Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática
Desenvolvimento de Sistemas de Software

Relatório 3ª Fase

Grupo 15 - 2020/21

Carlos Filipe Coelho Ferreira, a89542
João Filipe Oliveira Gomes, a89489
José Miguel Ferreira Gomes, a89485
Mário Rui Mirra Arcipreste, a89507
Pedro Miguel Martins Guimarães, a89569

Índice

Índice de Figuras

1. Introdução

Nesta terceira e última fase do projeto de Desenvolvimento de Sistemas de Software, o objetivo principal foca-se na implementação em linguagem de programação de certos requisitos e funcionalidades identificados nas fases anteriores do mesmo. Para isso, foi introduzido o conceito de base de dados no contexto da Programação orientada aos objetos, elemento-chave que, forçando assim uma mudança de paradigma, ditou não só a necessidade de alteração de vários diagramas elaborados na segunda fase do projeto, como também a elaboração de novos diagramas de sequência.

Em termos de implementação, é importante referir que, acompanhando as respetivas reduções verificadas em cada etapa do projeto, a implementação concebida não corresponde à totalidade do funcionamento do sistema, uma vez que, para efeitos desta terceira fase, foi apenas tido em conta um grupo restrito de use cases, que a seguir se enumeram:

- Comunicar código QR (Ator: Leitor de códigos QR);
- Sistema comunica ordem de transporte (Ator: Robot / Iniciativa: Sistema);
- Notificar recolha de paletes (Ator: Robot);
- Notificar entrega de paletes (Ator: Robot);
- Consultar listagem de localizações (Ator: Gestor).

De acordo com os elementos acima descritos, nos capítulos seguintes deste relatório será explicitado todo o processo de desenvolvimento destas funcionalidades, realçando as alterações face às etapas anteriores do projeto e concretizando as escolhas de implementação adotadas.

2. Objetivos e abordagem

Tal como referido na secção anterior, os objetivos desta fase do projeto passam pela implementação em linguagem de programação de um conjunto restrito de funcionalidades relacionadas com o funcionamento de um sistema de gestão de stocks de um armazém, recorrendo a uma base de dados capaz de dar resposta a este requisito. Nesse sentido, é essencial não só a revisão dos diagramas elaborados na fase anterior, em particular aqueles que têm uma ligação direta com as funcionalidades a implementar, como também uma abordagem mais específica para os objetivos que se pretendem alcançar.

Assim sendo, comparativamente à fase anterior, verificam-se alterações ao nível do diagrama de classes, destacando-se a introdução de DAO's (*Data Access Objects*) e a maior especificidade de acordo com os use cases a implementar com a adição de novos métodos, o que espelha, por um lado, o reaproveitamento, em certa medida, dos diagramas de sequência já existentes, e por outro, a adição de novos métodos necessários para satisfazer os requisitos.

Tendo isso em consideração, segue-se uma breve exposição das alterações efetuadas comparativamente à segunda fase do projeto, bem como as devidas adições.

3. Diagramas

Diagrama de Classes

No sentido de desenvolver o sistema com a implementação da vertente relacionada com um sistema de base de dados, o diagrama de classes proposto na fase anterior sofreu alterações, de onde se destaca a introdução de classes DAO em substituição dos Map's de armazenamento de todas as paletes, prateleiras, robots e utilizadores, de forma a garantir a segurança e conservação dos dados do sistema após cada encerramento do programa. Uma vantagem da vertente introduzida traduz-se na eliminação da necessidade de trazer para memória uma enorme quantidade de dados por cada vez que o sistema é iniciado. Contudo, considera-se pertinente a utilização deste processo no que diz respeito a mapeamentos de tamanho mais reduzido, e em certa medida imutáveis, como é o caso do Map que guarda todos os vértices que compõem o grafo representativo do mapa do armazém, continuando este a ser carregado em memória.

De igual maneira, foi considerada pertinente a introdução de um conjunto de classes localizadas acima da lógica de negócio, nomeadamente a classe Controller, com o intuito de abrigar o trabalho pesado de cada funcionalidade a implementar através de pedidos à interface da Facade em utilização, e classes relativas à interação entre utilizador e sistema através de interfaces do utilizador.

Finalmente, foram removidas algumas classes consideradas obsoletas no contexto das funcionalidades a implementar para a fase 3 do projeto, como se verifica relativamente à classe Zona, ou com o intuito de facilitar o armazenamento de algumas informações na base de dados, como a classe Localizacao, substituída por uma variável privada do tipo String denominada Vértice em cada entidade que a utilizava, permitindo assim uma estruturação mais objetiva do problema.

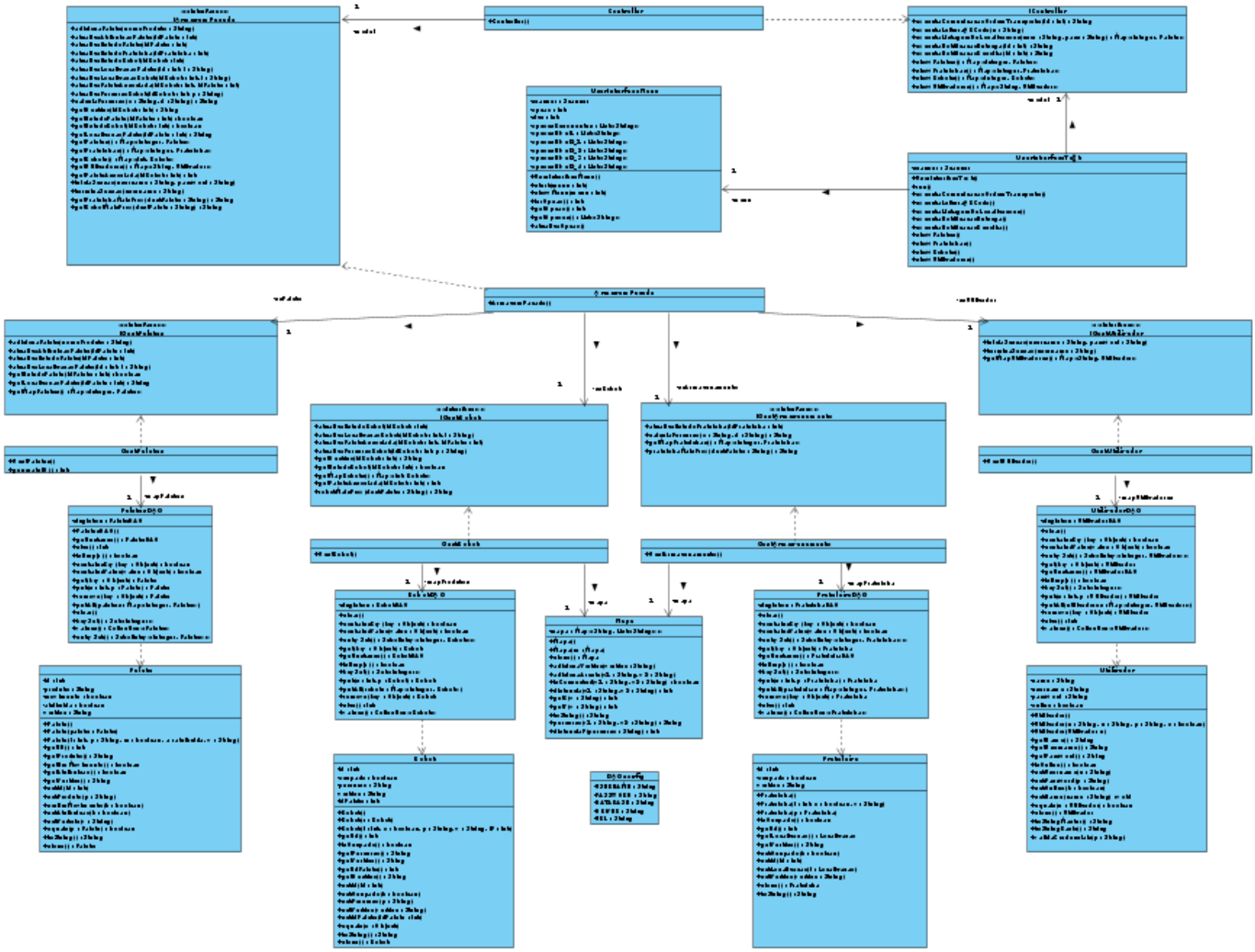

Figura 1 - Diagrama de Classes

Diagramas de Sequência

Assim como o diagrama de classes, também os diagramas de sequência criados na fase anterior do projeto passaram por um processo de atualização de forma a corresponderem às mudanças de estratégia adotadas neste ponto. Deste processo, é possível destacar, a título de exemplo, a queda da necessidade de distinguir entre produtos perecíveis e não perecíveis, e a mudança em termos do tratamento da localização tanto de robots, como de prateleiras.

Assim sendo, entre diagramas de sequência reestruturados e diagramas de sequência recém-criados, observa-se, associados a 5 use cases distintos, um total de 27 diagramas cuja ação é brevemente descrita em seguida.

adicionaPalete_DS: O método adicionaPalete() serve o propósito de inserir no sistema uma nova palete em função de um valor do tipo String, correspondente ao nome do produto. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde gera um identificador para a palete e a adiciona à base de dados em questão

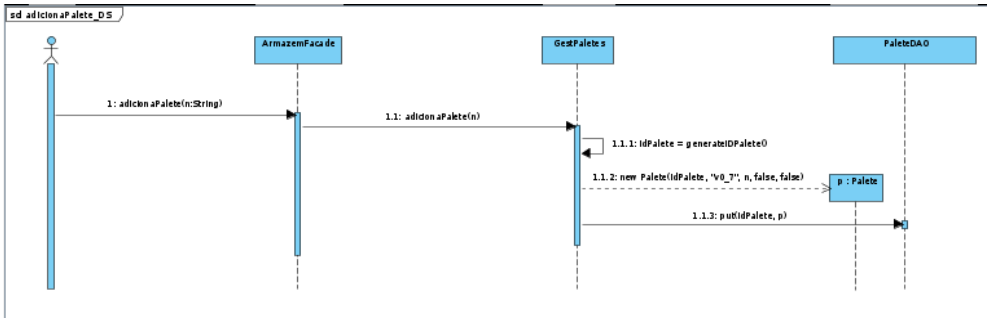

Figura 2 - Diagrama de Sequência adicionaPalete

atualizaAtribuicaoPalete_DS: O método atualizaAtribuicaoPalete() serve o propósito de atualizar o estado de atribuição a um robot de uma palete no sistema em função de um valor do tipo int, correspondente ao identificador da palete. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde verifica a existência da palete e, em caso verdadeiro, altera o seu estado de atribuição para o seu oposto.

