Relatório de Desenvolvimento de Sistemas de Software – 2º Fase

Grupo 30

Unidade Curricular: Desenvolvimento de Sistemas de Software [J305N2]

Coordenador: Professor Doutor José Francisco Creissac Freitas Campos

Ano Letivo: 2021/2022

Nome: Carlos Filipe Almeida Dias
Número: 93185
Contacto: a93185@alunos.uminho.pt
Curso: Licenciatura em Engenharia Informática, Universidade do Minho

Nome: José Pedro Martins Magalhães
Número: 93273
Contacto: a93273@alunos.uminho.pt
Curso: : Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Francisco Reis Izquierdo
Número: a93241
Contacto: a93241@alunos.uminho.pt
Curso: : Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Duarte Augusto Rodrigues Lucas
Número: a89526
Contacto: a89526@alunos.uminho.pt
Curso: : Licenciatura em Engenharia Informática, Universidade do Minho

Conteúdo

Lista de Figuras

Introdução

No âmbito da disciplina Desenvolvimento de Sistema de Software foi proposto o desenvolvimento de um Sistema de Gestão para Centros de Reparação de equipamentos eletrónicos, sistema que fosse capaz de garantir a gestão de todo o processo de reparação de equipamentos, desde a orçamentação até à entrega do equipamento.

Trata-se de um projeto que foi executado em duas fases:

- Na primeira fase e posterior entrega consistiu na elaboração do diagrama de Use Case e no diagrama de Domínio. A equipa começou por reunir-se e discutir o enunciado disponibilizado pelos docentes da disciplina, com o propósito de uma boa interpretação e posterior elaboração dos diagramas UML, de forma a desenvolver o sistema de software pretendido.

- A segunda fase foi responsável por alterar e aprimorar algumas decisões tomadas aquando da primeira fase, tendo por base a visão do grupo em relação a vários aspetos relacionados com a conceção, levando a alterações notáveis na estruturação, descrição e também nos diagramas supramencionados. Além disso, nesta fase foram estendidos vários conceitos de conceção que se revelaram importantes para a implementação do sistema de software objetivo, levando, no seguimento disto, à resolução de diversos outros diagramas para sustentarem a modulação e conceção efetuada nesta fase, tais como o diagrama de Componentes, o diagrama de Máquina de Estado, os diversos diagramas de Sequência e o diagrama de Package. Ainda nesta fase do trabalho, com base nos diagramas UML realizados, foi desenvolvido e acima de tudo implementado o sistema de software de gestão para centros de reparações e equipamentos, passando

da conceção ao físico, criando. A aplicação foi desenvolvida na linguagem de programação Java, com recurso ao IDEA "IntelliJ".

Objetivos e Abordagem

Inicialmente, enquanto grupo e após várias discussões construtivas, foram definidos vários objetivos para um bom desenvolvimento do projeto. Foi então, nesta linha de raciocínio, que concluímos que seria de teor fundamental o desenvolvimento do software proposto seguindo vertentes respetivas à conceção, tais como comportamento, estrutura e interação, suportados por vários diagramas UML que elucidassem de forma clara e objetiva todo o enunciado que nos fora proposto. Deste modo, a posterior implementação do código tornar-se-ia mais facilitada e organizada.

- Arquitetura: A primeira abordagem foi escolher qual a arquitetura que achávamos mais adequada e que iria sustentar o sistema a desenvolver, pelo que o grupo decidiu que a arquitetura de todo o sistema seria a conhecida arquitetura "Facade". Esta arquitetura mostra ser bastante vantajosa em diversos aspetos, tais como a fácil implementação e simplicidade, pois apenas tem de ser "definida" uma fachada que faça a conexão e de certo modo a separação do que visa ser a lógica de negócio e a interface do utilizador.

Posto isto, numa fase inicial, começámos por definir as várias entidades e os seus relacionamentos de forma a abstrair o enunciado apresentado e alado a isto o correspondente Modelo de Domínio. Posto isto, o seguinte passo passou por identificar quem iria interagir com o sistema a implementar, isto é, os utilizadores para o qual a aplicação seria projetada e que interações iriam descrever. Assim, foram identificados os diversos atores bem como as interações que os mesmos poderiam ter com o sistema, e com estes dados foi possível criar o diagrama de Use Case, bem como descrever detalhadamente cada use case. Este diagrama revelou-se um dos grandes motores para os diagramas futuros, pois foi bastante esclarecedor ajudando-nos a ter uma perceção mais objetiva e clara sobre o que iria ser o "produto final".

Após o planeamento sobre os atores e as suas interações com o sistema, de forma a complementar o diagrama de Use Case, foi desenvolvido e estendido em "Excel" todos os use cases pensados e os seus respetivos fluxos. Desta forma, foi possível atribuir/identificar responsabilidades da lógica de negócio, identificar métodos e agrupar os métodos em subsistemas, representando estes subsistemas e as suas interfaces, bem como as relações entre si no diagrama de Componentes.

