Universidade do Minho

Departamento de Informática

Desenvolvimento de Sistemas de Software

Grupo nº 5

Carlos Preto (a89587)

Maria João Moreira (a89540)

Pedro Veloso (a89557)

Rui Fernandes (a89138)

Índice

1. Introdução

O presente trabalho foi desenvolvido no âmbito da Unidade curricular de Desenvolvimento de Sistemas de Software e pretende apresentar um modelo para gestão de paletes num armazém.

Na primeira fase apresentou-se um modelo de domínios e, esta segunda fase, trata a arquitetura conceitual do problema proposto. Para tal, foi tido em consideração todo o processo de gestão de paletes do armazém, desde a leitura do *QR-Code* das paletes a serem armazenadas até à altura em que estas são colocadas na zona destinada a entregas.

Após enunciar os componentes que farão parte da lógica de negócio do projeto, cada um terá também a sua própria API e arquitetura interna cuja contribuição para o funcionamento do sistema será representada através da modelação comportamental.

Para esta modelação foi escolhido o tipo de diagrama lecionado durante as aulas, o diagrama de sequência do sistema, que nos permite iniciar a análise do sistema propriamente dito com uma visão de mais alto nível. Aqui, cada Use Case representa o sistema como uma "caixa preta", os atores que interagem com o sistema e os eventos gerados pelos mesmos, e por sua vez, a resposta que o sistema dá a cada um desses eventos.

2. Identificação de APIs e subsistemas

Analisando os *uses cases,* definidos na Fase 1, identificaram-se as seguintes funções que são responsáveis pela lógica de negócio a implementar.

Use Case	**Funcionalidades Identificadas**
Consultar inventário do armazém	Listar Paletes e as suas Localizações;
Login	Verificar existência de *Username;* Verificar se *Password* corresponde à correta;
Logout	Realizar *Logout;*
Ler o *QR-Code* da Palete	Registar os dados relativos à entrada da palete no armazém; Encontrar uma prateleira na respetiva zona (refrigerada ou não) para guardar a palete; Mudar o estado da prateleira para "Espera"; Encontrar o robot que está "Livre"; Adicionar palete à *Queue* da zona de receção;
Notificar o *Robot* para transportar palete	Calcular percurso até à Palete; Mudar a disponibilidade do *Robot*;
Notificar a recolha das paletes	Verificar se produto é para entregar; Associar o *robot* ao transporte da palete; Localizar a prateleira onde a palete será colocada; Calcular o novo percurso; Mudar a disponibilidade da prateleira;
Notificar a entrega das paletes	Marcar o *robot* como "Livre"; Verificar que a palete foi levada para a zona de entregas; Retirar a palete da *Queue* de entregas; Verificar se existe prioridade para transportar as paletes na *Queue* relativa às entregas; Verificar se existem paletes na *Queue* relativa à receção; Calcular o percurso para a base do *Robot;* Verificar se a palete está na *Queue* de pedidos em espera; Retirar a prioridade de entrega;
Requisição de Paletes	Verificar a disponibilidade das paletes; Registar as paletes na *Queue* de entregas; Registar as paletes inexistentes na *Queue* de pedidos em espera.

2.1 Identificação dos métodos

Uma vez identificadas as funcionalidades, realizou-se uma transformação das mesmas para métodos capazes de ser implementados em linguagem java. Dividindo-os pelos subsistemas mais convenientes, como apresentado na secção seguinte.

2.2 Junção dos métodos encontrados

Após a identificação da API global da logica de negócio, realizou-se uma análise dos seus métodos de modo a perceber as semelhanças existentes entre eles e, deste modo, dividir a API global em várias API's parciais.

Assim sendo, percebendo que existem funções relacionadas com usuários, com a disponibilidade do armazém e com os *robots,* dividiu-se a API global em 3 parciais, como se pode observar na figura seguinte.

2.2 Diagramas de Componentes/Interfaces

A seguinte figura descreve os componentes do sistema, bem como as dependências entre cada componente. Assim, percebe-se que existe um componente principal denominado ArmazenamentoLN, que implementa as interfaces dos subsistemas SSRobot_Facade, SSUsers_Facade e SSArmazenamento_Facade.

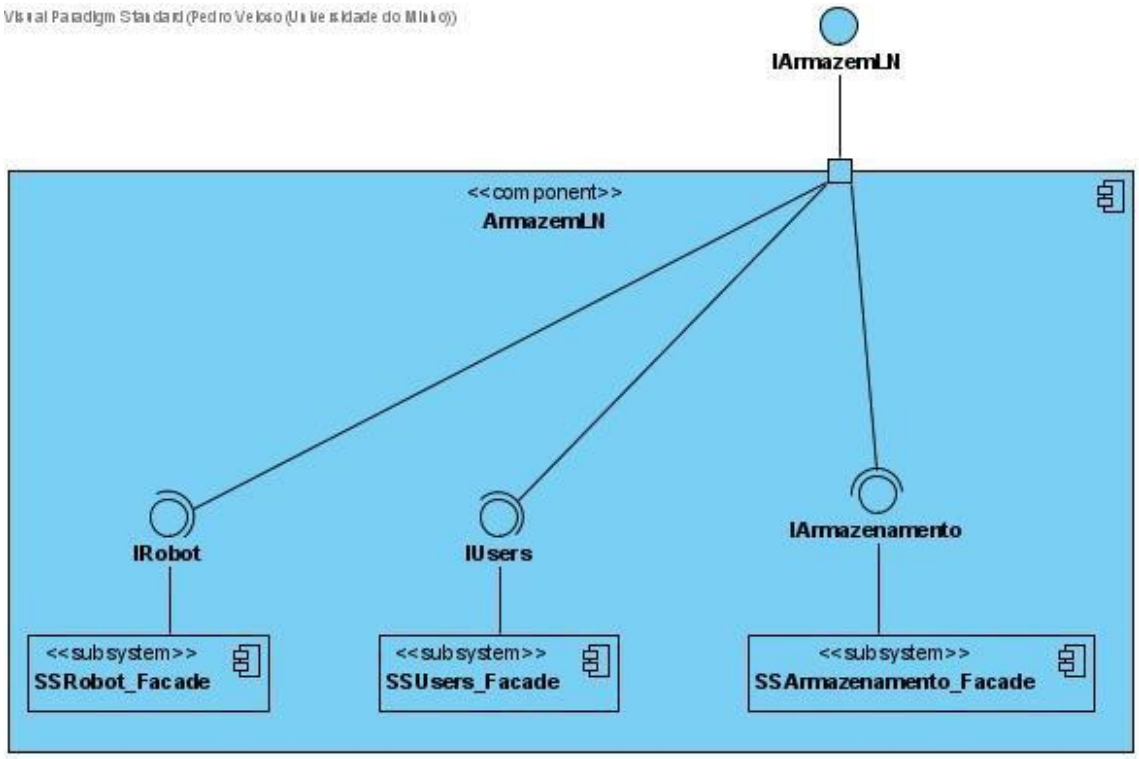

Figura 1: Diagramas de Componentes/Interfaces

3. Análise da estrutura do Sistema

Após a identificação das API global e parciais, identificaram-se os componentes do sistema, contudo falta definir as classes – atributos e métodos – existentes no sistema, bem como a forma que elas se interligam entre si. Para tal, procedeu-se à realização de diagramas de classes para cada componente identificado.

3.1 Diagrama de Classes do SSRobot_Facade

Uma vez que o sistema opera sobre *robots*, foi necessário criar uma forma de os guardar em memória. Adotou-se, então, um *map*, onde a *key* é o código do robot e o *value* é a classe *Robot*. Os *robots* podem ter uma palete associada, contudo a palete no *robot* é apenas um apontador de memoria para a palete, guardada no subsistema SSArmazenamento_Facade. Este

subsistema também apresenta uma classe Mapa, onde constam valores relativos às localizações gerais do armazém e o grafo que representa o mapa do armazém. Representa-se, então, o mapa do armazém como um grafo (Grafo de Listas de Adjacências), onde cada nodo representa um ponto de interesse (diz-se ponto de interesse porque representa uma zona a que o *robot* precisa de aceder). Na próxima figura apresenta-se o diagrama de classes do subsistema.

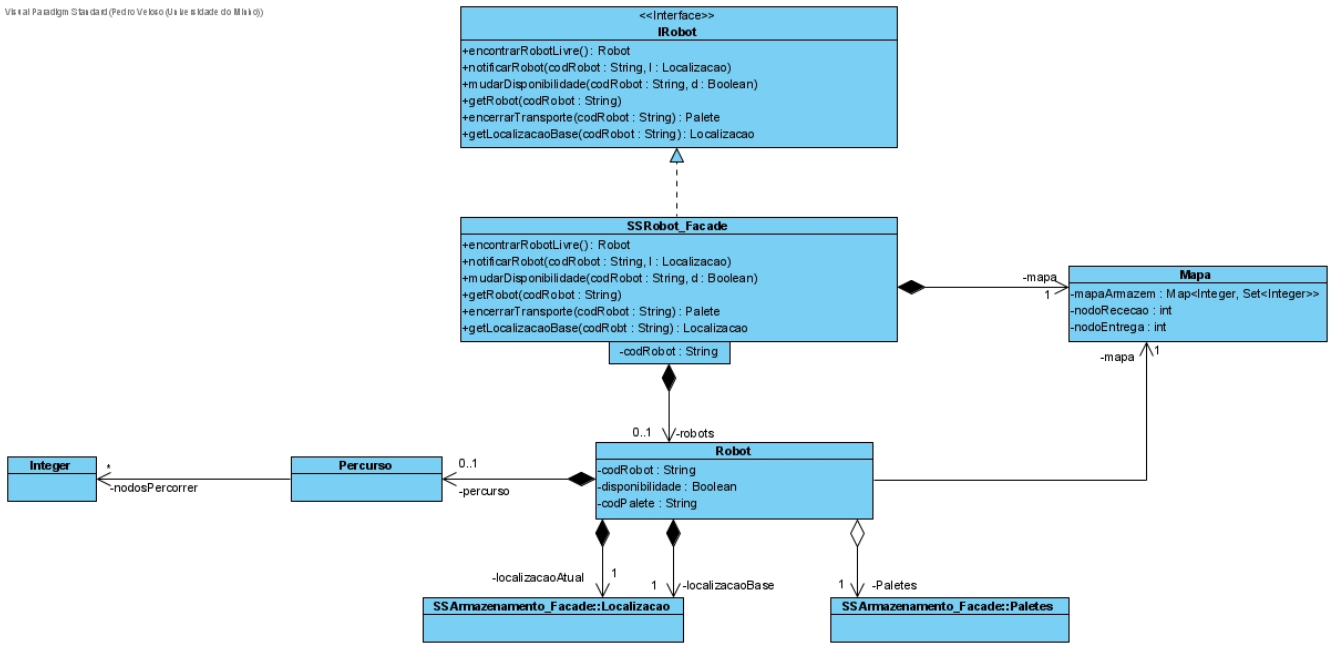

Figura 2: Diagrama de classes do subsistema SSRobot_Facade

3.2 Diagrama de Classes do SSUsers_Facade

Uma vez que no sistema existe a possibilidade de efetuar *login* e *logout*, foi necessário guardar os dados relativos aos usuários. Essa manipulação de dados é feita na SSUsers_Facade, que tem um *map* de *Users*, onde a *key* é o *username* e o *value* é a classe *User.*

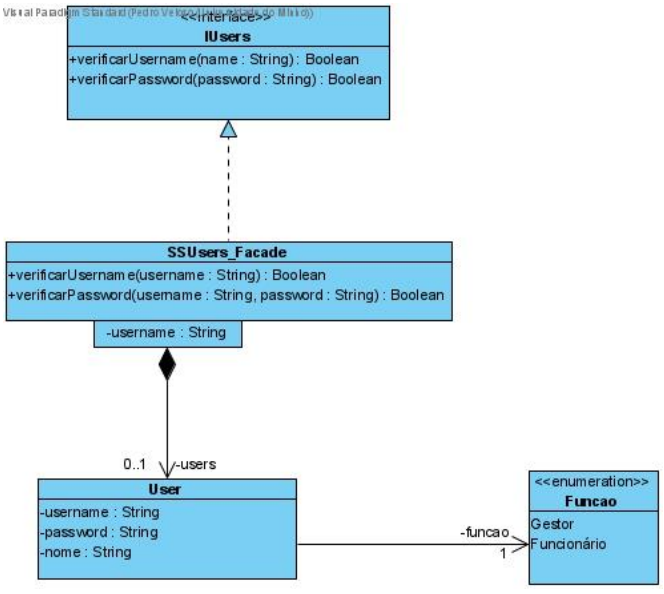

Figura 3: Diagrama de classes do subsistema SSUsers_Facade

Refere-se também a existência de um *enum* que referencia a função que um usuário exerce no armazém. Este *enum* pode servir para diferenciar as funcionalidades que o sistema apresenta a cada utilizador.

