Desenvolvimento de Sistemas de Software

Licenciatura em Engenharia Informática

Universidade do Minho
Grupo 25
2021/2022

Abstract

Este documento diz respeito à 2ª Fase do trabalho prático da Unidade Curricular de Desenvolvimento de Sistemas de Software.

Figure 1: Grupo 25

1. Objetivos

Neste documento, apresentamos os seguintes aspetos relativos a um Sistema de Gestão para Centros de Reparação de equipamentos eletrónicos:

i. Arquitetura conceptual, capaz de suportar os requisitos identificados.
ii. Modelação comportamental dos métodos necessários à implementação desses requisitos.
iii. Descrição da implementação da solução.

2. Metodologia

A abordagem seguida para a implementação da solução consistiu em:

i. Identificação da API da lógica de negócio a partir dos Use Cases desenvolvidos na 1ª Fase do trabalho.
ii. Análise das operações da API da lógica de negócio, via diagramas de sequência.
iii. Identificação dos subsistemas da solução, representado por um diagrama de componentes.
iv. Definição da arquitetura dos subsistemas, representada por um diagrama de classe.

3. Alterações da 1ª Fase

Na 1ª Fase desenvolvemos um Diagrama de Use Case (e sua respetiva descrição textual) e um Modelo de Domínio. Alteramos ambos, ligeiramente.

3.1. Alterações no Diagrama de Use Case

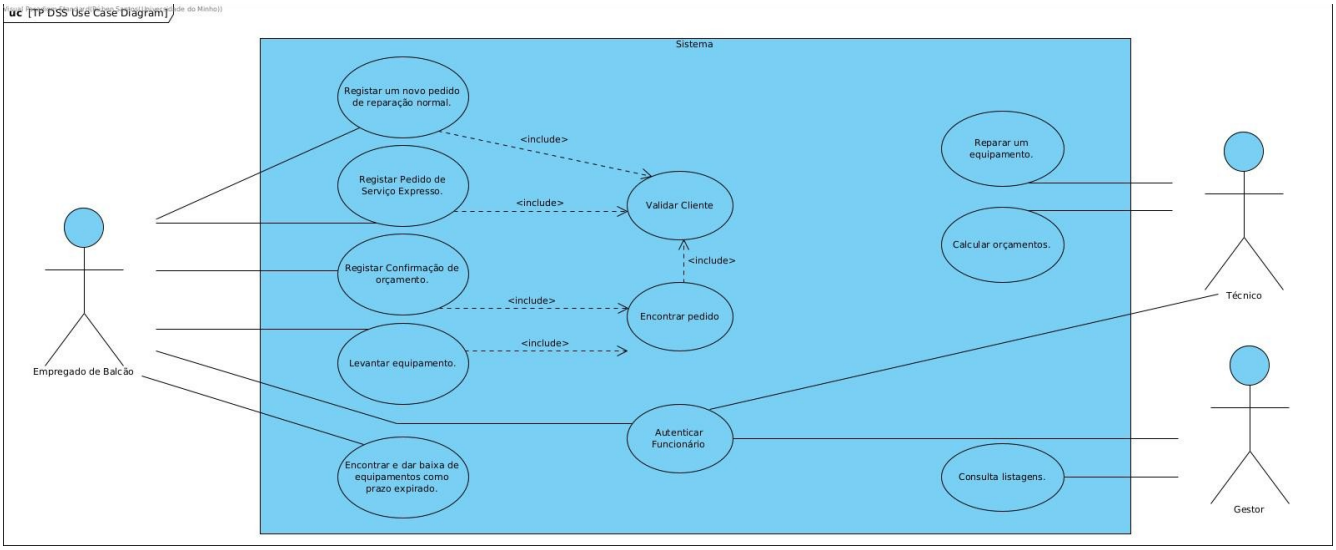

Figure 2: Diagrama de Use Cases 1ª Fase

Em relação ao Diagrama apresentado na figura acima, foram feitas algumas inserções, maioritariamente de Use Cases que retratam particularidades da implementação, como p. ex. *Adicionar preço de reparação expresso*. Outros Use Case mencionados abaixo vieram apenas substituir os da 1ª Fase, sem nenhuma alteração profunda (4., 5. e 6.). Os novos Use Case são:

1. *Adicionar Funcionário*, associado ao *Gestor*.
2. *Adicionar preço de reparação expresso*, associado ao Gestor.
3. *Alterar preço por hora*, associado ao Gestor.
4. Encontrar pedidos, associado ao Empregado de Balcão.
5. *Dar Baixa de Equipamentos*, associado ao Empregado de Balcão.
6. *Registar Pedido*, associado ao Empregado de Balcão.

Os Use Case 1., 2. e 3., trazem novas funcionalidades, mas não tão interessantes como os da segunda metade da lista, que iremos abordar em detalhe neste documento.

O Use Case *Encontrar Pedido* é agora acessível ao Empregado de Balcão diretamente. Desta forma deixam de existir todos os *includes* ligados ao Use Case Encontrar Pedido. Decidimos expandir este Use Case para o funcionário ter melhor acesso à informação, permitindo assim 5 tipos de procura diferente: Por cliente, Expresso, Normal, Fora de Prazo ou todos os anteriores. Desta forma, o novo Use Case fica da seguinte forma:

USE CASE: Encontrar pedidos.
CENÁRIOS: Cenários 3 e 4.
PRÉ-CONDIÇÃO: Funcionário está autenticado.
PÓS-CONDIÇÃO: O sistema gera uma lista de pedidos.
FLUXO NORMAL:
1. Funcionário seleciona que tipo de pedidos deseja ver (Por cliente, Expresso, Normal ou Fora de Prazo).
2. Sistema mostra os pedidos.

FLUXO ALTERNATIVO 1: [Funcionário quer procurar por cliente] (passo 2):
1.1. «include» Validar Cliente.
1.2. Funcionário fornece dados do cliente
1.3. Sistema mostra os pedidos do cliente.