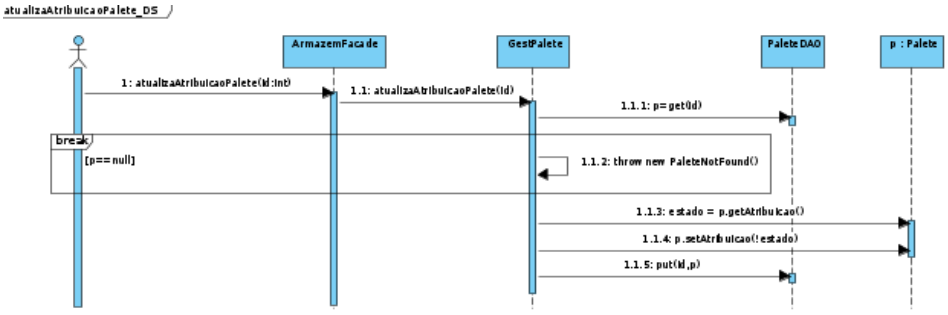

Figura 3 - Diagrama de Sequência atualiazAtribuicaoPalete

atualizaEstadoPalete_DS: O método atualizaEstadoPalete() serve o propósito de atualizar o estado de movimentação de uma palete no sistema em função de um valor do tipo int, correspondente ao identificador da palete. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde verifica a existência da palete e, em caso verdadeiro, altera o seu estado de movimentação para o seu oposto.

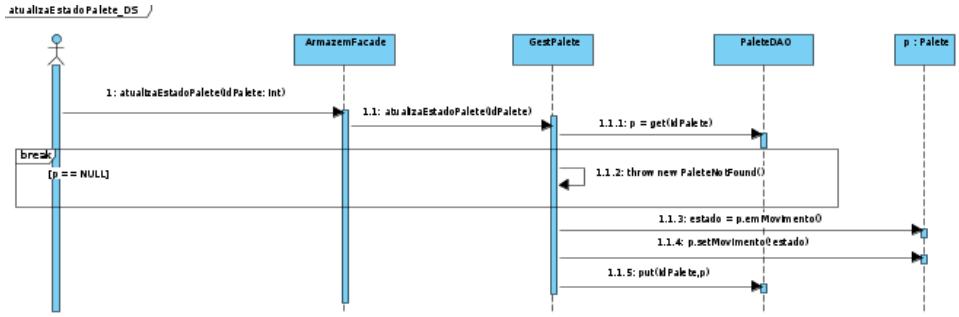

Figura 4 - Diagrama de Sequência atualizaEstadoPalete

atualizaEstadoPrateleira_DS: O método atualizaEstadoPrateleira() serve o propósito de atualizar o estado de ocupação de uma prateleira no sistema em função de um valor do tipo int, correspondente ao identificador da prateleira. A partir do Facade, o método dirige-se ao subsistema GestArmazenamento, onde verifica a existência da prateleira e, em caso verdadeiro, altera o seu estado de ocupação para o seu oposto.

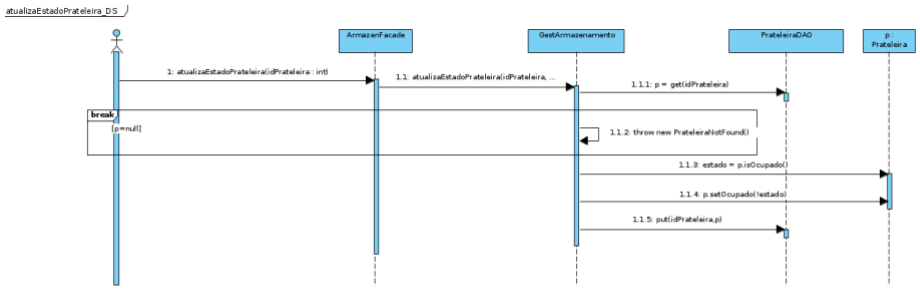

Figura 5 - Diagrama de Sequência atualizaEstadoPrateleira

atualizaEstadoRobot_DS: O método atualizaEstadoRobot() serve o propósito de atualizar o estado de ocupação de um robot no sistema em função de um valor do tipo int, correspondente ao identificador do robot. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, altera o seu estado de ocupação para o seu oposto.

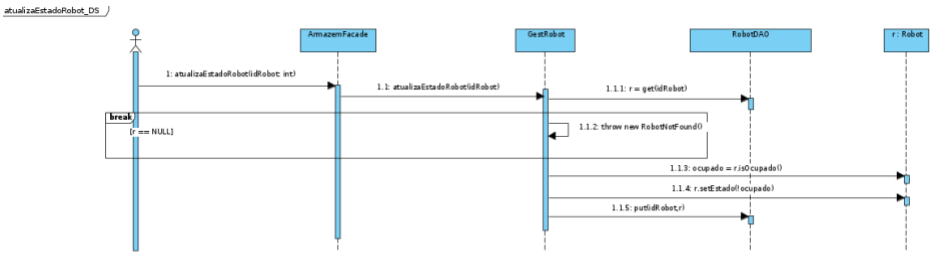

Figura 6 - Diagrama de Sequência atualizaEstadoRobot

atualizaLocalizacaoPalete_DS: O método atualizaLocalizacaoPalete() serve o propósito de atualizar a localização de uma palete no sistema em função de um valor do tipo int, correspondente ao identificador da palete e de um outro valor string, representativo do vértice em que a palete se encontra. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde verifica a existência da palete e, em caso verdadeiro, altera a sua localização.

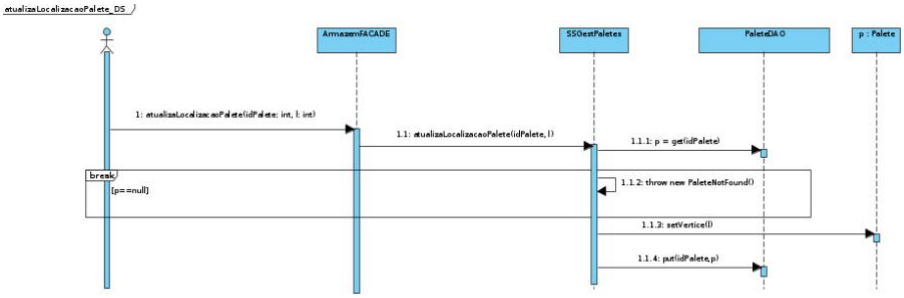

Figura 7 - Diagrama de Sequência atualizaLocalizacaoPalete

atualizaLocalizacaoRobot_DS: O método atualizaLocalizacaoRobot() serve o propósito de atualizar a localização de um robot no sistema em função de um valor do tipo int, correspondente ao identificador da palete e de um outro valor string, representativo do vértice em que este se encontra. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, altera a sua localização

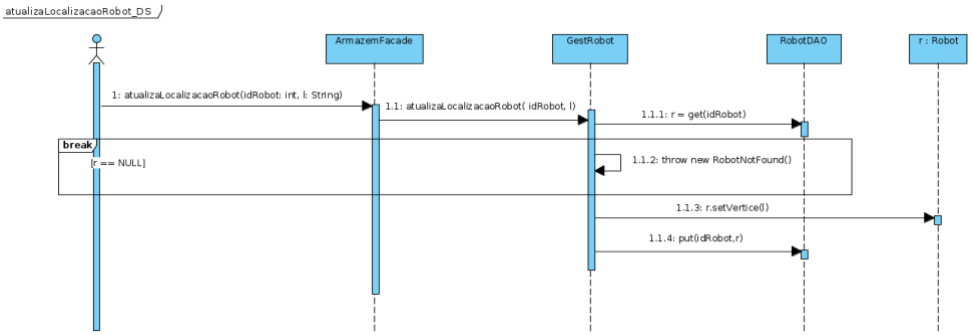

Figura 8 - Diagrama de Sequência atualizaLocalizacaoRobot

atualizaPaleteAssociada_DS: O método atualizaPaleteAssociada() serve o propósito de atualizar a palete associada a um robot do sistema em função de um valor do tipo int, correspondente ao identificador da palete e de um outro valor que representa o identificador da palete. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, altera o identificador da palete a si associada.

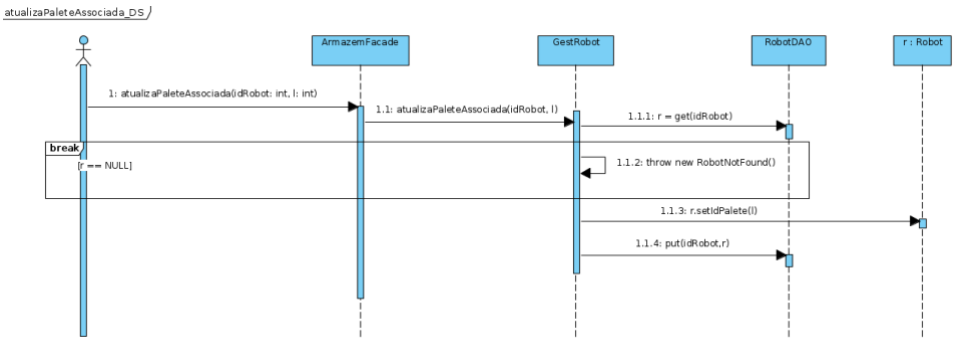

Figura 9 - Diagrama de Sequência atualizaPaleteAssociada

atualizaPercursoRobot_DS: O método atualizaPercursoRobot() serve o propósito de atualizar o percurso de um robot no sistema em função de um valor do tipo int, correspondente ao identificador da palete e de um outro valor string que representa o percurso. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, altera o seu percurso.

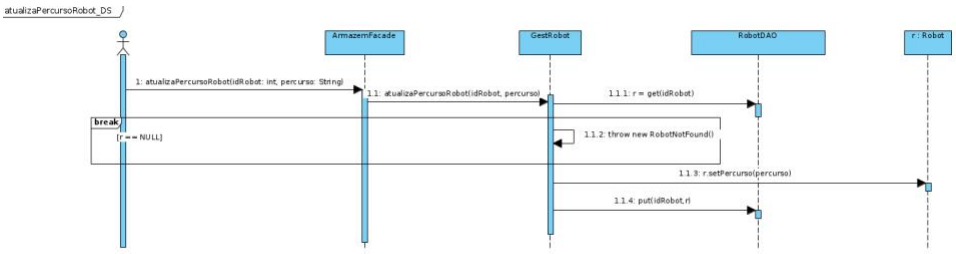

Figura 10 - Diagrama de Sequência atualizaPercursoRobot

calculaPercurso_DS: O método calculaPercurso() serve o propósito de calcular o percurso que um robot terá de percorrer, através de duas Strings, vértice inicial e final. A partir do Facade, o método dirige-se ao subsistema GestArmazenamento, onde aplica o método percurso(), explicado de seguida, que devolverá o percurso referido.