De seguida e dado o supramencionado, passámos para a fase de conceção onde partimos da identificação dos vários subsistemas e responsabilidades da lógica de negócio, onde tivemos em vista a lógica. Deste modo e dados os diferentes subsistemas e responsabilidades por estes encarregues, conseguimos abstraí-los em classes respetivas, surgindo deste modo o diagrama de Classes. Com isto, conseguimos posteriormente e ainda nesta fase, detalhar e especificar como iriam ser descritos os diversos use cases, isto é, quais classes e respetivos métodos iriam estar envolvidos na resolução dos mesmos, representando cada use case com o Diagrama de Sequência respetivo. Todos os diagramas mencionados

revelaram-se especialmente importantes, pois a sua conceção tornou-se num “guia” fundamental para a realização de um código, que por consequente se tornou mais simples e facilitado. Por fim, foi criado, com base nos diagramas supramencionados, o diagrama de Máquina de Estado.

Alterações Realizadas nos Modelos da Fase 1

Com o objetivo de retificar algumas nuances aquando do planeamento da primeira fase do projeto, decidimos fazer algumas alterações importantes no nosso planeamento, de forma a garantir uma implementação concisa e correta, uma vez que caso não retificássemos as diversas nuances, a implementação final do sistema seria posta em causa.

Um dos diagramas que sofreu mais alterações foi o diagrama de Use Cases. Começámos por reformular a nossa visão em relação ao enunciado proposto, nomeadamente recaindo sobre os atores. A reformulação passou por remover o ator cliente, uma vez que a sua interação com o sistema não acontece diretamente, mas sim por meio do ator funcionário.

Seguidamente retificámos os restantes atores e essencialmente as suas interações, isto é, reformulámos alguns dos use cases associados a estes, de forma a haver coerência e não haver redundância, uma vez que era algo notório na descrição de alguns use cases, mas apenas tendo em conta a atribuição/identificação de responsabilidades feita na segunda fase do projeto.

Primeiramente, no que diz respeito ao funcionário fundimos os use cases "Registar Equipamento" e "Registar pedido de orçamento" num só denominado "Registar Serviço", uma vez que de acordo com o cenário um, sempre que o cliente pretende arranjar um equipamento através de um serviço não expresso este pede sempre um orçamento antes de dar autorização para avançar com a reparação.

Relativamente ao técnico decidimos remover os use cases "Aceder à lista de pedidos" e "Registar plano de trabalhos para a reparação”, pois o primeiro é agora feito de forma automática pelo sistema e o segundo está agora incorporado no use case "Calcular orçamento".

Acerca do gestor mantivemos todos os seus use cases, sendo apenas adicionado mais um, o "Avalia centro de reparações" que tal como sugerido pelo nome permite a este ator gerar uma avaliação do centro tal como explicitado no cenário cinco.

Por fim, adicionámos ainda um use case partilhado por todos os atores, o "Autenticar", mencionado em vários cenários e que não foi adicionado na primeira fase do projeto. Este use case, vista a englobar a interação inicial que os diversos atores têm com o sistema, dado que para puderem ter acesso às restantes funcionalidades do mesmo, têm de porventura

autenticar-se neste, podendo mesmo os vários atores serem identificados numa fase inicial como um só ator, o utilizador.