Também alteramos o Use Case de "Encontrar e dar baixa de equipamentos com o prazo expirado", pois o nosso sistema faz a gestão dos prazos do sistema por si mesmo e não é efetuado especificamente só para este caso, como inicialmente esperado. Assim temos o novo Use Case *Dar Baixa de equipamentos*, da forma:

USE CASE: Dar Baixa de equipamentos.
CENÁRIOS: Cenários 1.
PRÉ-CONDIÇÃO: Empregado de Balcão está autenticado.
PÓS-CONDIÇÃO: O sistema fica com mais um pedido no estado "Arquivado".
FLUXO NORMAL:
1. Empregado de Balcão fornece o id de um pedido e confirma a baixa desse pedido.
2. Sistema muda arquiva o pedido de reparação.

FLUXO EXCEÇÃO 1: [O pedido não necessita de baixa] (passo 1):
2.1. Sistema verifica que não existe equipamentos no estado "Necessita baixa do equipamento".
2.2. Sistema cancela a operação.

Temos ainda, e mais importante, o novo Use Case *Registar Pedido*, que junta os dois antigos (*Registar um novo pedido de reparação normal/expresso*). O novo Use Case fica da seguinte forma.

USE CASE: Registar um novo pedido de reparação.
CENÁRIOS: Cenários 1.
PRÉ-CONDIÇÃO: Empregado de Balcão está autenticado.
PÓS-CONDIÇÃO: O sistema fica com mais um pedido de reparação na base de dados.
FLUXO NORMAL:
1. «include» Validar Cliente.
2. Empregado de Balcão regista novo pedido de reparação.
3. Sistema associa cliente a este pedido.
4. Empregado de Balcão adiciona a descrição do pedido.
5. Sistema associa um código ao pedido.
6. Sistema imprime o talão com o código associado ao pedido do cliente.
7. Sistema adiciona o pedido na base de dados no estado "Orçamento por calcular".

FLUXO ALTERNATIVO 1 [O novo pedido é um pedido expresso] (passo 2):
2.1. Empregado de Balcão verifica disponibilidade do Serviço Expresso.
2.2. Empregado de Balcão regista novo pedido de Serviço Expresso, associado ao cliente.
2.3. Sistema associa um código ao pedido e vai buscar o preço a tabela de preços do serviço expresso.
2.4. Sistema imprime o talão com o código e preço associado ao pedido do cliente.
2.5. Sistema adiciona o pedido na base de dados no estado "Necessita reparação".

FLUXO EXCEÇÃO 1 [Não há disponibilidade para realizar o pedido expresso] (passo 2.2):
2.2.1. Empregado de Balcão verifica que não há disponibilidade para realizar o pedido.

2.2.2. Sistema cancela Serviço Expresso.

Na figura abaixo podemos visualizar as alterações referidas anteriormente, no novo Diagrama de Use Case da 2ª Fase.

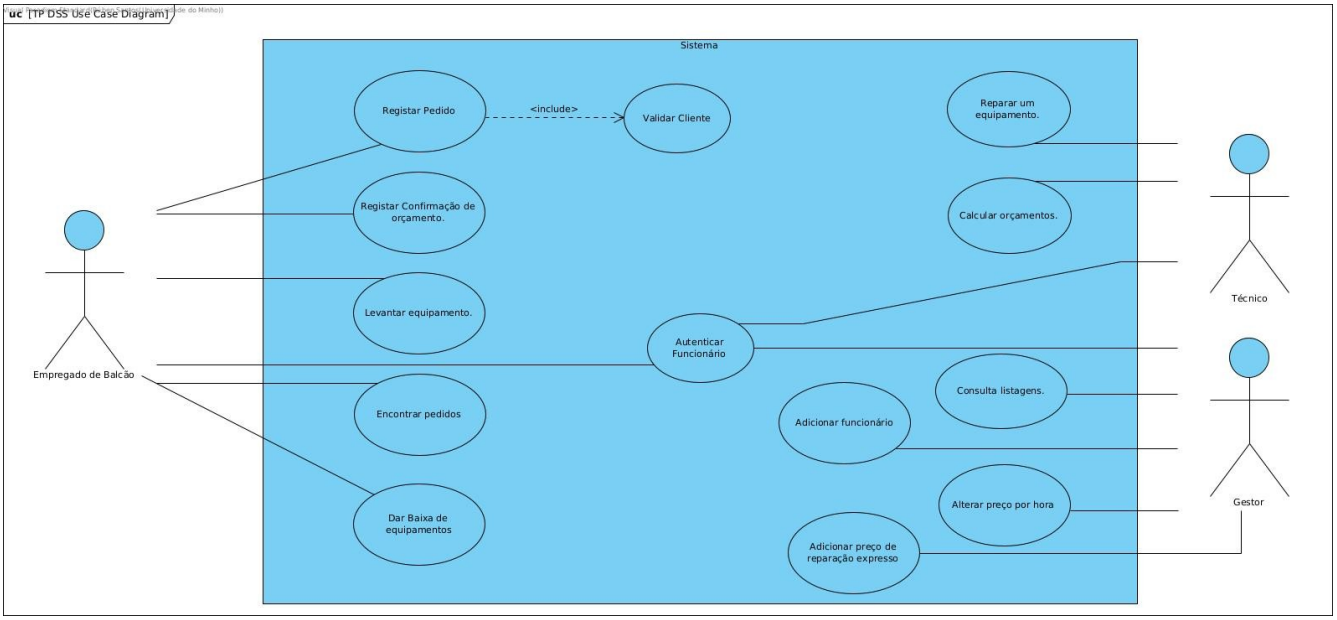

Figure 3: Diagrama de Use Cases 2ª Fase

3.2. Alterações no Modelo de Domínio

No novo Modelo de Domínio, deixado em anexo, mudamos apenas uma nota. No novo modelo não existe o Estado *Arquivado*.