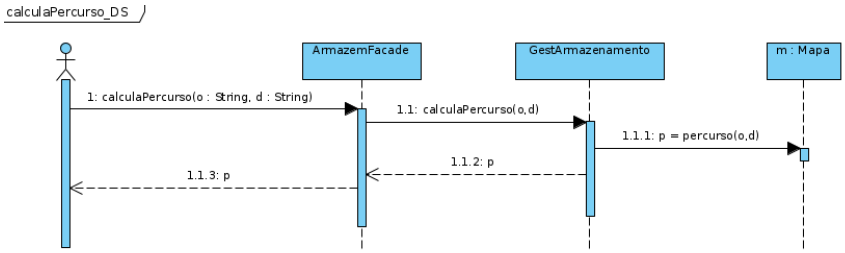

Figura 11 - Diagrama de Sequência calculaPercurso

getDestino_DS: O método getDestino() serve o propósito de obter o vértice de destino ao qual o robot chegará no final do percurso, em função de um valor int, correspondente ao identificador de um robot. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, retorna o seu vértice de destino.

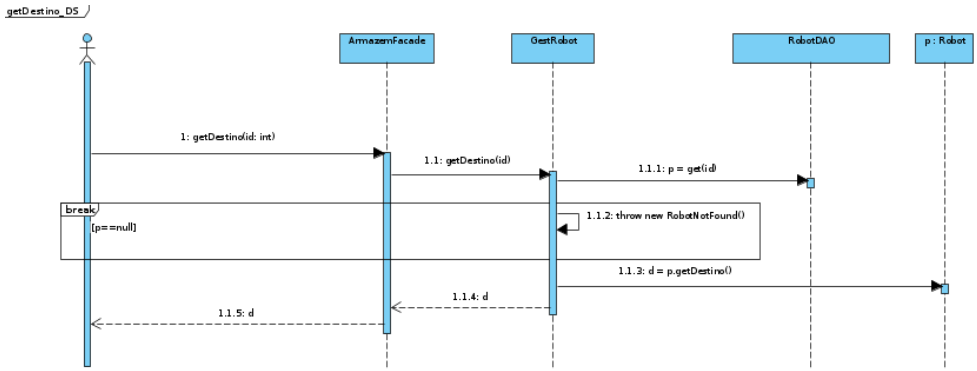

Figura 12 - Diagrama de Sequência getDestino

distanciaP_DS: O método distanciaP() serve o propósito de calcular a distância de um percurso, dado em forma de String. Este método, desenvolvido ao nível da classe Mapa, realiza um split, separando o percurso pelas vírgulas que separam os vértices, e calcula a soma das distâncias de Manhattan entre cada dois vértices que constituem o percurso.

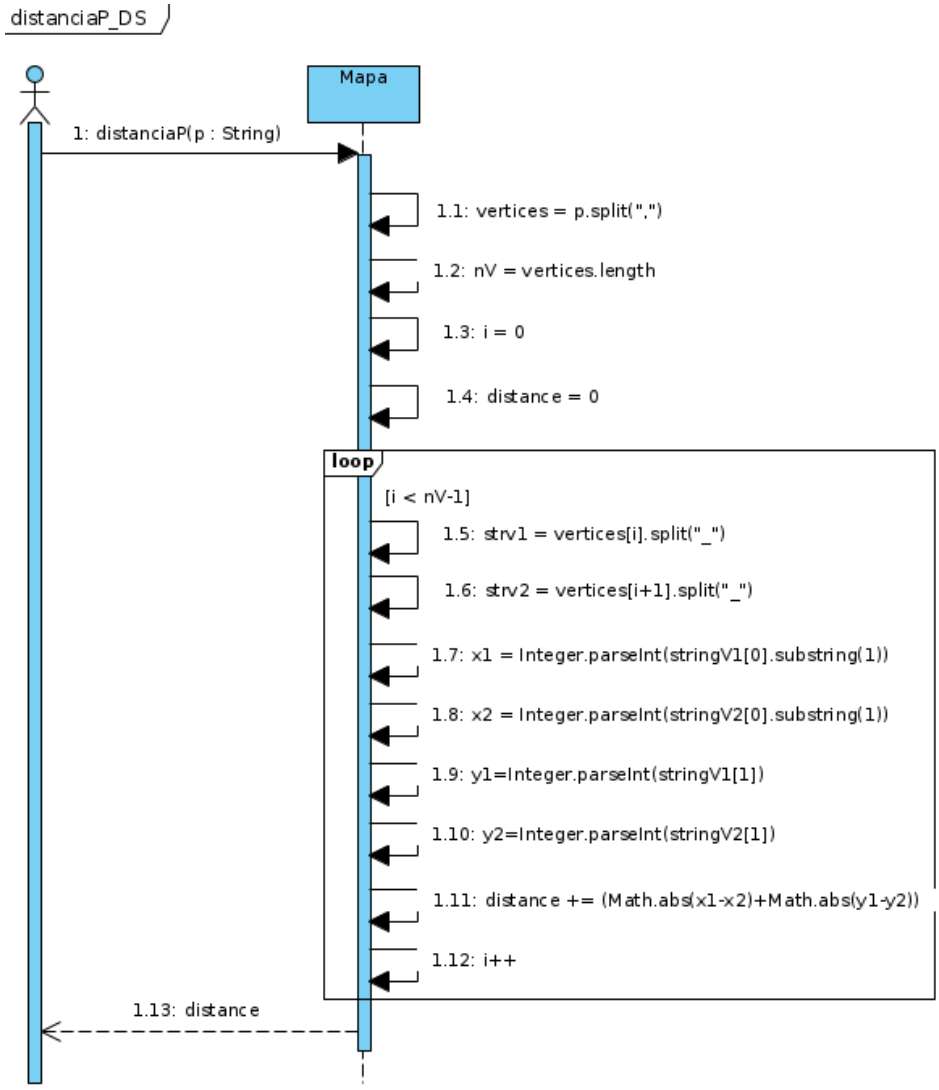

Figura 13 - Diagrama de Sequência distanciaP

getEstadoPalete_DS: O método getEstadoPalete() serve o propósito de obter o estado de uma palete, em função de um valor int, correspondente a um identificador desta. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde verifica a existência da palete e, em caso verdadeiro, retorna o seu estado.

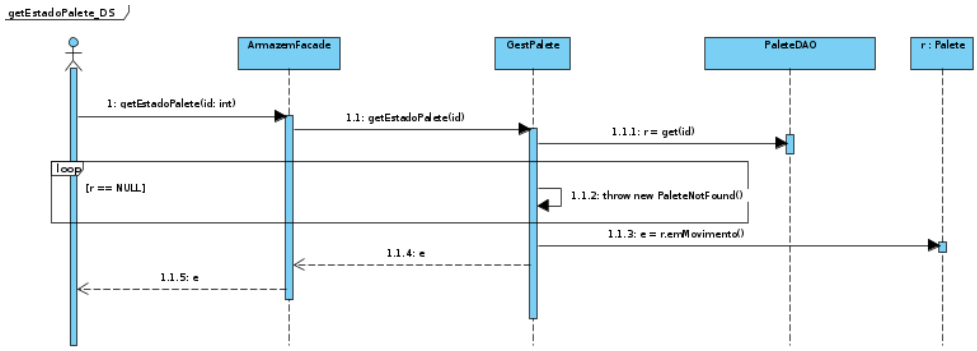

Figura 14 - Diagrama de Sequência getEstadoPalete

getEstadoRobot_DS: O método getEstadoRobot() serve o propósito de obter o estado de um robot, em função de um valor int, correspondente ao identificador de um robot. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, retorna o seu estado.

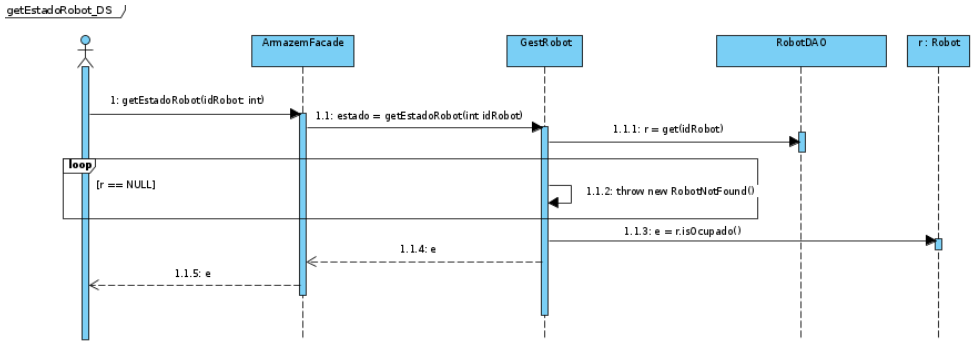

Figura 15 - Diagrama de Sequência getEstadoRobot

getLocalizacaoPalete_DS: O método getLocalizacaoPalete() serve o propósito de obter a localização de uma palete, em função de um valor int, correspondente ao identificador desta. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde extrai a palete pretendida e devolve a localização desta.

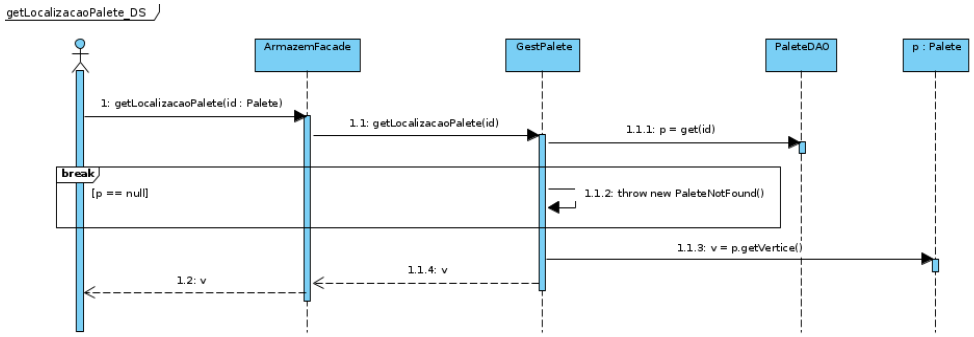

Figura 16 - Diagrama de Sequência getLocalizacaoPalete

getPaleteAssociada_DS: O método getPaleteAssociada() serve o propósito de obter a palete associada a um robot, em função de um valor int, correspondente a um identificador de um robot. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde verifica a existência do robot e, em caso verdadeiro, retorna a palete a si associada.

