condição: Sistema fica com o registo das paletes que chegaram.

• Fluxo normal:

1. Leitor lê o código QR.
2. Sistema valida código QR.
3. Palete é registada no sistema.

• Fluxo Exceção 1: [Código não está na base de dados] Passo 2

2.1 Sistema comunica que produto não faz parte do armazém.

Consultar listagem de localização

• Ator: Gestor.

• Pré-Condição: Gestor tem de estar autenticado.

• Pós-Condição: Sistema disponibiliza lista.

• Fluxo normal:

1. Gestor pede listagem das paletes ao Sistema.
2. Sistema disponibiliza a listagem.

Notifica recolha de palete

• Ator: Robot.

• Pré-Condição: Robot já foi solicitado pelo Sistema para transportar a palete.

• Pós-Condição: Sistema atualiza a localização da palete para o robot.

• Fluxo normal:

1. Robot indica ao Sistema que já chegou à localização da palete e que já a recolheu.

Notifica entrega de paletes ao destino

• Ator: Robot.

• Pré-Condição: Palete tem de estar localizada num robot.

• Pós-Condição: Sistema atualiza a localização da palete para uma das zonas (destino).

• Fluxo normal:

1. Robot indica ao Sistema que já chegou ao destino da palete e que já a entregou.

Comunica Ordem de Transporte / Notifica Transporte da Palete

• Ator: Robot.

• Interação: Sistema → Robot.

• Pré-Condição: Robot está disponível.

• Pós-Condição: Estado do Robot atualiza para não disponível.

• Fluxo normal:

1. Sistema indica ao robot a palete a transportar.
2. Sistema indica ao robot onde a palete se encontra.
3. Sistema indica ao robot destino da palete.
4. Sistema calcula percurso a efetuar pelo robot.
5. Sistema envia o percurso ao robot.

Diagrama de Componentes

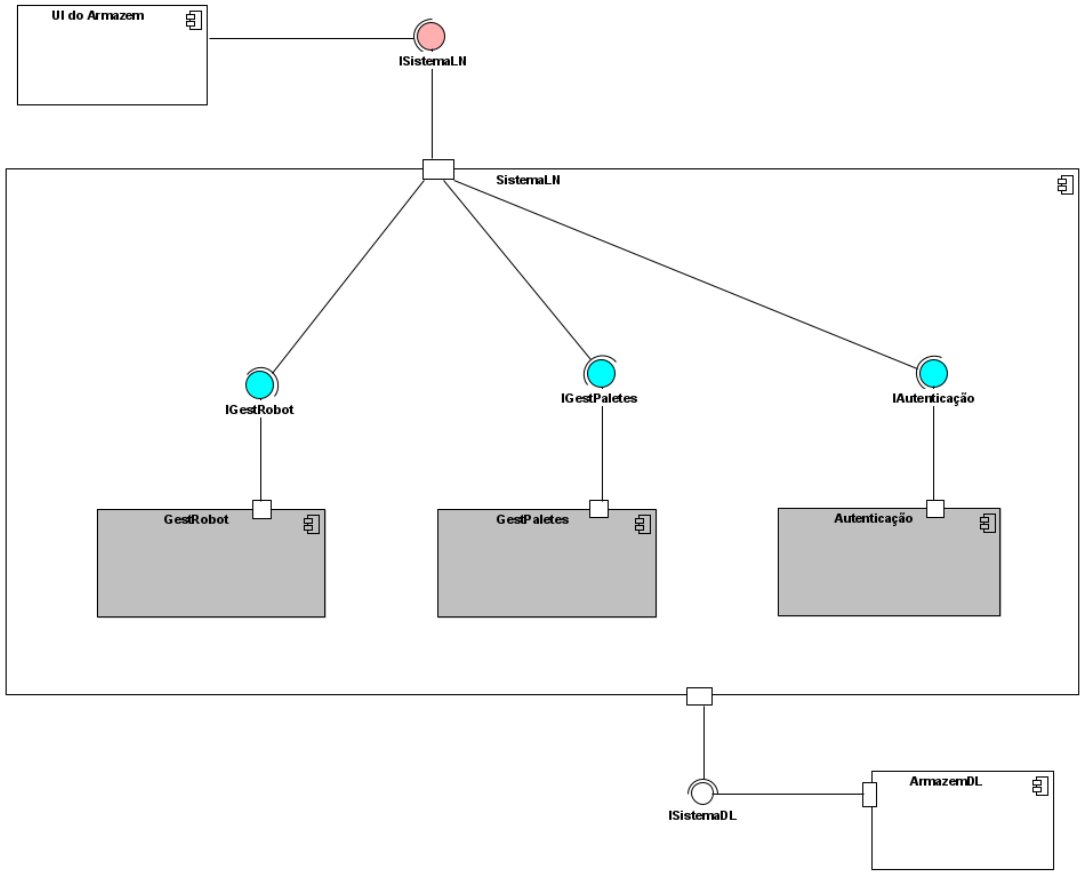


Figura 1 - Diagrama de Componentes

Como foi referido anteriormente, nesta fase do projeto foi necessário guardar os dados em bases de dados. Sendo assim, no diagrama de componentes foi necessário acrescentar um novo sistema ArmazemDL que é responsável por aceder e efetuar quaisquer alterações em relação aos dados armazenados na base de dados. Consequentemente, foi também acrescentada a interface ISistemaDL que é responsável por fazer a ligação entre os sistemas da logica de negócio (SistemaLN) e da camada de dados (ArmazemDL).

Diagrama de Packages

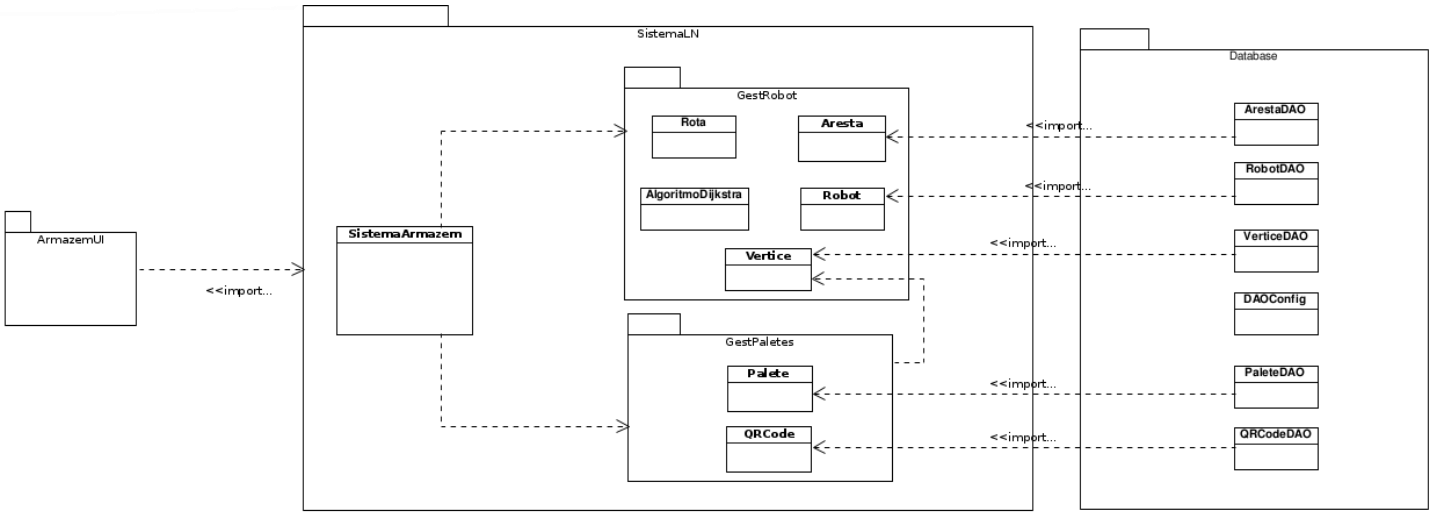


Figura 2 - Diagrama de Packages

Nesta fase, um dos requisitos foi a persistência de dados, ou seja, estes teriam de ficar guardados numa base de dados e não apenas em memória. Para isto, foi preciso utilizar classes DAO que salvaguardam os objetos das classes a que correspondem. Ao todo foram criadas 5 classes DAO: *RobotDAO*, *VerticeDAO*, *PaleteDAO*, *QrCodeDAO*, *ArestaDAO*. De forma a melhor organizar a estrutura do projeto foi criado um package, denominado “database”, com todos os DAOs referidos.

Concluindo, teremos dois packages principais: o package database, que contém todas as classes necessárias á preservação de dados, e o package da lógica de negócio, no qual estão incluídos o package relativo às paletes e o package relativo aos robots.