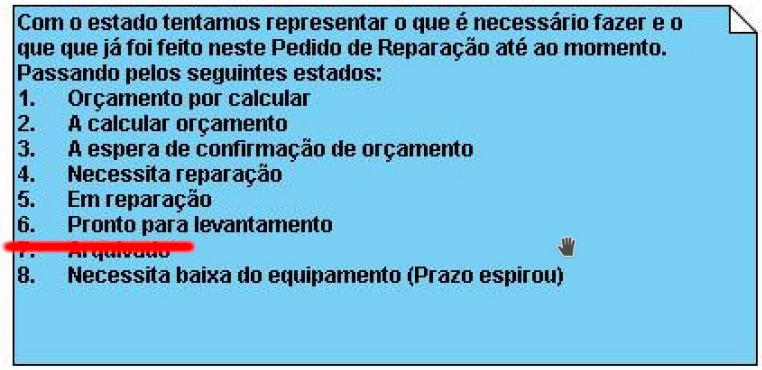

Figure 4: Alteração na nota do modelo de domínio antigo (a vermelho).

4. Implementação

A descrição da implementação da solução é dividida em 3 visões distintas: uma visão geral, orientada aos componentes do sistema, uma segunda visão, ainda bastante alargada, da arquitetura dos subsistemas e, finalmente, uma perspetiva das operações da API da lógica de negócio.

4.1. Componentes do Sistema

Com o objetivo de generalizar e melhor estruturar o nosso projeto, decidimos dividir a nossa camada de negócio em 3 componentes:

i. O componente *IGestorEntidades* vai guardar e gerir todas as entidades físicas associadas ao sistema: Clientes, Equipamentos e Funcionários.
ii. O componente *IGestorPedidosReparacao* vai gerir todos os pedidos de reparação do sistema, tanto os expressos como os normais. Para além de guardar e gerir, vai disponibilizar métodos que permitam, por exemplo: calcular um orçamento, reparar um equipamento ou levantar um pedido.
iii. O maior componente do nosso projeto, o *ISistemaFacade*, vai integrar os componentes referidos anteriormente num só. Vai também ter a ele autenticado um funcionário, registando assim as ações realizadas por ele.

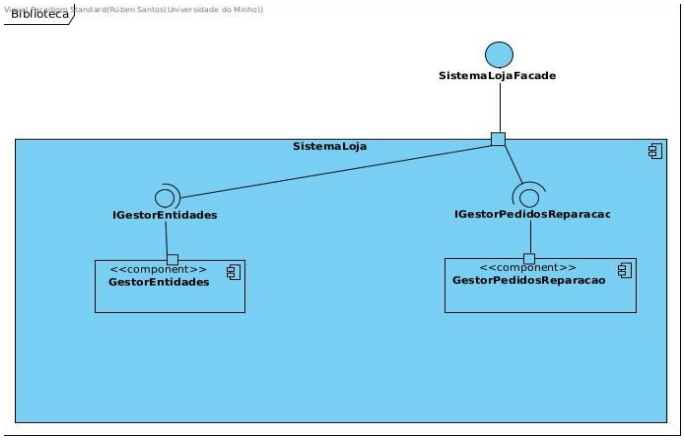

Figure 5: Diagrama de Componentes

4.2. Arquitetura dos subsistemas

Nota: Os packages do diagrama de classe apresentados suprimem detalhes da implementação do sistema com o objetivo de facilitar a sua leitura no presente documento. Os diagramas "mais completos" são enviados em anexo em formato *.vpp*.

As classes que compõe o componente "IGestorEntidades" são bastante simples, permitindo apenas a adição e consulta das entidades no sistema. Cada tipo de entidade é guardada num *HashMap*, usando o seu id no sistema como uma chave. Para este efeito faz uso de uma exceção *EntidadeNaoExiste*, para impedir a procura de entidades que não existam no sistema.