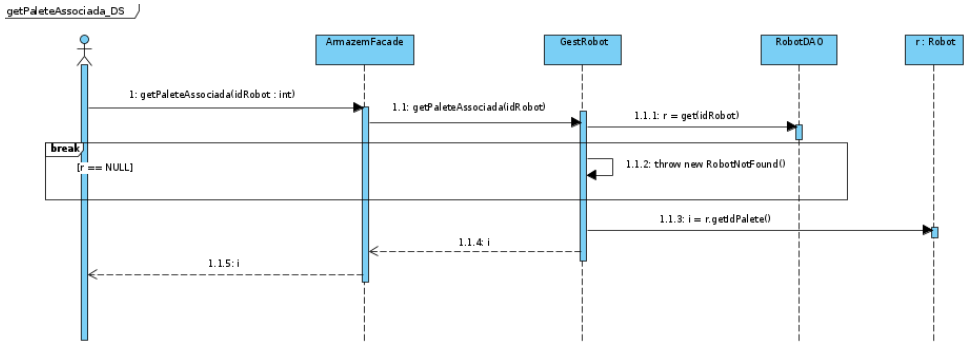

Figura 17 - Diagrama de Sequência getPaleteAssociada

getPaletes_DS: O método getPaletes_DS() serve o propósito de obter uma listagem com o conjunto das informações de todas os paletes existentes no sistema. A partir do Facade, o método dirige-se ao subsistema GestPalete, onde chama o método getMapPalete(), que conta com a verificação da existência de paletes no sistema e, em caso afirmativo, inicia o processo de cópia dos dados das paletes guardadas em RobotDAO, devolvendo-as num novo Map<Integer, Palete>.

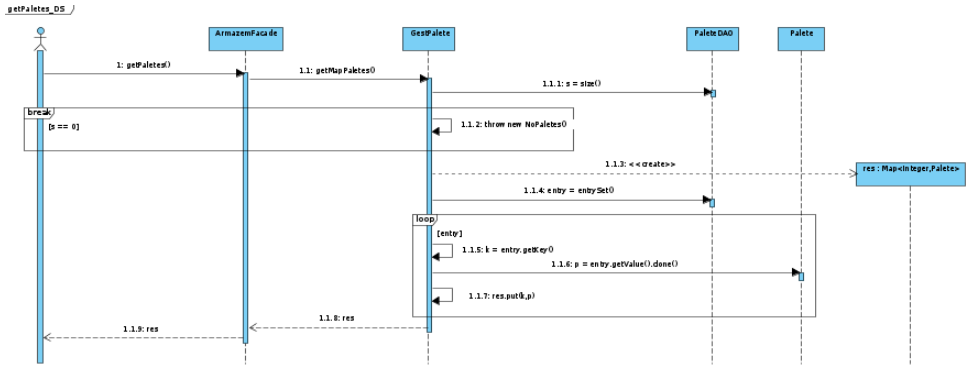

Figura 18 - Diagrama de Sequência getPaletes

getPrateleiraMaisProx_DS: O método getPrateleiraMaisProx_DS() serve o propósito de obter a prateleira mais próxima de uma certa localização, dada em forma de String. A partir do Facade, o método dirige-se ao subsistema GestArmazenamento, onde chama o método prateleiraMaisProx(), que devolve, então, a prateleira que se encontra mais perto de um dado vértice.

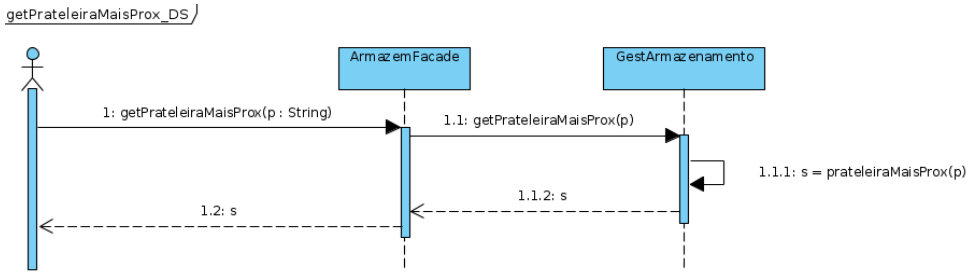

Figura 19 - Diagrama de Sequência getPrateleiraMaisProx

getRobotMaisProx_DS: O método getRobotMaisProx() serve o propósito de obter o robot mais próximo de uma certa localização, dada em forma de String. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde chama o método robotMaisProx(), que devolve, então, o robot que se encontra mais perto de um dado vértice.

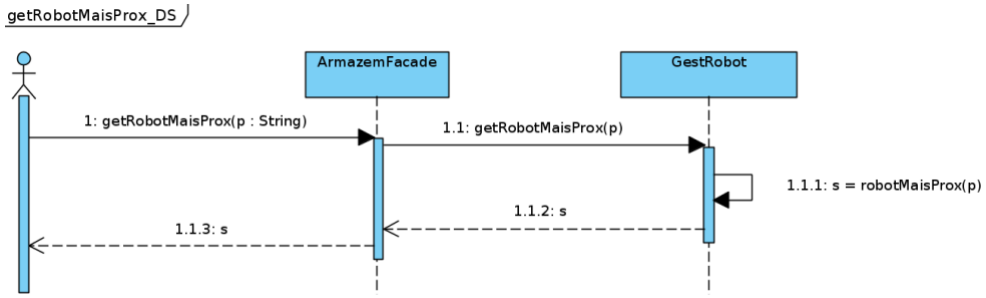

Figura 20 - Diagrama de Sequência getRobotMaisProx

getPrateleiras_DS: O método getPrateleiras_DS() serve o propósito de obter uma listagem com o conjunto das informações de todas as prateleiras existentes no sistema. A partir do Facade, o método dirige-se ao subsistema GestArmazenamento, onde chama o método getMapPrateleiras(), que conta com a verificação da existência de paletes no sistema e, em caso afirmativo, inicia o processo de cópia dos dados das prateleiras guardadas em PrateleiraDAO, devolvendo-as num novo Map<Integer, Prateleira>.

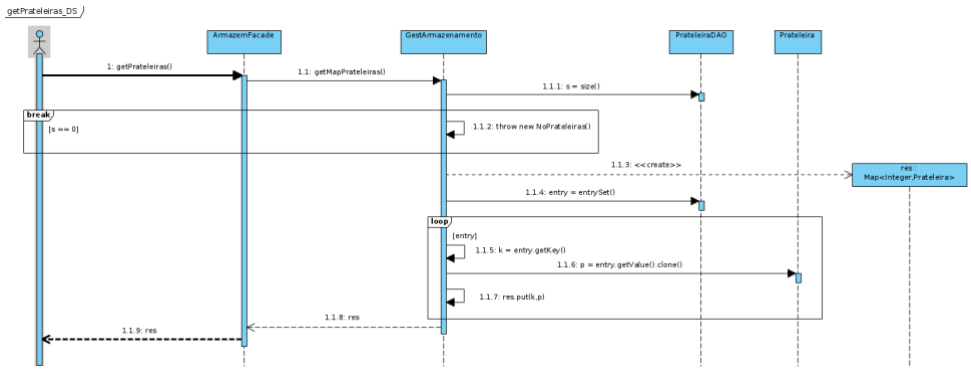

Figura 21 - Diagrama de Sequência getPrateleiras

getRobots_DS: O método getRobots_DS() serve o propósito de obter uma listagem com o conjunto das informações de todos os robots existentes no sistema. A partir do Facade, o método dirige-se ao subsistema GestRobot, onde chama o método getMapRobots(), que conta com a verificação da existência de robots no sistema e, em caso afirmativo, inicia o processo de cópia dos dados dos robots guardados em RobotDAO, devolvendo-os num novo Map<Integer, Robot>.

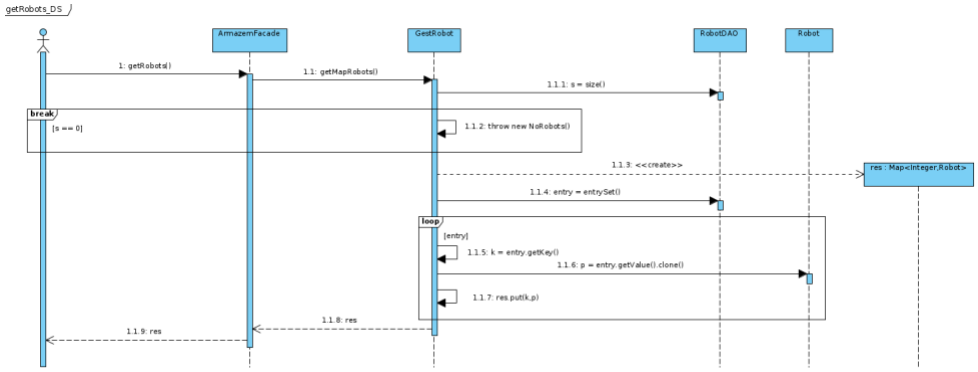

Figura 22 - Diagrama de Sequência getRobots

getUtilizadores_DS: O método getUtilizadores_DS() serve o propósito de obter uma listagem com o conjunto das informações de todos os utilizadores existentes no sistema. A partir do Facade, o método dirige-se ao subsistema GestUtilizadores, onde chama o método getMapUtilizadores(), que conta com a verificação da existência de utilizadores no sistema e, em caso afirmativo, inicia o processo de cópia dos dados dos utilizadores guardados em UtilizadorDAO, devolvendo-os num novo Map<Integer, Utilizador>.

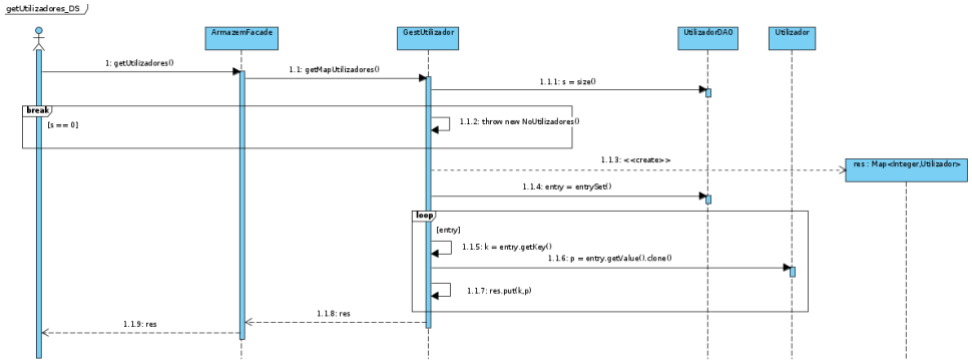

Figura 23 - Diagrama de Sequência getUtilizadores

iniciaSessão_DS: O método iniciaSessao() serve o propósito de efetuar a verificação dos dados de um utilizador. A partir do Facade, o método dirige-se ao subsistema GestUtilizador, onde verifica a existência do utilizador e, em caso verdadeiro, altera o seu estado para online.