Diagramas De Classes

GestRobot

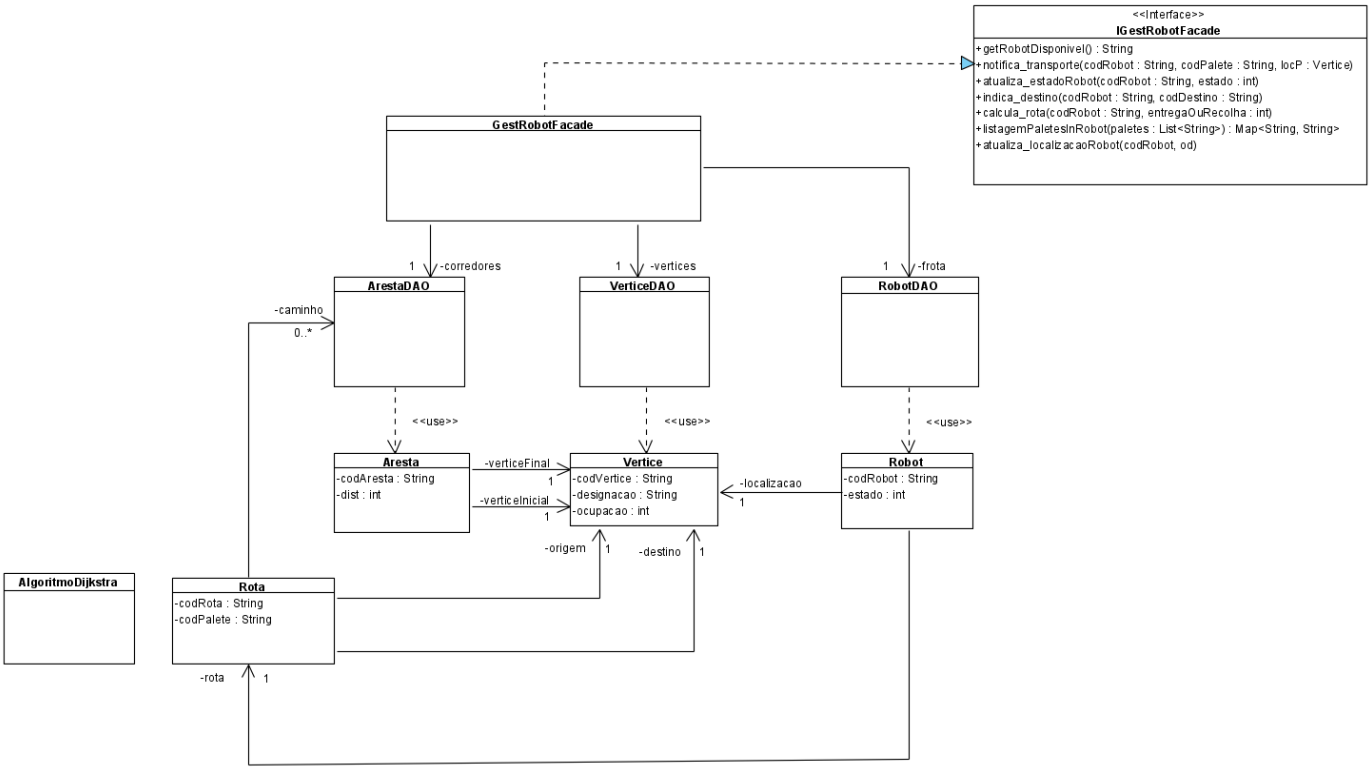


Figura 3 - Diagrama de Classes : GestRobot

Depois de uma análise e de um estudo mais detalhado dos requisitos necessários, e de uma reconstrução dos Use Cases, como também dos diagramas que nos permitem obter uma estrutura geral do trabalho, houve a necessidade de reformular os diagramas de classes feitos na fase passada. Tendo em conta o surgimento do novo requisito relacionado com a persistência dos dados do programa em desenvolvimento, foi necessário substituir as relações de composição que antes tínhamos com o *Facade*, cujo objetivo era guardar os dados do programa em memória, por classes DAO. Estas classes não só permitem a persistência dos objetos na Bases de Dados, como também criam objetos a partir da informação disponível na BD, além disso também encapsulam *queries* SQL. Desta forma obtemos uma separação clara e concisa entre a Camada de Dados e a Lógica de Negócio, permitindo assegurar a facilidade de manutenção do código a implementar.

Além da adição das classes DAO, fizemos também algumas mudanças e melhorias já pensadas em realizar no final da 2ª fase. Antes considerávamos que os corredores, tinham prateleiras para armazenar apenas um tipo de produto, mas agora com um novo levantamento dos requisitos e exploração dos mesmos, decidimos considerar que cada prateleira corresponde a um vértice de um

Grafo representativo do Mapa do Armazém em causa. Antes os vértices eram apenas pontos de interseção dos corredores, agora consideramos que os corredores são arestas e que cada canto, zona ou prateleira do Armazém é um vértice. Devido a isto houve uma simplificação e clarificação da abordagem do grupo, que na nossa opinião reflete-se no diagrama apresentado.

Achamos importante considerar que as classes Aresta, Vértice e Robot deveriam ter um DAO correspondente a cada uma delas de modo a persisti-las, já que são consideradas informações que nós devemos ter disponível na base de dados para que possamos aceder sempre que quisermos. Além disso, são dados que devemos manter e guardar, mesmo que o programa termine para que depois seja possível retomar o mesmo sem perder o progresso que foi feito durante a sua execução.

É também possível observar o surgimento de duas novas classes, *Rota* e *AlgoritmoDijkstra*. A classe *Rota* é onde iremos guardar a lista de arestas que representam o caminho que um Robot deverá tomar, sendo assim, cada robot tem um objeto Rota que poderá ser null ou não dependendo da ação que o sistema lhe atribui. É importante referir que esta informação é armazenada na Base de Dados, mas o grupo considerou que não era necessário um DAO para tal já que era informação que pertencia ao Robot, veremos mais à frente como é que esta situação foi tratada. Em relação à classe *AlgoritmoDijkstra*, esta surgiu durante a fase de implementação do programa, uma vez que houve a necessidade de criar uma classe "auxiliar" de forma a organizar e a clarificar o código, originando assim uma divisão concisa, já que a estratégia adaptada pelo grupo para a resolução do problema de encontrar o caminho mais curto entre dois vértices é algo complexa. Como os diagramas não só servem para planear um trabalho, mas como também para documentá-lo, acrescentou-se esta classe, é possível observar que ela não se encontra associada a nenhuma classe, isto porque ela é apenas "chamada" no método do facade "calculaRota"

GestPaletes

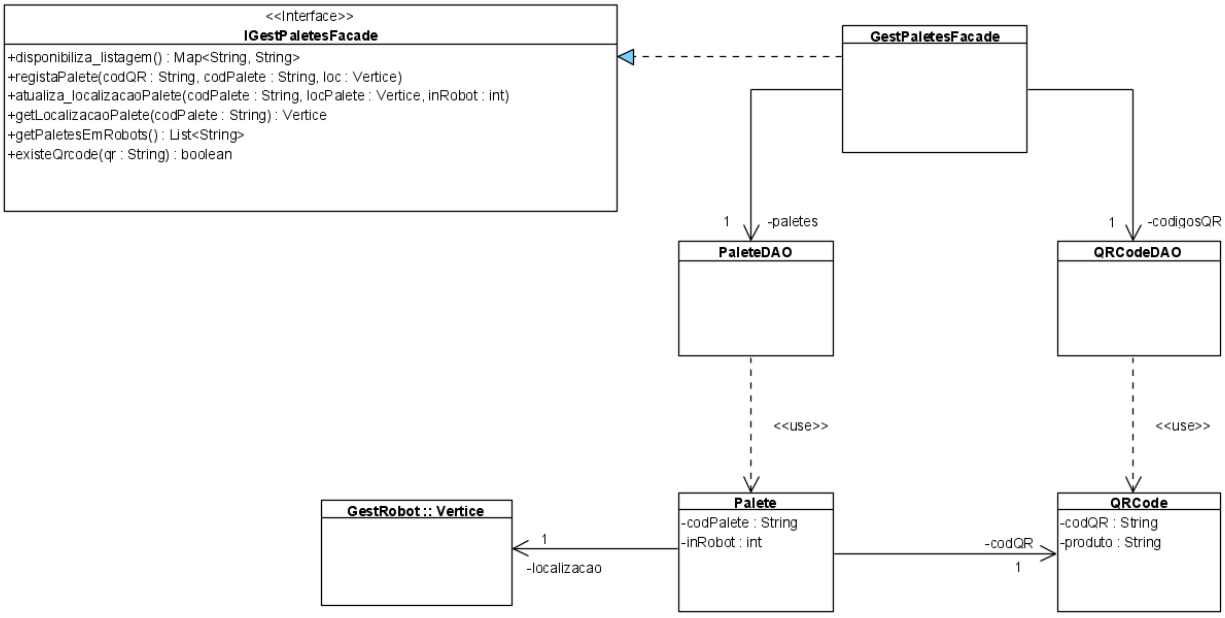


Figura 4 - Diagrama de Classes: GestPaletes

Tal como vimos, foi necessário acrescentar também aqui classes DAO, com os mesmos propósitos referidos no tópico anterior. Este diagrama foi reformulado tendo apenas em conta os Use Cases selecionados para implementar nesta fase. Hipoteticamente, caso houvesse uma próxima fase onde tivéssemos de implementar mais Use Cases dentro daqueles que foram definidos na primeira fase deste projeto, teríamos de acrescentar aqui classes (ou não) de forma a completar este diagrama e a responder aos requisitos estabelecidos.

Além disso, é possível também observar que houve uma simplificação referente à classe *QRCode*. Na fase anterior, tínhamos definido que cada corredor armazenava apenas paletes que tivessem o mesmo QRCode, sendo assim a classe em questão tinha um atributo que identificava qual seria esse corredor onde as paletes deveriam ser armazenadas. Agora deixou de existir, uma vez que, tratamos todas as paletes da mesma maneira independentemente do seu Código QR ou do tipo da matéria-prima.

Na classe *Palete* também surgiram algumas alterações. Foi eliminado o atributo "codigoRobot", onde considerávamos que se estivesse preenchido saberíamos automaticamente em que Robot é que a palete se encontrava. Sendo que, depois de rever e avaliar essa opção, chegamos à conclusão que seria informação repetida uma vez que já temos essa informação armazenada através da classe Rota no *GestRobots*. Portando, adicionamos o atributo "inRobot" de modo a indicar-nos se a palete se encontra num Robot, ou "reservada" para ser levantada por um, ou então nenhuma destas opções.

Finalmente, mas não menos importante, acrescentamos o método "existeQRCode" à interface do facade, de modo a indicar-nos se o QRCode lido pertence ou não à base de dados do sistema.

Diagramas De Sequência

Devido a todas as alterações feitas, foi também necessário modificar, retirar e acrescentar alguns métodos em relação à fase 2. Com a especificação detalhada dos Use Cases analisados nesta fase, decidimos retirar alguns métodos que tínhamos planeado anteriormente, o *ValidaRota*, *AdicionaRota*, e por fim, o *CancelaRota*, uma vez que recebemos a informação de que os robots eram autónomos e eles próprios tinham um Sistema externo ao nosso. Por outro lado, foram acrescentados dois métodos (*ListagemPaletesInRobot* e *GetPaletesEmRobots*) que achamos necessários para esta fase do projeto. Quanto aos restantes foi necessário, por um lado, corrigir alguns erros cometidos, mas também efetuar as alterações necessárias devido às novas classes adicionadas, nomeadamente a utilização de DAO's uma vez que que foi utilizada uma nova forma de guardar os dados.

Assim, para esta fase do trabalho, os diagramas de sequência dos métodos relevantes de cada subsistema são os seguintes:

1. DisponibilizaListagem

Listagem de paletes que não estão a ser transportadas associadas ao vértice onde se localizam.