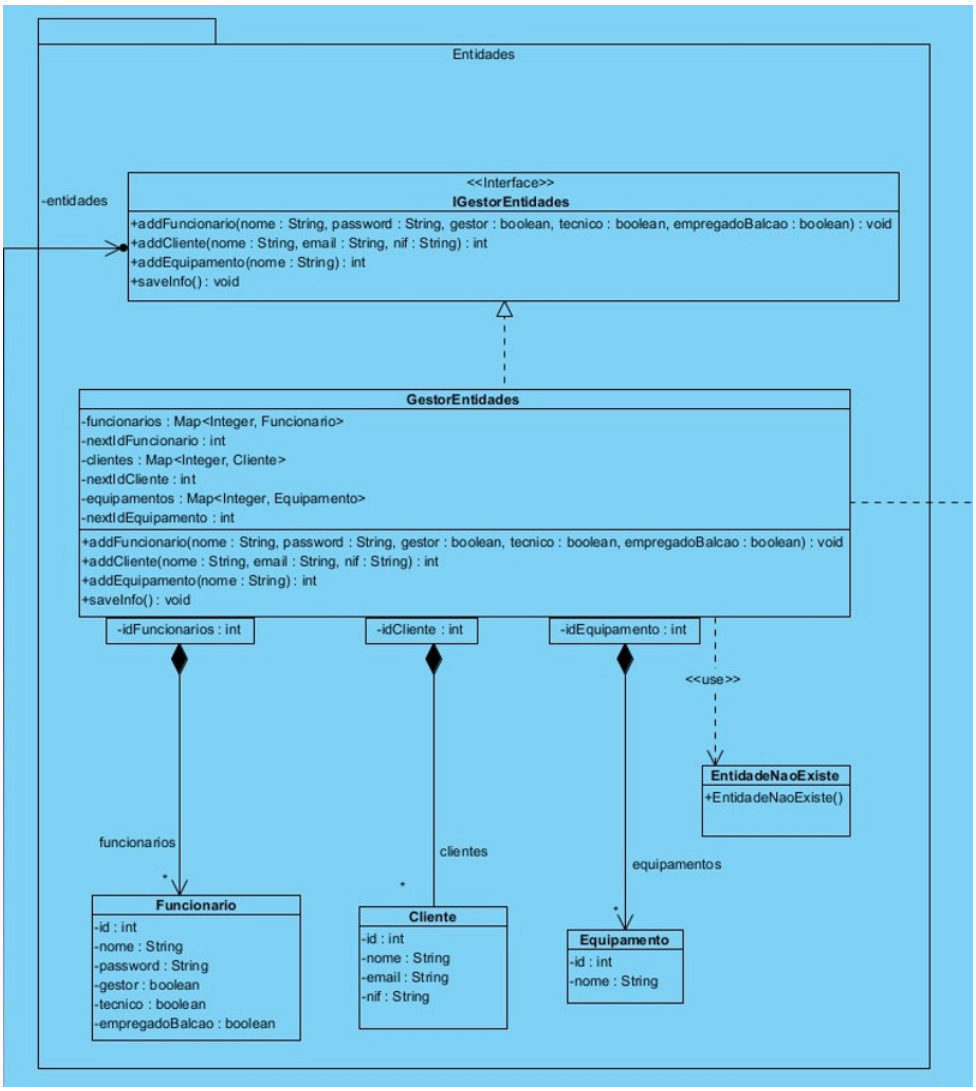

Figure 6: Package Entidades

O componente que cuida dos pedidos (*IGestorPedidosReparacao*) é um pouco mais complexo. A classe *Pedido de Reparação* é a classe abstrata que representa qualquer pedido, contém assim as suas informações principais e implementa os uses cases, relacionados com pedidos, apresentados na primeira fase.
Um pedido de reparação normal percorre sempre todos os estados presentes na enumeração, exceto o último que está reservado a pedidos cujos prazos de confirmação do orçamento ou levantamento do equipamento expiraram. Por exemplo, o método *orcamentoAceite*, se o pedido estiver no estado *EsperaConfirmacaoOrcamento* transita para o estado *NecessitaReparacao*, ficando assim elegível para ser reparado. Caso contrário, o pedido não está no estado correto e, como tal, invocamos uma das exceções presentes no package Exceptions.
O pedido de reparação expresso, ao contrário do normal, não tem orçamento, ou seja, parte logo do estado *NecessitaReparacao*.
Para permitir a adição de passos no orçamento pelo método *addPassoOrcamento* é necessário estar no estado *ACalcularOrcamento*, isto é conseguido pelo método *comecarOrcamento.* Esta atividade é repetida no caso de uma reparação, utilizando os métodos *comecarReparacao* e *terminadoProximoPasso*.
Para fazer esta gestão, o componente *IGestorPedidosReparacao* guarda os pedidos por processar como instâncias da classe *PedidoReparacao* e os pedidos já terminados como instâncias do record *InfoPedidoTerminado*. Isto deve-se ao facto de não haver necessidade de guardar certas informações, por exemplo já não haver necessidade de haver um estado nem um prazo.

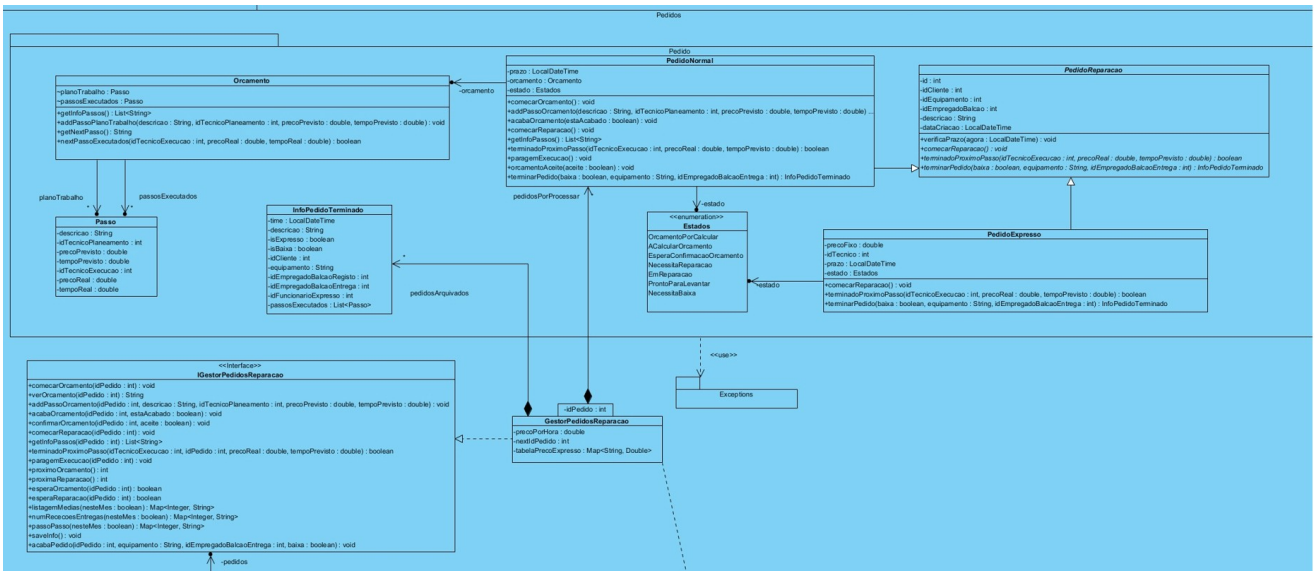

Figure 7: Package Pedidos.

O componente apresentado abaixo permite a união dos dois anteriores. Vamos explicar este componente com dois exemplos.

i. No caso de levantar um equipamento da loja precisamos de arquivar o pedido. O pedido contém o id do Equipamento, mas este vai ser inútil, e problemático, pois o equipamento vai ser removido do sistema. Ou seja, vamos aceder ao componente *IGestorEntidades* para obter e remover o equipamento e usar o *IGestorPedidosReparacao* para arquivar o pedido com esta informação.

ii. No caso de acabar um orçamento é necessário notificar o cliente. Para isto vamos usar a classe auxiliar *SendEmails*, mas é necessário obter informações do cliente e do pedido, ou seja, ir buscar as informações aos dois outros componentes.

