Universidade do Minho
Escola de Engenharia

DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE

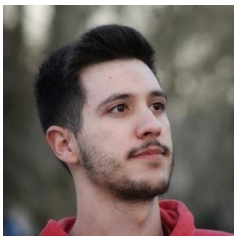
Relatório Projeto Fase III

Sistema de gestão de stocks de um armazém

GRUPO 40

Luís Miguel Pinto A89506

Ana Luísa Carneiro A89533



Pedro Almeida Fernandes A89574

Ana Rita Peixoto A89612

ÍNDICE

INTRODUÇÃO

Na **Fase III** deste projeto propõe-se a **implementação da solução** modelada nas fases I e II**.** Para isso, esta última fase do projeto tem como objetivo inicial definir quais os objetos a serem persistidos através da **criação de DAOs** que implementam a base de dados de cada classe. De seguida, deve-se atualizar os **modelos arquiteturais e comportamentais**, necessários á implementação do subconjunto de *use cases* definidos pelos docentes. O objetivo final do projeto será **implementar em código** o subconjunto de *use cases* que se referem à chegada, transporte e armazenamento de paletes no armazém. Para além disso, como forma de tornar o projeto mais completo implementou-se um **diagrama de estados** que representa a interface do sistema.

ALTERAÇÕES FASE II

Ao longo da realização da fase III, notou-se que havia algumas alterações que teriam de ser feitas nos diagramas da fase II de forma a tornar a última fase do projeto mais completa e funcional. Duas dessas alterações passaram por acrescentar nos diagramas de packages e componentes a representação dos DAOs. Para além disso, verificou-se que alguns métodos que se encontravam nos diagramas de classes não estavam corretos ou teriam de ser alterados de forma a completar a sua implementação em código. Por fim atualizou-se os diagramas de classes e de sequência, devido à criação dos DAOs. Apresenta-se na próxima secção os diagramas de packages e componentes atualizados. Os restantes diagramas encontram-se representados nas respetivas etapas em que foram atualizados.

ARQUITETURA EM TRÊS CAMADAS

Diagrama de Packages

Em relação à fase anterior, foi necessário alterar o diagrama de *packages* de modo a incluir a base de dados. O nosso projeto encontra-se organizado através de uma arquitetura em 3 camadas, visível através da existência dos *packages* UI, Business e Data, que constituem as camadas de apresentação, de lógica de negócio e de dados, respetivamente. Deste modo, obteve-se o seguinte diagrama:

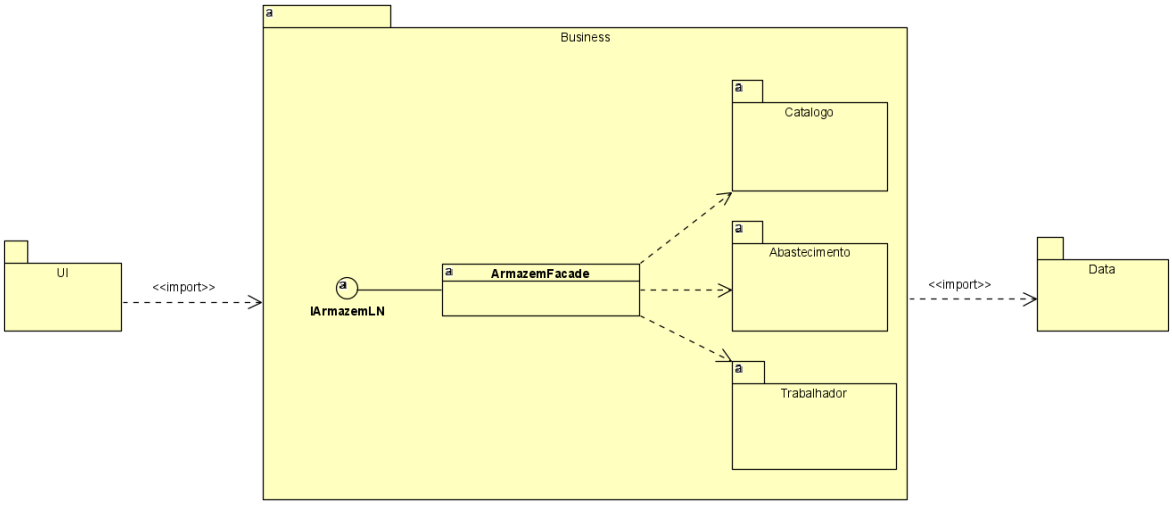


Figura 1: Diagrama de *packages* atualizado

Diagrama de Componentes

Foi necessário atualizar o diagrama de componentes de modo a considerar a base de dados do armazém. Deste modo, obteve-se o seguinte diagrama:

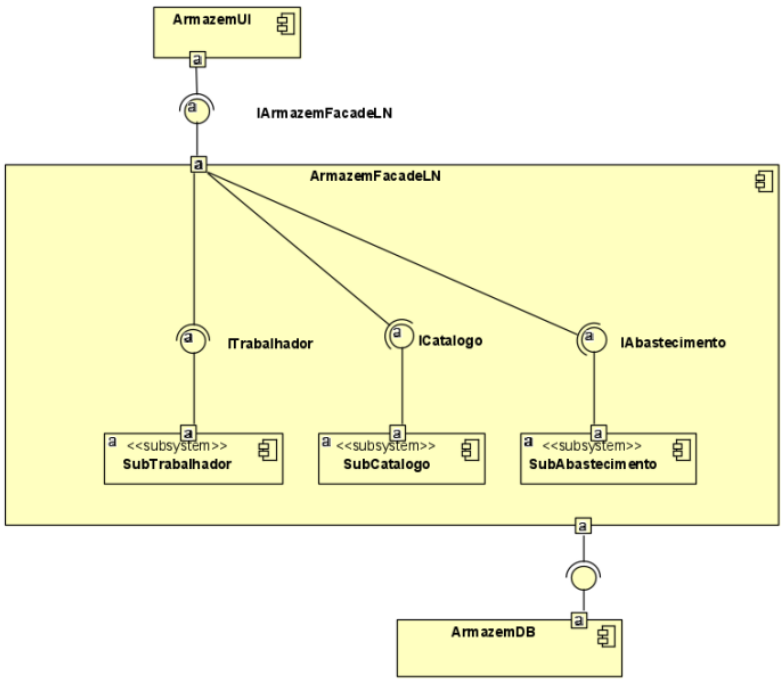


Figura 2: Diagrama de componentes atualizado

DAOs

De modo a implementar uma Base de Dados no programa, foi necessário considerar DAOs (*Data Access Objets*) e selecionar classes a serem persistidas. Os DAO permitem persistir objetos na base de dados, criar objetos a partir da informação da BD e encapsular *queries sql.* Além disso, para cada DAO, foi necessário implementar um conjunto de operações base que permitem a comunicação com a base de dados e, consequentemente, acesso aos dados armazenados. Os métodos efetivamente implementados foram os métodos utilizados para suportar os *use case* selecionados.

Base de Dados

Os DAOs implementados possuem uma tabela na base de dados. Na seguinte figura está presente o modelo lógico da base de dados que suporta o programa, onde é possível observar quais as **chaves primárias** de cada entidade e também as **chaves estrangeiras**, caso existam.