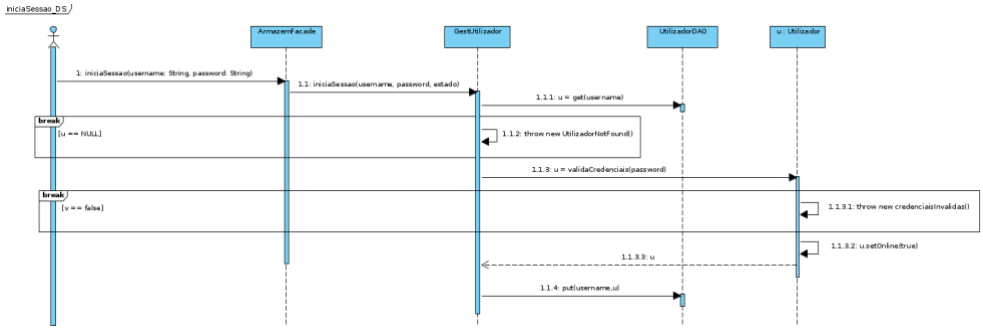

Figura 24 - Diagrama de Sequência iniciaSessão

prateleiraMaisProx_DS: O método prateleiraMaisProx_DS(String p) serve o propósito de determinar qual o destino de uma palete em função da sua posição atual: se a palete se encontra na zona de descargas (v0_7), considera-se que o seu destino é a prateleira mais próxima; se a palete se encontra numa prateleira, considera-se que o seu destino é a zona de entregas (v26_5). A partir de GestArmazenamento, ocorre a verificação de todas as prateleiras existentes no sistema de forma a verificar qual a prateleira que satisfaz o requisito que se pretende, através da utilização de variáveis atualizadas a cada prateleira que surja com uma localização melhor que a prateleira com localização mais próxima até ao momento. Este método devolve um valor do tipo String contendo o vértice de destino da palete e o identificador da palete, separados por uma vírgula.

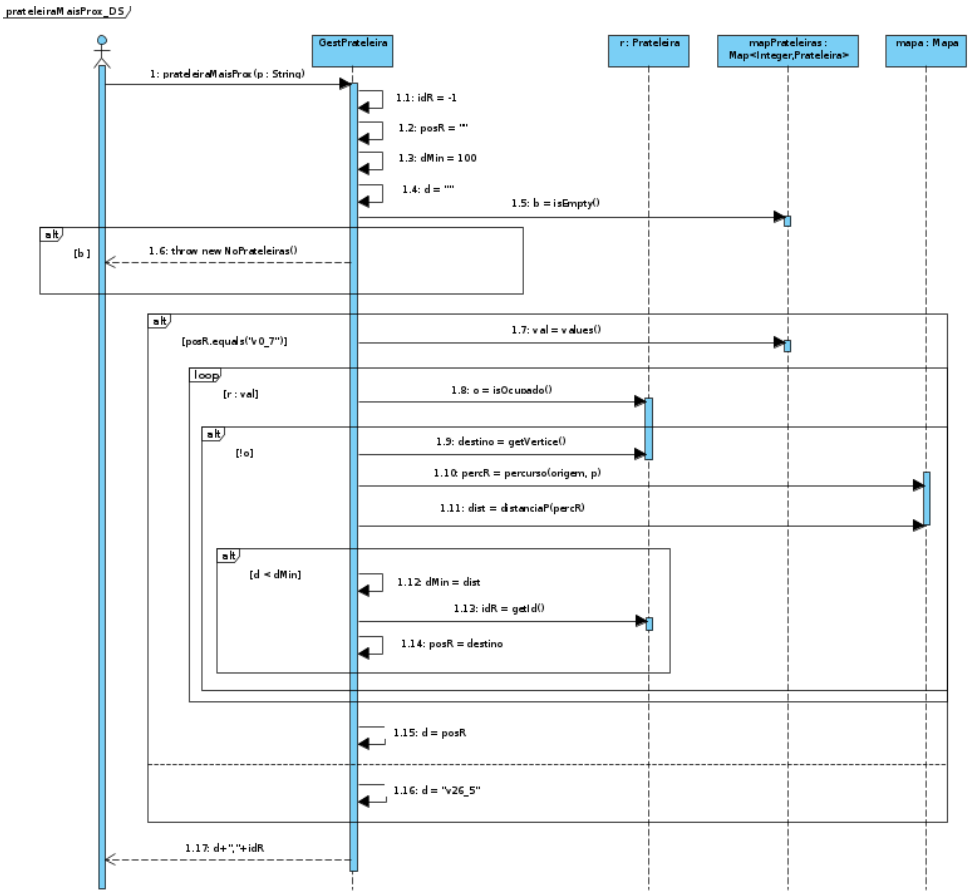

Figura 25 - Diagrama de Sequência prateleiraMaisProx

percurso_DS: O método percurso_DS(), localizado na classe Mapa, serve o propósito de obter o percurso mais curto entre dois pontos de um mapa no formato do tipo String, pretendendo simular o algoritmo de Dijkstra.

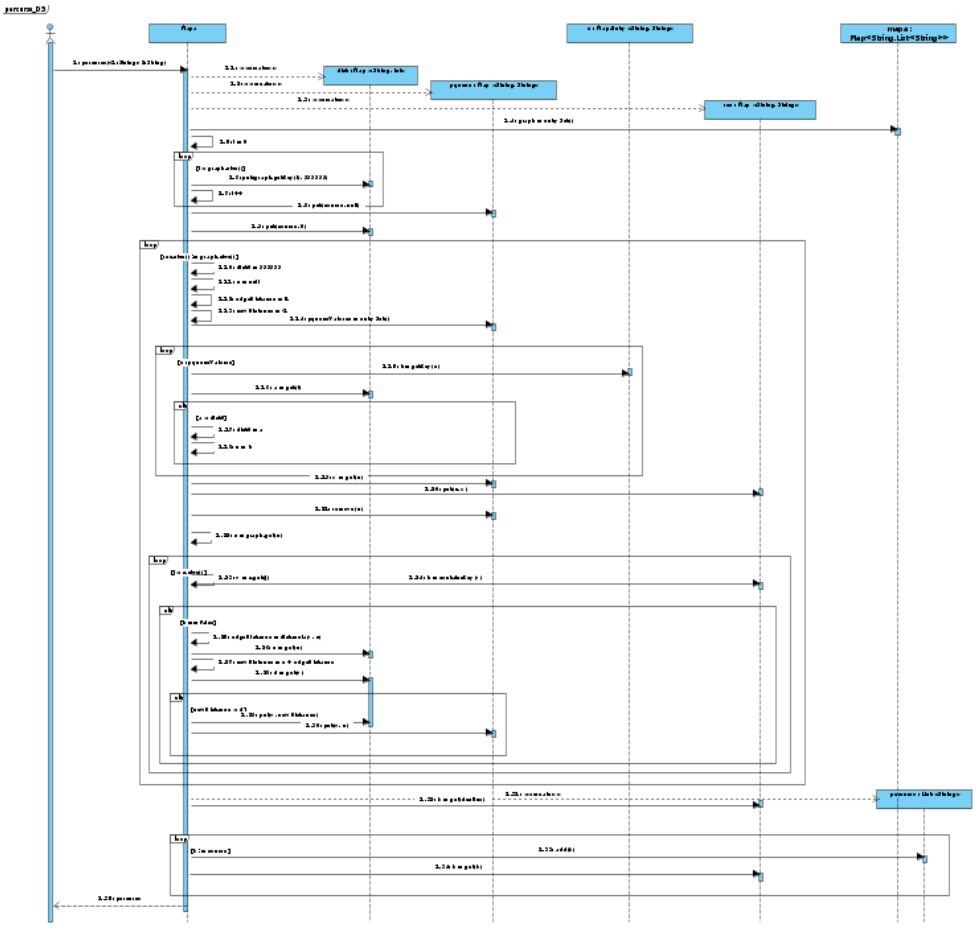

Figura 26 - Diagrama de Sequência percurso

terminaSesao_DS: O método terminaSessao() serve o propósito de efetuar o logout de um utilizador. A partir do Facade, o método dirige-se ao subsistema GestUtilizador, onde verifica a existência do utilizador e, em caso verdadeiro, altera o seu estado para offline.

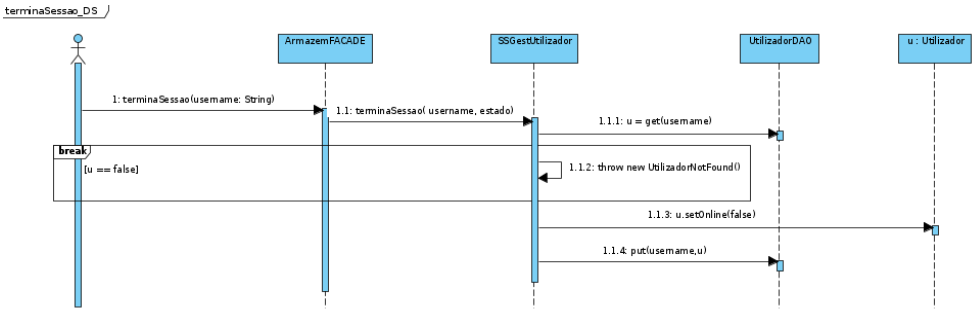

Figura 27 - Diagrama de Sequência terminaSessao

robotMaisProx_DS: O método robotMaisProx_DS(String p) serve o propósito de determinar qual o robot livre mais próximo em função da posição de uma palete. A partir de GestRobot, ocorre a verificação de todos os robots existentes no sistema de forma a verificar qual o robot que satisfaz o requisito que se pretende, através da utilização de variáveis atualizadas a cada robot que surja com uma localização melhor que o robot com localização mais próxima até ao momento. Este método devolve um valor do tipo String contendo a posição do robot e o identificador do robot, separados por uma vírgula.