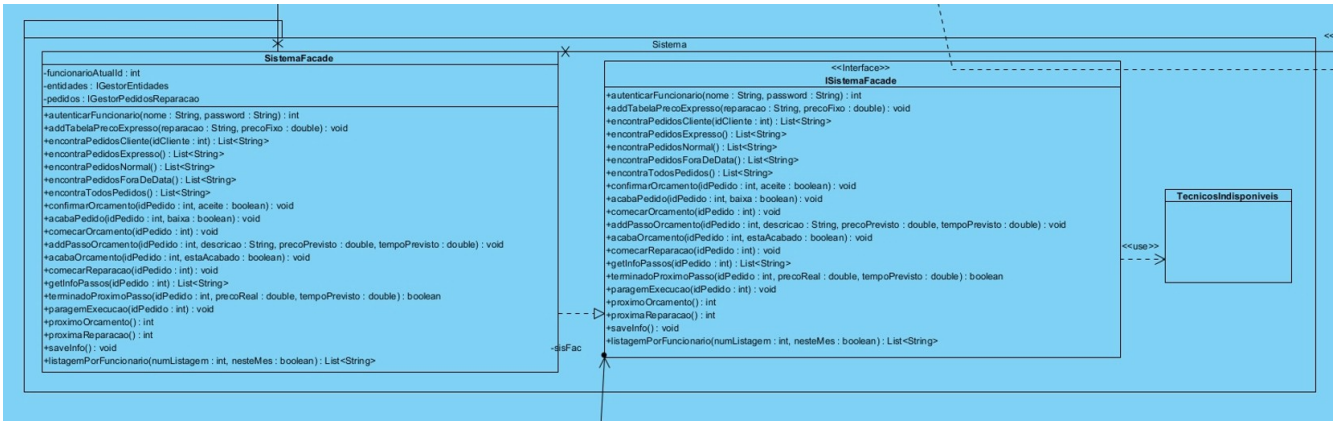

Figure 8: Package Sistema

Os dois *packages* abaixo vão conter classes auxiliares. A classe *dataDAO* vai permitir aos diferentes componentes escrever e ler informação em ficheiros.

A classe *SendEmails*, envia emails do email da empresa para os clientes. Para isto usamos a API *javaMail*, inserimos as propriedades necessárias do serviço e-mail que utilizamos, como por exemplo, o *host smtp* e a porta do *smtp.* Para enviar um e-mail, a classe cria uma *EmailSession* com as propriedades fornecidas anteriormente e autentica-se no e-mail da empresa. Posteriormente, o método *sendEmailCode* é utilizado para construir o e-mail com a informação necessária, ambos no caso de fim do orçamento e de fim da reparação, e utiliza o *Transport*,

que é um objeto fornecido pela API, que se encarrega de enviar para o e-mail do cliente.

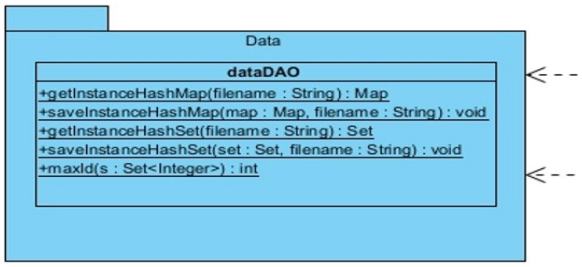

Figure 9: Package Data

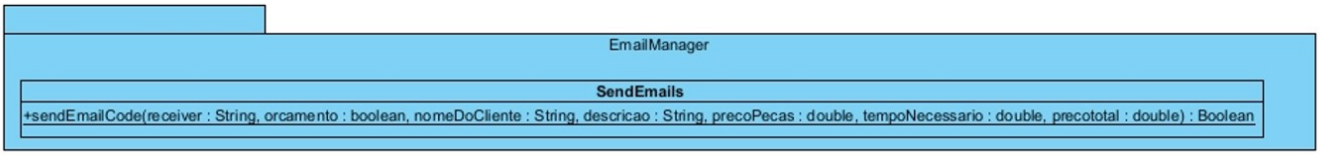

Figure 10: Package EmailManager

De modo a manter a estrutura MVC, temos, finalmente, os packages Controller e View.
No package View temos duas classes, Menu e ReaderWriter. A classe Menu representa os menus da View do sistema e a classe ReaderWriter faz todas as leituras e prints no terminal. Então, a classe Menu utiliza a classe ReaderWriter para mostrar o seu menu e obter as opções escolhidas pelo utilizador.
O package Controller tem a classe Controlador que liga a View do sistema ao modelo, esta classe utiliza a classe Menu para os seus diferentes menus e a classe ReaderWriter para mostrar os resultados das queries do modelo e obter inputs do utilizador.