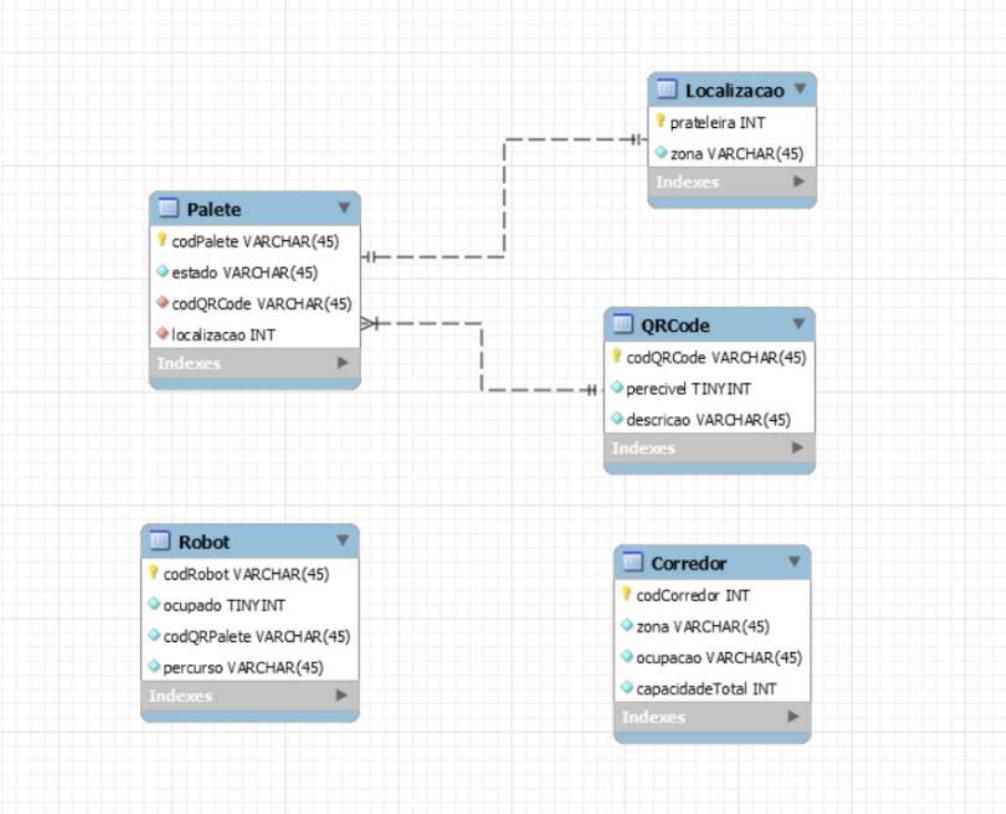


Figura 3: Modelo lógico da Base de Dados

Diagrama de classes

A seguir serão expostos os diagramas de classes respetivos aos *packages* atualizados na implementação do projeto (Fase III).

Relativamente ao *package* utilizador, apesar de não ser necessária a sua implementação nesta fase, adicionamos também o DAO para a classe utilizador. Não será considerada a sua explicação no presente relatório, dado que não era um requisito necessário. No entanto, a implementação do seu DAO permite abrir portas para uma implementação futura e mais ampla.

Armazém

Em relação ao *package* Armazem, foram acrescentados alguns métodos e também variáveis de instância relativas aos *packages* anteriores, tal como visível na seguinte figura:

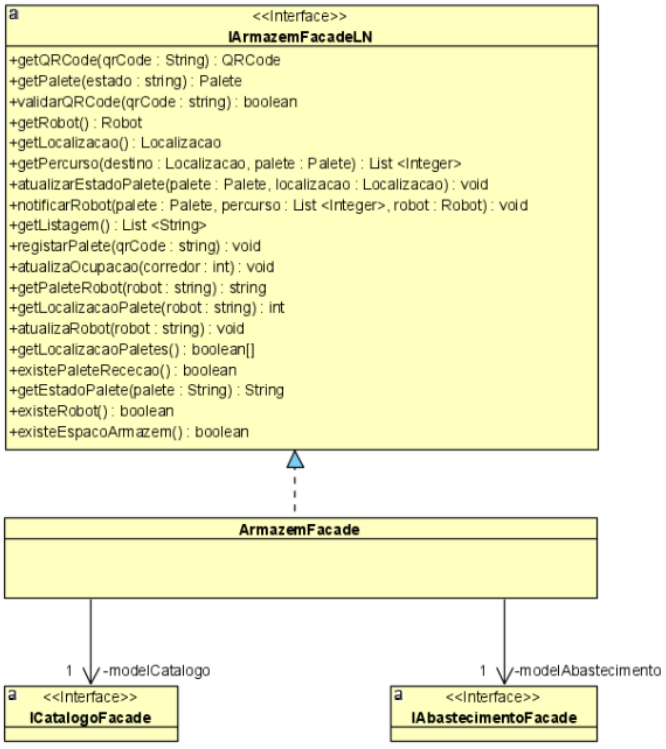


Figura 4: Diagrama de classes do armazém

Abastecimento

Neste *package* foram persistidas as classes Corredor e Robot através das classes CorredorDAO e RobotDAO. De forma a mapear estas classes em tabelas de base de dados representadas pelos DAO's, elegeu-se o codCorredor e o codRobot como **chaves primárias** da tabela para o CorredorDAO e o RobotDAO, respetivamente. Os restantes atributos da tabela são as variáveis de decisão de cada classe. O seguinte diagrama de classes contém a atualização relativa aos DAO, do diagrama de classes do *package* abastecimento:

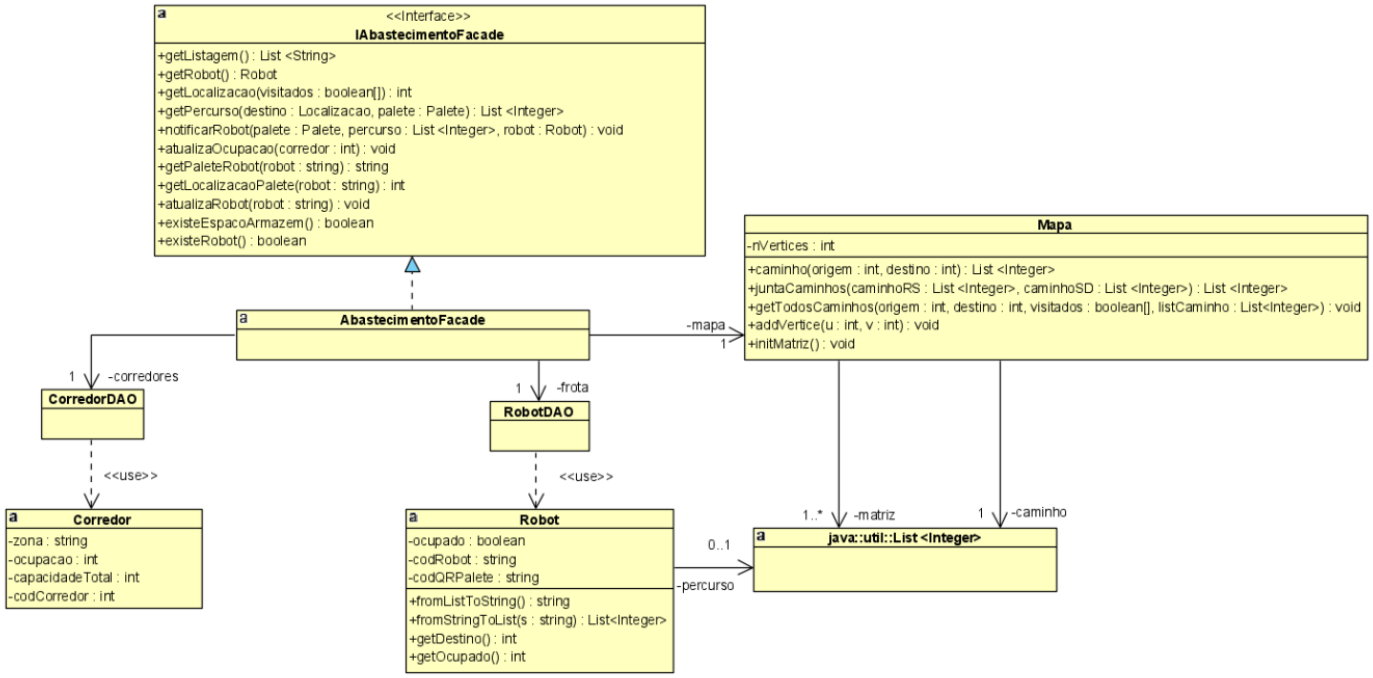


Figura 5: Diagrama de classes do abastecimento

Na seguinte figura está presente a versão final do mapa do armazém. A atribuição do número de prateleira (1-10) teve em consideração a distância a que se encontra da receção, sendo 1 o mais próximo e 10 o mais distante. Além disso, o vértice 11 representa o ponto de partida dos *robots*, o vértice 0 representa a zona de receção e o 12 a zona de entrega (saída) do armazém.