3.3 Diagrama de Classes do SSArmazenamento_Facade

Tendo em conta as API's encontradas, decidiu-se que o SSArmazenamento_Facade deve ter um *map* com paletes, onde estão guardadas todas as paletes existentes no armazém. Foi definido, também, uma lista de Movimentos, onde ficam registadas todas as entradas e saídas do armazém. Existem ainda 2 zonas, uma refrigerada e uma não refrigerada, constituídas por um *map* de prateleiras. É de notar que foram criadas 2 listas de paletes, uma referente às paletes que estão para entrega e outra referente às que estão na receção, sem um *robot* associado. Na seguinte figura encontra-se o diagrama de classes do subsistema.

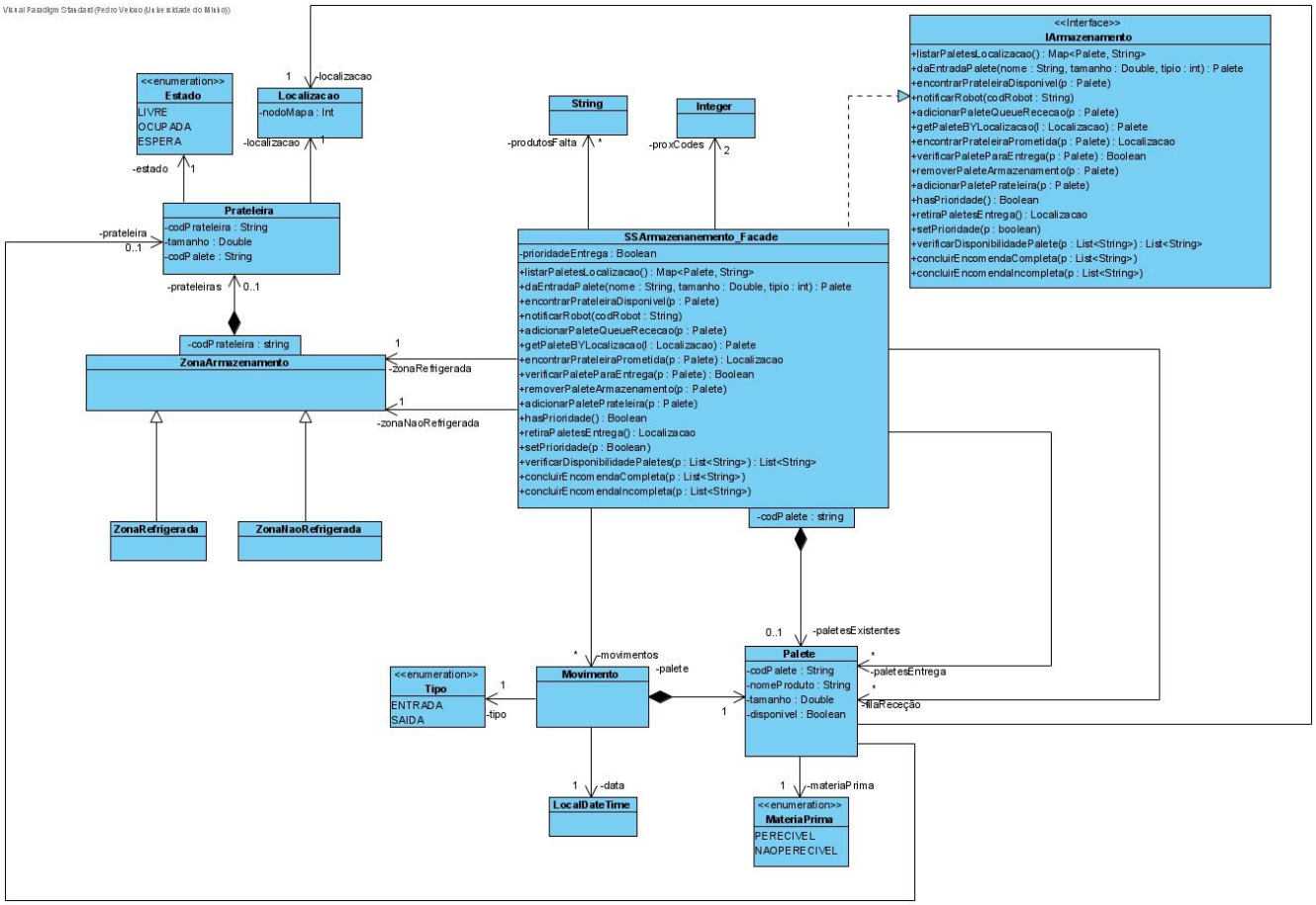

Figura 4:Diagrama de classes do subsistema SSArmazenamento_Facade

Por fim adotaram-se 3 *enums*, para diferenciar diferentes os estados que as classes podem ter. Pois, por exemplo, a palete pode ser constituída por matérias-primas perecíveis ou não perecíveis.

4. Diagrama de Sequências

4.1 Consultar inventário do armazém

Uma vez que um gestor pode solicitar ao sistema a informação de onde se encontram as paletes, identificou-se um *Use Case* (denominado por Consultar inventario do armazém) durante a Fase 1 do projeto que daria suporte à implementação da funcionalidade. Sabendo que o gestor ao expressar tal intenção não necessita de fornecer nenhuma informação, o método emitido tem então zero parâmetros de *input* e recebe como *output* um *map* onde a chave irá ser a palete e a *key* será a sua localização em formato *String*. Assim, quando uma palete se encontrar na receção a sua localização terá como valor "Receção", quando se encontrar num *robot* terá como valor "Robot" e quando se encontrar na prateleira terá como valor o código da prateleira. Evitou-se enviar como *key* do *map output* uma localização física, uma vez que tanto a receção como o *robot* são localizações especiais. A localização da receção é única e, como tal, ao receber a informação que se encontra na receção o gestor tem de imediato uma localização especifica e precisa. Por outro lado, a localização de um *robot* não é constante, uma vez que o *robot* iria-se encontrar em movimento.

Assim a classe ArmazenamentoLN ao intercetar o pedido descrito, terá de o reencaminhar ao subsistema SSArmazenameto_Facade, de modo a obter o resultado.

Por sua vez o SSArmazenamento_Facade, tendo registado todas as paletes, terá de iterar sobre todas as paletes existentes e perceber se a sua localização corresponde à Receção, Robot ou a uma Prateleira. Este processo descreve-se no seguinte diagrama.

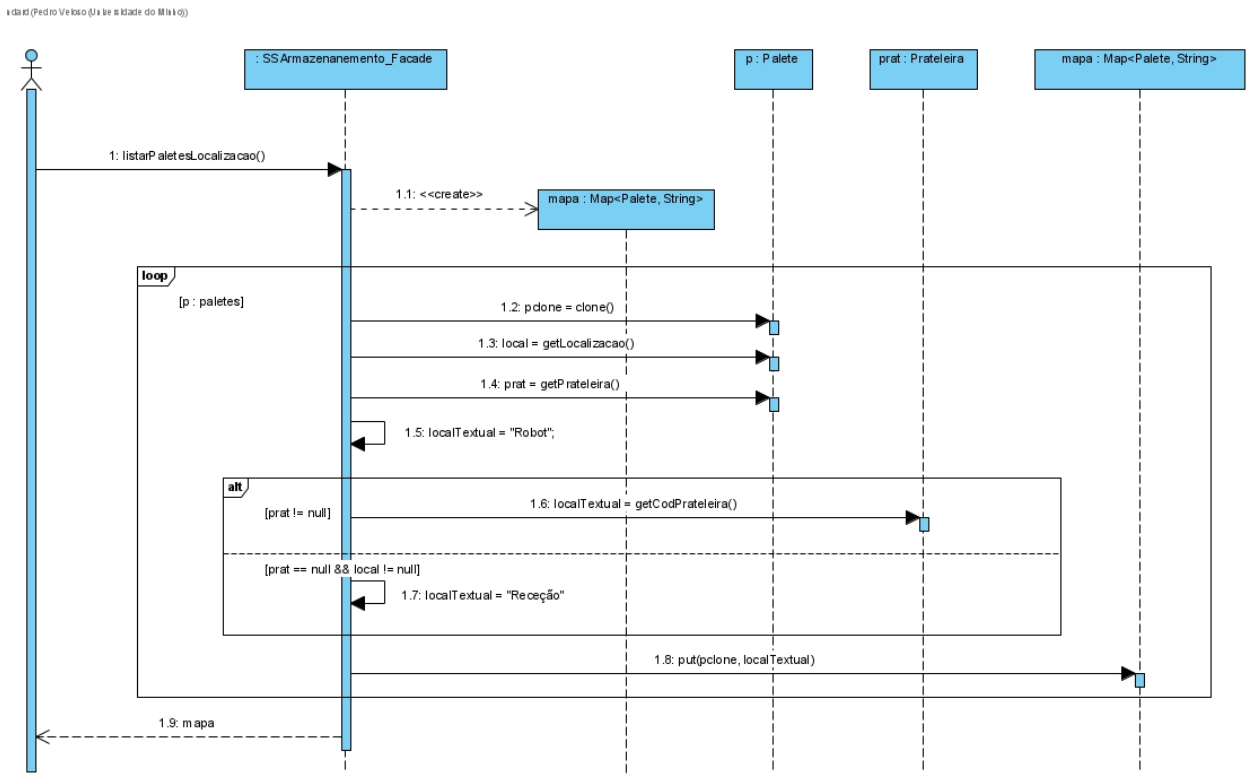

Figura 5: Diagrama de Sequência - ListarPaletesLocalizacao

4.2 Login

Para um gestor poder utilizar o sistema, é necessário estar registado. Desta forma, sempre que um gestor inicia o sistema é necessário efetuar *login*. Para tal, é necessário o gestor fornecer dados como o seu *username* ou a sua *password.* Uma vez fornecidos esses dados o ArmazenamentoLN terá de verificar se o *username* existe no sistema. Caso não exista o sistema deve lançar uma exceção a avisar que o *username* não existe. Após a verificação do *username* é necessário verificar se a password é a correta para esse usuário, caso não seja é lançada outra exceção. Contudo, como o ArmazenamentoLN não conhece nenhum utilizador, deve invocar métodos conhecidos e analisar os seus retornos.

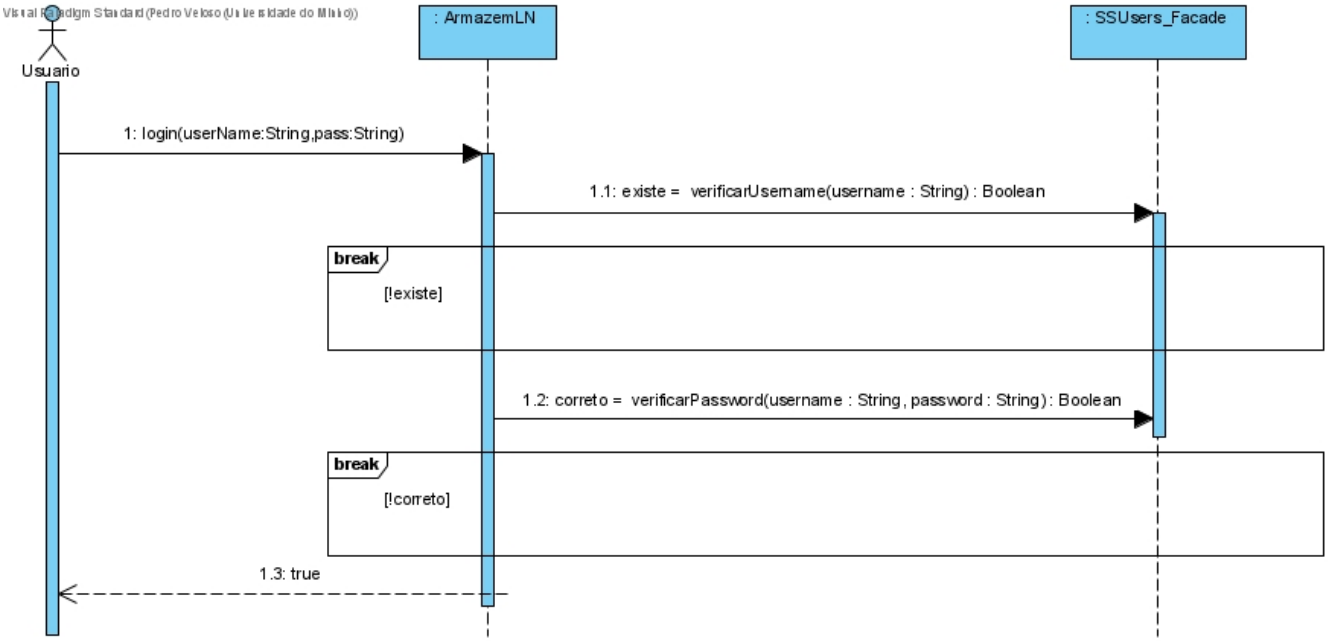

Figura 6: Diagrama de Sequência - Login

Tais métodos estão definidos no SSUsers_Facade e devem ter o seguinte comportamento identificado na primeira figura (método responsável por verificar se a password está correta) e na segunda figura (método responsável por verificar se o *username* existe no sistema).