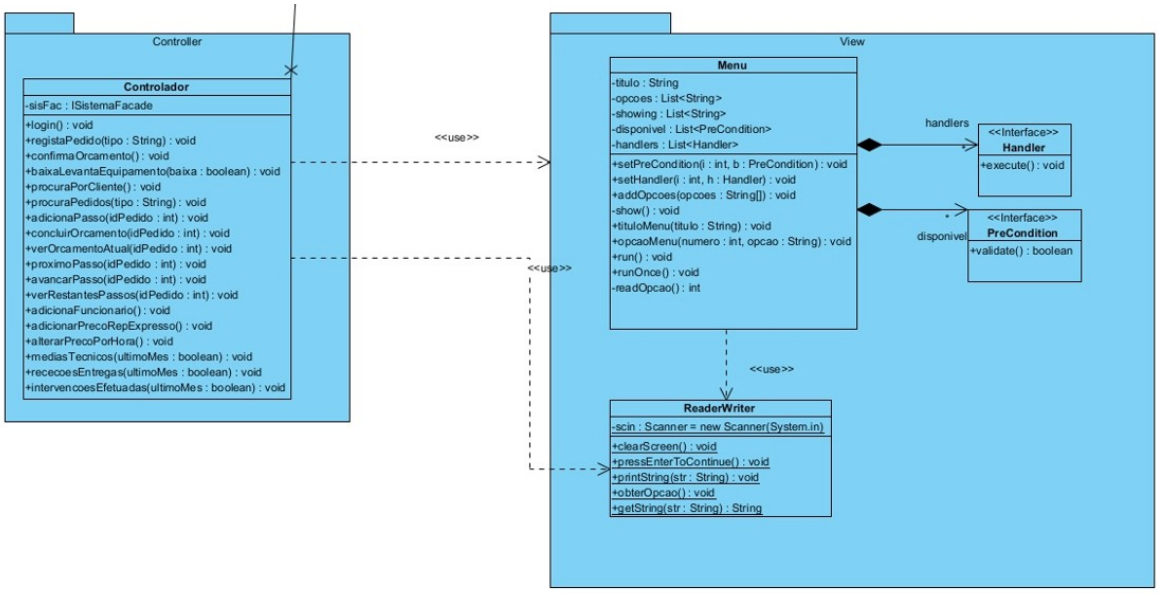

Figure 11: Packages View e Controller

4.3. Análise das operações da API da Lógica de Negócio

Para este ponto da descrição da implementação do sistema, decidimos selecionar os Use Case que desempenham uma função importante e fundamental no sistema. Apresentamos abaixo os Use Cases escolhidos, acompanhados pela sua descrição e por um diagrama de sequência.

Nota: Os diagramas apresentados suprimem detalhes da implementação do sistema com o objetivo de facilitar a sua leitura no presente documento. Os diagramas "mais completos" são enviados em anexo em formato *.vpp*.

4.3.1. Registar pedido (normal e expresso)

O diagrama de sequência apresentado na figura abaixo retrata o processo de registo de um pedido de reparação. Depois de inserirmos o NIF do cliente (método *getString*), procuramos pelo ID de cliente através da interface *ISistemaFacade*. No caso do cliente não se encontrar registado, procedemos ao seu registo (esta situação não está representada na figura). Inserimos o nome/descrição breve do equipamento (método *getString*) e chamamos o método *addEquipamento* que adiciona o equipamento ao sistema, via a interface *IGestorEntidades*. De seguida adicionamos o pedido, que pode ser normal ou expresso.

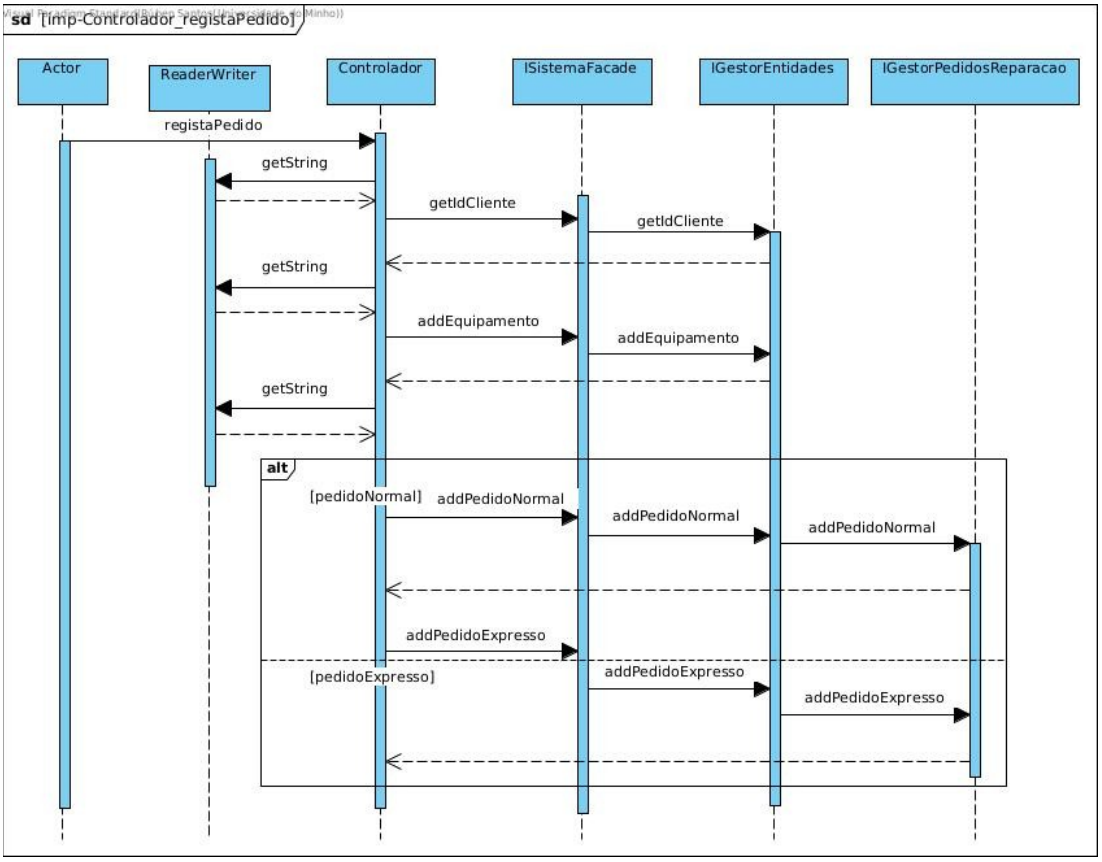

Figure 12: Versão simplificada do Diagrama de Sequência que regista um pedido.

4.3.2. Calcular Orçamento

O diagrama de sequência apresentado na figura abaixo representa o cálculo do orçamento de um pedido normal de reparação. O Técnico tem a possibilidade, através do Controlador, de adicionar passos ao orçamento (*addPassoOrcamento*) e de acabar o orçamento (*acabaOrcamento*). Sempre que pretende adicionar um passo ao orçamento, o método *getString* é utilizado. O orçamento é constituído por um conjunto de passos. Finalmente, após o Técnico ter acabado o orçamento, o cliente é notificado.
Neste diagrama falta uma primeira parte relativa à procura do orçamento a calcular. A versão completa segue em

anexo.