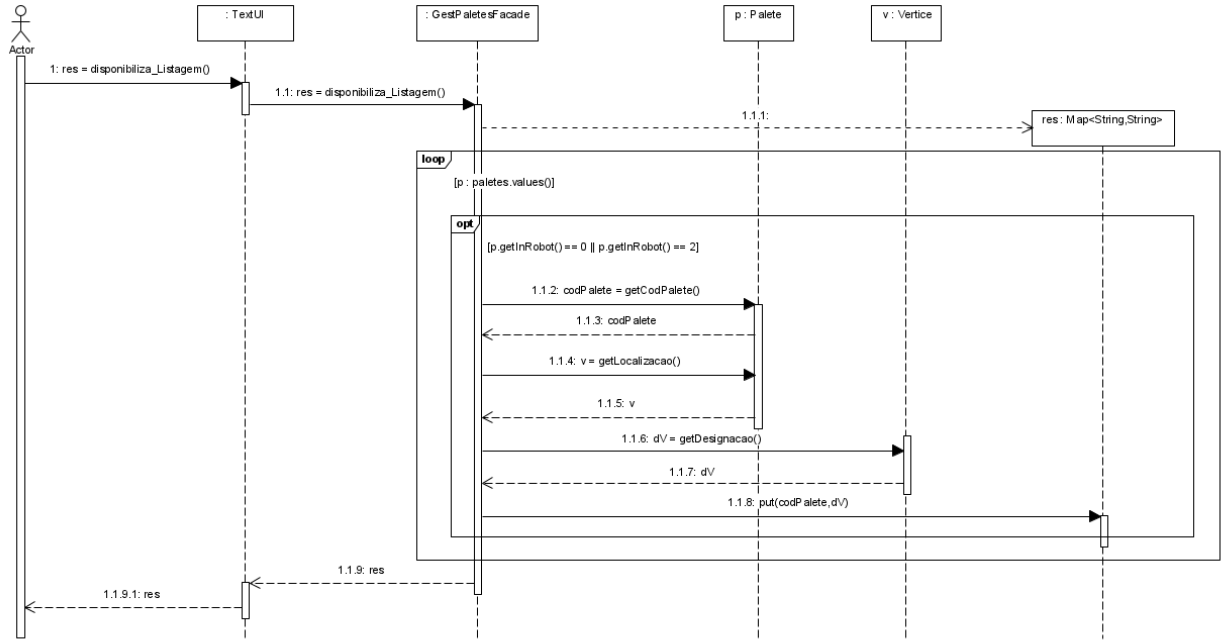


Figura 5 - DisponibilizaListagem

2. RegistaPalete

Regista nova palete no sistema.

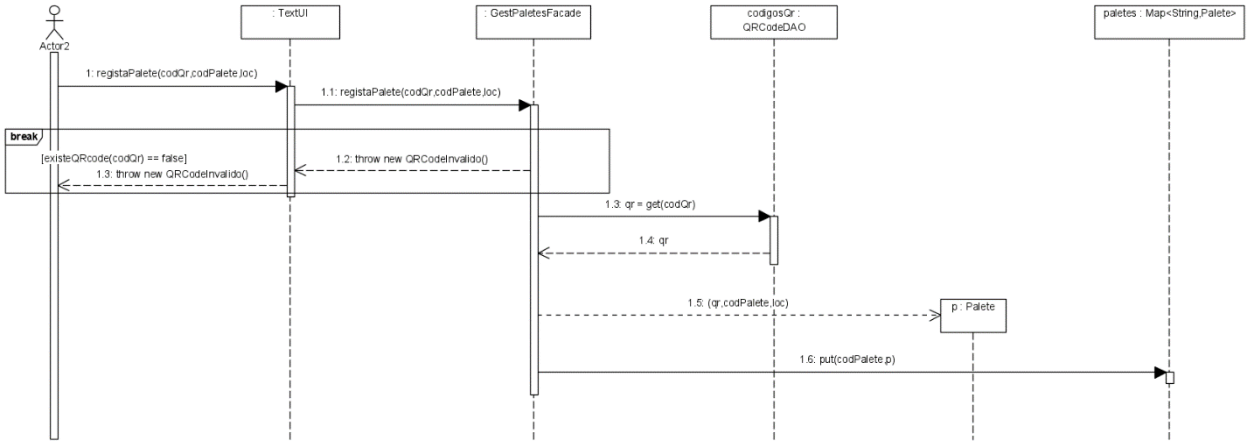


Figura 6 - RegistaPalete

3. AtualizaLocalizaçãoPalete

Atualiza a localização de uma Palete

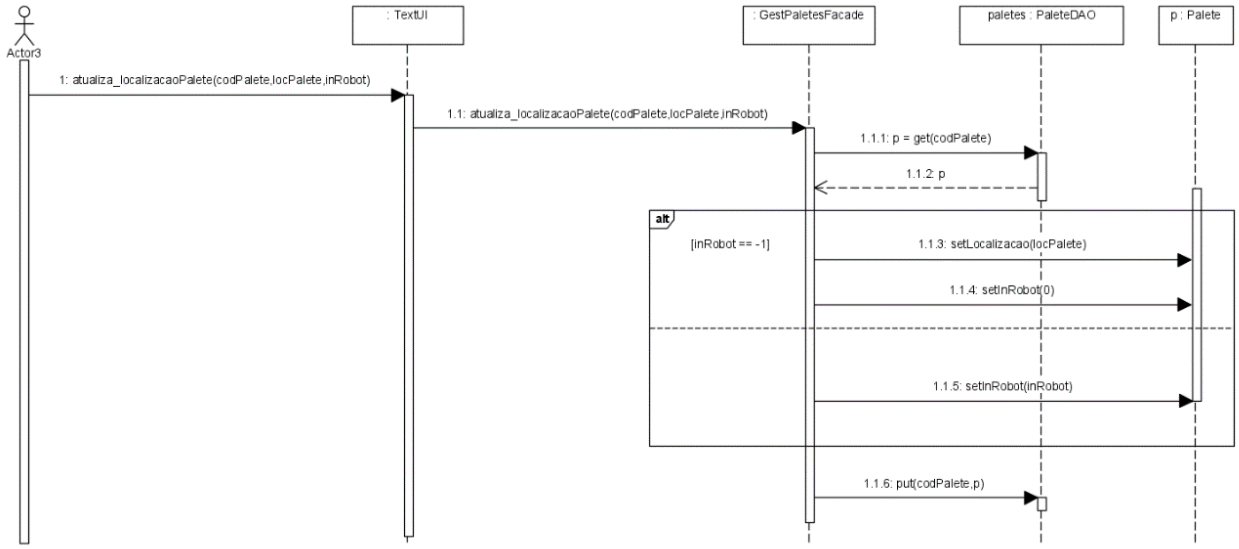


Figura 7 - AtualizaLocalizaçãoPalete

4. GetPaletesEmRobots

Lista dos códigos das paletes que estão a ser transportadas em Robots

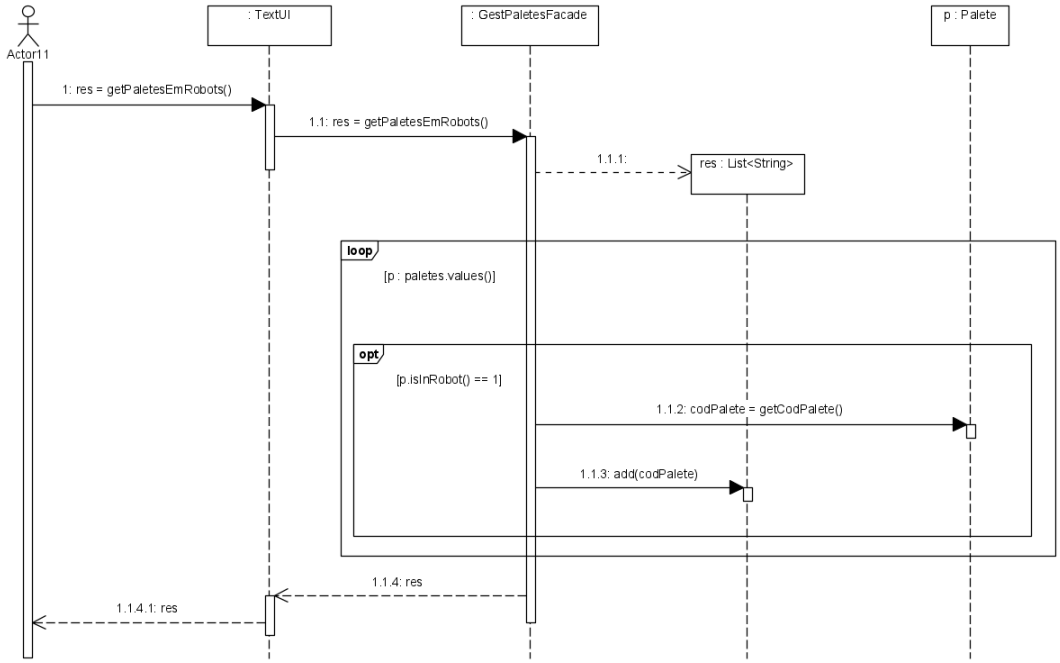


Figura 8 - GetPaletesEmRobots

5. GetRobotDisponivel

Código de um Robot que esteja disponível para iniciar novo percurso.