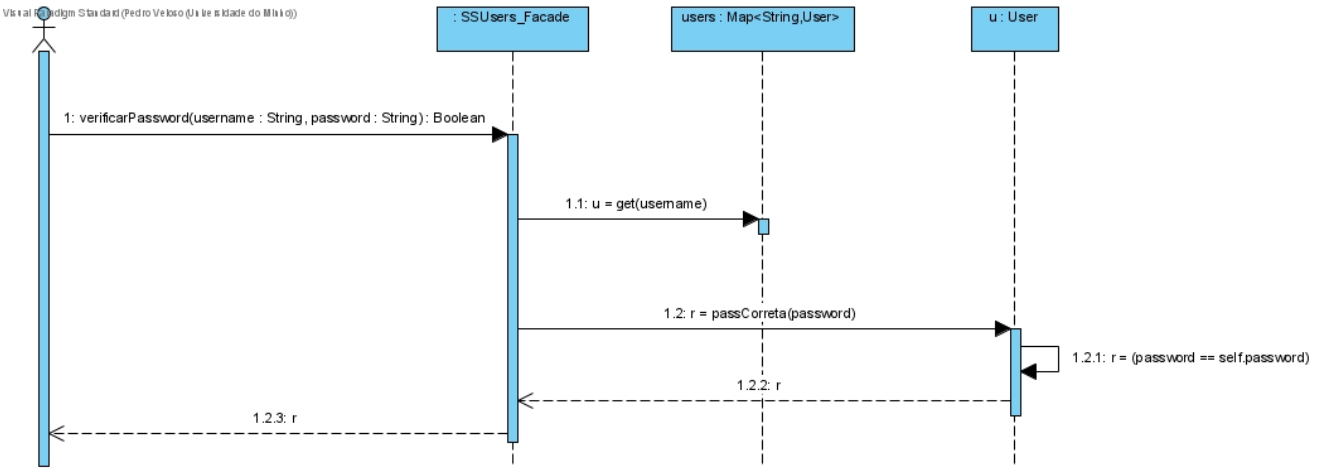

Figura 7: Diagrama de Sequência - VerificarPassword

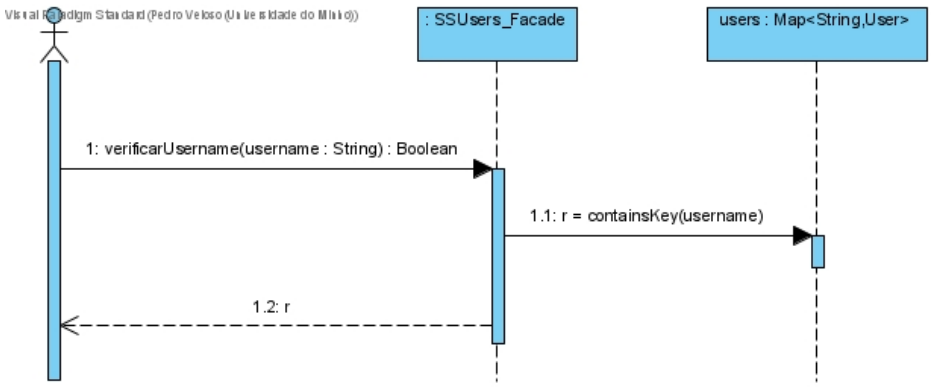

Figura 8: Diagrama de Sequência - VerificarUsername

4.4 Ler o QR-Code da Palete

Sempre que uma palete entra no armazém o seu *QR-Code* é lido pelo leitor que notifica o sistema, dando ordem para efetuar o processo de entrada da palete. Uma vez que ao ler o *QR- Code* o leitor recebe a informação necessária – nome, tamanho, tipo da matéria-prima (0-Perecível, 1-Não Perecível) – pode invocar o método responsável por iniciar o processo. Durante este processo de leitura do QR-Code da Palete existem algumas etapas da responsabilidade do ArmazenamentoLN que devem ser efetuadas, como:

- Dar a entrada da palete no sistema;
- Encontrar prateleira disponível;
- Encontrar um *robot* livre;
- Consoante o sucesso dessa operação anterior, notificar o robot para entrega ou adicionar a palete à *queue* de receção,

Cada uma destas etapas é um método definido nos subsistemas. Na figura seguinte verifica-se como o ArmazenamentoLN deve proceder.

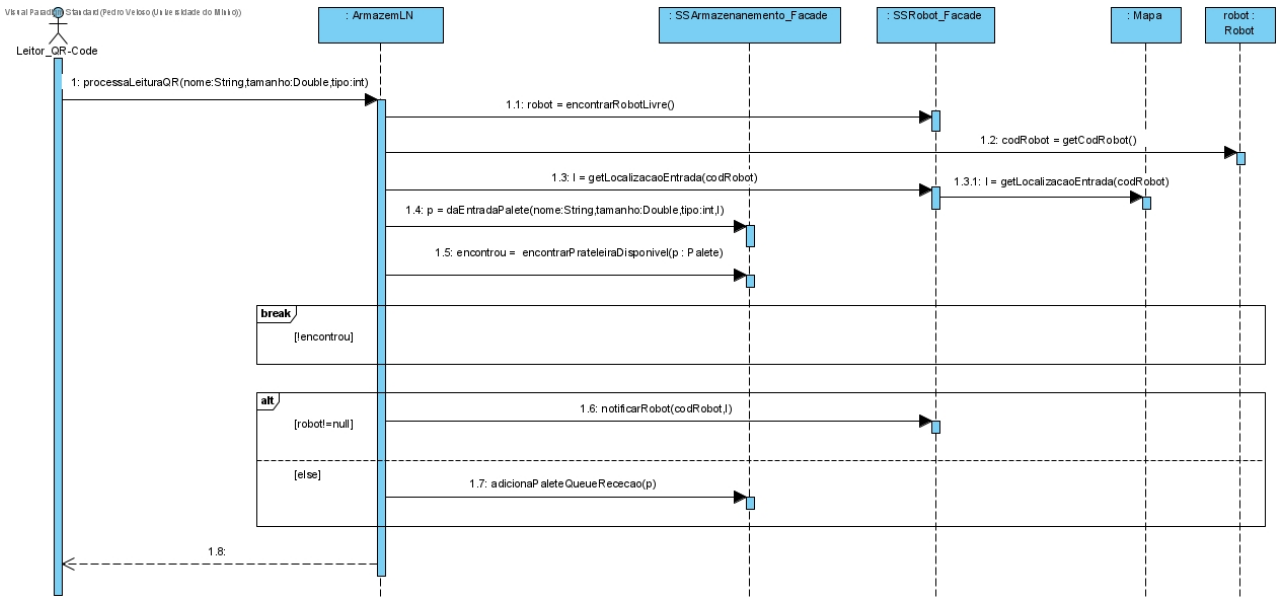

Figura 9: Diagrama de Sequência - PrecessaLeituraQR

A primeira etapa é responsabilidade do SSArmazenamento_Facade e define-se em criar a palete e introduzi-la nas paletes existentes, bem como criar um movimento, com a data atual (em java utiliza-se o LocalDateTime.now), com o *enum* Tipo igual a ENTRADA (declara-se em java como Tipo.ENTRADA) e com um clone da palete.

Na criação do movimento é importante que a palete seja um clone pois, quando a palete sair do armazém e for apagada das paletes existentes é necessário que continue presente no histórico, pois pode existir interesse em saber, por exemplo, a data de entrada e saída de uma palete especifica. É também importante referir que cada palete tem como identificador um código único, calculado pela inicial (p-Perecível ou n-Não Perecível) mais o próximo número do tipo da palete.

Esta etapa define-se no seguinte diagrama de sequência:

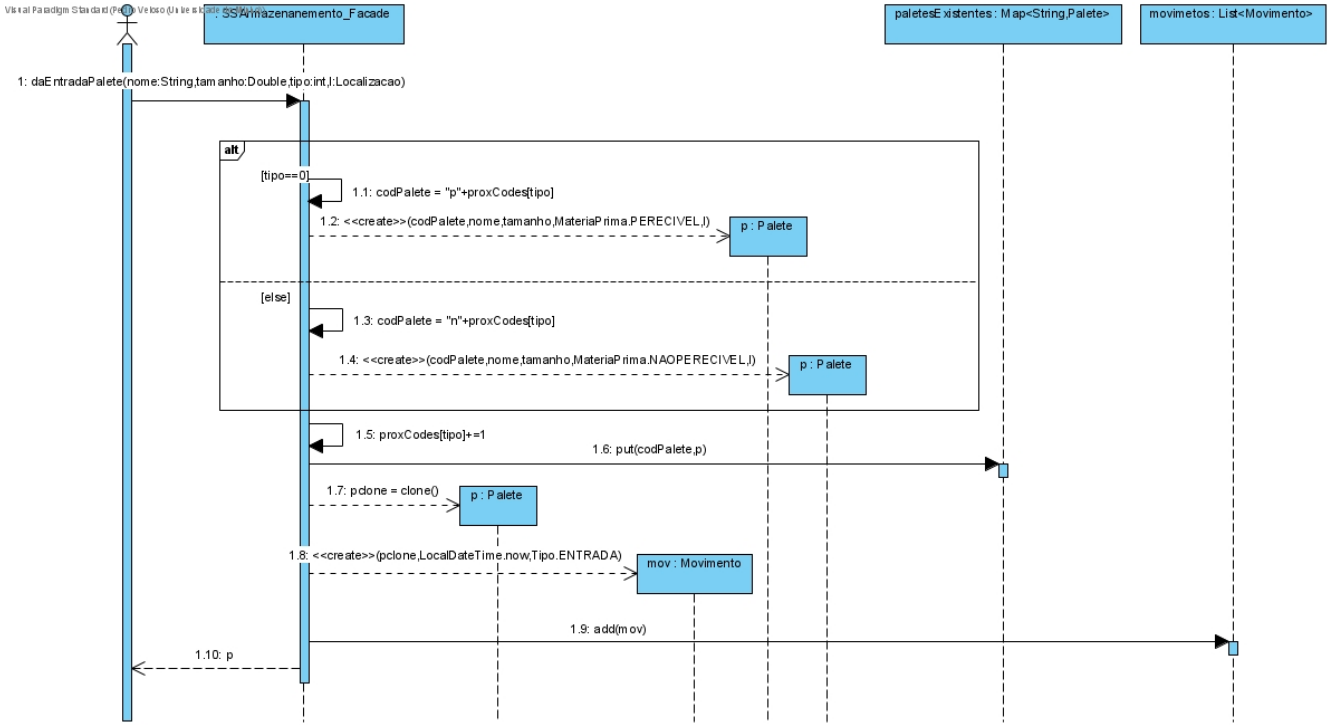

Figura 10: Diagrama de Sequência - DaEntradaPalete

Na segunda etapa, para encontrar uma prateleira disponível é necessário percorrer as prateleiras de modo a perceber onde a palete, dada como *input* se encaixa. Deve-se ter em conta o tipo da zona, se é uma zona refrigerada ou não, e o tamanho da palete. Uma vez encontrada a prateleira define-se o estado da prateleira como “Espera” (mais uma vez deve-se ter em conta que o estado é um *enum*) e coloca-se o código da palete na prateleira, para mais tarde se poder identificar a prateleira prometida à palete.