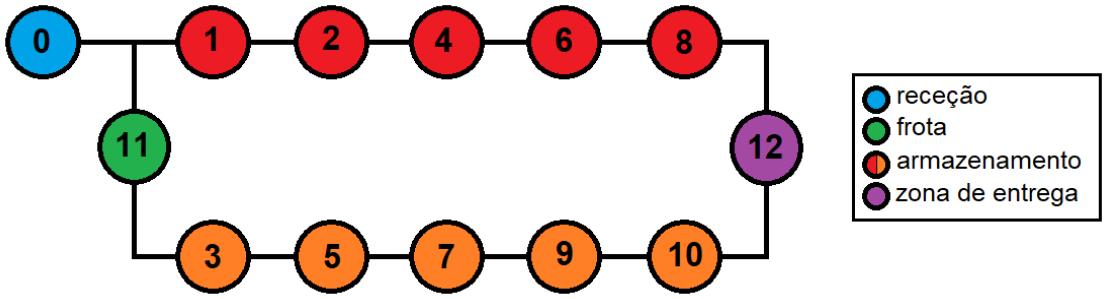


Figura 6: Mapa final do armazém

Na classe Mapa foi implementada uma matriz de adjacência, baseada num grafo orientado (figura 6), capaz de representar o mapa do armazém (figura 5). Com esta representação é possível representar armazéns com diferentes dimensões.

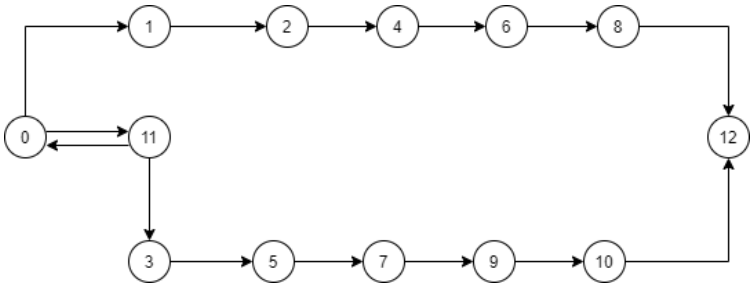


Figura 7: Representação do mapa num grafo orientado

Catálogo

Em relação ao *package* Catálogo, foram adicionadas as classes PaleteDAO e QRCodeDAO, de modo a persistir os objetos das classes Palete e QRCode. Os **identificadores** das tabelas persistidas na base de dados são o codPalete, no caso da classe PaleteDAO, e o codQRCode, para a classe QRCodeDAO.

Em relação à fase II foram feitas algumas alterações. A implementação anterior considerava uma estrutura para cada estado da palete, ou seja, uma lista onde seriam guardadas temporariamente as paletes presentes na receção, uma para paletes a aguardar transporte, outra para paletes no robot e, por fim, uma para as paletes requisitadas. Nesta fase, considerou-se melhor prática prescindir desta implementação e adotar uma *string* "estado" presente na palete que identifica qual o seu estado naquele momento. Esta alteração baseou-se na necessidade de simplificar o armazenamento em base de dados. Em adição, consideramos duas estruturas "paletes" e "armazenamento", com o propósito de representar o *map* de listas considerado na Fase II. Para isso, foi necessário acrescentar uma *Collection* à classe QRCode de modo que seja possível efetuar a correspondência entre as paletes e o respetivo QRCode.

Assim, o seguinte diagrama de classes contém as alterações feitas neste *package*:

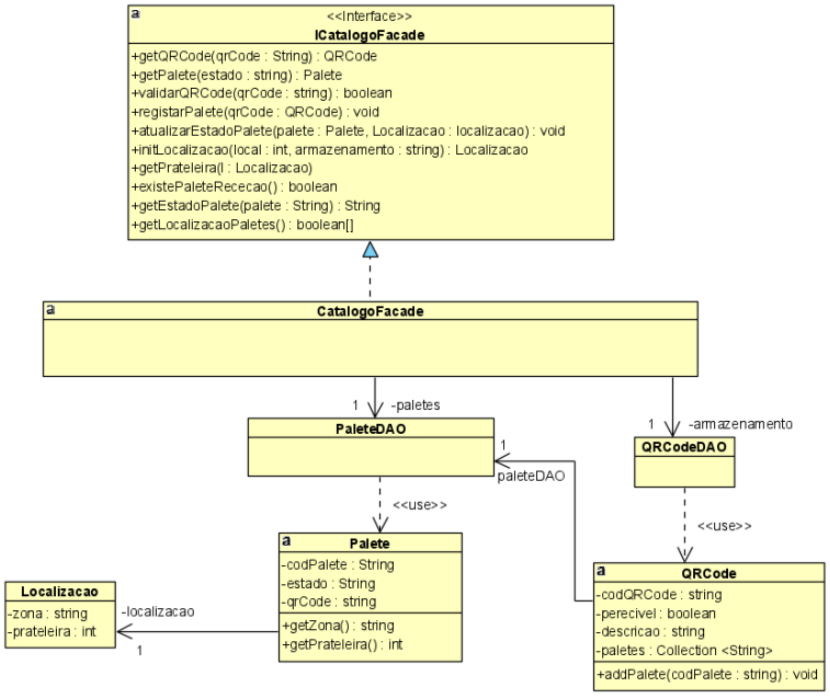


Figura 8: Diagrama de classes do catálogo

DIAGRAMAS DE SEQUÊNCIA

Foi necessário atualizar os diagramas de sequência elaborados na fase II, de modo a considerar os DAO implementados. Deste modo, os diagramas obtiveram a seguinte estrutura:

Use Case: Consultar listagem

Obter listagem

Neste método não houve alterações em relação à fase anterior.

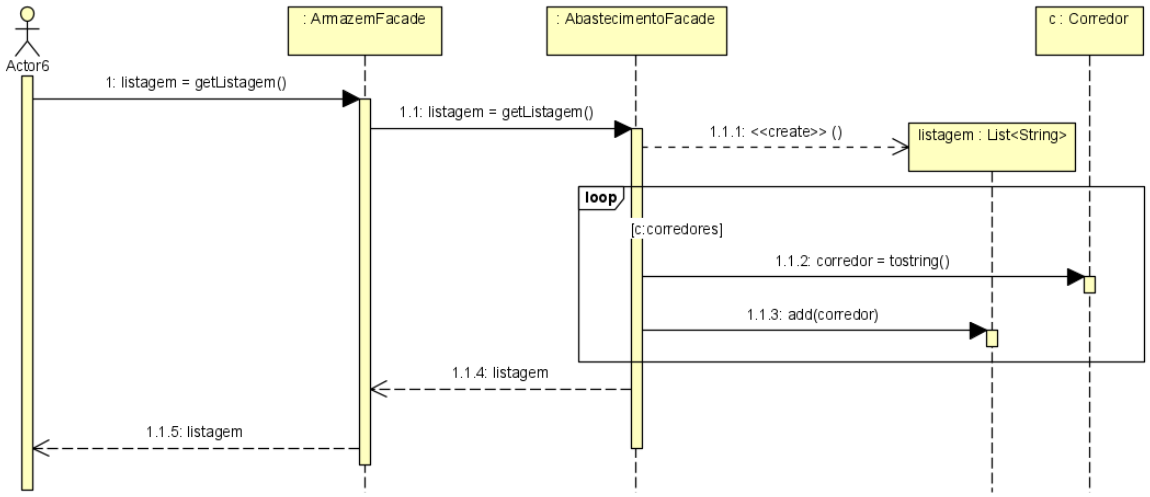


Figura 9: Diagrama de sequência do método *getListagem*

Use Case: Comunicar QRCode

Validar QRCode

A alteração presente neste diagrama de sequência residiu na verificação de existência do QRCode na base de dados, a partir da classe PaleteDAO.

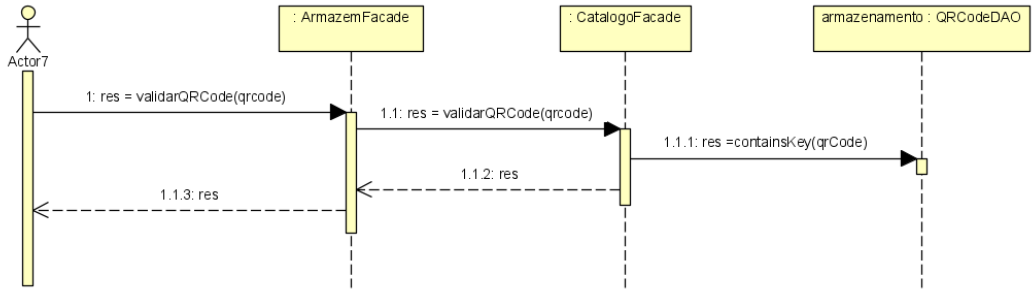


Figura 10: Diagrama de sequência *ValidarQRCode*

Registar Palete

Em relação à fase II foi necessário efetuar algumas alterações neste método. Além de ser necessário recorrer aos DAOs implementados, foi acrescentado uma parte de código que visa manter a unicidade de códigos de paletes.