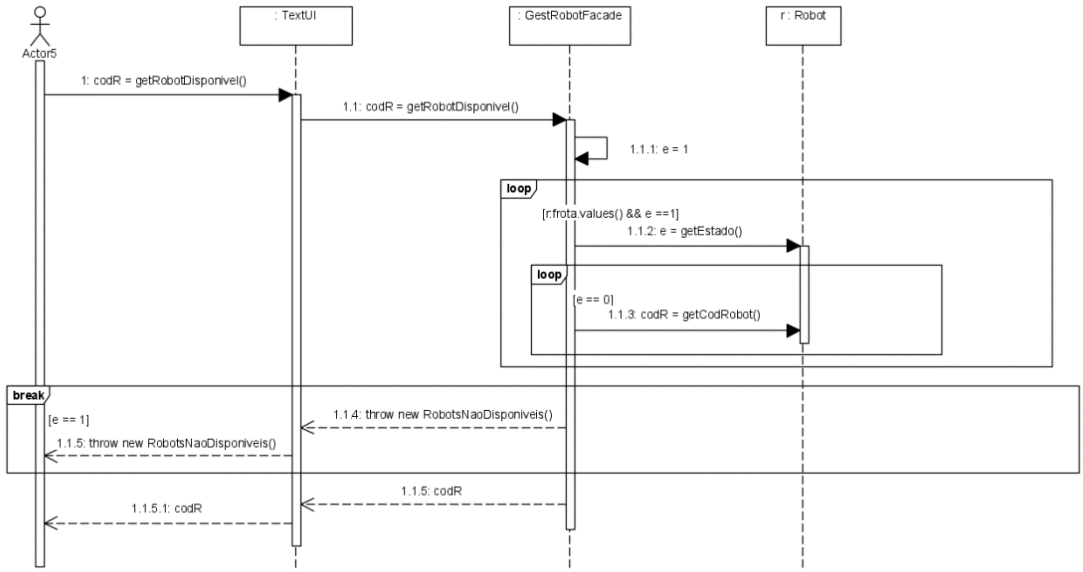


Figura 9 - GetRobotDisponivel

6. Notifica_Transporte

Notificar o transporte de uma palete a um Robot

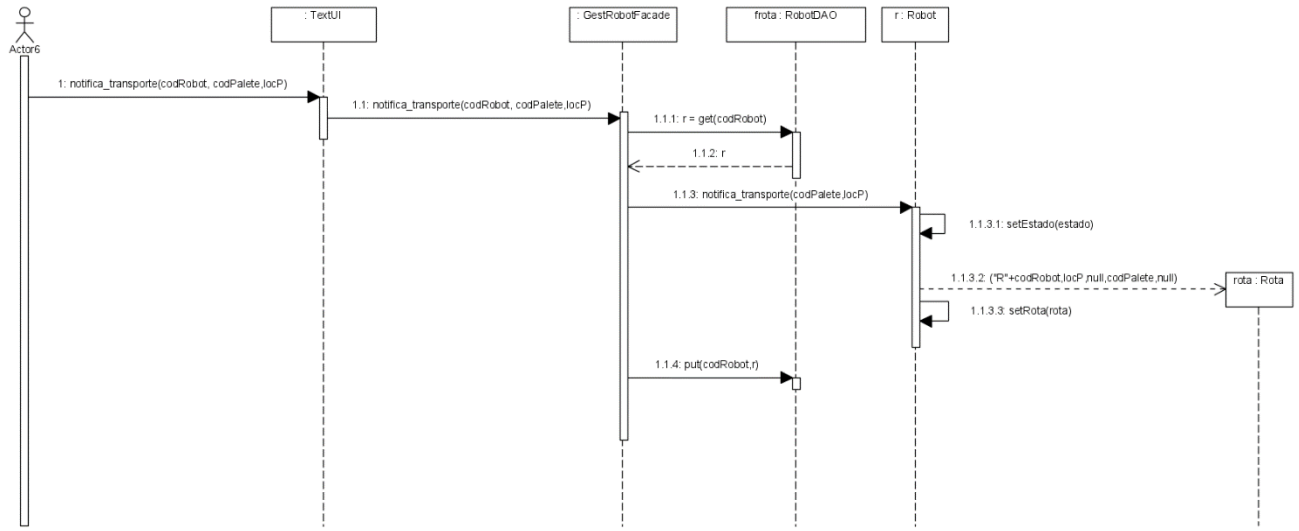


Figura 10 - NotificaTransporte

7. AtualizaEstadoRobot

Atualizar estado de um Robot

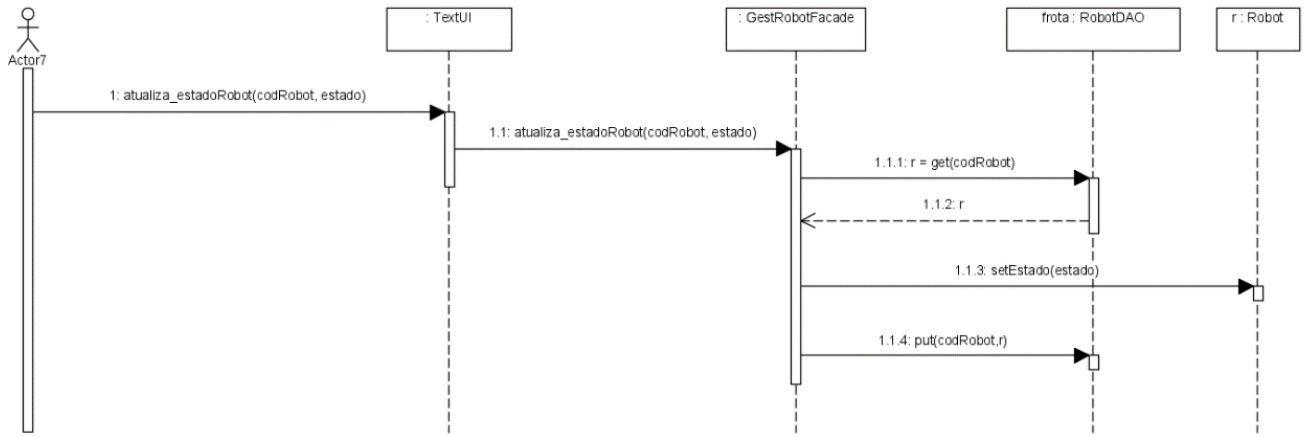


Figura 11 - AtualizaEstadoRobot

8. IndicaDestino

Indicar destino ao Robot que pretende efetuar um percurso

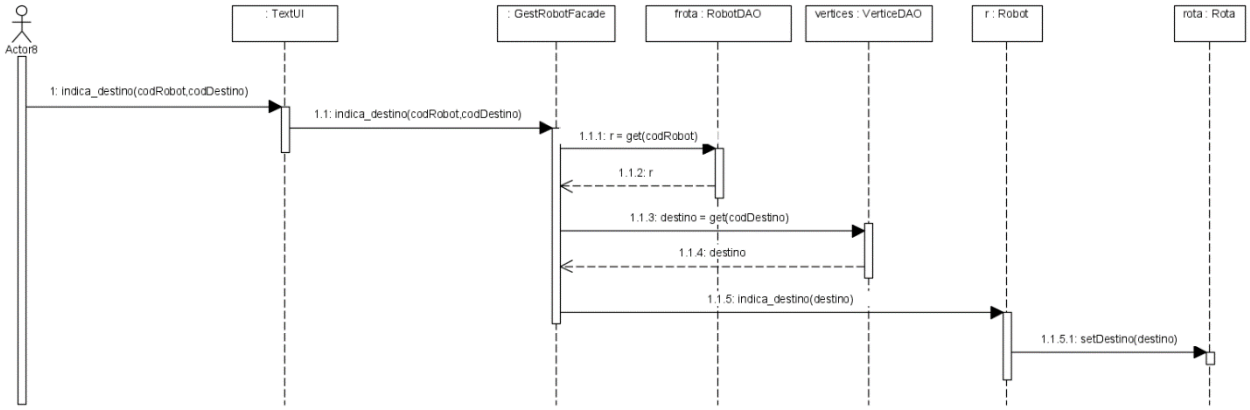


Figura 12 - IndicaDestino

9. AtualizaLocalizacaoRobot

Atualizar localização de um Robot.

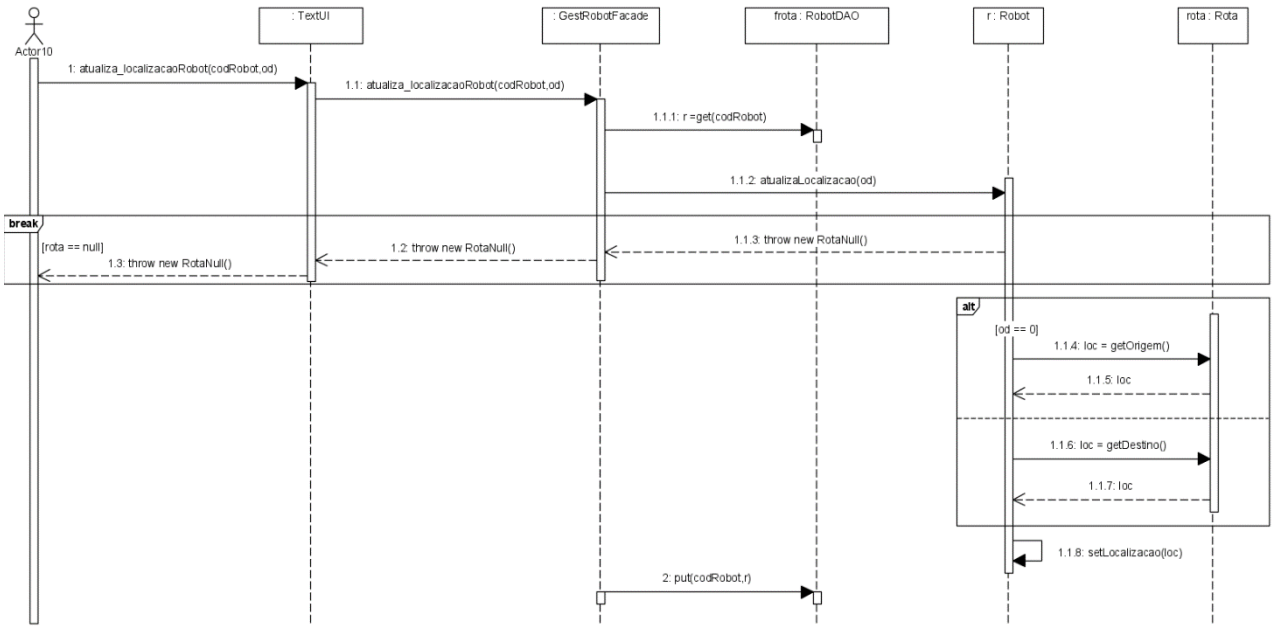


Figura 13 - AtualizaLocalizacaoRobot

10. CalculaRota

Calcular o caminho a percorrer pelo Robot para a recolha ou entrega de uma palete. Para efetuar este cálculo foi utilizada a classe AlgoritmoDijkstra que possui os métodos do algoritmo de *Dijkstra* que são necessários para calcular a lista dos vértices a percorrer para efetuar o caminho mais curto entre dois vértices de um grafo.