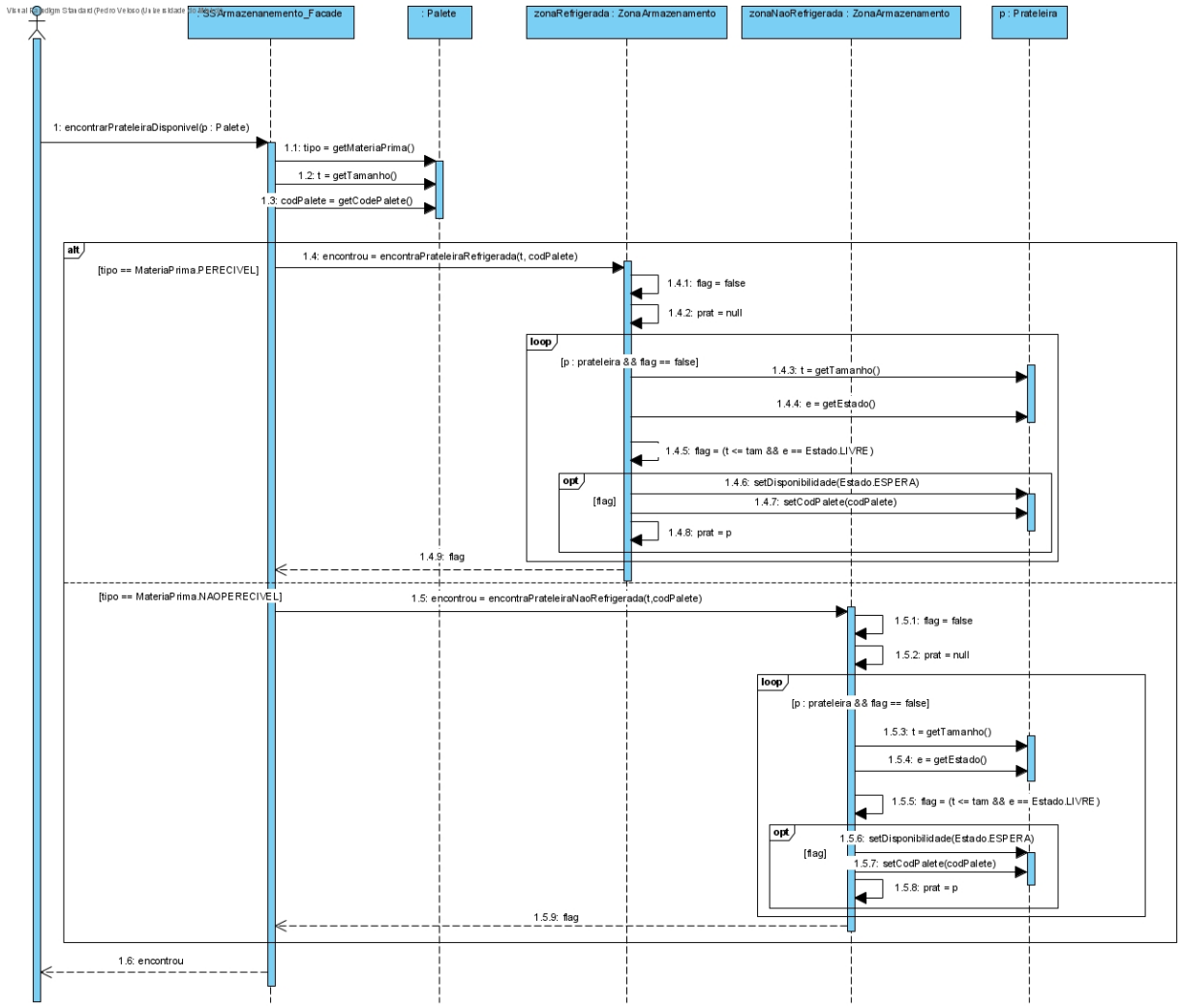

Figura 11: Diagrama de Sequência - EncontraPrateleiraDisponivel

Na penúltima etapa desta operação deve-se encontrar um *robot* disponível, para transportar a palete, no armazém. Para tal, basta iterar sobre os *robots* e verificar qual tem a bandeira disponível a verdadeiro, como se pode observar na figura seguinte.

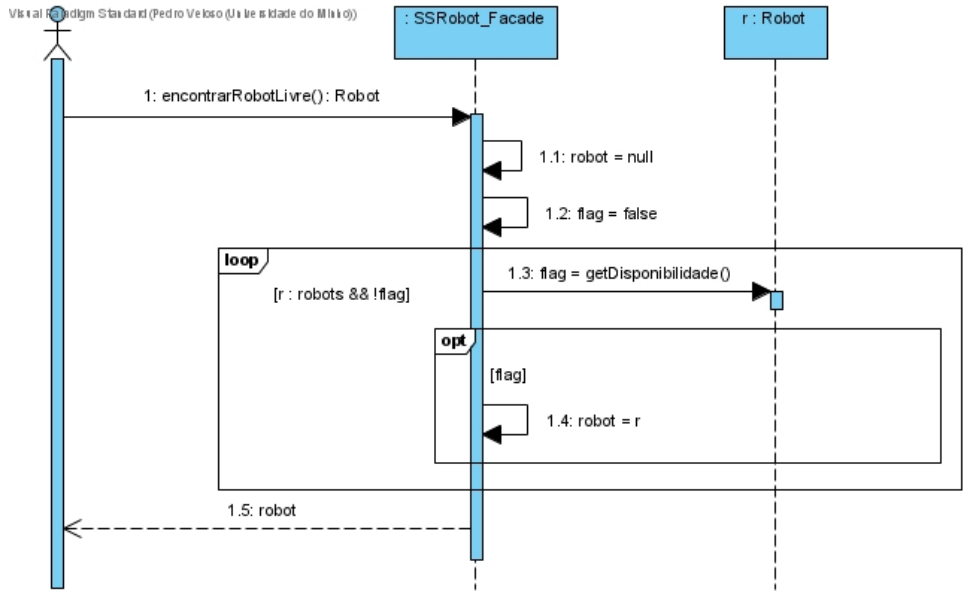

Figura 12: Diagrama de Sequência - EncontrarRobotLivre

Por fim, é realizada a análise desta última operação, e caso exista um *robot* livre notifica- se o *robot*, se não houver *robots* livres adiciona-se a palete na *queue* de receção, sendo essa operação uma adição simples a uma lista.

4.5 Notificar Robot para transportar palete

Um processo que pode ocorrer no sistema é a notificação de *robots* para transportar uma palete. Assim é necessário saber qual o *robot* a notificar e a localização a deslocar-se. Durante esta operação define-se o *robot* como indisponível, calcula-se e atribui-se ao *robot* o percurso a realizar, como se pode ver no diagrama de sequência seguinte:

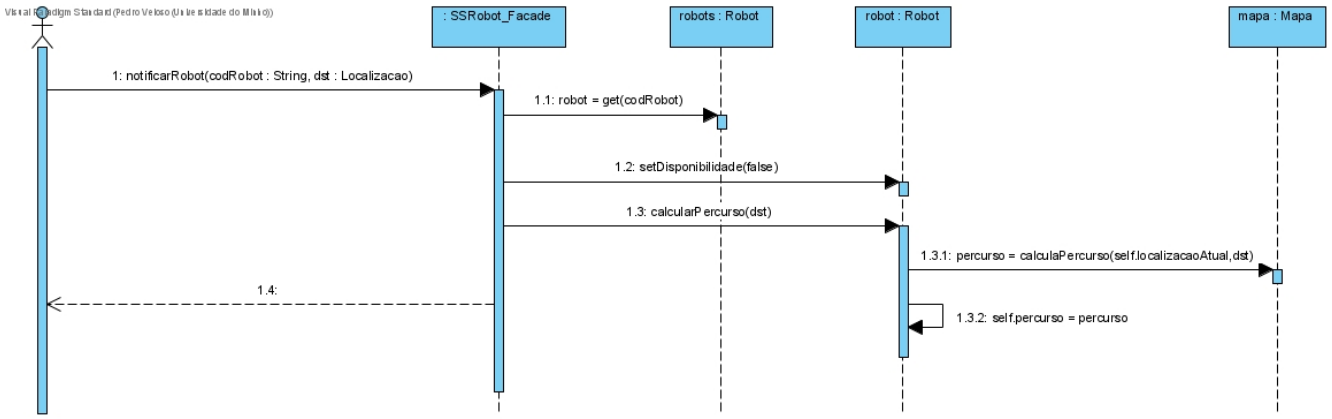

Figura 13: Diagrama de Sequência - NotificarRobot

Para mudar a disponibilidade do robot realiza-se as etapas presentes no seguinte diagrama de sequência:

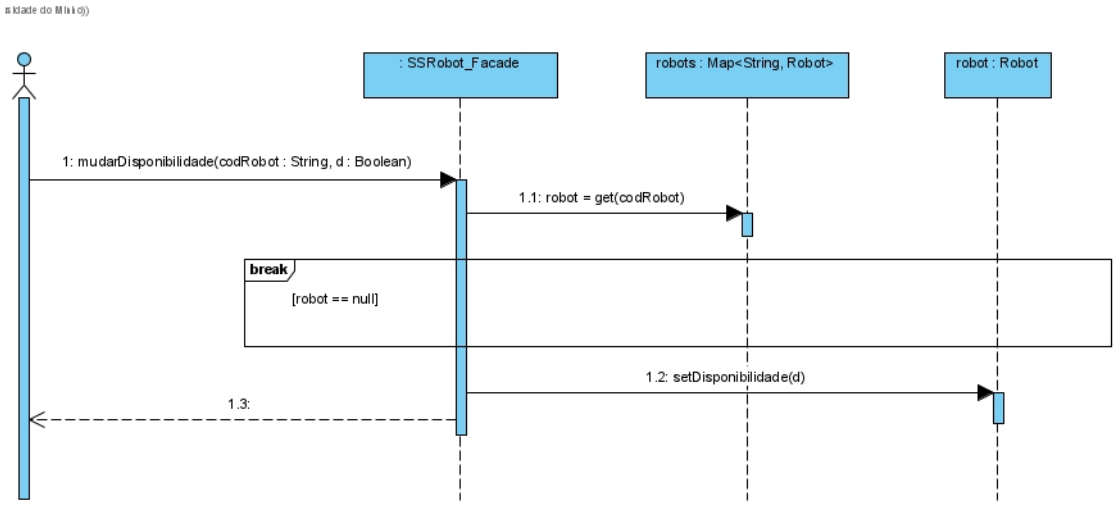

Figura 14: Diagrama de Sequência - MudarDisponibilidade

Para calcular o percurso, uma vez que o mapa do armazém é representado por um grafo, utilizou-se um algoritmo, denominado por *Dijkstra,* que permite encontrar um caminho mais curto num grafo. O algoritmo baseia-se nos passos demostrados na próxima figura.