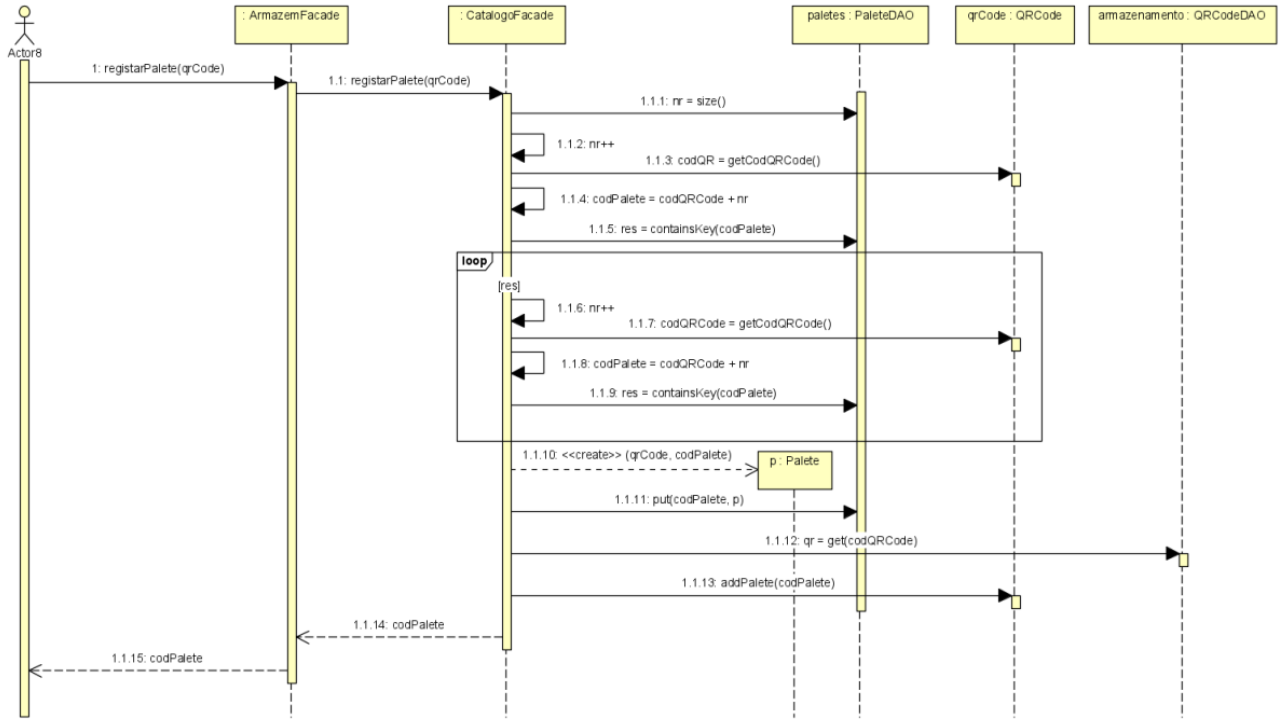


Figura 11: Diagrama de sequência *registarPalete*

Use Case: Ordenar transporte

Obter Palete

Em relação ao método *getPalete* as alterações efetuadas surgiram em consequência da alteração da representação do estado da palete.

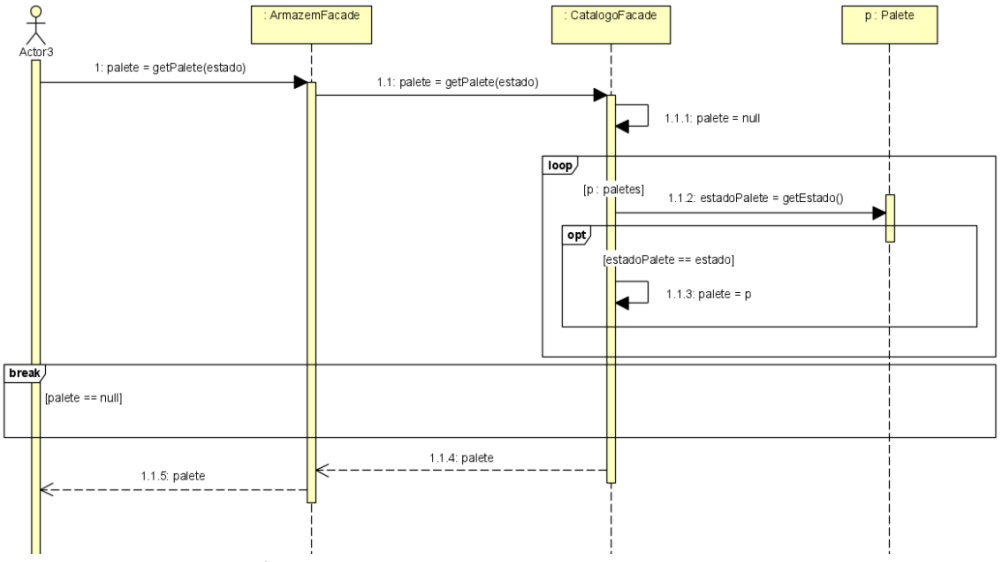


Figura 12: Diagrama de sequência *getPalete*

Obter Localização

No caso do método *getLocalizacao*, foram efetuadas algumas mudanças. De modo a verificar quais as prateleiras que estavam ocupadas e qual a localização futura da palete foram utilizados 3 métodos distintos que são chamados no *Armazemfacade*.

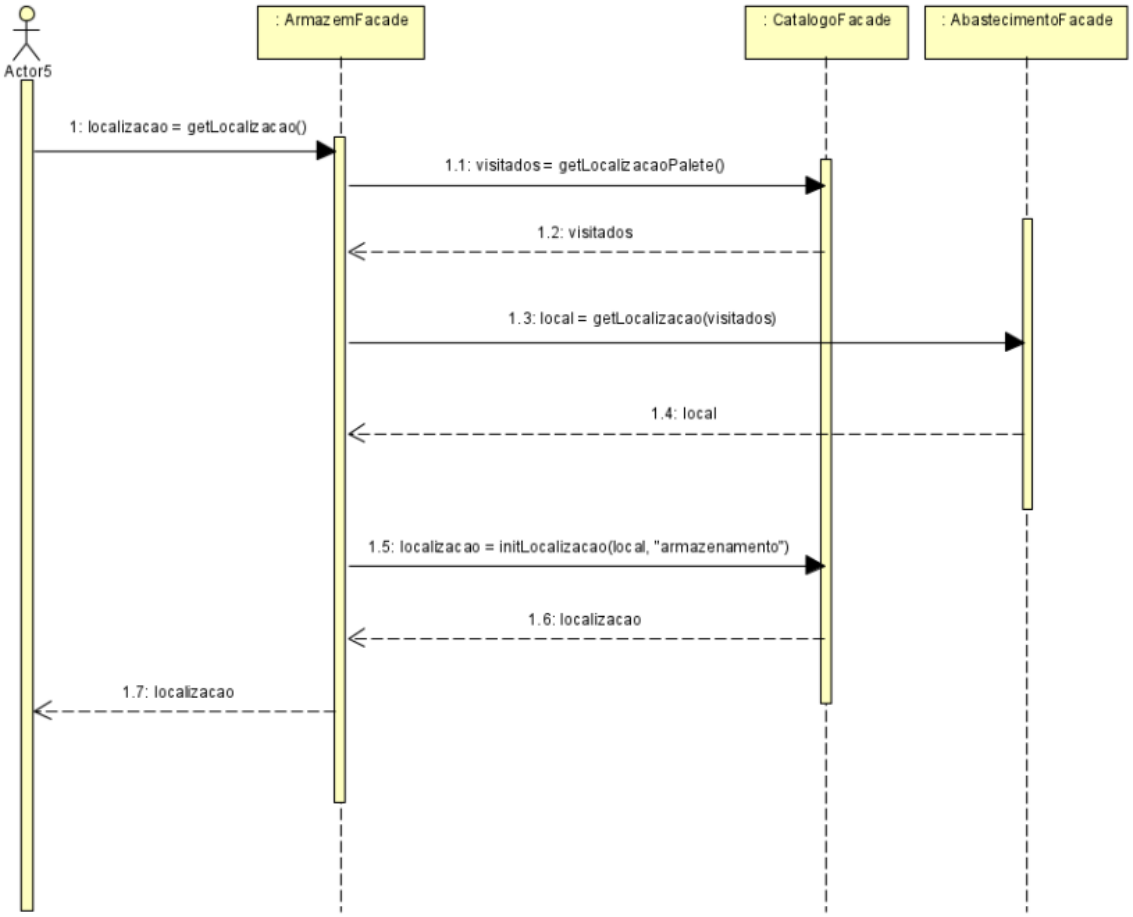


Figura 13: Diagrama de sequência do *getLocalizacao*

No método *getLocalizacaoPalete* é criado um *array* de *boolean* que indica se uma dada prateleira está ocupada ou não. Esse *array* é constituído por 10 entradas, sendo que o índice 0 é a prateleira mais próxima da receção, e o 9 a mais distante.