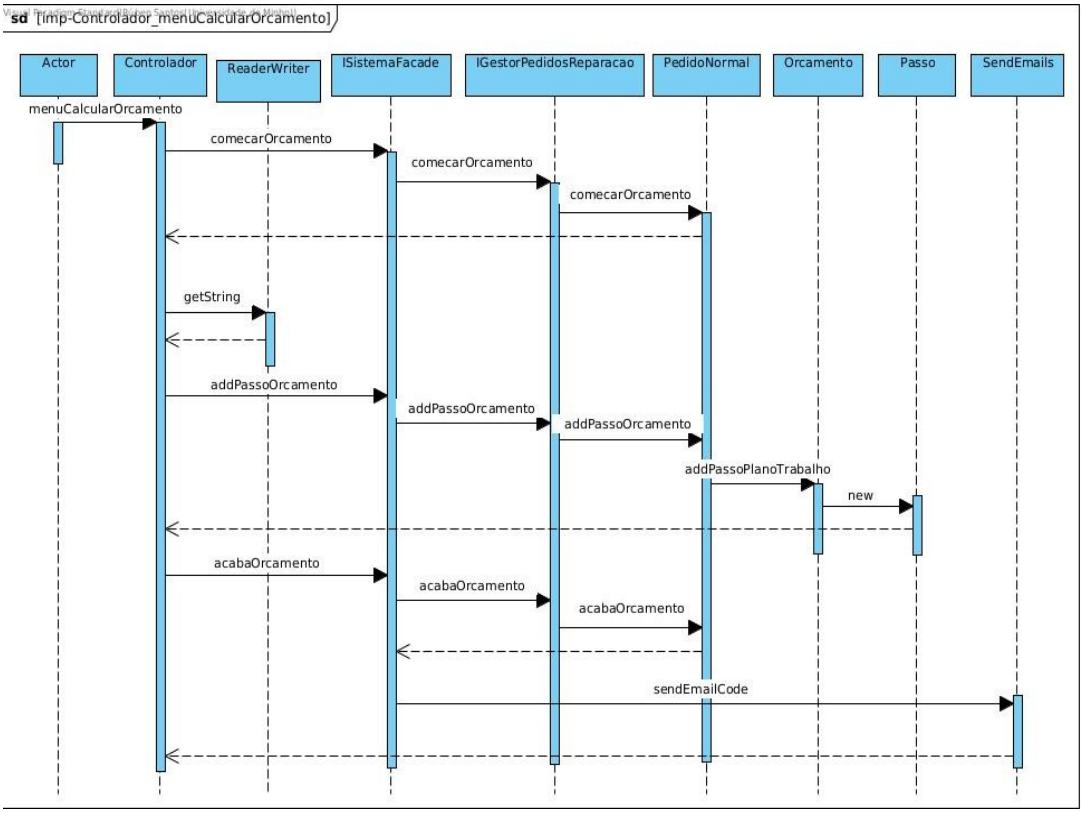

Figure 13: Versão simplificada do Diagrama de Sequência que calcula um orçamento.

4.3.3. Registar Confirmação Orçamento

O diagrama de sequência apresentado na figura abaixo diz respeito à confirmação do orçamento por parte do cliente. Assim que o Empregado de Balcão recebe a confirmação, este insere essa informação no sistema.

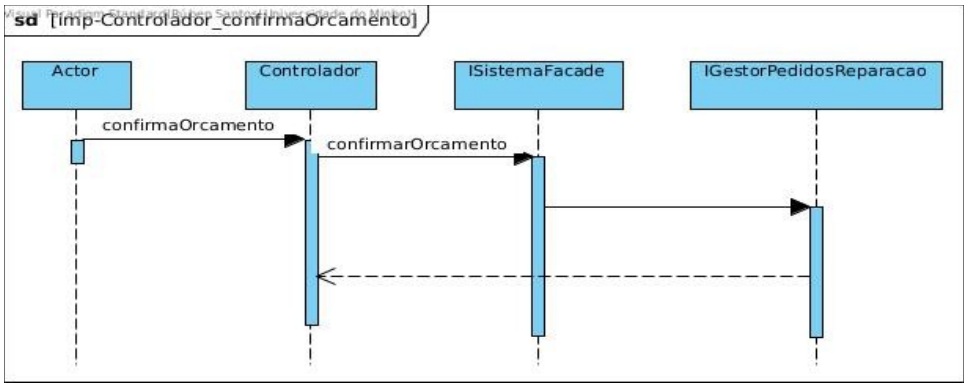

Figure 14: Versão simplificada do Diagrama de Sequência que confirma um orçamento.

4.3.4. Reparar Equipamento

O diagrama de sequência apresentado na figura abaixo descreve o processo de reparação, onde o Técnico repara o equipamento passo a passo, expresso pelo método *getProximoPasso*. A reparação termina assim que não

existam mais passos para serem executados, e o cliente é notificado para vir levantar o equipamento.
Neste diagrama falta uma primeira parte relativa à procura da próxima reparação a efetuar. A versão completa segue em anexo.