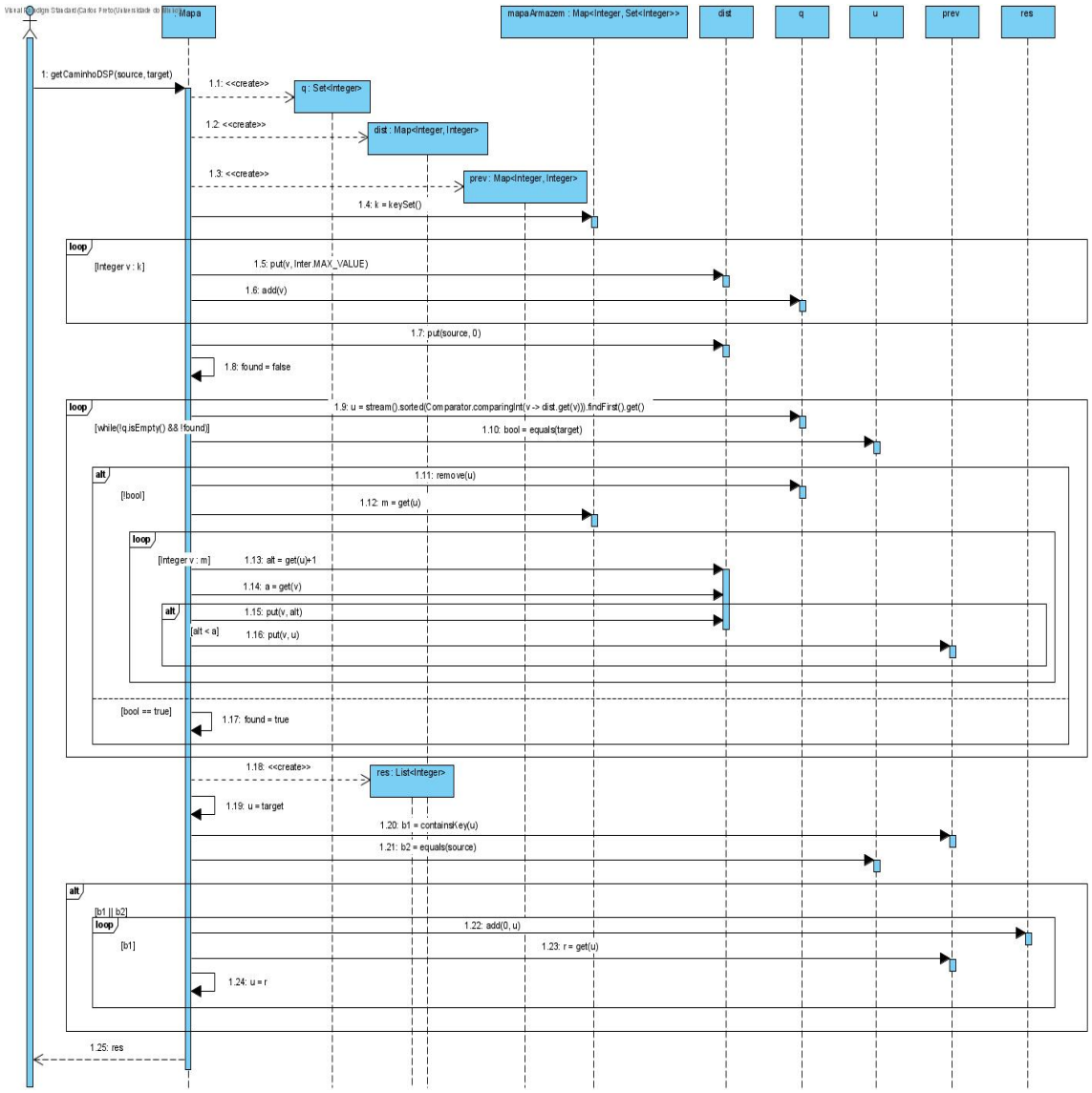

Figura 15: Diagrama de Sequência - Algoritmo de Dijkstra

Admite-se que o *robot* tem capacidade para se deslocar nas arestas do grafo de forma autónoma, conseguindo desviar-se de outros robots, ou de obstáculos que apareçam, bastando assim saber os próximos nodos a que tem de se deslocar.

4.6 Notificar recolha das paletes

Quando um robot recolhe uma palete, segundo o seu *Use Case*, é necessário associar a palete ao *robot* e verificar se a mesma é para entrega ou para as prateleiras. Contudo, para tal, é necessário saber qual é a palete que o *robot* recolheu. Assim, deve-se realizar uma busca nas paletes existentes e procurar por uma que tenha a mesma localização que o *robot*. Quando encontrada, pode-se então associar a palete ao *robot* e verificar se a palete se encontra na *queue* de entrega. Consoante o resultado encontra-se a próxima localização e notifica-se o robot para transporte.

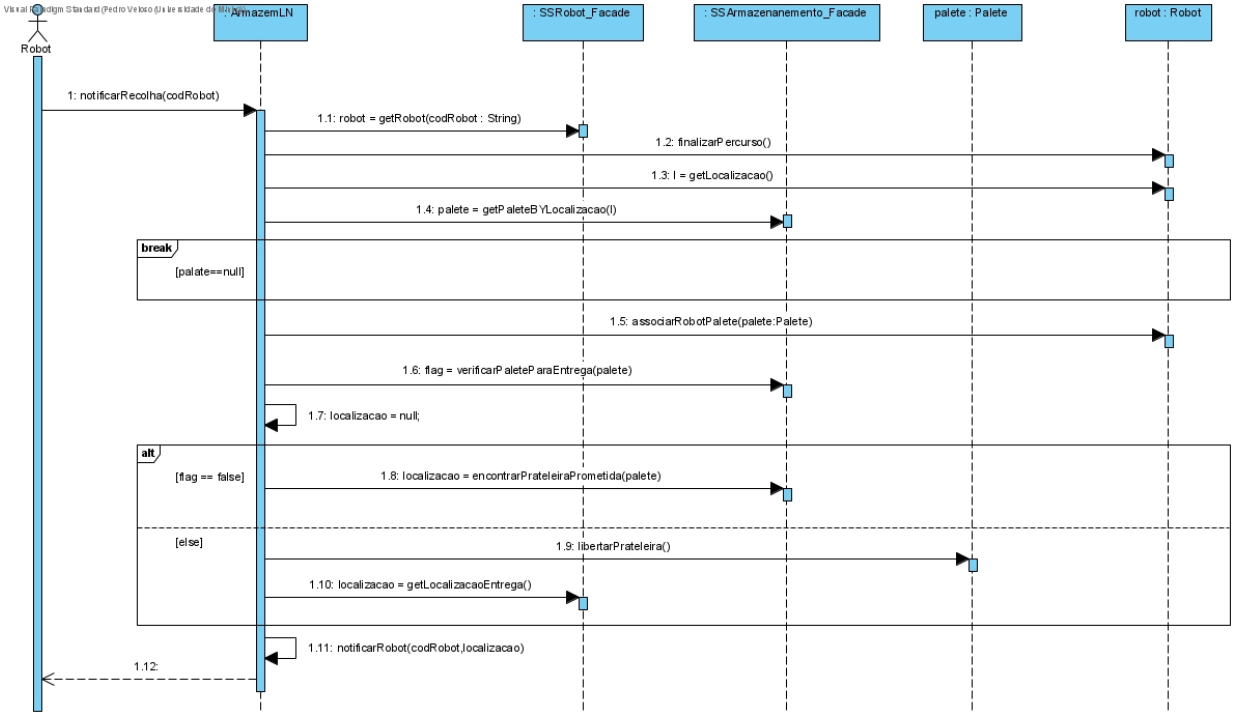

Figura 16: Diagrama de Sequência - NotificarRecolha

De modo a obter a palete através da sua localização é necessário percorrer as paletes existentes e verificar a que tem a mesma localização que a do *robot*, tal como indica o seguinte diagrama:

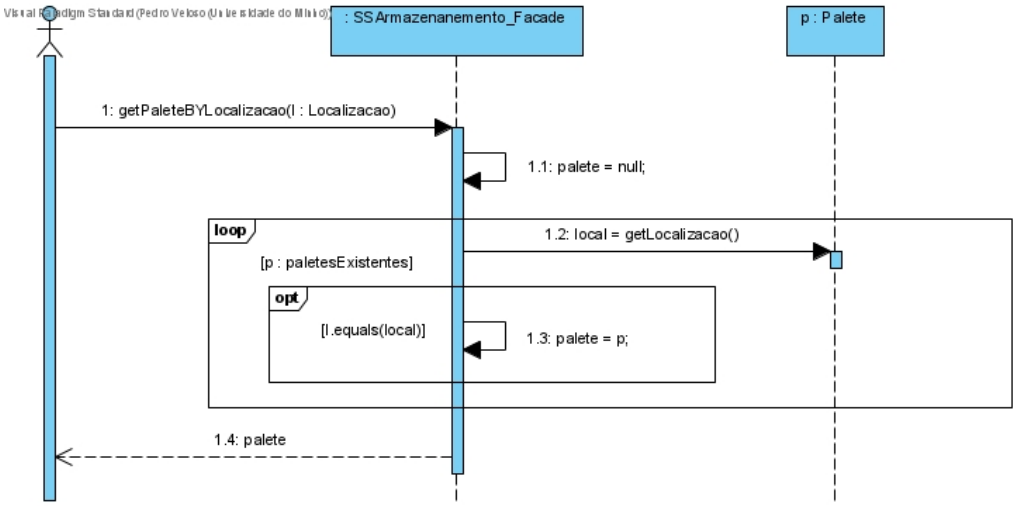

Figura 17: Diagrama de Sequência - GetPaleteBYLocalizacao

No processo de associar o *robot* à palete realiza-se os passos referidos no diagrama da figura seguinte.

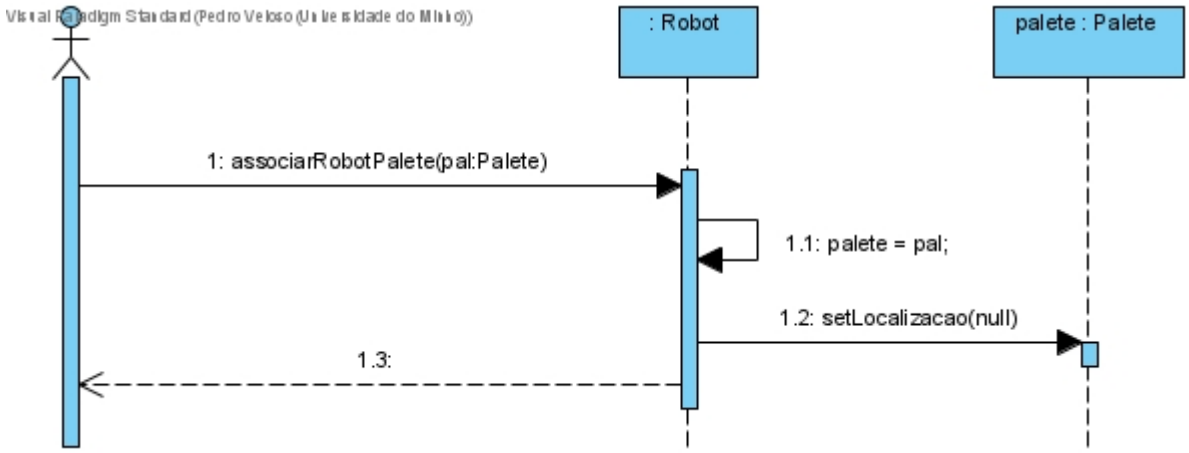

Figura 18: Diagrama de Sequência - AssociarRobotPalete

É necessário colocar a localização da palete a *null*, pois uma vez que esta se encontra em movimento no *robot*, não existe uma localização estática para colocar. Assim pode-se deduzir que quando uma palete não apresentar uma localização é porque se encontra num *robot*.

Após este passo, o *Use Case* diz-nos que ser feita a verificação de que palete é para entrega, logo segue se os seguintes passos:

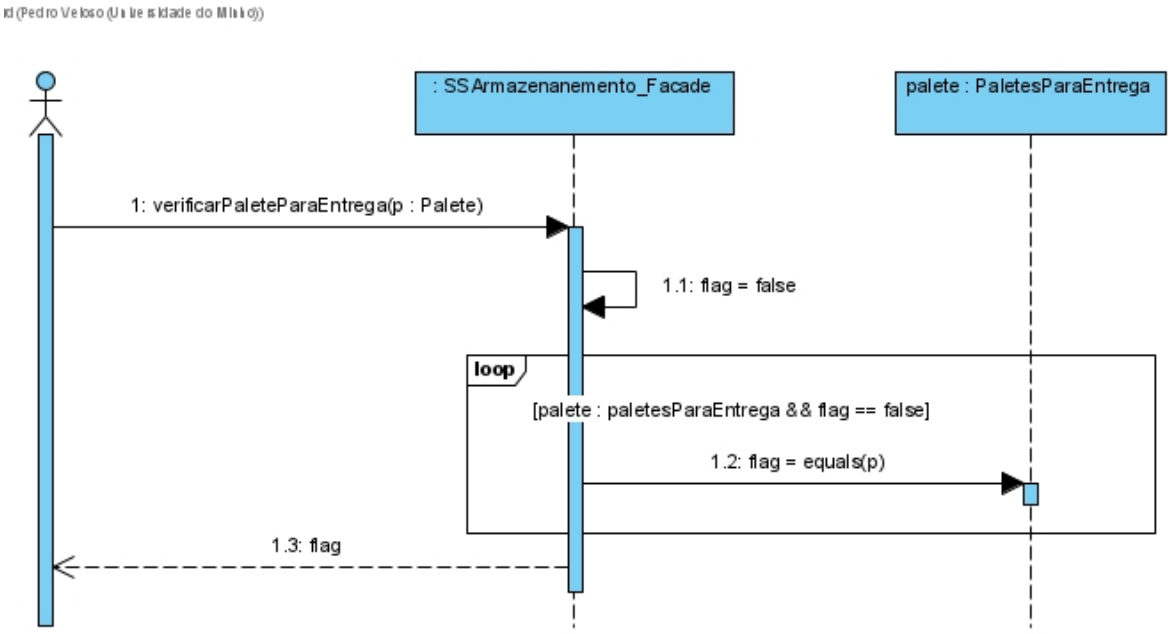

Figura 19: Diagrama de Sequência - VerificarPaleteParaEntrega

Analisando o resultado do método acima, escolhe-se o conjunto de operações a realizar.