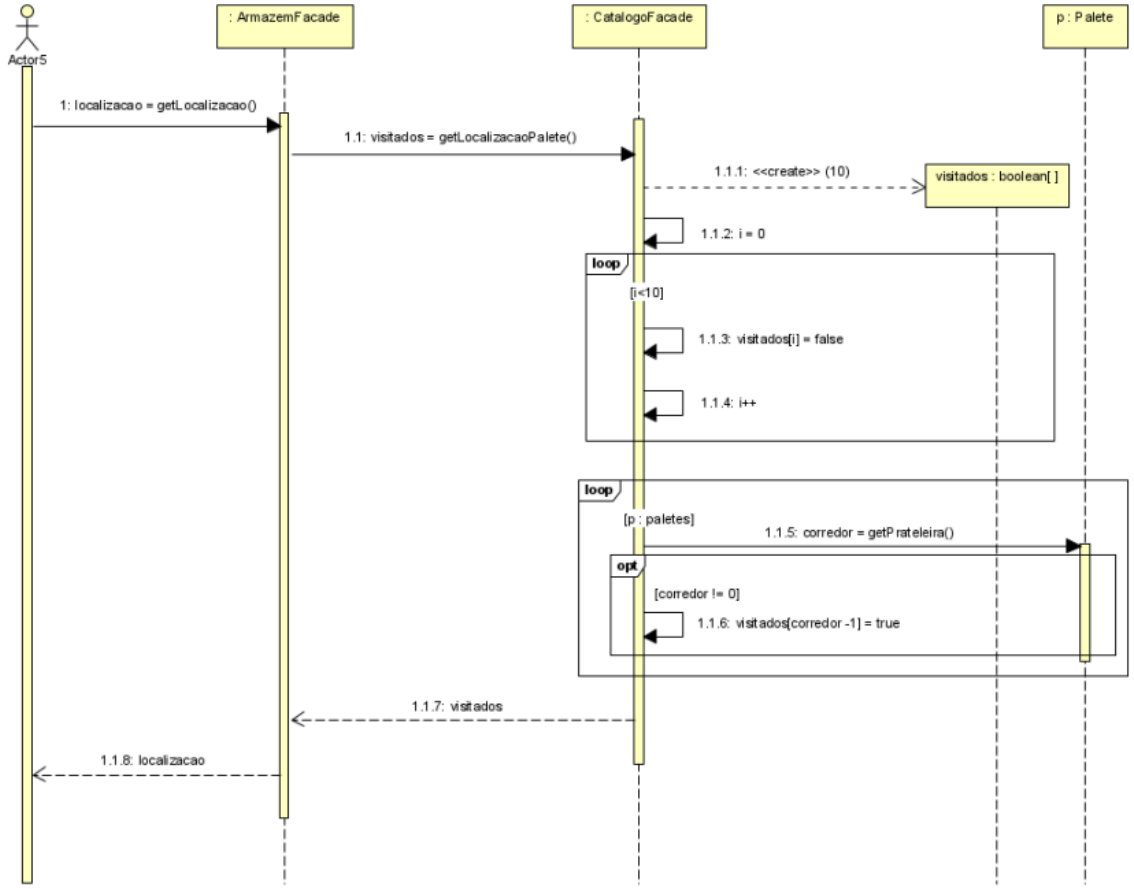


Figura 14: Diagrama de sequência *getLocalizacaoPalete*

No método *getLocalizacao* da classe *AbastecimentoFacade*, vai se percorrer o *array* de *boolean* e verificar qual a prateleira mais próximo que esteja disponível, devolvendo o identificador da prateleira.

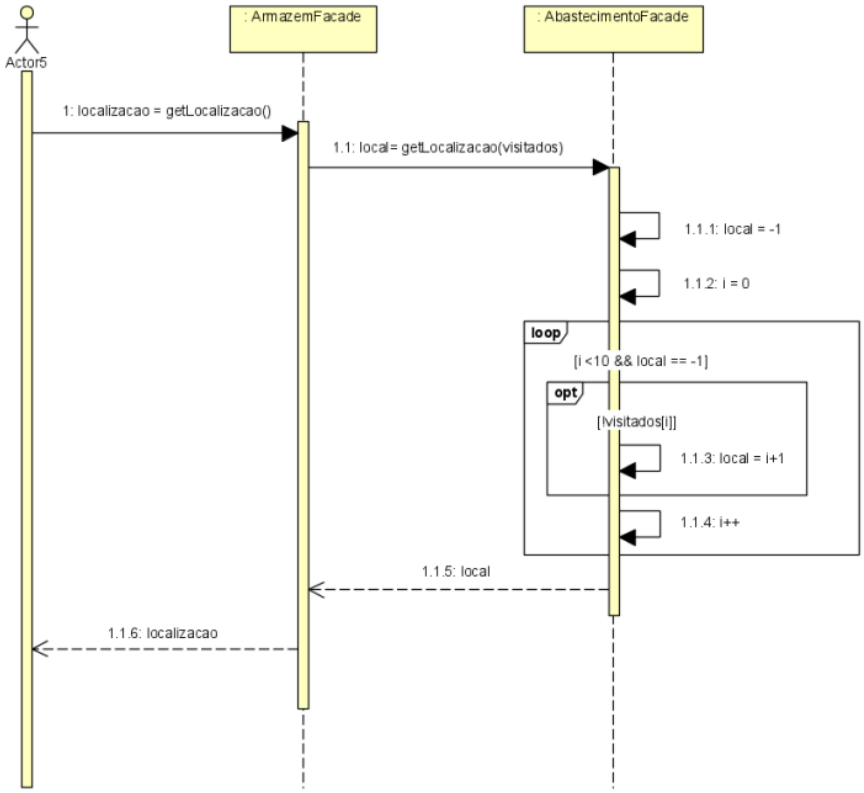


Figura 15: Diagrama de sequência *getLocalizacao*

No método de *initLocalizacao* vai criar uma localização para a palete com os parâmetros fornecidos pelas métodos em cima representados.

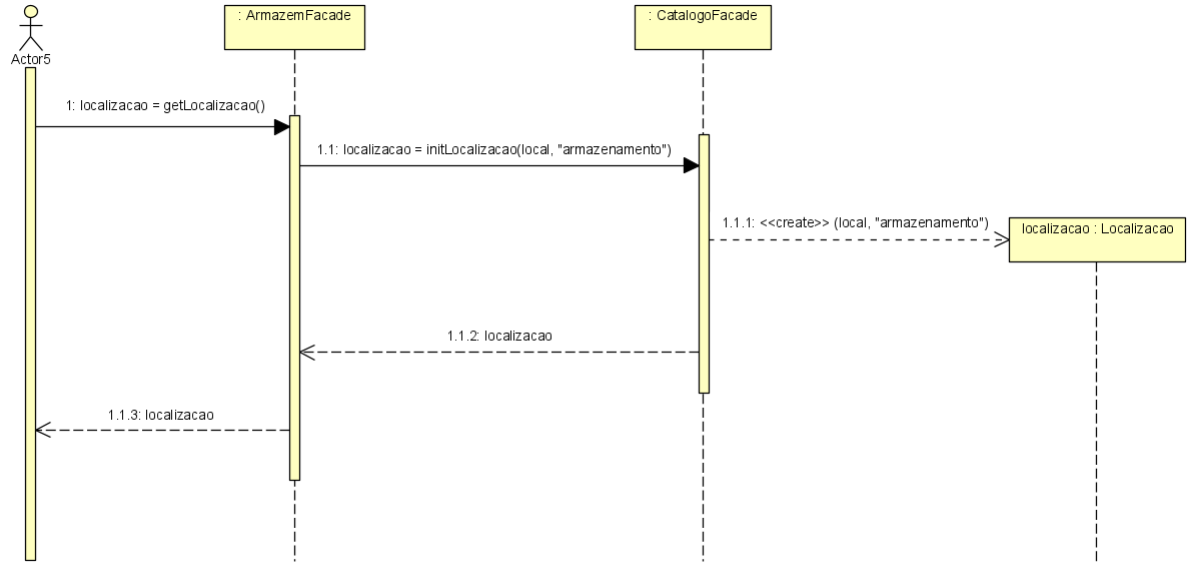


Figura 16: Diagrama sequência *initLocalizacao*

Obter Robot

Neste método não houve alterações em relação à fase anterior.