Conceção

- **Modelo de Domínio**

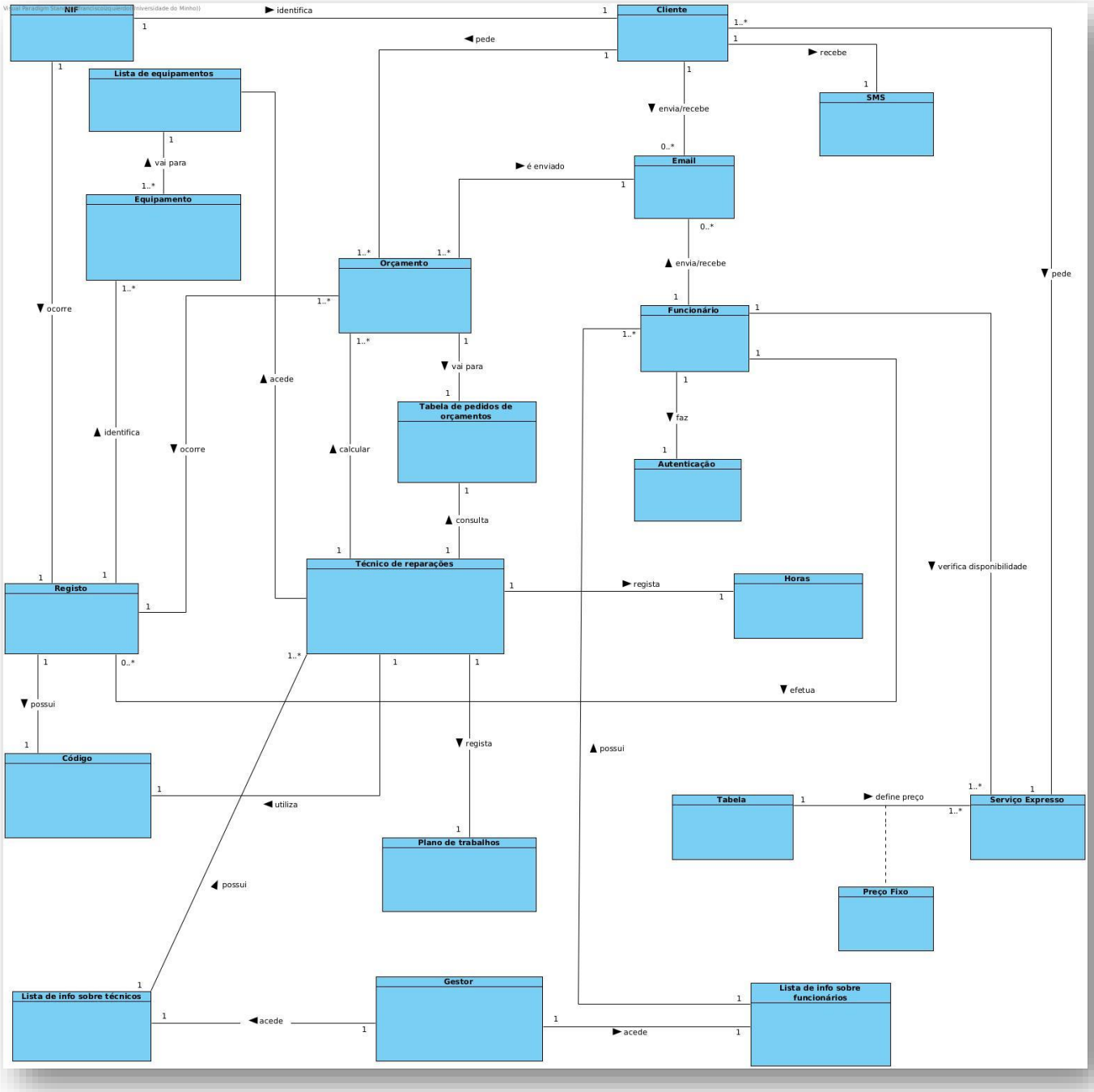

Figura 1: Diagrama de Domínio

- Modelo Use Case:

1ª Etapa: Identificar os atores e use cases do sistema:

Atores:

- Utilizador:

  - Funcionário do Balcão;
  - Técnico de Reparações;
  - Gestor.

Use cases para cada ator:

➢ Utilizador

- Autenticar

➢Funcionário do Balção

- Registar Serviço;
- Registar Serviço Expresso™;
- Levantar Equipamento;

➢Técnico de Reparações

- Calcular orçamento;
- Executar Reparação;
- Retomar Reparação;
- Confirmar orçamentos.
-

➢Gestor

- Faz uma consulta a uma listagem dos técnicos;
- Faz uma consulta a uma listagem dos funcionários de balção;
- Consultar lista de reparações;
- Avaliar Centro.

2ª Etapa: Diagrama de Use Cases:

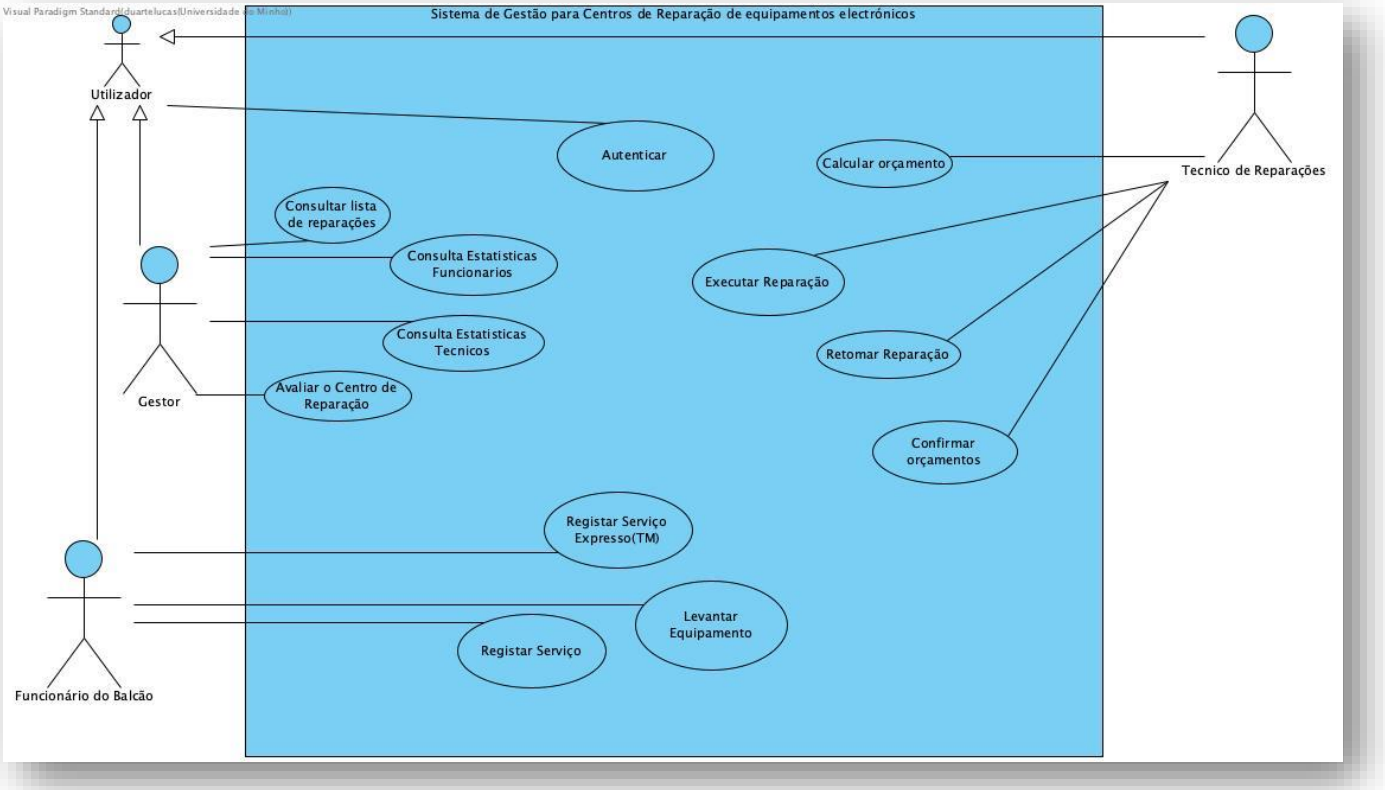

Figura 2: Diagrama de Use Cases

3ª Etapa: Descrever Cada um dos Use Cases:

- **Use case**: Autenticar.

Cenário: Cenário 1.

Pré-Condição: True.

Pós-Condição: Entrada no sistema.

Fluxo Normal:

1. O utilizador introduz o ID e a palavra-passe;
2. O sistema verifica que o ID existe e corresponde à palavra-passe;
3. O utilizador autentica-se com sucesso.

Fluxo de Exceção [Id introduzido não existe] (Passo 2):

1.1. O ID introduzido não existe no sistema;
1.2. O sistema informa que o ID não existe;
1.3. O utilizador não se autêntica.

Fluxo de Exceção [A palavra-passe e o id não correspondem] (Passo 2):

1.1. O ID e/ou a palavra-passe não correspondem;
1.2. O sistema informa que o ID e a palavra-passe não correspondem;
1.3. O utilizador não se autêntica.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Autenticar	autenticação()	Utilizador
UI			
	Autenticar	autenticação()	Utilizador
UI			
	verifica a correspondencia do id e da palavra passe	isValid()	Utilizador
UI			
UI			

Figura 3: Autenticar – Componentes

- **Use case**: Registar Serviço.

Cenário: Cenário 1.

Pré-Condição: True.

Pós-Condição: Serviço Registado.

Fluxo Normal:

1. O cliente dirige-se ao estabelecimento com um equipamento avariado e pede um orçamento para uma reparação;

2. O funcionário do balcão após se ter autenticado, introduz as informações relativas ao cliente o serviço pedido pelo cliente;

3. O sistema cria um registo com as informações relativas ao cliente.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
UI			
	Registar servico	registarServico()	Funcionario

Figura 4: Registar Serviço – Componentes

***Use case**: Registar Serviço Expresso (TM).*

Cenário: Cenário 2.

Pré-Condição: True

Pós-Condição: Serviço Expresso (TM) Registado.

Fluxo Normal:

1. O cliente dirige-se ao estabelecimento com um equipamento avariado e pede um serviço Expresso;

2. O funcionário do balcão após se ter autenticado, introduz as informações relativas ao cliente o serviço pedido pelo cliente.

3. Certifica-se da disponibilidade para a realização do serviço.

4. O sistema cria um registo com as informações relativas ao cliente.

Fluxo de Exceção [Não há disponibilidade para realizar o serviço do cliente] (Passo 2):

2.1. O funcionário do balcão verifica que não há disponibilidade para realizar o serviço;

2.2. O serviço é recusado.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
UI			
	Registar servico expresso	registarServicoExpresso()	Funcionario
UI			
UI			

Figura 5: Registar Serviço Expresso – Componentes

- **Use case**: Levantar Equipamento.

Cenário: Cenário 1.

Pré-Condição: Haver equipamento a levantar.

Pós-Condição: Haver um registo do levantamento e pagamento do equipamento.

Fluxo Normal:

1. Após a conclusão da reparação do equipamento, é enviado uma notificação com o código do registo ao cliente;
2. O cliente vai à loja levantar o equipamento, disponibilizando o código fornecido;
3. O funcionário do balcão procura o equipamento com o dado código.
4. O funcionário entrega equipamento após ter recebido o pagamento.
5. O funcionário regista entrega do equipamento e o pagamento.

Fluxo Alternativo [O cliente rejeita o pagamento da reparação] (Passo 1):

1.1. O cliente após receber o orçamento rejeita o pagamento da reparação.
1.2. O cliente vai à loja levantar o equipamento;
1.3. O funcionário do balcão regista a entrega do equipamento.

Fluxo de Exceção [O cliente não levanta o equipamento] (Passo 2):

3.1. Após 90 dias da conclusão da reparação do equipamento o cliente não levantou o equipamento;
3.2. O equipamento vai para uma lista de abandono;
3.3. Pode ser dada baixa do equipamento.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
UI			
UI			
UI			
UI			
UI			
	Verifica o abandono o equipamento	verificarAbandono()	Funcionario
UI			

Figura 6: Levantar Equipamento - Componentes

- **<u>Use case</u>**: Calcular Orçamento.

Cenário: Cenário 1.

Pré-Condição: Haver um pedido de orçamento.

Pós-Condição: Orçamento enviado ao cliente.