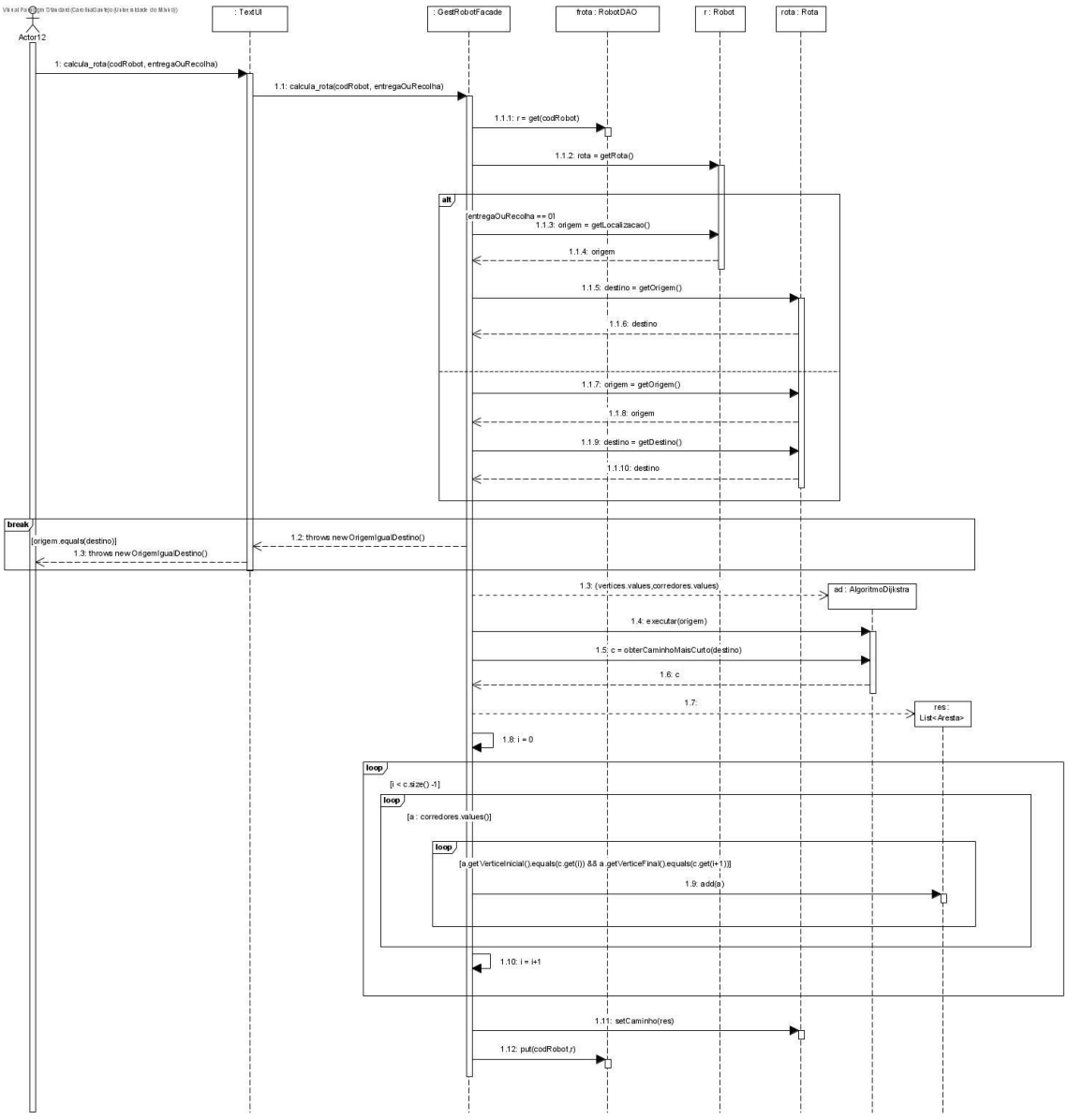


Figura 14 - CalculaRota

11. ListagemPaletesInRobot

Listagem de paletes associadas ao código do Robot que a transporta.

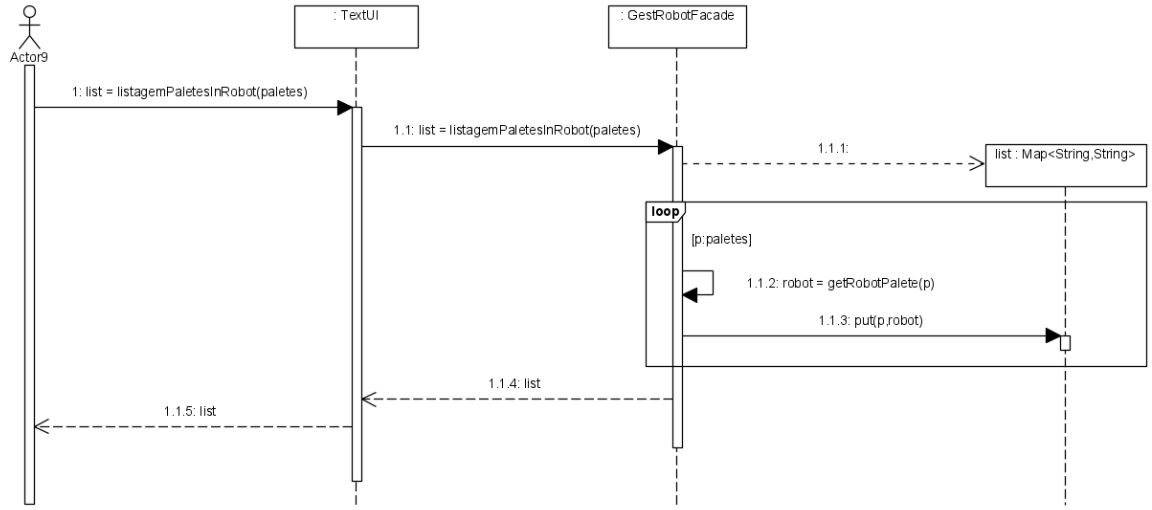


Figura 15 - ListagemPaletesInRobot

Implementação

Interface com o Utilizador

Modelo Máquina de Estado do Sistema

Quando um utilizador usa o *software* tem disponível dois submenus. O "Gestão de Paletes" que possui todas as opções relativas às paletes e o "Notificações de Robots", onde o utilizador pode solicitar as várias notificações que um Robot pode mandar ao Sistema, sendo que esta comunicação é simulada na interface.

Quanto às paletes existem duas funcionalidades: "Registar Nova Palete" e "Consultar Listagem de Paletes". No entanto, enquanto a primeira está sempre disponível, a segunda apenas aparece como opção caso haja paletes noa armazém.

Relativamente aos Robots existem também duas funcionalidades: "Notificar Recolha Paletes" e "Notificar Entrega Paletes". A primeira é invocada para que o robot notifique ao sistema que pegou na palete, e a segunda é invocada para que o robot notifique ao sistema que pousou essa mesma palete.

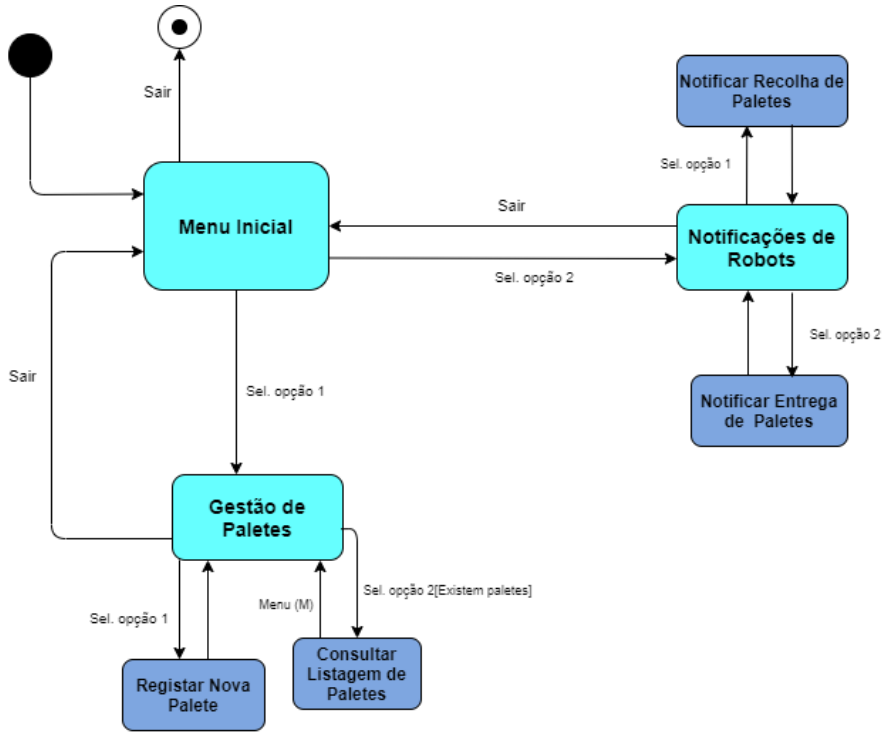


Figura 16 - Máquina de Estado

Base de Dados

De modo a guardar toda a informação e dados necessários ao funcionamento do nosso programa, implementamos uma Base de Dados, capaz de armazenar tudo isto, deixando assim o armazenamento em memória de lado. Desta forma, conseguimos garantir a persistência dos dados do nosso programa. Para implementar a Base de Dados escolhemos o "MySql" como sistema de gestão de base de dados. Foram criados 3 scripts, um para criar um utilizador e dar-lhe acesso à BD construída, outro para povoar as tabelas com dados já previamente definidos, tais como os robots, arestas, vértices do mapa do Armazém e *QrCodes* de cada produto. E, finalmente, um script adicional com várias queries que nos permitem aceder a toda a informação que está a ser armazenada ao longo do decorrer do programa.

Modelo Lógico

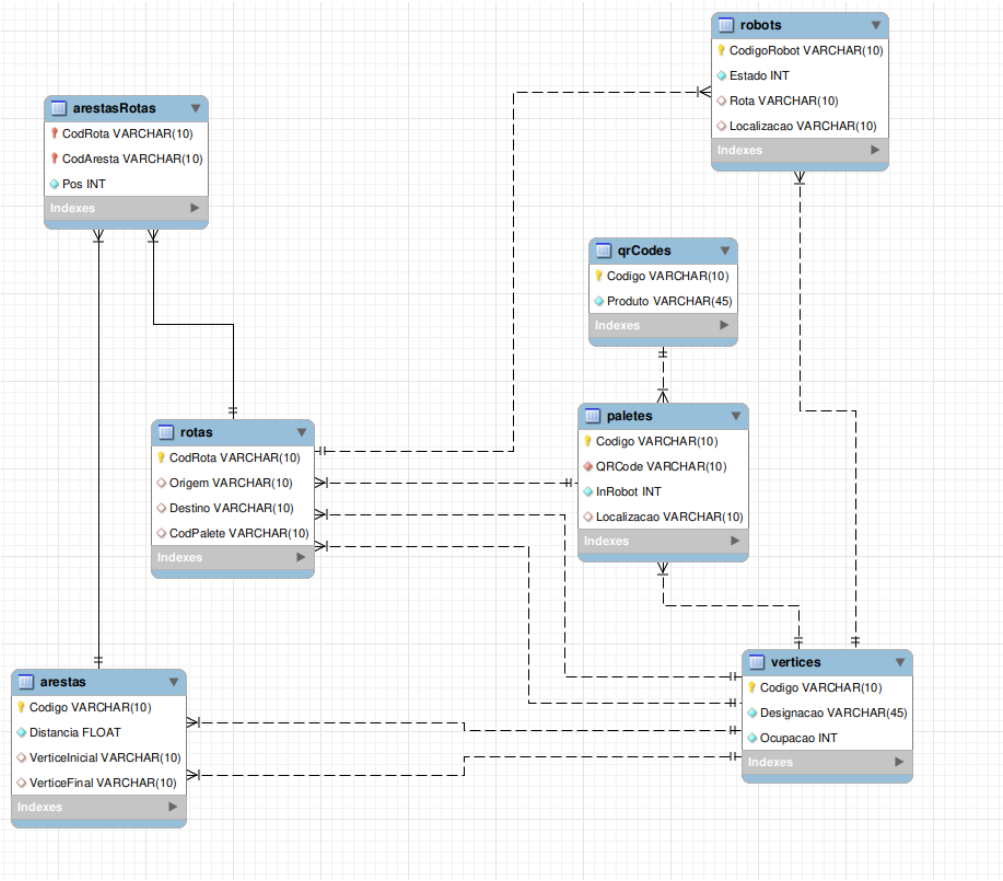


Figura 17 - Modelo Lógico da Base de Dados

- **QRCodes:**

Esta tabela tem como chave primária o seu código, e como atributo a designação do produto que representa. É importante referir que o grupo considerou que um armazém apenas recebe um grupo limitado de produtos, isto é um armazém de uma fábrica têxtil não irá receber produtos que uma fábrica de indústria alimentar receberia. Consideramos então que o armazém está preparado para receber uns certos produtos e que qualquer produto que não esteja dentro do esperado não deverá ser registado ou entrar no armazém. Para tal foi necessário povoar esta tabela com produtos que o armazém está pronto a receber.