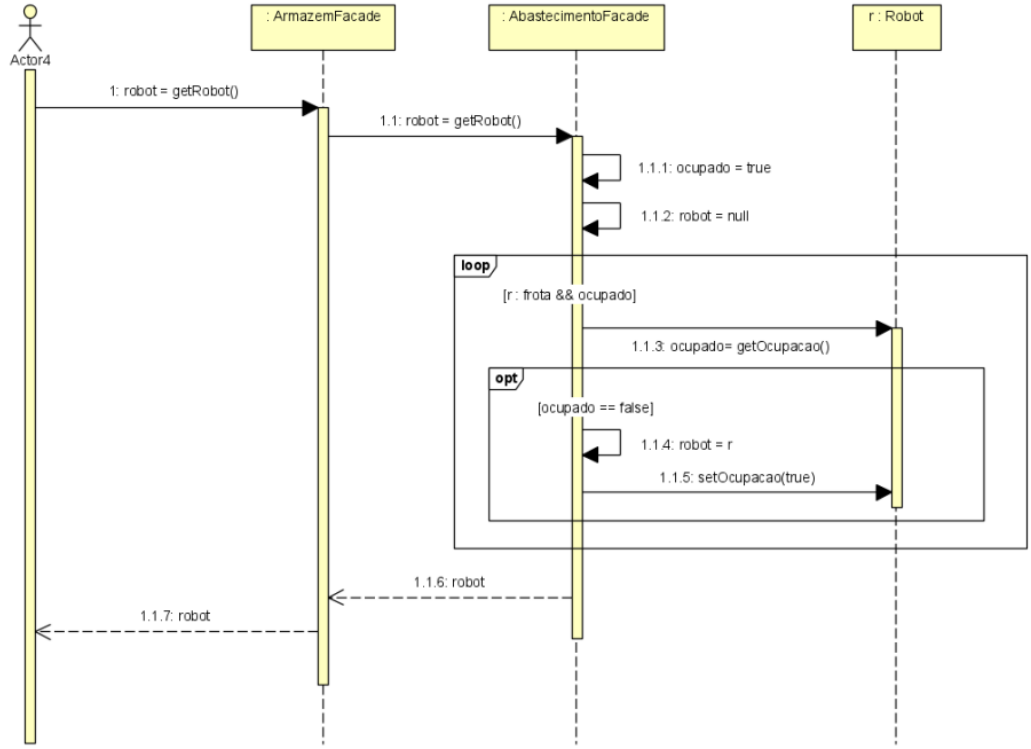


Figura 17: Diagrama de sequência *getRobot*

Obter Percurso

Neste método não houve alterações em relação à fase anterior.

O algoritmo utilizado para obter o percurso baseou-se numa travessia *breadth-first*, recorrendo à função recursiva "caminho".

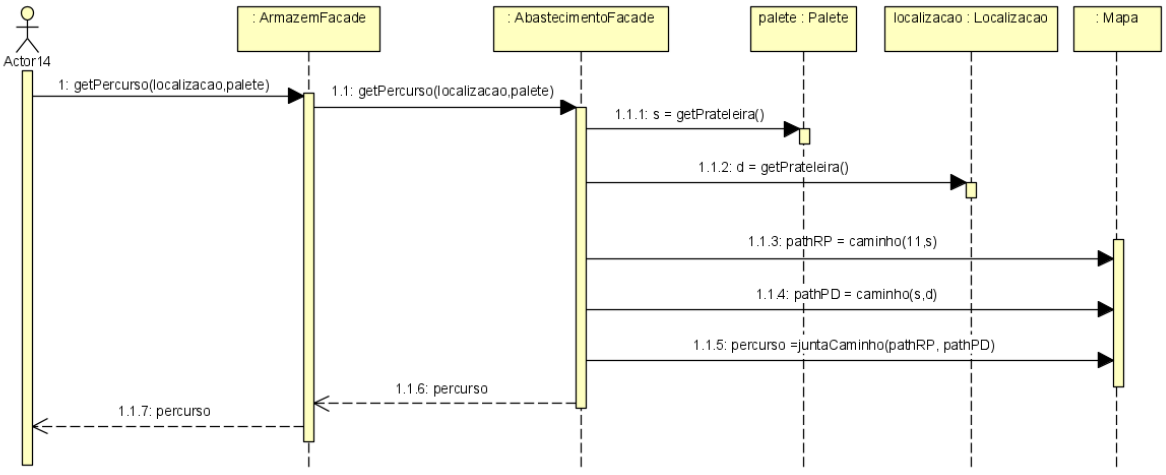


Figura 18: Diagrama de sequência *getPercurso*

Atualizar Estado da Palete

Uma das alterações feitas neste método foi acrescentar a classe PaleteDAO para aceder ao seu conteúdo. Além disso, foi efetuada a junção de dois métodos em um só, eliminando redundâncias que surgiram a partir das alterações feitas e dado que não foi necessário implementar métodos para a requisição de paletes.

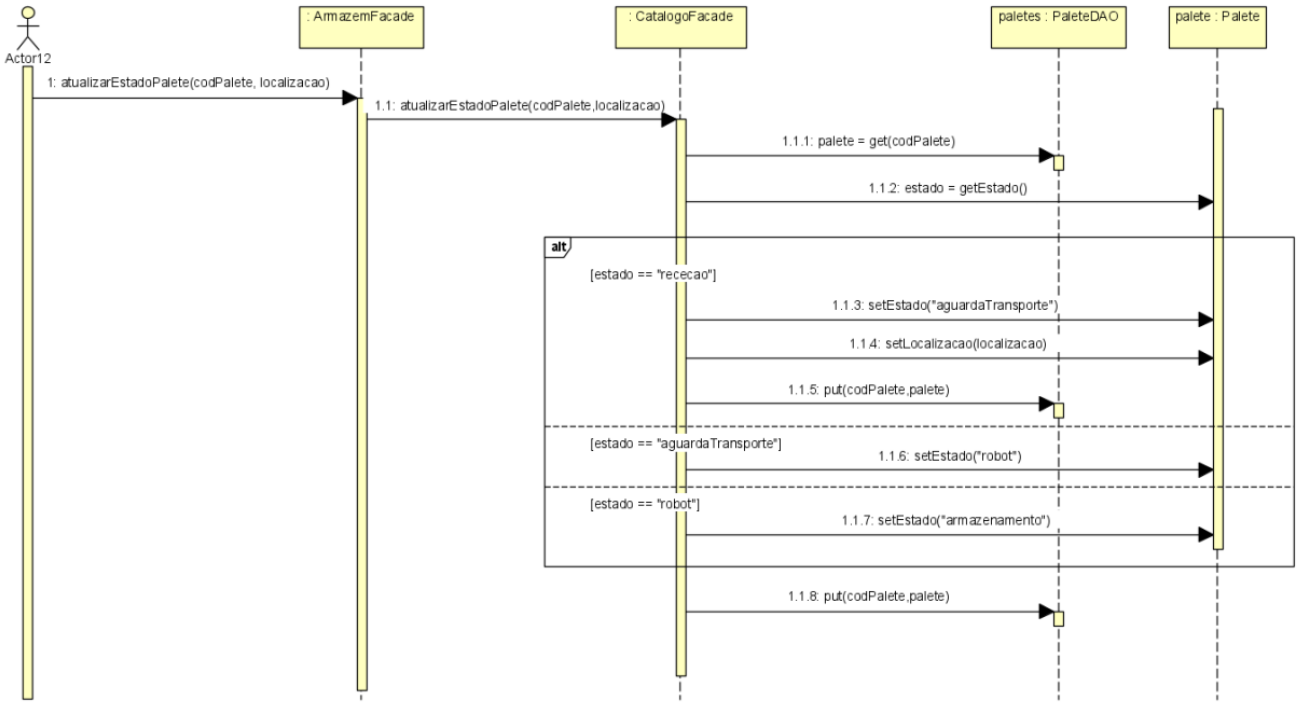


Figura 19: Diagrama de sequência *atualizarEstadoPalete*

Notificar Robot

A alteração deste método permitiu atualizar a ocupação do armazém, para além de notificar o robot.