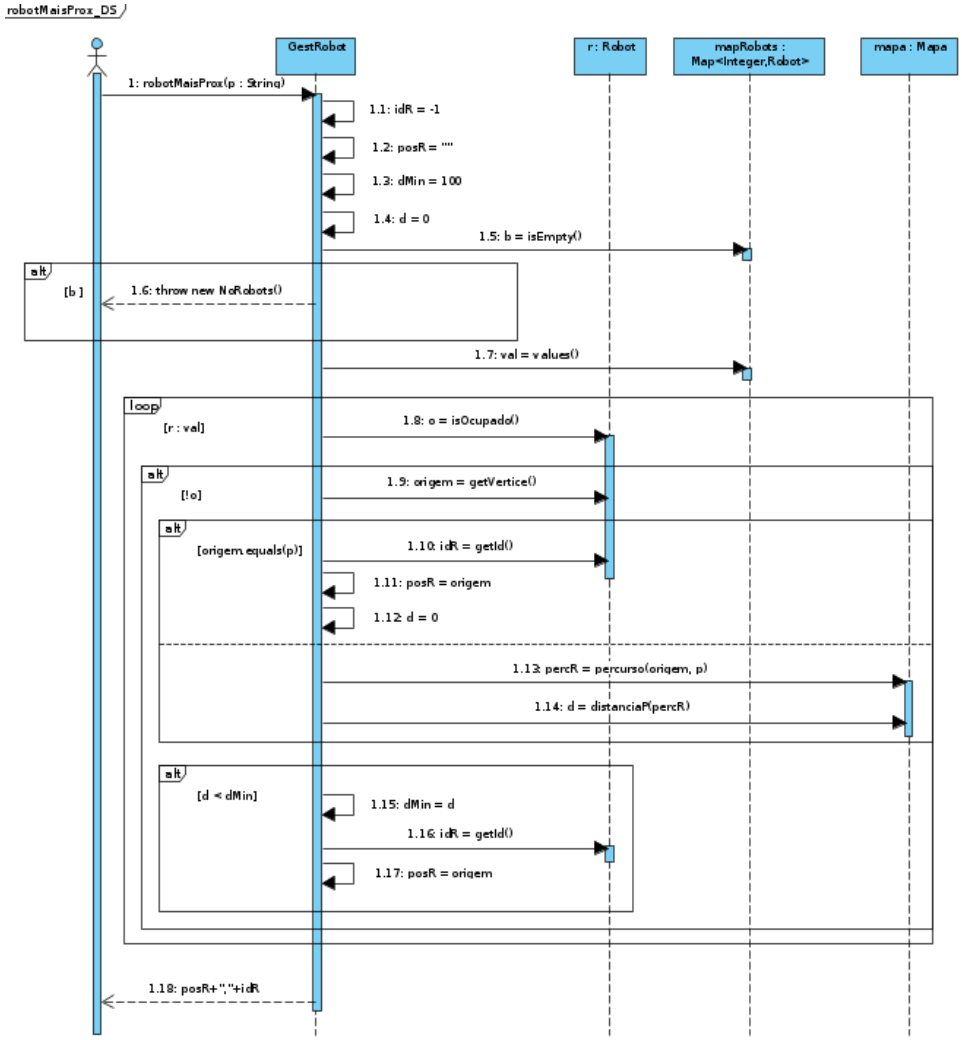

Figura 28 - Diagrama de Sequência robotMaisProx

Diagramas de Packages

Considerando o processo de reestruturação efetuado, torna-se necessário organizar novamente o diagrama de pacotes, de forma a abrigar todas as novas classes adicionadas. Comparativamente ao diagrama de pacotes anterior, é possível denotar que as principais alterações se prendem com as camadas de base de dados e interface do utilizador. Deste modo, o diagrama de pacotes resultante pode ser observado na figura que se segue.

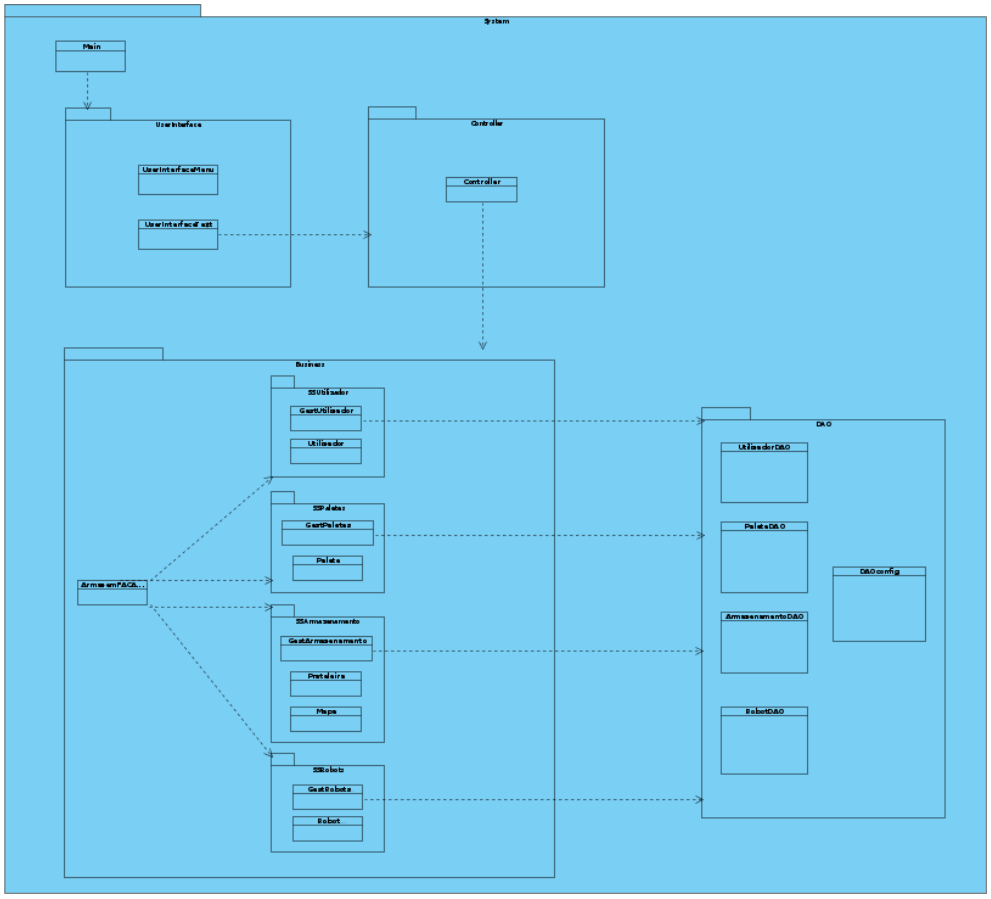

Figura 29 - Diagrama de Packages

Diagrama de Estado

No que à componente dos diagramas diz respeito, neste ponto é possível identificar a implementação de diagramas de estado, componentes que possibilitam a modelação de vários comportamentos de um dado sistema, modelando todos os estados possíveis que o sistema atravessa em resposta aos eventos que possam ocorrer. Assim sendo, no contexto do presente projeto, considera-se relevante a construção de um diagrama de estado que traduza a forma como o sistema reage às diferentes mensagens obtidas não só a partir do menu principal, como também a partir dos respetivos menus secundários. Desta forma, o diagrama de estado que se segue representa os princípios de funcionamento associados à interface implementada.

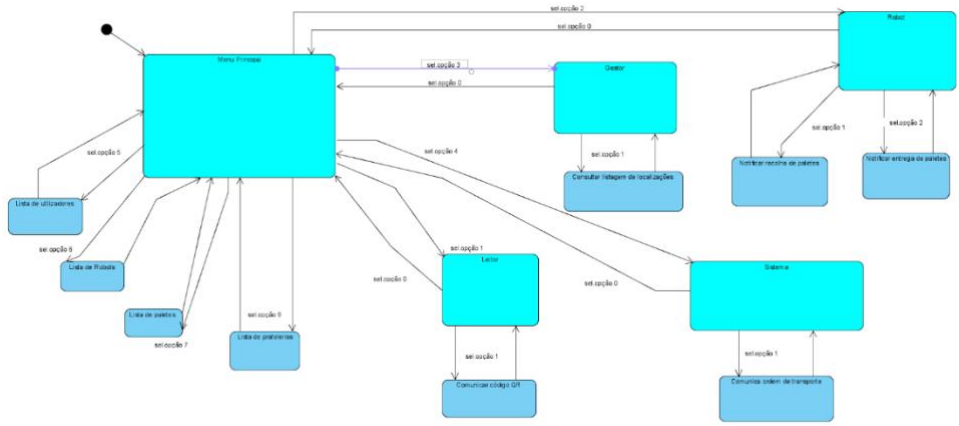

Figura 30 - Diagrama de Estado

4. Implementação

No que à linguagem de programação diz respeito, as classes desenvolvidas vão de encontro ao previamente idealizado no diagrama de classes. No entanto, conforme as mesmas foram sendo implementadas, foram surgindo certas implicações na forma de concretizar certas funcionalidades, o que se traduziu na elaboração de algumas alterações, nomeadamente em termos de métodos e de classes, particularmente relacionadas com a ligação à base de dados.

Como evidenciado anteriormente, o sistema a implementar apresenta uma estrutura tripartida - interface do utilizador, lógica de negócio e base de dados - e tem como foco um conjunto restrito de use cases que podem ser divididos em quatro subconjuntos de acordo com a entidade à qual cada funcionalidade está associada: Sistema, Gestor, Robot e Leitor de QR-Code's. No entanto, face às limitações existentes em termos da automação da aplicação, cada entidade referida é vista como um papel a ser desempenhado por um utilizador singular.

Tendo estes elementos em consideração, e para uma melhor exposição do cariz da implementação, são então explicitadas de seguida as classes constituintes da mesma, divididas segundo a sua camada particular, com maior destaque para a interface e para os subsistemas.

Interface

No que diz respeito à camada da interface do utilizador, é possível observar um grupo de dois componentes cujas designações são UserInterfaceMenu e UserInterfaceText. Tomando a primeira como exemplo, esta serve o propósito de apresentação de informações relativas aos menus no ecrã e a leitura correta de inputs por parte do utilizador no que diz respeito à seleção dos mesmos. Nesse sentido, e de forma a satisfazer as obrigatoriedades impostas pelos papéis enunciados no parágrafo anterior, foi idealizado um conjunto de cinco menus: um menu para cada entidade e um menu principal, cada um com as funcionalidades que lhe são atribuídas, como se pode ver nas imagens que se seguem:

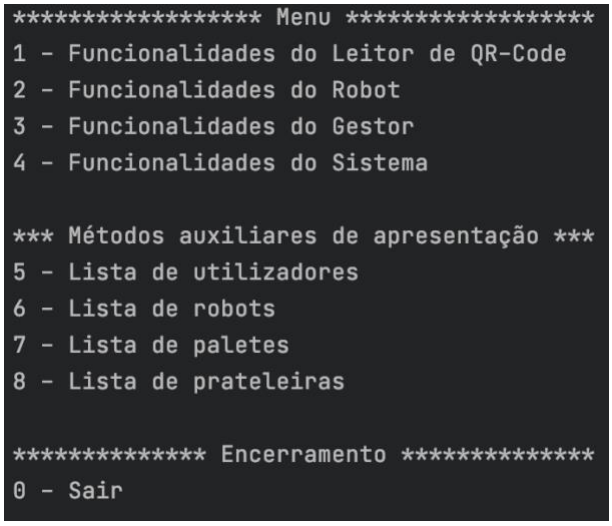

Figura 31 - Menu principal

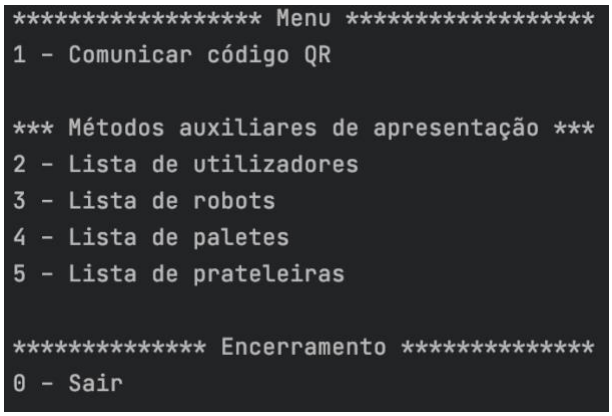

Figura 32 - Menu Funcionalidades do Leitor de QR-Code

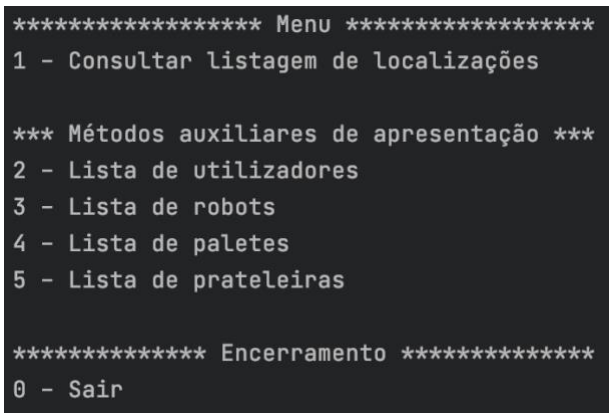

Figura 33 - Menu Funcionalidades do Gestor

```
****************** Menu ******************
1 - Notificar recolha de paletes
2 - Notificar entrega de paletes

*** Métodos auxiliares de apresentação ***
3 - Lista de utilizadores
4 - Lista de robots
5 - Lista de paletes
6 - Lista de prateleiras

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

Figura 34 - Menu Funcionalidades do Robot

```
****************** Menu ******************
1 - Sistema comunica ordem de transporte

*** Métodos auxiliares de apresentação ***
2 - Lista de utilizadores
3 - Lista de robots
4 - Lista de paletes
5 - Lista de prateleiras

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

Figura 35 - Menu Funcionalidades do Sistema

Lógica de Negócio

Por sua vez, relativamente à lógica de negócio é possível observar a existência de seis elementos principais: subsistema GestPaletes, subsistema GestArmazenamento, subsistema GestUtilizador, subsistema GestRobot, ArmazemFacade e Controller, identificados segundo uma ordem crescente em termos de nível e cuja conjugação permite o normal funcionamento da globalidade do sistema.

Controller:

A classe Controller, como o próprio nome indica, comanda o funcionamento do sistema, através da implementação das cinco principais funcionalidades pretendidas para o sistema de gestão de stocks de um armazém. Para tal, esta classe conta com um conjunto diminuto de métodos onde é efetuado o trabalho, isto é, em colaboração com a classe ArmazemFacade, fazendo-lhe pedidos de certas informações, o Controller trata essas informações recebidas de acordo com a funcionalidade pretendida e, por fim, encaminha os resultados para a devida interface.

ArmazemFacade:

Por sua vez, e à semelhança do que acontece com a classe anterior, ArmazemFacade colabora com as classes do nível imediatamente inferior com o objetivo de dar resposta a um requisito de maior nível. Um dos principais benefícios da sua utilização prende-se com o facto de esta permitir ocultar a complexidade dos subsistemas implementados ao comprimir os métodos destes numa interface mais simples. Esta classe faz a ponte entre a classe Controller e as funcionalidades que este pretende executar, ou seja, conecta-a aos vários subsistemas que existem no sistema como um todo.

GestArmazenamento:

Estando presente no package SSArmazenamento, esta classe concentra em si tudo o que se encontra relacionado com o armazenamento e disposição de paletes. Neste sentido, podemos facilmente conectar-lhe as prateleiras do armazém e o mapa do mesmo, como se pode observar nas classes mais atómicas Prateleira e Mapa - a última possui certas opções que a distinguem das demais classes atómicas, a verificarem-se de seguida.

É nesta classe que ocorre a verificação de prateleiras, ou seja, estão aqui definidos os métodos que permitem inferir se uma prateleira se encontra livre e onde é feita a gestão da mesma. É também aqui que são calculadas distâncias entre uma palete e uma prateleira ou então entre um robot e uma prateleira. Importa também salientar que esta classe contribui para o cálculo do percurso de um certo robot, tendo em conta a prateleira mais próxima (método aqui implementado).

Mapa:

A classe Mapa destaca-se de outras classes atómicas como Palete, Prateleira, etc, uma vez que a sua ação não se resume à inicialização e representação de um mapa. Nesta classe é construído um mapa pré-definido tendo por base a ideia disponibilizada de um armazém, onde os vértices que se encontram na inicialização deste mesmo mapa são retirados tendo por base a seguinte figura:

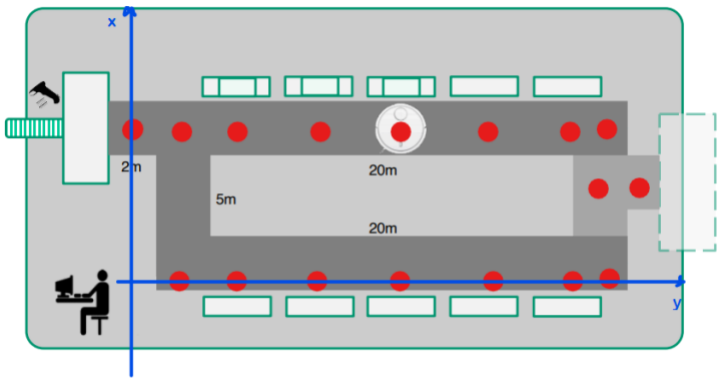

Figura 36 - Mapa do Armazém

Como se pode observar na imagem anterior, o mapa fornecido foi contextualizado num referencial ortonormado, o que permitiu uma interpretação simples e mais direta do mesmo, conduzindo a uma simplificação significativa dos processos ao nível da inserção de informação na base de dados e ao nível dos conceitos de zonas de entrega e recolha, uma vez que estas, deste modo, passam a ser representadas apenas por um vértice.

Nesta classe são ainda definidos métodos que permitem a conexão de vértices, criando arestas, no sentido de construir um grafo que permita simular a movimentação do robot nas suas tarefas de entrega e recolha de paletes. Merecem ainda destaque alguns métodos como percurso, que calcula o menor percurso entre dois vértices, e como distancia e distanciaP, que permitem obter informação acerca da distância entre dois vértices ou acerca da distância total de um percurso, tendo em conta o referencial idealizado.

GestPalete:

Situada no package SSPaletes, esta classe gere toda a informação relativa a paletes. Esta gestão implica a implementação de métodos de atualização e inserção de informações na tabela "paletes" presente na base de dados implementada, podendo estes modificar os dados característicos de uma palete, evidenciados na classe Palete, também ela presente no mesmo package. É ainda relevante mencionar o método generateID(), uma vez que é através deste que são atribuídos os ID's de cada palete aquando da sua chegada ao sistema através do leitor de códigos QR-Code.

GestRobot:

Presente no package SSRobots, a classe GestRobot, à semelhança do que acontece com a classe anterior, faz a gestão do conjunto de dados dos robots, tendo a capacidade de atualização de todas as variáveis de um certo robot através de métodos implementados neste ponto. Para além disto, é pertinente referir dois métodos em particular: getPaleteAssociada e robotMaisProx. A primeira permite obter a palete que se encontra associada a um certo robot, e, por sua vez, a segunda efetua o cálculo do robot mais próximo de uma dada palete.

GestUtilizadores:

Localizada no package SSUtilizador, a classe GestUtilizadores é em muito semelhante a todas aquelas que têm vindo a ser descritas, uma vez que concentra a modificação de dados dos utilizadores do sistema na base de dados. Além destes já habituais métodos de definição, existem ainda outros dois métodos que merecem algum destaque visto que efetuam o início e término de sessão de um certo utilizador no sistema, lançando eventuais exceções.

Exceções

Com o intuito de obter uma experiência moldada às especificações das funcionalidades no que se refere ao tratamento de exceções que possam ocorrer no sistema, foi criado um conjunto de Exceptions relacionado com classes específicas, sendo, assim, possível identificar a exceção de uma forma mais precisa. As Exceptions implementadas são identificadas de seguida, com uma breve descrição do seu propósito:

CredenciaisInvalidas:

Esta exceção diz respeito à questão do utilizador, evidenciando uma incorreta inserção dos dados do mesmo ou a inexistência dessa mesma informação na base de dados.

NoPaletes:

Esta exceção destina-se a destacar a inexistência de paletes no sistema.

NoPrateleiras:

Esta exceção destina-se a destacar a inexistência de prateleiras no sistema.

NoRobots:

Esta exceção destina-se a destacar a inexistência de robots no sistema.

NoUtilizadores:

Esta exceção destina-se a destacar a inexistência de utilizadores no sistema.

PaleteNotFound:

Esta exceção verifica-se quando não existe uma certa palete que se pretende obter.

PrateleiraNotFound:

Esta exceção verifica-se quando não existe uma certa prateleira que se pretende obter.

RobotNotFound:

Esta exceção verifica-se quando não existe um certo robot que se pretende obter.

UtilizadorNotFound:

Esta exceção verifica-se quando não existe um certo utilizador que se pretende obter.

Base de dados

Um dos principais requisitos do projeto passa pela concretização de um sistema de base de dados capaz de assimilar todos os dados do sistema em função da implementação concebida. É possível destacar que, no que à categorização dos dados diz respeito, foram consideradas como necessárias quatro tabelas, divididas segundo a sua área principal de ação: utilizadores, robots, paletes e prateleiras. Nesse sentido, foi concebido, em igual número, um conjunto de quatro classes relacionadas com a terminologia DAO, cada uma representante de uma das entidades referidas acima - UtilizadorDAO, RobotDAO, PaleteDAO e PrateleiraDAO - cujos métodos constituintes visam as interações necessárias com as informações contidas, ou não, na base de dados, como a adição e remoção de elementos, levantamento da totalidade das informações, entre outros.

Deste modo, e considerando as tabelas apenas como bancos de dados sem a necessidade de ligações entre si para esta implementação em particular, foi concebido o modelo lógico que a seguir se observa.