- **Paletes:**

Contém como chave primária o seu código, e tem como atributos a chave estrangeira correspondente ao código QR que a palete tem. A sua localização, ou seja, um ponto/vértice do Mapa do Armazém, que poderá ser NULL, caso a Palete se encontre num Robot. Além disso, temos ainda o atributo "inRobot" que poderá tomar o valor 0, caso a palete não esteja num robot, 1 caso a palete esteja efetivamente a ser transportada por um robot, e 2 nas situações em que a palete está á espera de ser levantada pelo robot ao qual ela já foi destinada.

- **Vertices:**

Cada vértice tem um código que é a chave primária da tabela, e tem uma designação, podendo este ser uma prateleira, uma zona (descarga ou entrega, mas para já só consideramos a zona de entrega), ou então um canto do mapa. Além disso, contém também o atributo ocupação que poderá tomar o valor 0 para o caso de não ter atingindo a sua capacidade máxima de paletes, e 1 assim que essa capacidade é atingida. É importante referir que a capacidade de cada prateleira é 1 palete, para as zonas e para os cantos não consideramos qualquer tipo de limite na capacidade, portanto nestes vértices a ocupação é sempre 0. Também acrescentamos o valor 2 para os casos em que uma prateleira fica reservada para uma determinada palete, evitando desta forma que o sistema atribua a mesma prateleira a duas paletes diferentes.

- **Arestas:**

As arestas representam os corredores existentes no armazém, entre as várias prateleiras, zonas e cantos. Cada uma tem o seu código, distância, e ainda o vértice inicial e o vértice final que cada uma conecta.

- **Robots:**

Cada robot tem o seu código que é único, o estado, podendo ser 0 se o robot estiver livre e 1 caso ele esteja ocupado a transportar uma palete, ou então tenha sido solicitado pelo sistema. Além disso, contém ainda a Rota que o sistema lhe atribui caso ele seja notificado para transportar alguma palete, podendo esta ser NULL, caso ele esteja livre e sem tarefas para realizar. Tem também uma localização, sendo esta, como vimos, um vértice do mapa do armazém.

- **Rotas:**

Para calcular uma rota ou percurso que um robot tem de tomar temos de ter em consideração 3 localizações, a localização do Robot, onde a Palete está, e o destino pretendido para a palete. Para tal consideramos o atributo "origem" como o vértice onde a palete se encontra, e o "destino" o vértice para o qual a palete deve ser transportada. Além disso guardamos ainda o código da palete

que deverá ser transportada. É importante referir que cada robot só tem uma rota pois um robot não consegue percorrer mais do que um percurso ao mesmo tempo e apenas transporta uma palete de cada vez, portanto sempre que este é notificado para fazer um transporte, é criada uma nova rota e é criada uma nova entrada na Base de Dados, mas assim que ele termina, esta rota é retirada da BD.

- **ArestasRotas:**

Para representar o caminho, isto é as arestas que um robot deverá percorrer, sentimos a necessidade de criar esta tabela, uma vez que, a relação entre a tabela Arestas e a tabela Rotas é de N:N, isto é para n arestas pode haver n rotas, sendo que cada uma dessas rotas tem n arestas. Portanto consideramos como chave primária o par das chaves estrangeiras referentes ao código da rota e do código de cada aresta. Além disso, adicionamos também o atributo "Pos" de forma a conseguirmos obter no final o caminho pela ordem correta.

Povoamento

```
USE `SistemaArmazem` ;
-- DELETE FROM `SistemaArmazem`.`qrCodes`;
-- SELECT * FROM `SistemaArmazem`.`qrCodes`;
INSERT INTO `SistemaArmazem`.`qrCodes`
(Codigo,Produto)
VALUES
('1','Ferrero Rocher'),
('2','Lindor'),
('3','Merci'),
('4','ChesterField Gold'),
('5','Mon Cheri'),
('6','Rafaellos'),
('7','Regina'),
('8','Guylian'),
('9','Champs Elysees'),
('10','Rondnoir');


-- DELETE FROM `SistemaArmazem`.`vertices`;
-- SELECT * FROM `SistemaArmazem`.`vertices`;
INSERT INTO `SistemaArmazem`.`vertices`
(Codigo,Designacao,Ocupacao)
VALUES
('1','Zona D',0),
('2','Canto A','0'),
('3','Canto B','0'),
('4','Canto C','0'),
('5','Canto D','0'),
('6','P1','0'),
('7','P2','0'),
('8','P3','0'),
('9','P4','0'),
('10','P5','0'),
('11','P6','0'),
('12','P7','0'),
('13','P8','0'),
('14','P9','0'),
('15','P10','0');
```

```
-- DELETE FROM `SistemaArmazem`.`arestas`;
-- SELECT * FROM `SistemaArmazem`.`arestas`;
INSERT INTO `SistemaArmazem`.`arestas`
(Codigo,Distancia,VerticeInicial,VerticeFinal)
VALUES
('1','2','1','2'),
('2','2','2','6'),
('3','4','6','7'),
('4','4','7','8'),
('5','4','8','9'),
('6','4','9','10'),
('7','2','10','4'),
('8','5','4','5'),
('9','2','5','15'),
('10','4','15','14'),
('11','4','14','13'),
('12','4','13','12'),
('13','4','12','11'),
('14','2','11','3'),
('15','5','3','2'),
('16','2','2','1'),
('17','2','6','2'),
('18','4','7','6'),
('19','4','8','7'),
('20','4','9','8'),
('21','4','10','9'),
('22','2','4','10'),
('23','5','5','4'),
('24','2','15','5'),
('25','4','14','15'),
('26','4','13','14'),
('27','4','12','13'),
('28','4','11','12'),
('29','2','3','11'),
('30','5','2','3');
```

```
-- DELETE FROM `SistemaArmazem`.`robots`;
-- SELECT * FROM `SistemaArmazem`.`robots`;
INSERT INTO `SistemaArmazem`.`robots`
(CodigoRobot,Estado,Rota,Localizacao)
VALUES
('Wall-E','0',null,'1'),
('Xiaomi','0',null,'1'),
('Etelvina','0',null,'1');
```

Figura 18 - Script do Povoamento da BD

Script Adicional e Criação de Utilizador

```
CREATE DATABASE SistemaArmazem;

CREATE USER 'g19'@'localhost' IDENTIFIED BY 'G19.1234567';
GRANT ALL PRIVILEGES ON * . * TO 'g19'@'localhost';
```

Figura 19 - Script Create User

```
USE SistemaArmazem;
SELECT * FROM vertices;
SELECT * FROM paletes;
SELECT * FROM arestas;
SELECT * FROM robots;
SELECT * FROM rotas;
SELECT * FROM qrCodes;
SELECT * FROM arestasRotas;
```

Figura 20 - Script Adicional

Lógica de Negócio

Alterações das Responsabilidades

Tal como vimos, para esta terceira fase foi apenas necessário implementar em Java algumas das responsabilidades levantadas nas fases anteriores, sendo que futuramente selecionaríamos um novo grupo de responsabilidades para implementar. Através da modelação detalhada de cada Use Case selecionado, tivemos de aperfeiçoar as responsabilidades vistas anteriormente, ou excluir alguma que realmente não fizesse sentido implementar ou que não estivesse de acordo com os requisitos. Desta forma, verificaram-se algumas alterações na lógica de negócio.

Durante a execução da fase 2, e visto que os produtos poderiam ser ou não perecíveis, e que cada um deles estava destinado a uma zona em específico, o grupo tinha decidido que cada corredor do armazém seria utilizado para guardar paletes de um só dado produto. Isto é, cada produto existente no armazém teria o seu corredor. No entanto, como nesta fase foi pedido para ignorar o tipo dos produtos e tratar todas as paletes da mesma forma, as paletes podem ser guardadas em qualquer prateleira e em qualquer corredor, desde que a prateleira em questão esteja disponível.

Além disto, o grupo desde o início do projeto, contemplou a hipótese de que o armazém poderia ter vários robots a movimentarem-se e que seria necessário evitar a sua colisão. Isto foi feito através de um método denominado “validaRota” que indicava se numa determinada rota haveria risco de choque. Contudo, foi-nos dito que os robots se moviam de forma autónoma, conseguindo “desviar-se” de outros robots. Assim, não necessário incluir este método na implementação.

Classe Auxiliar- Algoritmo Dijkstra

Um dos requisitos desta fase consistiu em que cada vez que um robot se movimentasse de um local para outro, deveria efetuar o percurso mais curto. Visto que a implementação do armazém se baseia num grafo, no qual as zonas, as prateleiras e cantos são vértices (Figura 21), o grupo decidiu utilizar o já estudado algoritmo de Dijkstra para calcular a rota mais curta. De forma a melhorar a organização do código, criou-se uma classe á parte, na qual estão definidos todos os métodos necessários ao funcionamento do algoritmo. Por sua vez, esta classe é chamada no "GestRobotFacade", de modo a executar o algoritmo e a calcular todos os caminhos mais curtos do vértice escolhido aos restantes vértices do Grafo, sendo que depois é escolhido o caminho cujo destino é aquele que pretendemos.