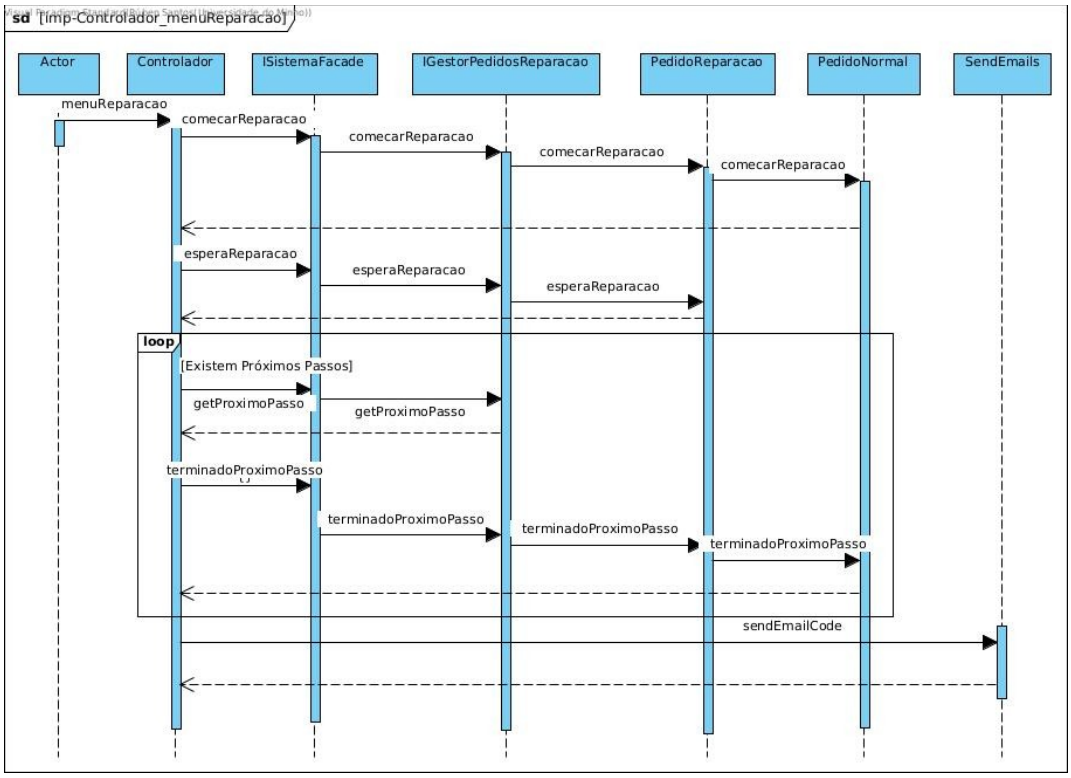

Figure 15: Versão simplificada do Diagrama de Sequência de reparação de um equipamento.

4.3.5. Levantar Equipamento e Baixar Pedido

O diagrama de sequência apresentado na figura abaixo exibe o comportamento do sistema quando pretendemos terminar um pedido e levantar o equipamento em questão. O método *getString* está a ler o ID do pedido. Dar baixa do pedido acontece quando o equipamento é abandonado na loja por mais de 30 dias, sendo que o processo é igual ao do terminar pedido/levantar equipamento (alteramos argumento de *baixaLevantaEquipamento*).

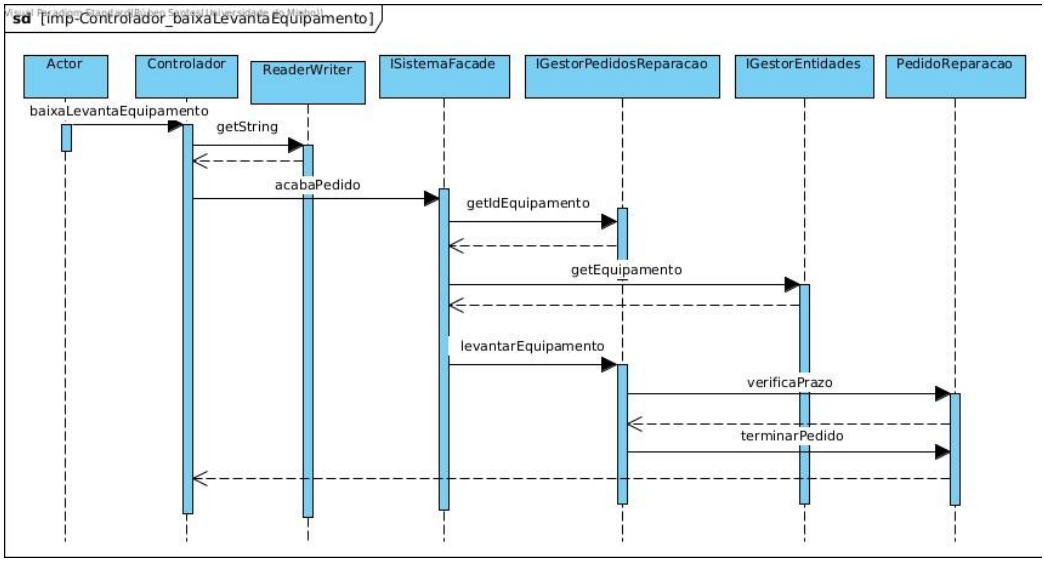

Figure 16: Versão simplificada do Diagrama de Sequência de levantamento/baixa de um equipamento.

5. Conclusão

No geral, o grupo está razoavelmente satisfeito com o que foi atingido ao longo das duas fases deste projeto. Tivemos dificuldades e desafios, alguns dos quais deixaremos nesta conclusão.
De modo a elaborar este documento precisamos de adaptar os diagramas de sequência que tínhamos feito. As figuras deste relatório relativas aos diagramas são portanto versões simplificadas dos diagramas completos disponíveis em anexo. Este processo foi moroso mas foi essencial para que possamos integrar diagramas legíveis e fáceis de entender neste documento.
O mesmo aconteceu para o diagrama de classe, sendo que deixamos em anexo duas versões do diagrama de classe: a versão completa e a versão simplificada (presente neste relatório).
Não achamos ter havido necessidade de fazer um diagrama de atividade, pois não existe nenhum algoritmo complexo o suficiente para representar.

Algumas das melhorias que poderíamos implementar tem que ver com o sistema de e-mail, como por exemplo, criar um sistema que confirme orçamentos a partir do *feedback* do cliente. Outras melhorias são: fazer uma Base de dados e ter mais opções para o gestor gerir o sistema (p. ex. Eliminar funcionários, eliminar preços de pedido expresso...)

Referências

Fowler. UML Distilled (3rd ed). Addison-Wesley, 2004.

Blaha & Rumbaught. Object-Oriented Modeling and Design with UML (2nd ed). Prentice Hall, 2005.

LucidChart. Disponível em LucidChart.