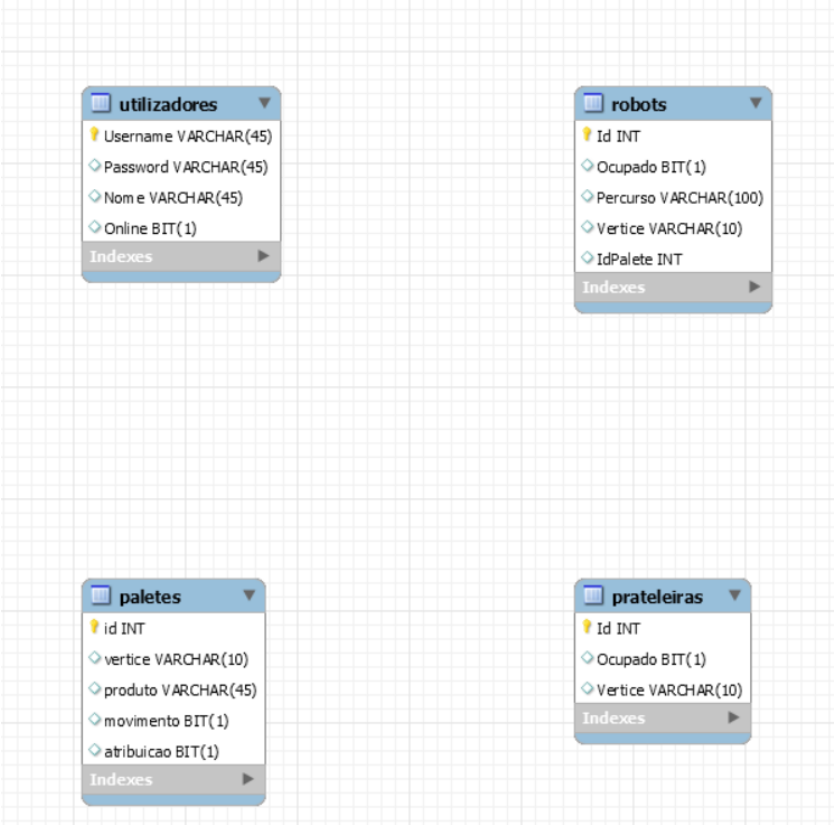

Figura 37 - Modelo Lógico da Base de Dados

5. Análise crítica dos resultados obtidos

De forma a analisar criticamente os resultados obtidos, é descrito o percurso completo do desenvolvimento de um use case em particular, desde o levantamento do seu fluxo, até à sua implementação: o use case "Comunicar código QR" , cuja finalidade se centra na adição de uma palete, identificada com o respetivo QR-Code, à base de dados do sistema.

A comunicação de um código QR-code, ou seja, a adição de uma palete ao sistema, ocorre quando, após uma descarga de novas paletes, os respetivos QR-Codes são lidos por um leitor, inserindo no sistema a palete e os seus respetivos dados, como é o caso do nome do produto. Após esta leitura, a palete é armazenada no sistema com um identificador gerado automaticamente, com as respetivas informações do produto e com a sua localização. A sua implementação é de extrema pertinência, uma vez que um sistema de gestão, em particular um sistema de gestão de stocks, necessita de ter, a todo o momento, o conhecimento exato da totalidade dos dados dos elementos que compõem o referido stock.

O primeiro passo do desenvolvimento de um use case, passa pela criação do seu fluxo. Nesta metodologia é realizada uma descrição do caso em destaque, descreve-se um possível cenário onde o use case é aplicado, identificam-se as condições de inicialização e finalização do mesmo, e explicitam-se todos os fluxos possíveis que deste podem derivar. Nesse sentido, considerando o levantamento efetuado na primeira fase do projeto e as atualizações sofridas na segunda fase do mesmo, o fluxo final para a comunicação de um código QR-Code pode ser observado na figura que se segue.

Use case: Ler QR-Code	
Descrição: Leitor comunica informações de chegada de palete ao sistema	
Cenário: O leitor comunica que chegou uma palete de tomates-cereja ao armazém	
Pré-condição: Leitor está operacional	
Pós-condição: Palete fica registada no sistema	
Fluxo normal:	1. Leitor envia informações da palete ao sistema 2. Sistema regista a entrega da palete na sua base de dados

Como se verifica, é apenas considerado o fluxo normal, uma vez que se entende que aquando de uma entrega, todas as paletes são inseridas no sistema.

Por sua vez, no segundo passo é necessário identificar as principais funcionalidades do use case em análise, caracterizando-as como responsabilidades do sistema. Neste caso foi entendido que o único requisito inerente a esta funcionalidade está relacionado com a adição da palete ao sistema através da sua inserção na base de dados. Nesse sentido, e classificada esta funcionalidade como uma responsabilidade do subsistema atribuído às paletes definido como GestPalete, foi elaborado um método que permitisse a adição da palete ao sistema a partir de um único valor do tipo String, representativo do produto nela contido. Este método possui, então, a seguinte assinatura:

- adicionaPalete(produto : String);

No seu momento, o terceiro passo passa pela criação de um diagrama de sequência em função do use case em questão, onde é descrito o modo como objetos de um determinado grupo interagem entre si, havendo uma particular atenção no que diz respeito à ordem pela qual esta interação ocorre. Seguindo essa linha de pensamento, tem-se que uma mensagem a solicitar a adição de uma palete chega ao ArmazemFacade; este, por sua vez, reencaminha esse mesmo pedido para o subsistema que trata a gestão de paletes: GestPaletes. Neste, ocorrem três eventos: em primeiro lugar, é gerado um identificador para a palete a adicionar, em segundo lugar, é efetuada a construção parametrizada de um novo valor do tipo Palete, através do identificador definido, da localização da zona de descargas (v0_7), do nome do produto passado anteriormente como argumento, e de dois atributos do tipo boolean iniciados a false, um referente ao seu estado de atribuição a um robot, e outro respeitante ao seu estado de movimentação, ou seja, se se encontra em movimento ou não. Finalmente, esta é inserida na base de dados através da chamada da função de inserção definida na classe PaleteDAO, classe de ligação entre o sistema e a componente das paletes no sistema de base de dados. Esta implementação através de um diagrama de sequência pode ser observada na figura 2.

Finalmente, no quarto e último passo, esta funcionalidade é convertida para linguagem de programação, garantindo que a aplicação do use case é satisfeita. Para tal, consideram-se quatro níveis fundamentais, organizados segundo a sua condição de especificidade. Para um melhor entendimento, atente-se na classe UserInterfaceText, em particular no método executaLeituraDeQRCode().

```
private void executaLeituraDeQRCode() {
    scanner.reset();
    System.out.print("\nNome do produto: ");
    String nome=scanner.nextLine();
    this.model.executaLeituraDeQRCode(nome);
}
```

Figura 38 - Método executaLeituraDeQRCode() na UserInterfaceText

Este é o nível mais alto da implementação efetuada, havendo uma conexão direta com o utilizador. Aqui, a classe UserInterfaceText comunica com a interface de Controller, solicitando a execução do método executaLeituraDeQRCode(String nome), onde nome refere o valor lido do input para o nome do produto da palete a adicionar. Desta forma, desce-se então um nível, passando agora para a classe Controller.

```
public void executaLeituraDeQRCode(String nome) { this.model.adicionaPalete(nome); }
```

Figura 39 - Método executaLeituraDeQRCode() no Controller

Na classe Controller existe a comunicação com a interface de ArmazemFacade, solicitando a execução do método adicionaPalete(String nome), onde nome refere o valor passado como argumento para o nome do produto da palete a adicionar. Desta forma, a execução do método é encaminhada para a classe ArmazemFacade e desce-se então um nível.

```
public void adicionaPalete(String nomeProduto) { ssPalete.adicionaPalete(nomeProduto); }
```

Figura 40 - Método adicionaPalete no ArmazemFacade

Já na classe ArmazemFacade, comunica com a interface de GestPaletes, solicitando a execução do método adicionaPalete(String nomeProduto), onde nomeProduto refere o valor passado como argumento para o nome do produto da palete a adicionar. Desta forma, a execução do método é uma vez mais encaminhada, desta vez para a classe GestPaletes e desce-se novamente um nível.

```
public void adicionaPalete(String nomeProduto) {
    int id = this.generateID();
    Palete p = new Palete(id, vertice: "v0_7", nomeProduto, movimento: false, atribuicao: false);
    this.mapPaletes.put(id, p);
}
```

Figura 41 - Método adicionaPalete no GestPaletes

Neste quarto e último nível, a classe GestPaletes aplica o método adicionaPalete(String nomeProduto), satisfazendo por último a execução da funcionalidade pretendida. Aqui dá-se a geração do identificador da palete a inserir no sistema, é criada um valor do tipo Palete seguindo as características descritas acima e é, finalmente, colocado no mapeamento das paletes, concluindo assim a aplicação do use case “Comunicar código QR”.

Tomando este use case como título de exemplo, foi demonstrada a forma como a sua implementação foi realizada, mantendo a coerência e respeitando a divisão do trabalho através da sua difusão por camadas sucessivas. Para os restantes use cases, a sua implementação foi análoga, pelo que se consideram como alcançados os requisitos apresentados pelo sistema e o correto funcionamento das suas funcionalidades.

6. Conclusão

A terceira e última fase deste projeto diz respeito à implementação em linguagem de programação de um conjunto restrito de use cases selecionados entre os requisitos levantados nas fases anteriores do mesmo.

Para corresponder aos objetivos propostos para esta fase, foi necessário ter em consideração a utilização de uma base de dados na programação orientada aos objetos, o que constituiu, desde logo, um fator determinante para a redefinição de diagramas de sequência previamente concebidos e a elaboração de novos diagramas do mesmo formato.

Por sua vez, a implementação efetuada concentra a sua ação numa estrutura dividida em três partes - interface do utilizador, lógica de negócio e base de dados. Face às limitações existentes em termos de recursos, a aceitação de quatro entidades principais como papéis a serem desempenhados por um utilizador único teve alguma influência na implementação da forma de utilização relativamente mais manual do sistema.

Em termos de possíveis pontos fortes da implementação do projeto, destacam-se não só a correção na elaboração dos métodos e a sua eficiência, como também a transposição dos métodos concebidos nos diagramas de sequência da fase anterior de forma generalizada.

Relativamente aos pontos de maior dificuldade, é possível afirmar que o elemento de maior dificuldade se prendeu com o facto de os diagramas concebidos na fase anterior não serem suficientes para a explicitação total das funcionalidades implementadas, e com a necessidade de reestruturação dos mesmos, considerando a introdução do sistema de base de dados.

Por fim, face a eventuais previsões de trabalho futuro, estas surgem maioritariamente associadas à implementação dos restantes use cases associados ao funcionamento do armazém em estudo, levantados nas primeira e segunda fases, e, possivelmente, relacionadas com a definição e implementação de novos requisitos e use cases.