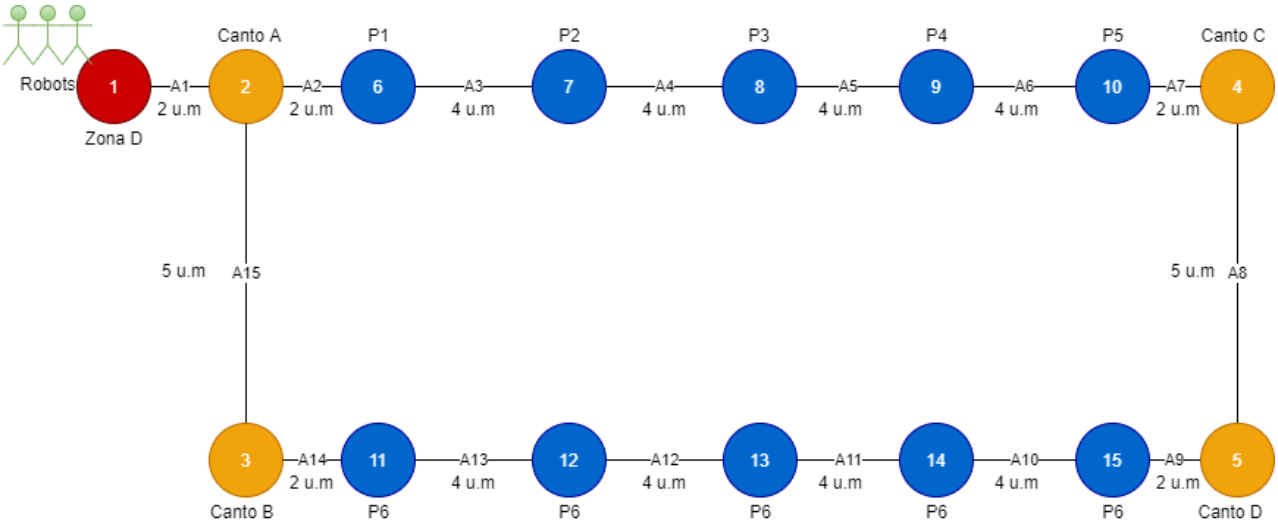


Figura 21 - Grafo/Mapa do Armazém

Dentro desta classe foram definidas as seguintes estruturas de dados necessárias para a aplicação do Algoritmo de Dijkstra:

List <Vertice> vértices -> Contem todos os vértices do grafo que representa o armazém.

List <Aresta> corredores -> Contem todos as arestas do grafo que representa o armazém.

Set <Vertice> visitados -> Contem todos os vértices do grafo que já foram "visitados" durante a execução do algoritmo.

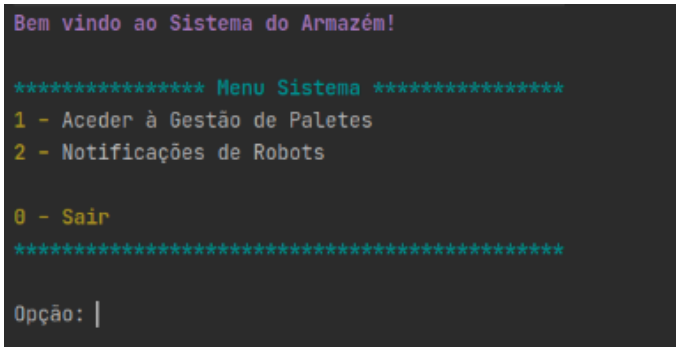
Set <Vertice> naoVisitados -> Contem todos os vértices do grafo que ainda não "visitados" durante a execução do algoritmo.

Map<Vertice, Vertice> precedentes -> Contem todas as relações de precedência, ou seja, todos os caminhos mais curtos a partir de um vértice origem.

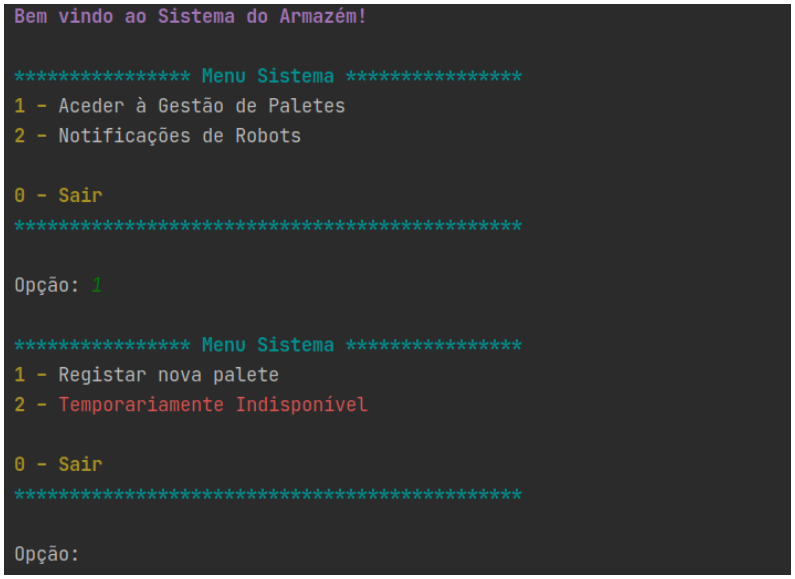
Map<Vertice, Float> distancia -> Contem as distâncias mínimas entre um dado vértice que se está a visitar e o vértice origem a partir do qual queremos calcular o percurso mais curto.

Manual de Utilização

Primeiramente o utilizador depara-se com a seguinte janela do menu principal da aplicação que, tal como vimos, tanto poderá desencadear no Submenu relacionado com a Gestão de Paletes como poderá resultar no Submenu das notificações dos Robots, dependendo da opção que ele escolha.



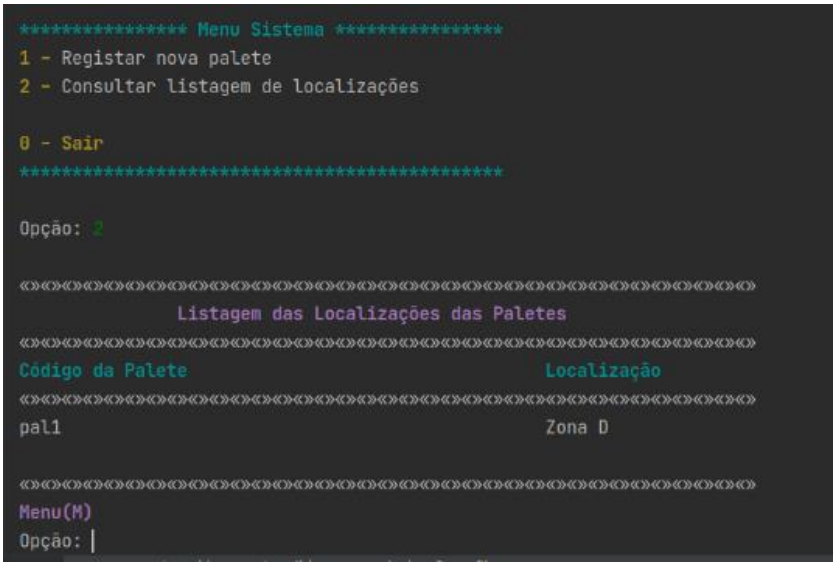
Caso o utilizador escolha a opção 1, irá ser apresentado o Submenu da Gestão das Paletes. Como ainda não existe qualquer tipo de registo de paletes no Armazém, isto é, o armazém encontra-se vazio, a opção 2 que corresponde à listagem das paletes existentes no armazém e a sua localização não está disponível. Vamos considerar que hipoteticamente o utilizador escolhe a Opção 1: Registar nova palete.



O utilizador terá de preencher as informações que forem solicitadas pelo programa. Como estamos a considerar que a comunicação com o leitor e os robots é simulada na interface. O utilizador terá de preencher a informação do CódigoQR e ainda de atribuir um código à palete, caso ele escreva um código que já exista será lançada uma exceção, e no caso do CódigoQR não existir na base de dados do sistema, o utilizador será informado de que a palete não poderá ser registada no armazém.

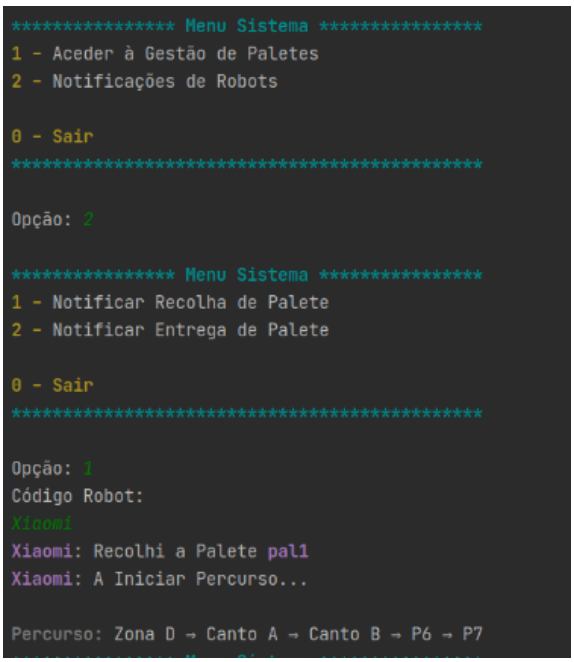
```
*************** Menu Sistema ***************
1 - Registar nova palete
2 - Temporariamente Indisponivel

0 - Sair
********************************************

Opção: 1
Código QR lido:
1
Inserir Código da Palete:
pal1
*Registo Concluído!!!
*Requisitando transporte...

O Robot Xiaomi foi notificado para transportar a palete pal1!! Em breve irá começar o seu percurso.
Robot já está no vertice pretendido.

*************** Menu Sistema ***************
1 - Registar nova palete
2 - Consultar listagem de localizações

0 - Sair
********************************************

Opção:
```

Assim que o registo é concluído, o sistema notifica o transporte da palete a um robot que esteja disponível. Caso o Robot já se encontre na Zona de Descarga, então não é necessário calcular um percurso, caso contrário então o sistema irá calcular o percurso mais curto que o Robot deverá tomar para recolher a palete.

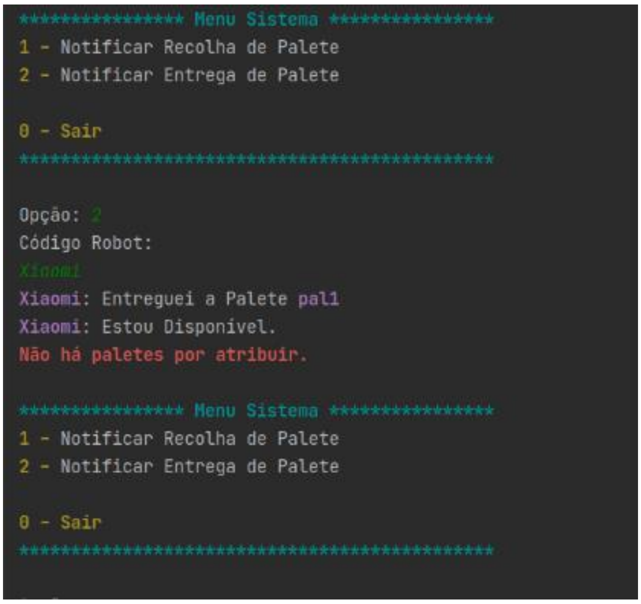


É apresentado de novo o submenu da Gestão das Paletes e podemos reparar que agora a opção "Consultar listagem de localizações" já está disponível. Se o utilizador escolher essa mesma opção já poderá observar que a palete já se encontra registada na base de dados do sistema, e qual a localização a mesma. Para voltar para o Submenu deverá escolher a opção *M*.