Fluxo Normal:

1. O técnico faz um plano de trabalhos para o serviço em questão;
2. O técnico envia ao cliente um email com o valor da reparação e o código do equipamento em questão.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Fazer um trabalho de planos para um serviço	setPlanoTrabalhos(List<String> plano)	Tecnico
UI			

Figura 7: Calcular Orçamento – Componentes

- **Use case**: Executar Reparação.

Cenário: Cenário 4.

Pré-Condição: Haver um equipamento para reparação.

Pós-Condição: Equipamento reparado.

Fluxo Normal:

1. O técnico verifica se que é um serviço não expresso.
2. O técnico vai assinalando a execução dos passos conforme os realiza, indicando o custo das peças efetivamente utilizadas e o tempo gasto;
3. O sistema vai atualizando a informação prestada pelo técnico

Fluxo Alternativo [O serviço é um serviço expresso] (Passo 1):

1.1. O serviço é um serviço expresso, logo não há plano de trabalhos, o técnico limita-se apenas a concluir a reparação.

Fluxo de Exceção [O técnico interrompe a reparação] (Passo 2):

2.1. O técnico a meio da reparação necessita de a interromper e coloca o equipamento em espera.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	O tecnico exxecuta a reparação do equipamento	executarReparacao()	Tecnico
	Reparaçao do serviço expresso	executarReparacao()	Tecnico
	O tecnico coloca em espera a reparação	colocaEmEspera(String codigo)	Tecnico

Figura 8: Executar Reparação – Componentes

- **Use case**: Retomar Reparação.

Cenário: Cenário 4.

Pré-Condição: Haver equipamento na lista de equipamentos em espera

Pós-Condição: Equipamento reparado.

Fluxo Normal:

1. O técnico escolhe um dos equipamentos da lista dos equipamentos em espera.

2. O técnico vai assinalando a execução dos passos conforme os realiza, indicando o custo das peças efetivamente utilizadas e o tempo gasto;

Fluxo de Exceção [O técnico interrompe a reparação] (Passo 2):

1.1. O técnico a meio da reparação necessita de a interromper e coloca o equipamento em espera.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	O tecnico exxecuta a reparação do equipamento	executarReparacao()	Tecnico
	O tecnico coloca em espera a reparação	colocaEmEspera(String codigo)	Tecnico

Figura 9: Retomar Reparação – Componentes

- **Use case**: Confirmar Orçamento

Cenário: Cenário 1.

Pré-Condição: Haver uma resposta por parte do cliente acerca do orçamento da reparação.

Pós-Condição: Informação do serviço atualizada.

Fluxo Normal:

1. O técnico acede a um regista que aguarda a confirmação do orçamento;
2. O técnico atualiza o registo com a reposta do cliente acerca do orçamento da reparação.
3. O sistema guarda a informação atualizada.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Atualizar a resposta ao pedido de orçamento do c	respostaAoPedidoOrcamento(boolean flag, String cod)	Tecnico

Figura 10: Confirmar Orçamento – Componentes

- **Use case**: Consulta estatísticas Técnicos.

Cenário: Cenário 5.

Pré-Condição: Haver estatísticas sobre os técnicos.

Pós-Condição: True.

Fluxo Normal:

1. O gestor acede às estatísticas sobre os técnicos.
2. O sistema fornece ao gestor as estatísticas relativas aos técnicos.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Aceder a estatísticas de tecnicos	estatisticasTecnicos()	Gestor

Figura 11: Consulta Estatísticas Técnicos – Componentes

- **Use case**: Consulta estatísticas Funcionários

Cenário: Cenário 5.

Pré-Condição: Haver estatísticas sobre os funcionários.

Pós-Condição: True.

Fluxo Normal:

1. O gestor acede às estatísticas sobre os funcionários.
2. O sistema fornece ao gestor as estatísticas relativas aos funcionários.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Aceder a estatísticas de funcionários	estatisticasFuncionarios()	Gestor

Figura 12: Consulta Estatísticas Funcionários – Componentes

- **Use case**: Consulta Lista de Reparações.

Cenário: Cenário 5.

Pré-Condição: Haver uma lista de reparações.

Pós-Condição: True.

Fluxo Normal:

1. O gestor acede a uma lista de reparações.
2. O sistema fornece ao gestor a lista de todas as reparações.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Aceder à lista de reparações	estatisticasTecnicosExaustiva()	Gestor

Figura 13: Consulta Listas de Reparações – Componentes

- **Use case**: Avaliar o Centro

Cenário: Cenário 5.

Pré-Condição: True.

Pós-Condição: Ter uma avaliação final do centro.

Fluxo Normal:

1. O gestor após aceder a todas as estatísticas, faz uma avaliação sobre o centro.
2. O sistema regista a avaliação efetuada pelo gestor.