Caso seja uma palete para entrega é realizada a libertação da prateleira e define-se a localização do destino como a da zona de entrega (é necessário pedir ao SSRobot_Facade para ir ao seu mapa e retornar à localização). A libertação da prateleira é feita da seguinte forma:

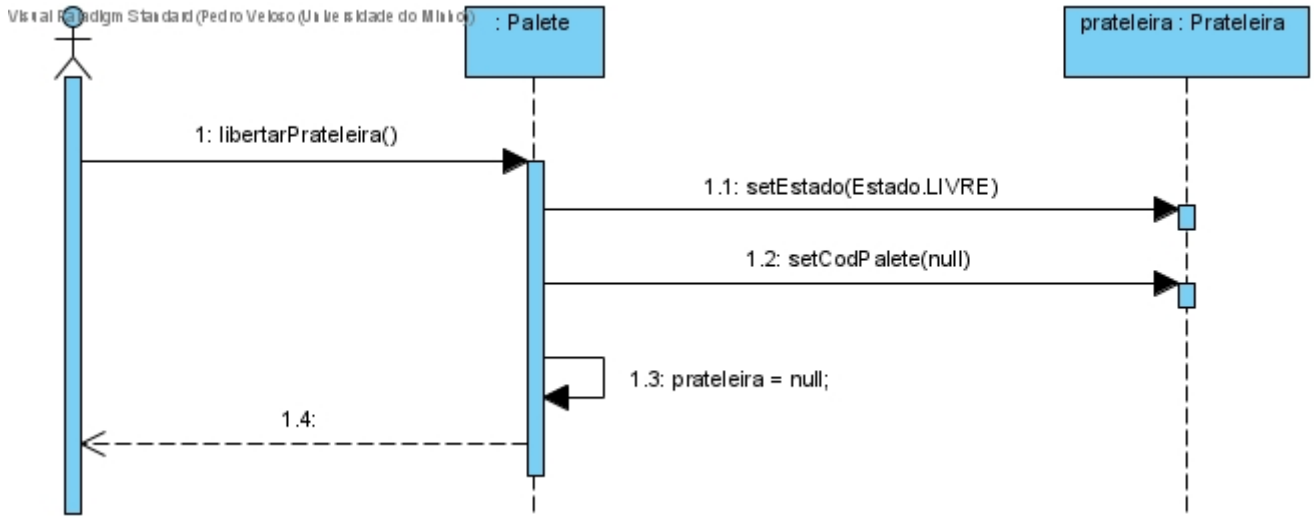

Figura 20: Diagrama de Sequência - LibertarPrateleira

Caso a palete seja para guardar numa prateleira é necessário encontrar a prateleira que lhe foi prometida aquando da sua chegada ao armazém. Este processo é feito da seguinte forma:

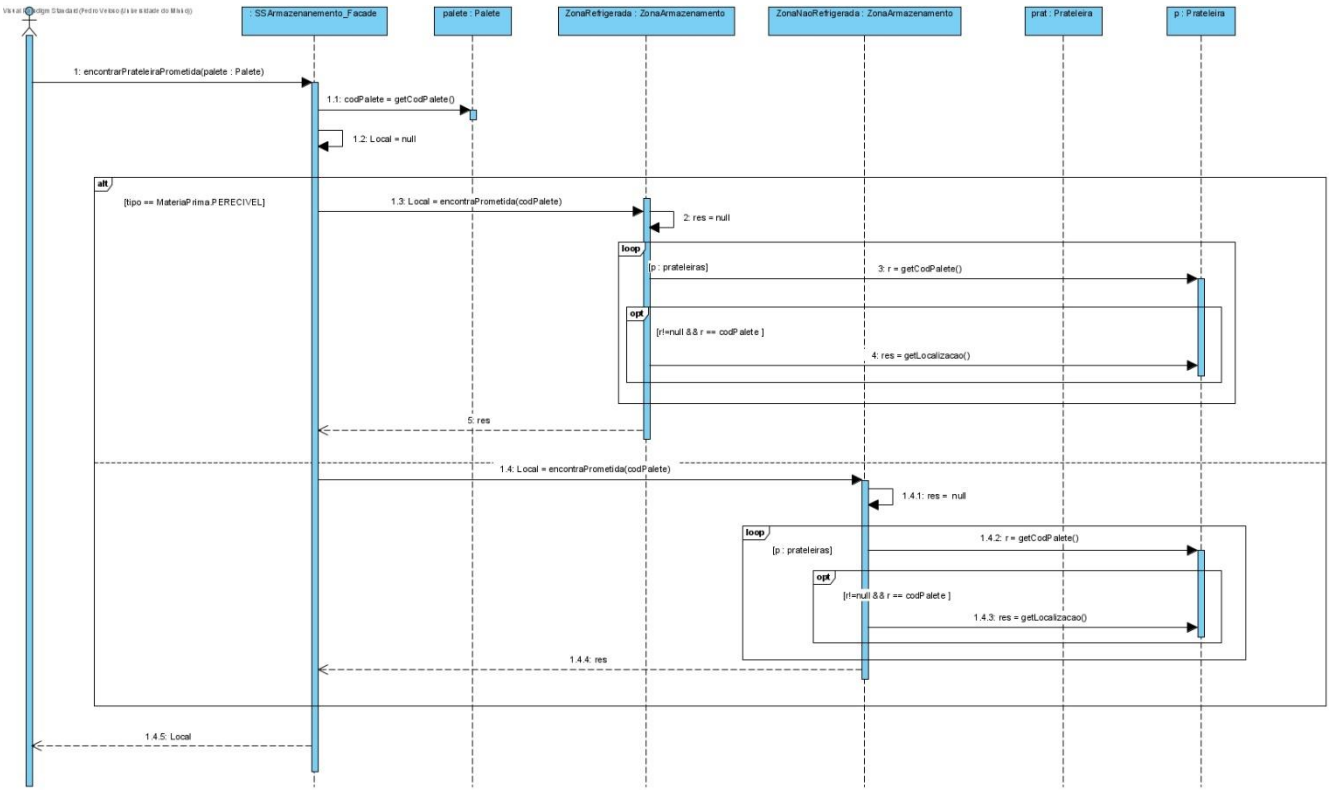

Figura 21: Diagrama de Sequência - EncontrarPrateleiraPrometida

Após isto é calculado o novo percurso atribuindo-o ao *robot*.

4.7 Notificar entrega das paletes

Sempre que um *robot* entrega uma palete, seja na zona de entrega ou na prateleira, é realizada a notificação da entrega. Segundo o *Use Case* seguem-se os seguintes passos:

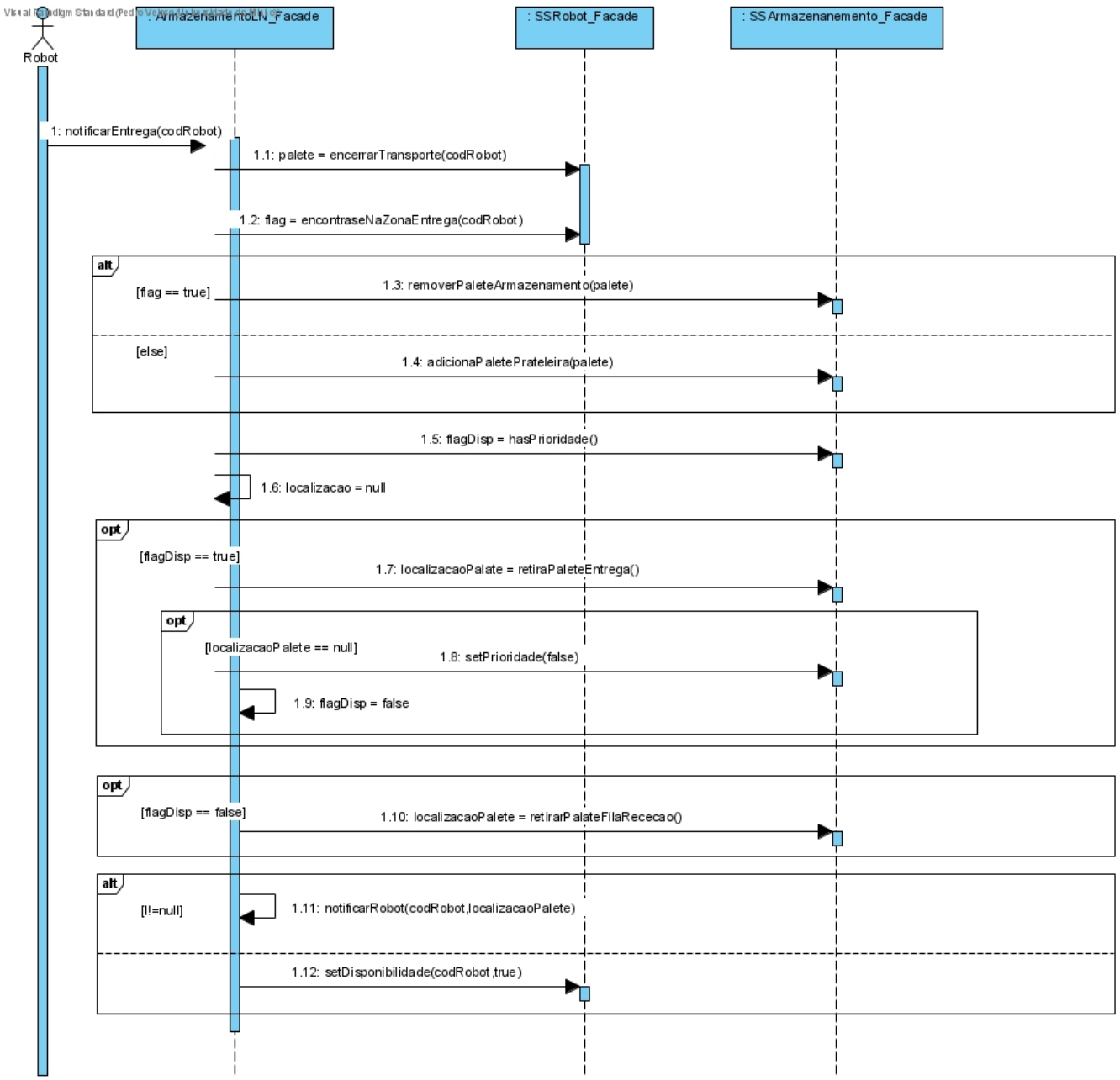

Figura 22: Diagrama de Sequência - NotificarEntrega

Primeiramente é encerrado o transporte do *robot*, colocando o *robot* como livre e retirando a palete ao *robot*.

A palete retirada deve ser retornada ao ArmazenamentoLN de modo a trabalhar com ela no decorrer da operação.

Assim temos os seguintes diagramas:

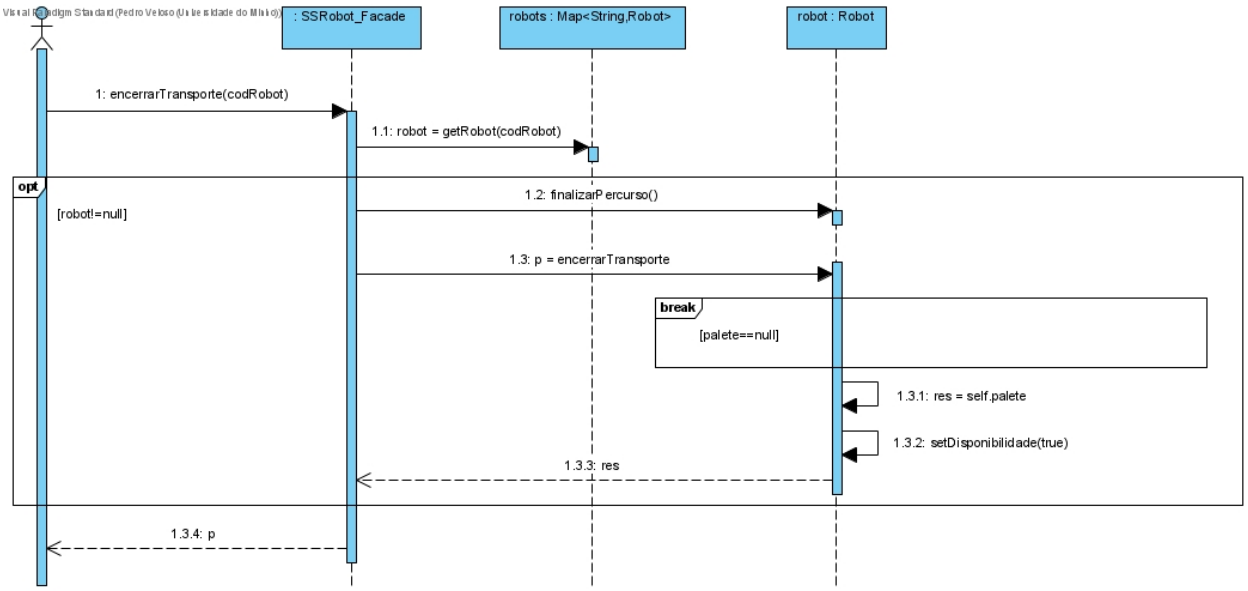

Figura 23: Diagrama de Sequência – EncerrarTransporte

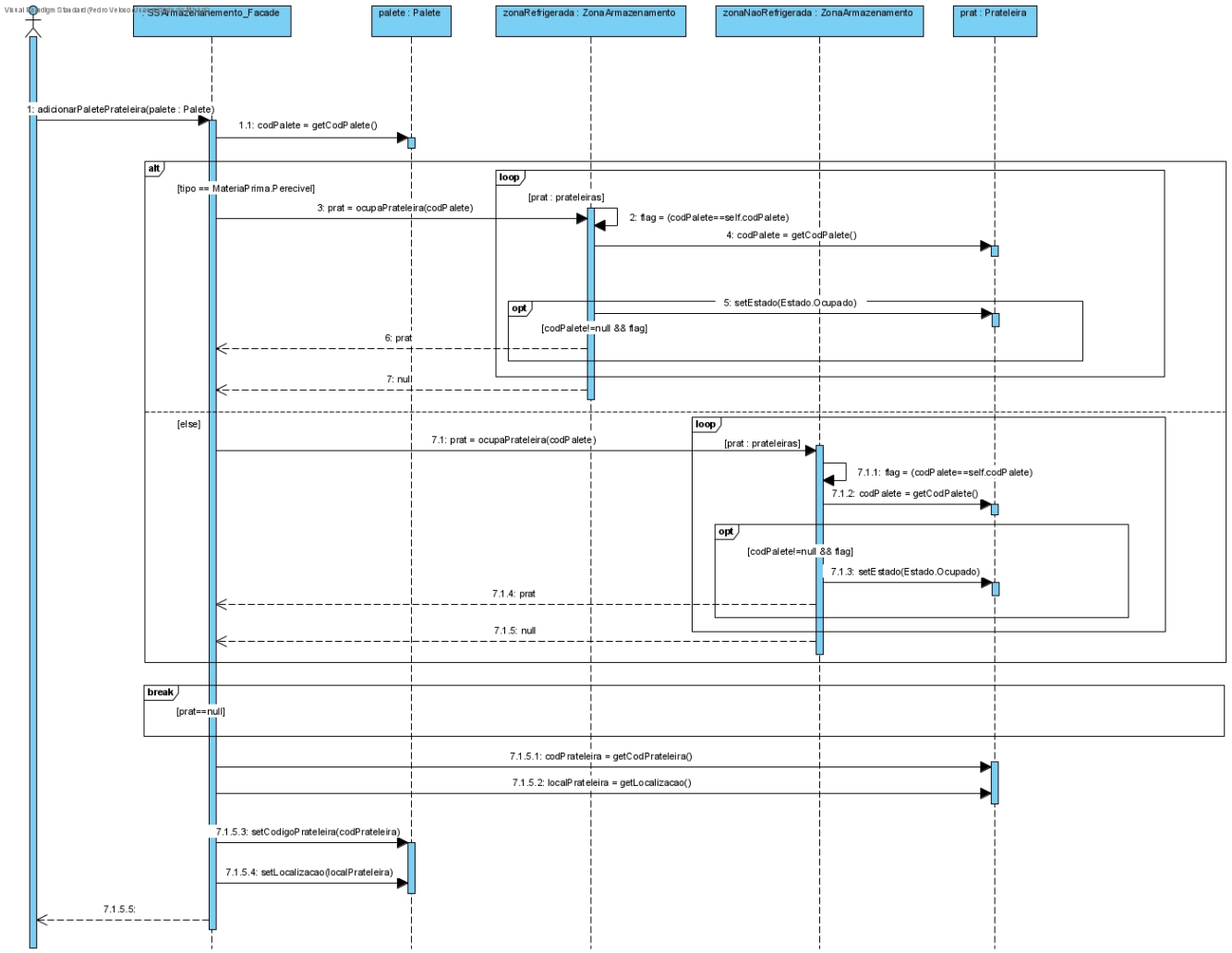

Figura 24: Diagrama de Sequência -AdicionarPaletePrateleira

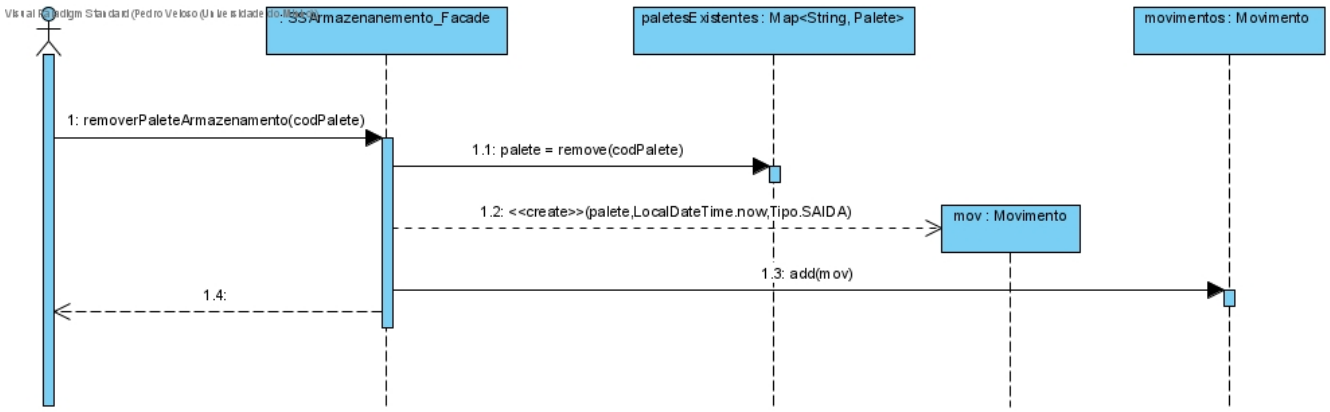

Figura 25: Diagrama de Sequência - RemoverPaleteArmazenamento

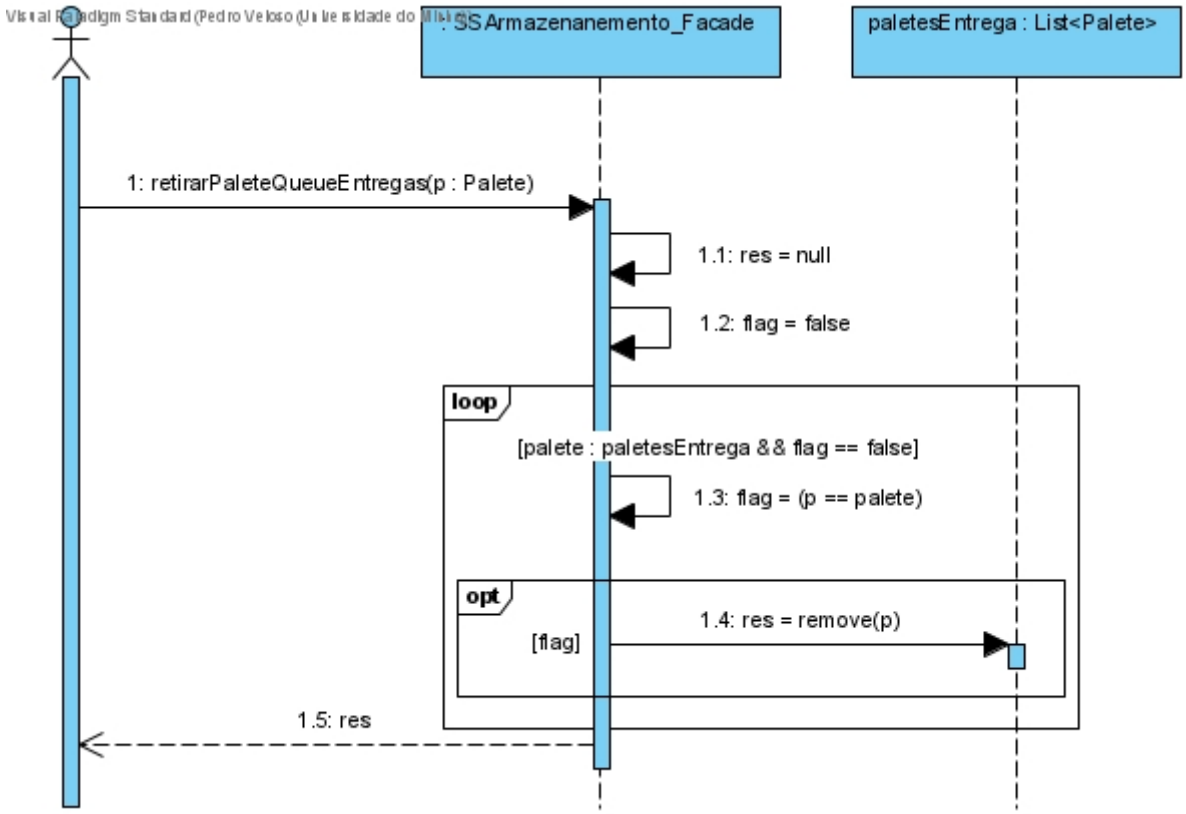

Figura 26: Diagrama de Sequência - RetirarPaleteQueueEntrega

Como se pode observar pelos diagramas de sequência acima, durante o processo de notificação de entrega é necessário distinguir a possibilidade de o *robot* entregar uma palete a uma prateleira (sendo necessário associar a palete à prateleira) da possibilidade de o *robot* entregar a palete na zona de entregas (sendo necessário removê-la do armazém e atualizar o histórico de movimentos). No fim é necessário verificar a próxima palete a transportar. Se existe prioridade de entrega, retira-se então uma palete da fila de entregas (caso a palete retirada seja nula, remove-se a prioridade de entrega e ignora-se a prioridade observada anteriormente). Se não existir prioridade procura-se por paletes na fila de entrada, que necessitam de ser guardadas nas prateleiras.

4.8 Requisição de Paletes

Analisando o *Use Case* referente, reconhece-se a possibilidade de os produtos pedidos não existirem todos no armazém, sendo assim necessária uma outra interação por parte do servidor da produção, confirmando se aceita pedidos incompletos ou não. Assim, o primeiro passo a realizar é a verificação da existência de todos os produtos. Com a noção de que produtos estão em falta, é necessário comunicá-los ao servidor, que por sua vez responde se aceita a encomenda incompleta ou não. Caso todos os produtos estejam disponíveis a encomenda é aceite de imediato, como se pode observar no diagrama seguinte.

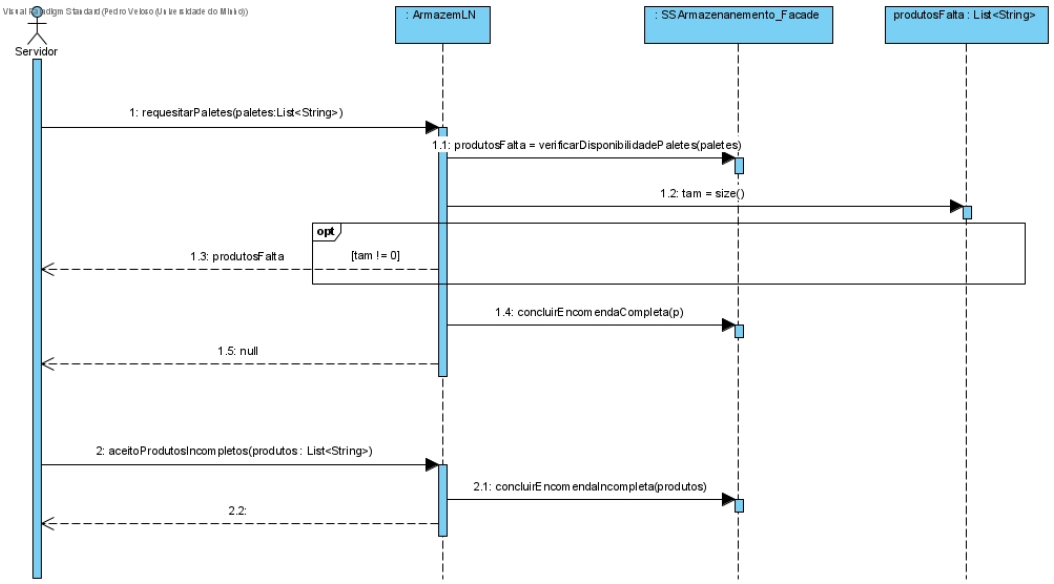

Figura 27: Diagrama de Sequência - RequesitarPaletes

Para verificar a disponibilidade das paletes existentes é necessário realizar os passos descritos no próximo diagrama.