Caso o utilizador não queira executar qualquer funcionalidade relacionada com a gestão de paletes, ele poderá escolher a opção 0, voltando assim para o Menu Principal. Se agora escolher a opção 2, irá ser encaminhado ao submenu das notificações dos robots, onde poderá consultar quais as notificações recebidas pelo sistema dos Robots. (Relembramos se trata apenas de uma simulação desta situação na interface implementada em modo texto.)



Se o utilizador escolher a opção 1, terá de indicar o código do robot que irá notificar a recolha da palete. Assim que isto acontece, o sistema calcula um novo percurso para o Robot, escolhendo uma prateleira que esteja disponível para armazenar a palete.

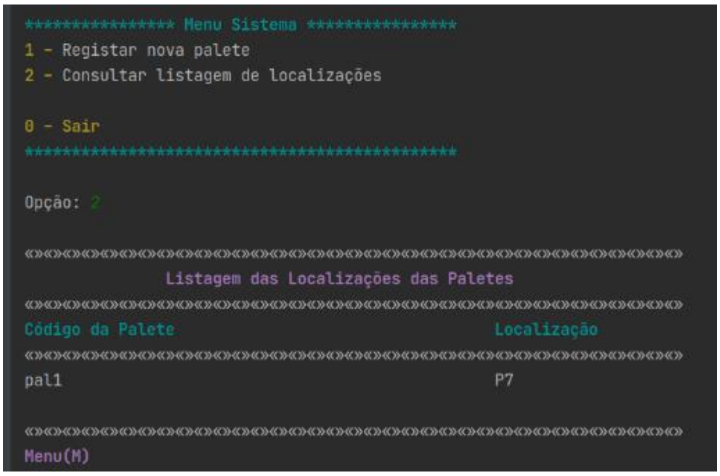


```
**************** Menu Sistema ****************
1 - Notificar Recolha de Palete
2 - Notificar Entrega de Palete

0 - Sair
**********************************************

Opção: 2
Código Robot:
Xiaomi
Xiaomi: Entreguei a Palete pal1
Xiaomi: Estou Disponível.
Não há paletes por atribuir.

**************** Menu Sistema ****************
1 - Notificar Recolha de Palete
2 - Notificar Entrega de Palete

0 - Sair
**********************************************
```

Após simular a notificação de recolha por parte do Robot, o utilizador poderá fazer o mesmo para o caso de o Robot notificar o sistema que já entregou a palete ao seu destino.

Assim que esta notificação é recebida, o Robot passa a estar “livre” e, portanto, caso haja paletes por armazenar na Zona de Descarga, este é encaminhando até lá para fazer o transporte destas, caso contrário o Sistema não atribui mais nenhuma tarefa ao robot.

Se agora o utilizador consultar a listagem de localizações das paletes do Armazém, ele poderá observar que a palete já se encontra na prateleira devida.



```
**************** Menu Sistema ****************
1 - Registar nova palete
2 - Consultar listagem de localizações

0 - Sair
**********************************************

Opção: 2

«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»
               Listagem das Localizações das Paletes
«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»
Código da Palete                                  Localização
«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»
pal1                                              P7

«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»«»
Menu(M)
```

Se agora consideramos que foram registadas várias paletes e que não existem robots disponíveis para transportar estas novas paletes. Então o utilizador irá receber o seguinte output:

```
*************** Menu Sistema ***************
1 - Registar nova palete
2 - Consultar listagem de localizações

0 - Sair
********************************************

Opção: 1
Código QR lido:
4
Inserir Código da Palete:
pal5
*Registo Concluído!!!
*Requisitando transporte...
Não existem robots disponiveis para transportar paletes.
```

Sendo assim, é necessário que o Robot acabe a tarefa que está a realizar no momento. Assim que ele acabar acontecerá a seguinte situação:

```
*************** Menu Sistema ***************
1 - Notificar Recolha de Palete
2 - Notificar Entrega de Palete

0 - Sair
********************************************

Opção: 2
Código Robot:
Xiaomi
Xiaomi: Entreguei a Palete pal4
Xiaomi: Estou Disponivel.
*Ainda existem paletes na Zona de Descarga!
Xiaomi: Nova Tarefa.

O Robot Xiaomi foi notificado para transportar a palete pal5!! Em breve irá começar o seu percurso.
Xiaomi: A Iniciar Percurso...

Percurso: P10 → P9 → P8 → P7 → P6 → Canto B → Canto A → Zona D
```

Tal como vimos, o robot assim que deixa a palete no seu destino, é notificado novamente pelo Sistema caso ainda existam paletes na Zona de Descarga por atribuir. É, portanto, gerada uma nova tarefa para o Robot, onde o Sistema calcula novamente o caminho que o Robot deverá tomar para ir buscar a palete.

Reflexão Final

Implementação do Use Case "Comunica Ordem de Transporte"

Para chegar à implementação final de cada use case foi necessário garantir diversos requisitos para que o resultado final correspondesse ao idealizado inicialmente pelo grupo. Assim, a implementação do use case "comunica ordem de transporte" foi feita sequencialmente seguindo os seguintes passos:

1. Como foi definido inicialmente pelo grupo, o nosso sistema é responsável por escolher um robot do armazém para iniciar uma nova tarefa (considerando que o armazém tem vários Robots). Por outro lado, seria necessário garantir a pré-condição do use case (Robot está disponível) por isso foi utilizado o método *getRobotDisponivel* que retorna o código de um Robot que esteja disponível. É enviada uma exceção no caso de não haver nenhum disponível.
2. Para além de um Robot, o sistema é também responsável por escolher a prateleira na qual a palete será armazenada. O método *getPrateleiraLivre* garante que é selecionada uma prateleira que esteja vazia e disponível para receber uma nova palete (ocupação = 0). É enviada uma exceção no caso de não haver nenhuma disponível. De seguida será imediatamente atualizada a ocupação desta prateleira/Vertice para -1 de modo a garantir que durante o transporte, nenhuma outra palete possa ser guardada nesta mesma prateleira.
3. Tal como no caso do vértice, também foi necessário garantir que mais nenhum Robot fosse transportar a palete em questão, logo foi atualizado o seu campo *inRobot* para 2 (método *atualiza_localizacaoPalete*), que sinaliza que já foi notificado o transporte da mesma.
4. De seguida foi atribuída uma nova classe Rota ao Robot (*notifica_transporte* e *indica_destino*) com os campos necessários (origem, destino e código da palete) para ser guardada na base de dados.
5. Para terminar foi implementado o método *calcula_rota* que vai indicar ao Robot quais as Arestas do armazém a percorrer para efetuar o caminho mais curto até ao Vertice no qual a palete a recolher se localiza. Se o Robot já se localiza no Vertice pretendido é lançada uma exceção.

Análise Crítica

Ao longo de todo este projeto, o grupo tentou sempre utilizar as metodologias de programação mais corretas e eficientes. Primeiro, rejeitou-se por completo a metodologia "code-and-fix". Visto que o projeto teria de ser desenvolvido em fases e de forma incremental, seria muito mais vantajoso planear uma solução inicial, que seria constantemente modificada e melhorada, do que começar imediatamente a gerar código. O grupo verificou que ao longo do tempo várias ideias que tinham sido tomadas em conta, acabavam por ser alteradas ou até mesmo descartadas. Se tivéssemos, já desenvolvido código e se este já fosse relativamente complexo, o processo de retificação seria bastante trabalhoso. Assim, o uso dos vários diagramas que foram sendo criados revelaram-se uma forma muito melhor de organizar o raciocínio, sendo que tanto ajudaram a pensar no sistema como a documentá-lo. No entanto, visto que é necessário estar constantemente a atualizar o diagramas, esta metodologia também se torna trabalhosa.

A título de exemplo, o grupo sentiu uma grande diferença entre o desenvolvimento do projeto da cadeira de POO do ano passado e o desenvolvimento do projeto da cadeira de DSS, uma vez que antes destas noções e novas metodologias de trabalho, muitas vezes tornava-se confuso a explicação do raciocínio e a própria organização do projeto a desenvolver. Desta forma, foi possível conciliar todas as ideias dos membros do grupo, discutir requisitos e no final, implementar tudo isto sem um grande nível de complexidade ou dependências, já que estava tudo planeado previamente.

Durante a fase 3, na qual era suposto desenvolver em java os use cases selecionados pela equipa docente, houve vários pontos nos quais o grupo prestou mais atenção. Um deles foi a tentativa de reutilização de código, ou seja, evitar a definição de métodos que façam praticamente a mesma coisa. Outro ponto a ter em conta foi o de que o programa deveria ser versátil e flexível. Este deveria ser capaz de evoluir e de se adaptar, independentemente do armazém ou no número de robots. Além disto, o grupo teve o cuidado de tentar não sobrecarregar os facades com métodos, distribuindo as suas responsabilidades pelas suas classes, de forma a realizar os métodos de forma sequencial e evitar concentrar a lógica de processamento numa única classe pois desta forma obtemos um algoritmo mais complexo e criamos mais dependências. Mesmo assim, o grupo tem consciência que poderia ter sido feita uma distribuição dos métodos ainda mais eficiente.

Nesta última fase, um dos desejos do grupo era desenvolver uma interface gráfica de forma a tornar o software mais apelativo. Visto que isto não era uma prioridade no projeto, acabou por não se realizar.

Conclusão

O trabalho prático da cadeira de DSS consistiu em desenvolver um software de gestão de um armazém, no qual trabalham robots autônomos e se guardam diversas paletes. No entanto, o projeto teve se ser desenvolvido em 3 fases, sendo que apenas a última implicou fazer código. Esta metodologia nunca tinha sido usada pelo grupo, mas mostrou-se bastante útil, na medida em que a definição dos use cases, dos subsistemas, dos diagramas de classes e outros, permite desenvolver progressivamente uma solução, sendo esta modificada sempre que necessário. Isto tem uma vantagem enorme, visto que se se implementasse código sem nenhuma análise prévia e caso este fosse complexo, se fossem encontradas incoerências, corrigi-las seria extremamente trabalhoso. Assim, apenas implementamos uma solução sólida que temos a certeza que satisfaz os requisitos.

Deste modo, ao longo deste projeto o grupo considera que fez um bom trabalho, uma vez que teve o cuidado de constantemente questionar se o projeto era coerente e realista, além de ter em conta que o software deveria ser versátil e por isso funcionar independentemente da topologia do armazém ou do número de robots. Para isto, sempre que se iniciava uma nova fase, foi fundamental reavaliar as anteriores.

Concluindo, este projeto foi uma consolidação dos conceitos abordados nas aulas de DSS, que serão de certeza utilizados em projetos futuros.