1. Dividir os fluxos em sequências de transacções	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subsistemas (agrupar métodos)
	Fazer uma avaliação do centro	fazAvaliacoes()	Gestor

Figura 14: Avaliar o Centro – Componentes

- **Diagrama de Componentes**

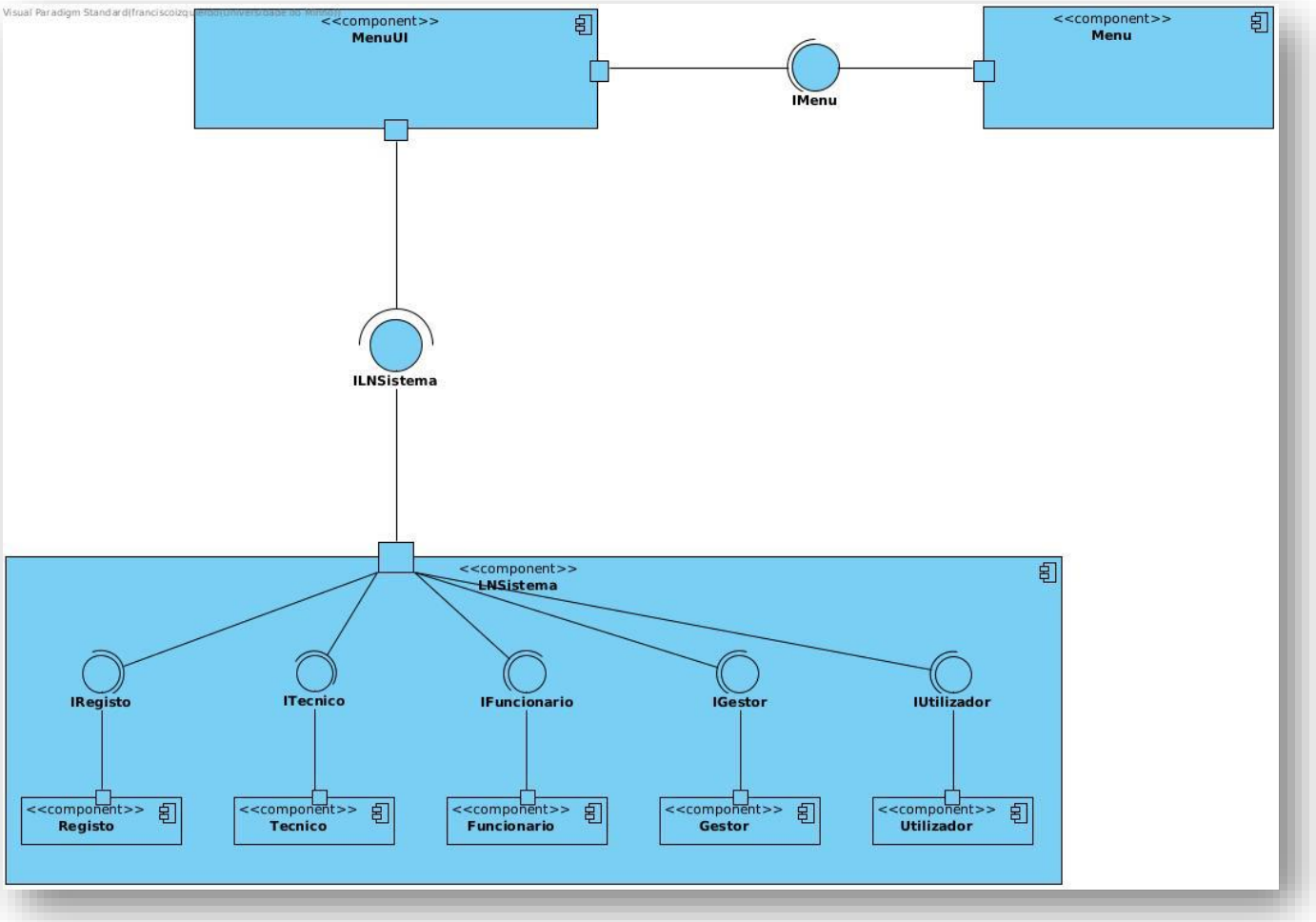

Figura 15: Diagrama de Componentes

- **Diagramas de Sequência**

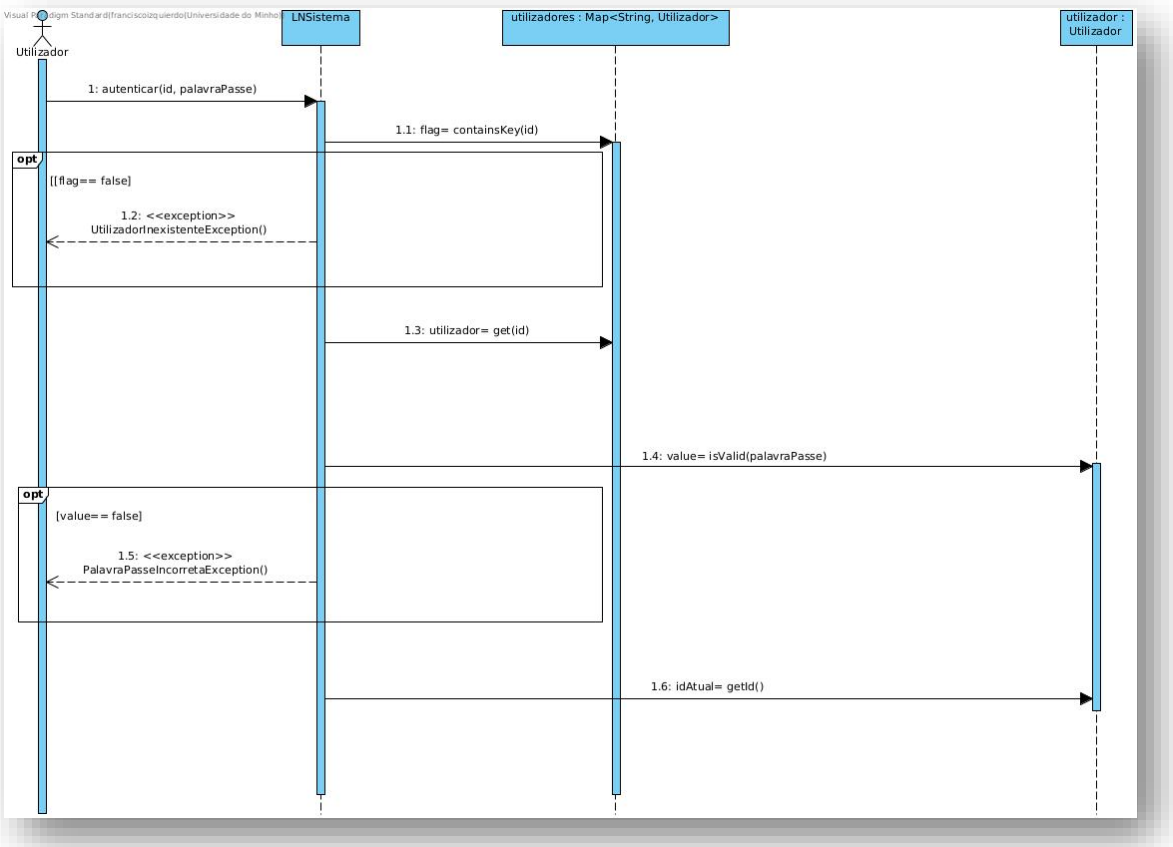

Figura 16: Autenticar