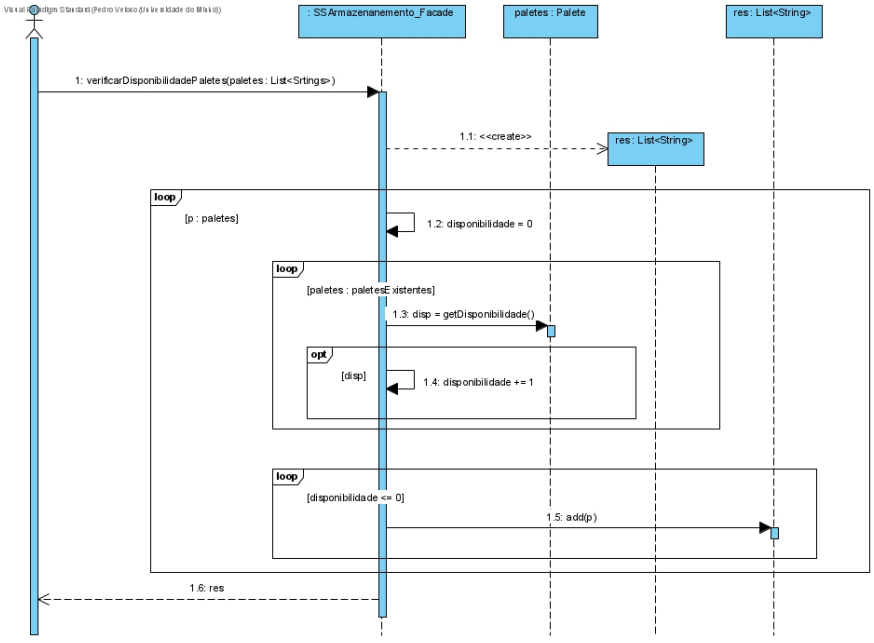

Figura 28: Diagrama de Sequência - VerificarDisponibilidade

Para finalizar a encomenda é necessário distinguir a finalização completa, que ocorre quando todos os produtos estão disponíveis, da conclusão incompleta, que ocorre quando é aceite a entrega incompleta da encomenda, como se pode verificar na primeira figura (conclusão incompleta) e na segunda figura (finalização completa).

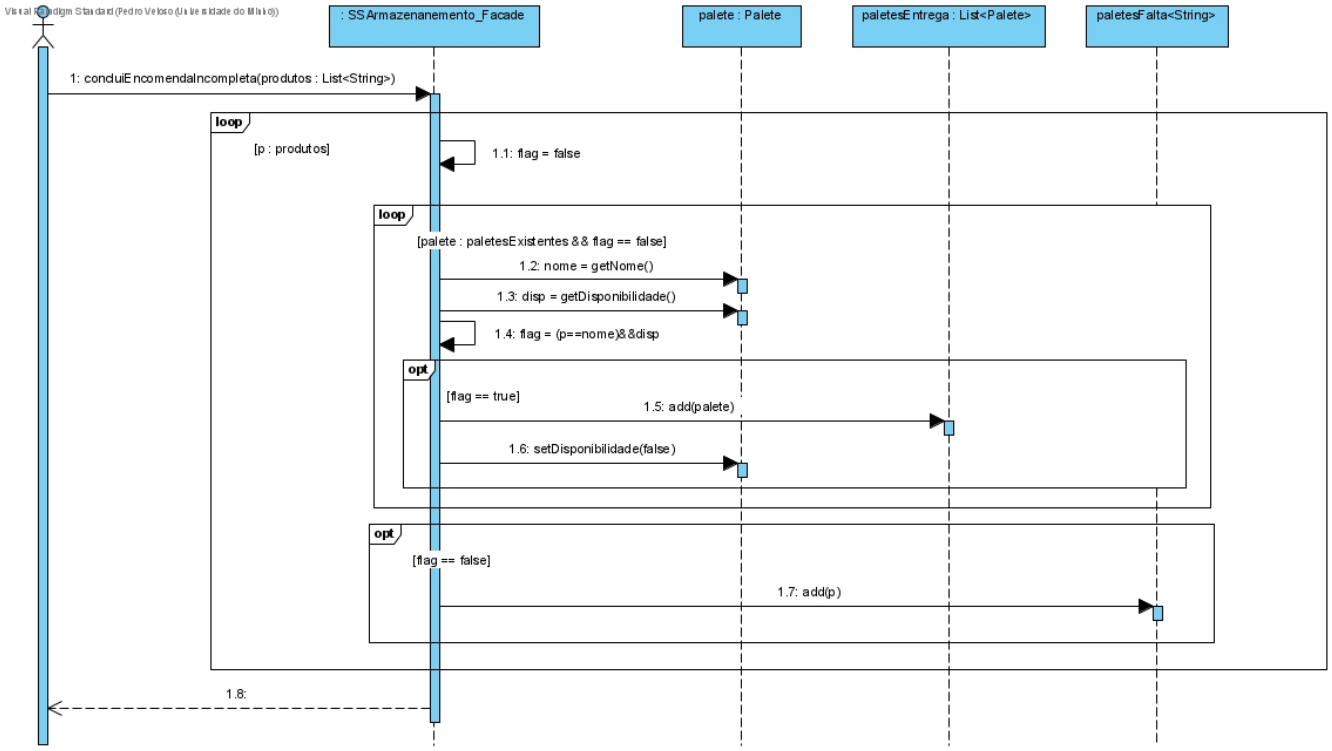

Figura 29: Diagrama de Sequência – ConcluirEncomendaIncompleta

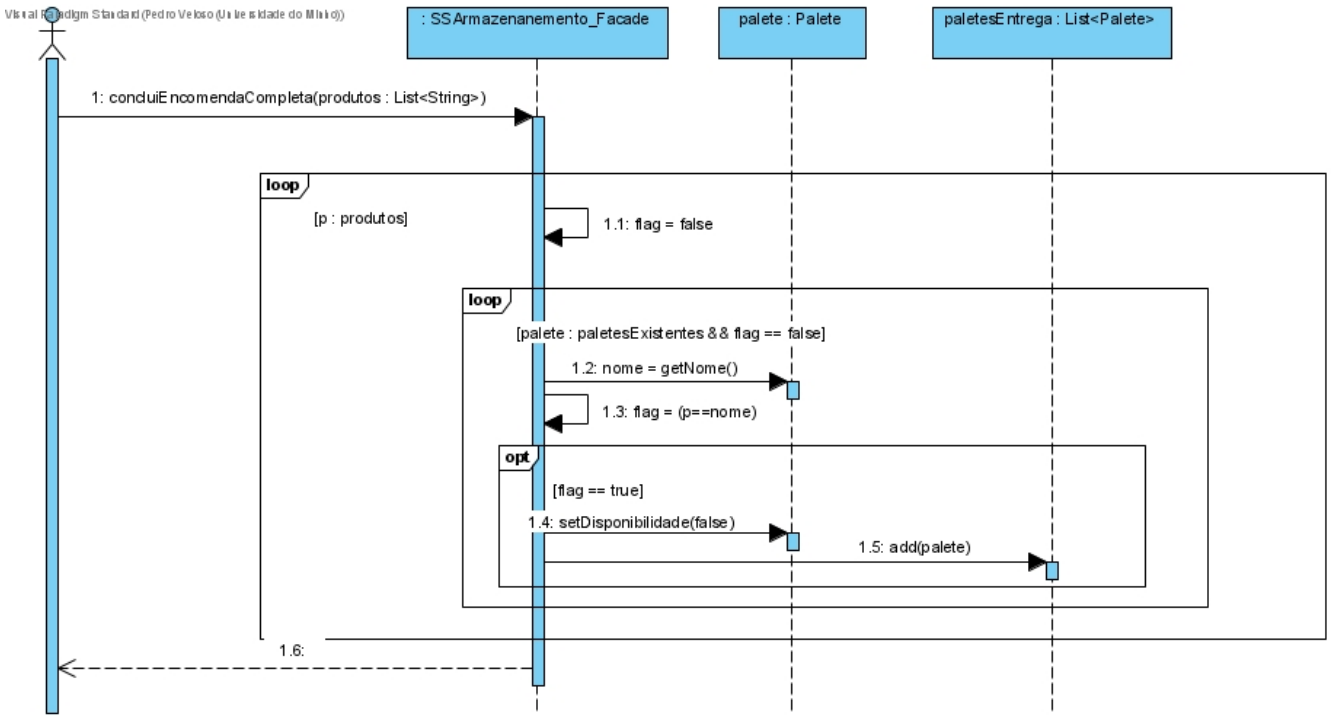

Figura 30: Diagrama de Sequência - ConcluirEncomendaCompleta

5. Diagrama de *Package*

Tendo em vista a evolução do sistema a implementar, adotou-se a utilização de packages, de modo a tornar mais fácil a gestão das classes.

Desta forma, a arquitetura do sistema deve seguir a seguinte estrutura.

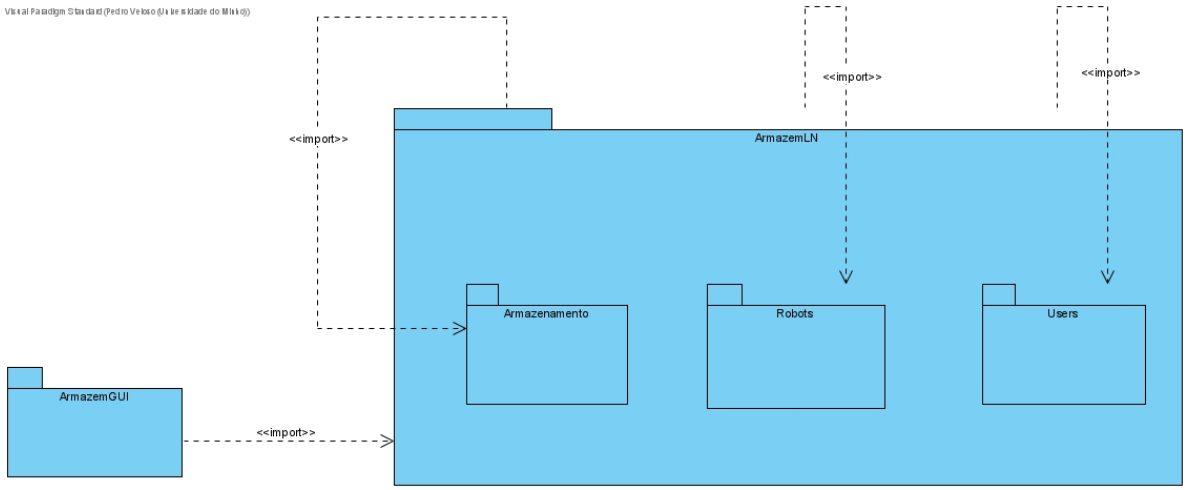

Figura 31: Diagrama de Package

Como se pode observar pela figura acima, existe um package denominado por ArmazemLN onde se encontra a logica de negócio que importa as *facades* dos outros subsistemas, cada um representado num package diferente. Posteriormente a lógica de negócio será importada por outras classes, como por exemplo as interfaces. Assim temos em cada package as seguintes classes:

Package Armazenamento:

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

Package Robots:

- IRobot
- Mapa
- Percurso
- Robot
- SSRobot_Facade

Package Users:

- User
- Funcao

6. Conclusão

A segunda fase possibilitou a construção de uma das partes mais essenciais ao desenvolvimento do programa proposto. Com o auxílio da modelação comportamental conseguimos visualizar mais concretamente aquilo a que o sistema deverá corresponder no fim da última fase.

Com a utilização dos diagramas de sequência foi possível dividir os fluxos provenientes de cada *Use Case* em sequências de transações, que consistem em trocas entre cada ator com o sistema ordenadas temporalmente.

Cada um dos *Use Cases* pode ser considerado como um comportamento, em que para cada um se descreveu o conjunto de objetos que o compõe e as interações entre estes para levar à execução do comportamento em questão.

Esta foi uma ferramenta fundamental para organizar de forma estruturada e visual, assim como o nome indica, o comportamento que cada fluxo de ações possíveis no domínio que diz respeito ao problema que estamos a resolver.

Por fim, todo o trabalho realizado nas primeira e segunda fases será determinante para o desenvolvimento da solução propriamente dita.