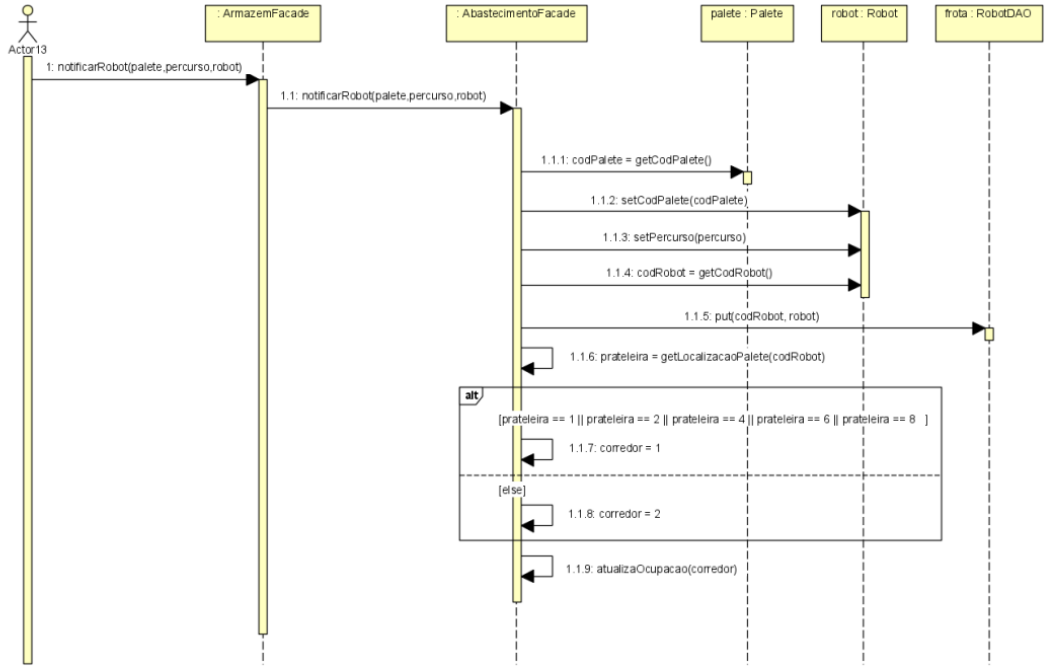


Figura 20: Diagrama de sequência *notificarRobot*

Use Case: Notificar Recolha de Paletes

Neste Use Case foi utilizado o método ***atualizarEstadoPalete*** mencionado anteriormente.

Use Case: Notificar Entrega de Paletes

Neste Use Case foi utilizado o método ***atualizarEstadoPalete*** mencionado anteriormente.

Atualizar Robot

O método *atualizaRobot* altera a ocupação do robot para *false* e o percurso para *null*, dado que a palete já foi entregue.

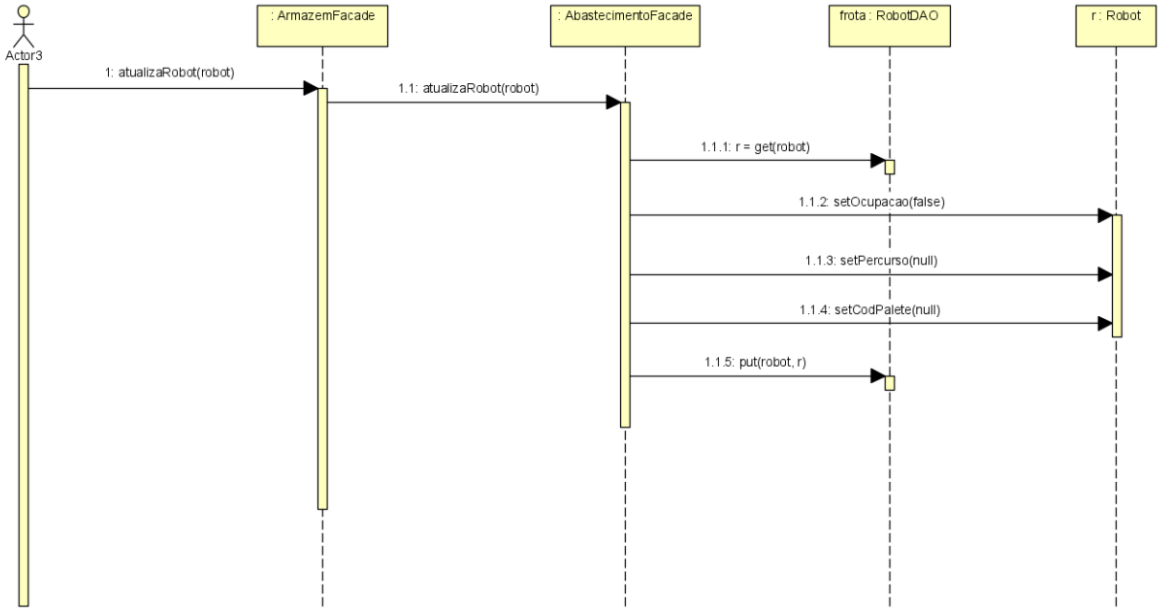


Figura 21: Diagrama de sequência *atualizaRobot*

DIAGRAMA DE ESTADO

Os diagramas de estado tem como objetivo modelar o comportamento de um dado objeto/sistema de forma global. Deste modo, são analisados todos os estados que um objeto pode adquirir e qual a resposta face aos eventos que podem ocorrer. O diagrama de estado elaborado foi do *package* UI, de modo a que seja possível compreender o comportamento do objeto de forma global ao sistema.

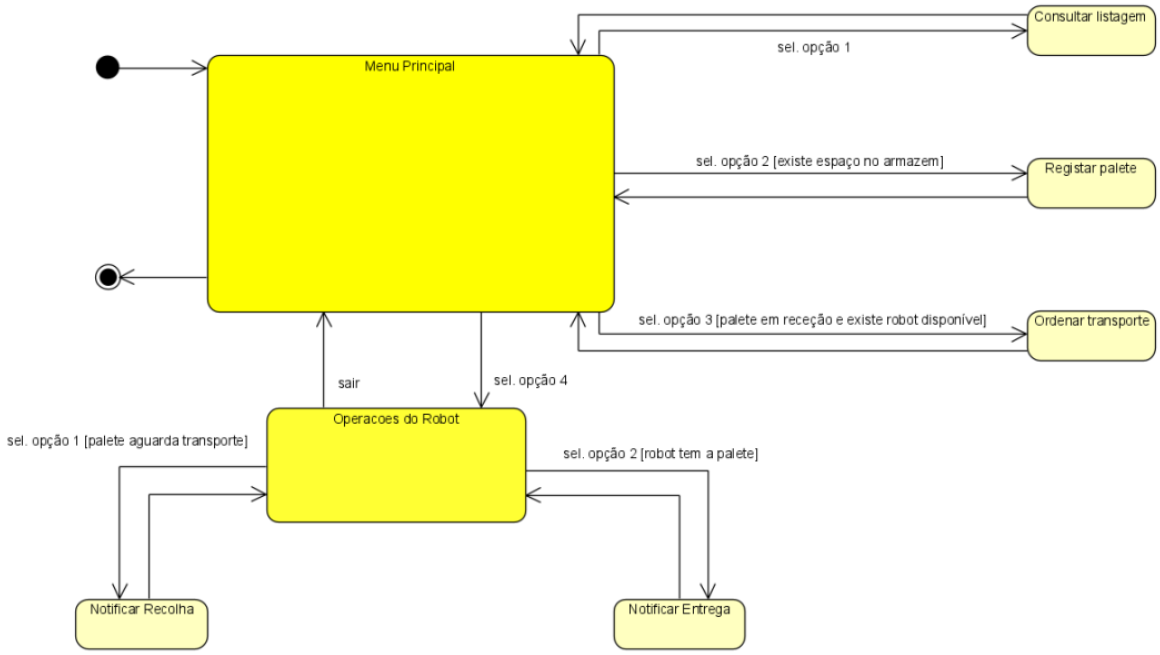


Figura 22: Diagrama de estado do menu

IMPLEMENTAÇÃO

Manual de utilização

O **processo de transporte de paletes** inicia-se no Use Case 1. Este Use Case corresponde à opção "1 Consultar listagem" presente no menu e deve ser efetuada pelo gestor.

```
********* Menu *********
1 Consultar listagem
2 Registar nova palete
3 ---
4 Operações do Robot
0 Sair
Opção:
```

Figura 23: 1ª instância do menu

Neste ponto, o gestor avalia a ocupação do armazém e tem capacidade para decidir se pode ou não haver entregas de paletes no armazém.

```
Opção: 1

Corredor 1:
   zona = armazenamento
   ocupacao = 3/5
Corredor 0:
   zona = rececao
   ocupacao = 0/10
Corredor 2:
   zona = armazenamento
   ocupacao = 2/5
```

Figura 24: Listagem fornecida ao gestor

De seguida, é possível efetuar o registo de uma nova palete (opção "2 Registar nova palete" do menu). A título de exemplo, efetuou-se o registo de uma palete cujo QRCode é A.

```
********* Menu *********
1 Consultar listagem
2 Registar nova palete
3 ---
4 Operações do Robot
0 Sair
Opção: 2
Insira o QRCode da palete a adicionar:
⇒ A
Código da palete adicionada: A6
```

Figura 25: Registo de uma nova palete

Após o registo de uma nova palete, podemos consultar novamente a listagem, para verificar as alterações ocorridas.

```
Opção: 1

Corredor 0:
   zona = rececao
   ocupacao = 1/10
Corredor 1:
   zona = armazenamento
   ocupacao = 3/5
Corredor 2:
   zona = armazenamento
   ocupacao = 2/5
```

Figura 26: Ocupação do armazém

Tal como é visível na figura 23, a ocupação da zona de receção aumentou 1 unidade.

O próximo passo a efetuar é atribuir um robot a uma palete e um percurso a um robot, de modo que seja possível transportar a palete a partir da zona da receção até ao local de armazenamento. Este Use Case está representado no menu através da opção "3 Comunicar ordem de transporte.", e apenas pode ser efetuado caso existam paletes na receção, ou seja, paletes que necessitam de transporte.

```
Opção: 3

 Robot B:
  ocupado = true
  codigo palete = A6
  percurso = [11, 0, 1, 2, 4]
```

Figura 27: Execução da opção 3 do menu

De seguida, é necessário efetuar as operações relativas à recolha e entrega de paletes, por parte do robot. Estas operações estão agregadas na opção "4 Operações do robot".

```
Opção: 4
Insira robot a notificar:
⇒ B

 ********* Menu *********
1 Notificar Recolha
2 ---
0 Sair
Opção: 1
Robot B notificou a recolha da palete A6
```

Figura 28: Notificar Recolha de Palete

O robot pode efetuar 2 ações: notificar a recolha e a entrega de paletes. No entanto, estas operações apenas podem ser efetuadas em determinadas circunstâncias: quando a palete possui o estado “aguardaTransporte” só é possível efetuar a opção “1 Notificar Recolha”, e quando o estado da palete é “robot” apenas faz sentido efetuar a opção “2 Notificar Entrega”. Assim, foi necessária a implementação de pré-condições no programa, de modo a cumprir os requisitos anteriormente referidos.

```
********* Menu *********
1 ---
2 Notificar Entrega
0 Sair
Opção: 2
Robot B notificou da entrega da palete A6 em
Localizacao: zona = armazenamento   prateleira = 4
```

Figura 29: Notificar Entrega de Palete

É possível observar que a ação foi efetuada com sucesso, pela ausência de mensagem de erro e pela presença de mensagem de confirmação de entrega.

Por fim, podemos observar novamente a listagem, de modo a confirmar a alteração das ocupações dos corredores respetivos.

```
Opção: 1

Corredor 2:
   zona = armazenamento
   ocupacao = 2/5
Corredor 1:
   zona = armazenamento
   ocupacao = 4/5
Corredor 0:
   zona = rececao
   ocupacao = 0/10
```

Figura 30: Listagem após armazenamento da palete

Gestão de frota de robots

De modo a gerir uma frota de robots, foi necessário manter diferentes correspondências robot-palete distintas.

Para compreender melhor a implementação da gestão de múltiplos robots, iremos começar por demonstrar como funciona o processo para um robot. Na opção "3 Comunicar ordem de transporte" do menu é selecionada uma palete presente na zona de receção, um robot disponível e uma localização livre. Posteriormente, é efetuada a atribuição da palete ao robot juntamente com o percurso que este terá de efetuar.

Agora estamos em condições de entender como é mantida a gestão de frota de robots. Assim, de modo a evitar uma solução periclitante, fundamentamos a gestão da frota em dois pilares:

1. uma palete não pode ser atribuída a dois robots distintos;
2. uma localização livre do armazém nunca poderá ser considerada como destino final de robots diferentes.

Dado que existe uma atribuição de uma dada palete e um dado destino a um robot em particular, é garantido o correto funcionamento do programa, na medida em que não haverá possibilidade de um robot interagir com uma palete que não lhe tenha sido atribuída ou com um destino que não lhe corresponda.

ANÁLISE CRÍTICA GLOBAL DOS RESULTADOS

Para expressar a analise crítica do trabalho realizado ao longo das três fases do projeto optamos por apresentar a construção passo a passo da funcionalidade de **obtenção da listagem do armazém**.

Fase 1 – Análise De Requisitos

Através de uma leitura atenta do enunciado do projeto, mais concretamente do cenário de utilização três, foi evidente a necessidade de cumprir o requisito acima mencionado. No modelo de domínio, criou-se as entidades **listagem**, nível de ocupação, gestor, pedido de descarga e localização, uma vez que a sua existência era fundamental para a coerência do modelo. Colmatamos esta primeira fase com o **use case Autorizar Pedido de Descarga**. Nele evidenciamos um fluxo de operações conferindo propósito às entidades e relações previamente estabelecidas. Sucintamente, após o gestor efetuar autenticação no sistema e aceder aos pedidos de descarga por analisar, pede uma listagem de localizações de modo a poder avaliar o nível de ocupação do armazém. Com esta informação poderá concluir se tem condições para autorizar o pedido de descarga.

Fase 2 - Modelação Conceptual Da Solução

No que toca à segunda fase do trabalho, foi crucial representar esquematicamente o use case anterior, de modo a melhor visualizar a ação em questão. Primeiramente, após identificar as responsabilidades da lógica de negócio e os métodos necessários (API), foi necessário agrupar os diferentes métodos em subsistemas. Dada a natureza do use case de obtenção de listagem, este foi inserido **package Abastecimento**, onde está armazenada toda a estrutura do armazém. Deste modo, foi criado um método **getListagem** na **interface IAbastecimento** e o seu respetivo diagrama de sequência. Neste diagrama está representada sequencialmente toda a troca de mensagens entre as diferentes classes de modo a retornar uma lista com as diversas ocupações dos corredores.

Fase 3 - Implementação Da Solução

Nesta última fase, implementou-se em código o método **getListagem** modelado na fase anterior. Esta implementação passou por aceder a uma base de dados que armazena os corredores do armazém, através da classe **CorredorDAO**. Nesta BD, está representada toda a informação relacionada com os corredores tal como a sua ocupação que vai ser apresentada ao gestor de forma a este verificar se pode aceitar ao não um pedido de descarga.

CONCLUSÃO

Dada por concluída a terceira fase e última fase do projeto, consideramos favorável elaborar uma análise crítica do trabalho realizado.

Um dos pontos que torna o projeto mais completo e eficiente foi a implementação de uma frota de robots que permite que seja possivel que mais do que um robot transporte paletes da receção até á prateleira. Além disso, consideramos que a interface criada é simples e intuitiva.

Apesar de termos sido consistentes na implementação dos métodos modelados nas fases precedentes, houve certos aspetos que tiveram de ser reformulados ou criados de forma a cumprir com o objetivo da fase III, mais concretamente o método *getLocalizacao*.

Como última fase, achamos que a realização do projeto por fases facilitou a organização do trabalho e desta forma foi possível uma implementação mais simples e eficaz do programa. Consideramos que as duas fases anteriores foram úteis para organização, exposição e discussão de ideias. Deste modo, na fase III apenas foi necessário implementar o que já tinha sido previamente discutido e pensado, facilitando todo o processo. Além disso, apesar de não ter sido possivel explorar todas as funcionalidades definidas em fases anteriores, sentimos que existe abertura para o fazer no futuro.

Deste modo, apesar das dificuldades sentidas, consideramos que conseguimos superá-las e obter um balanço positivo na globalidade do trabalho.