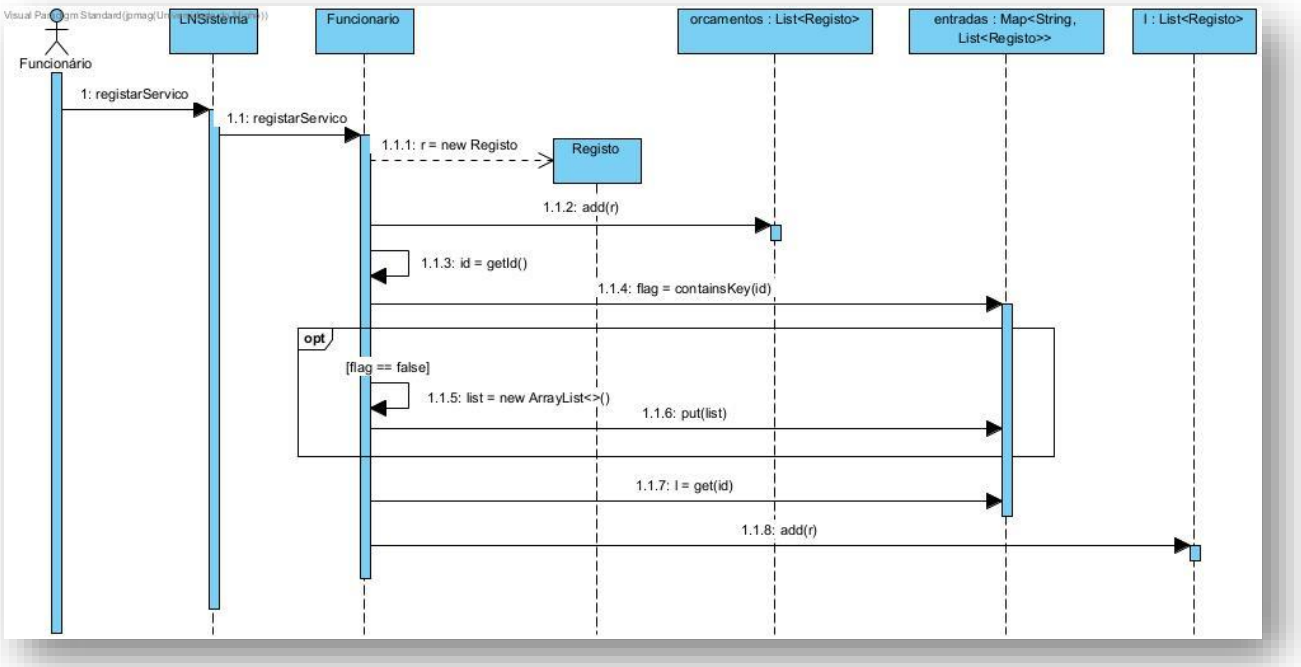

Figura 17: Registar Serviço

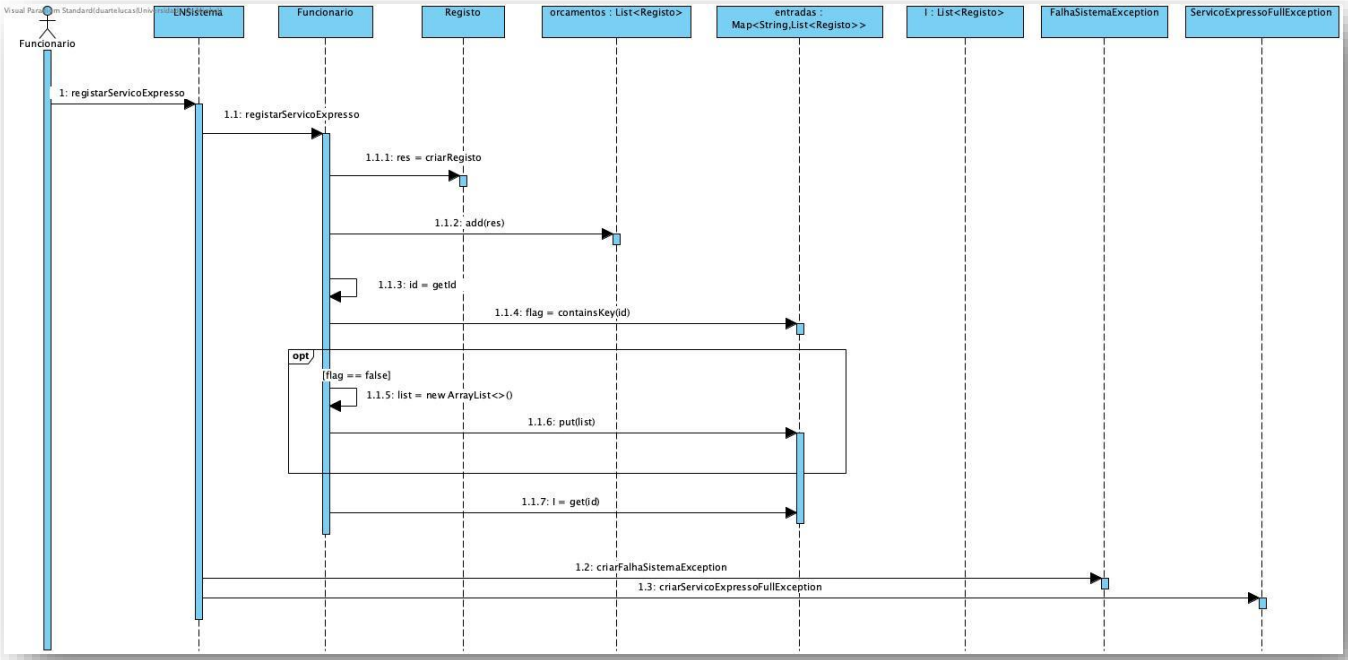

Figura 18: Registar Serviço Expresso

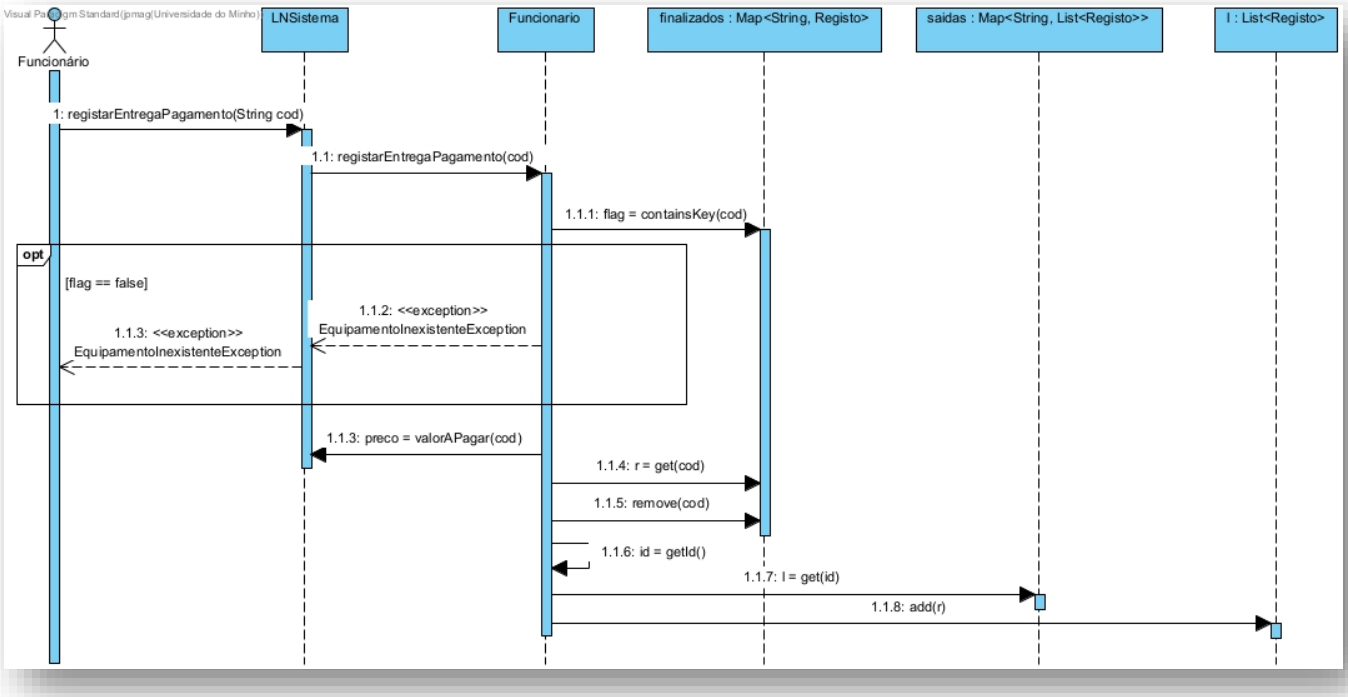

Figura 19: Levantar Equipamento

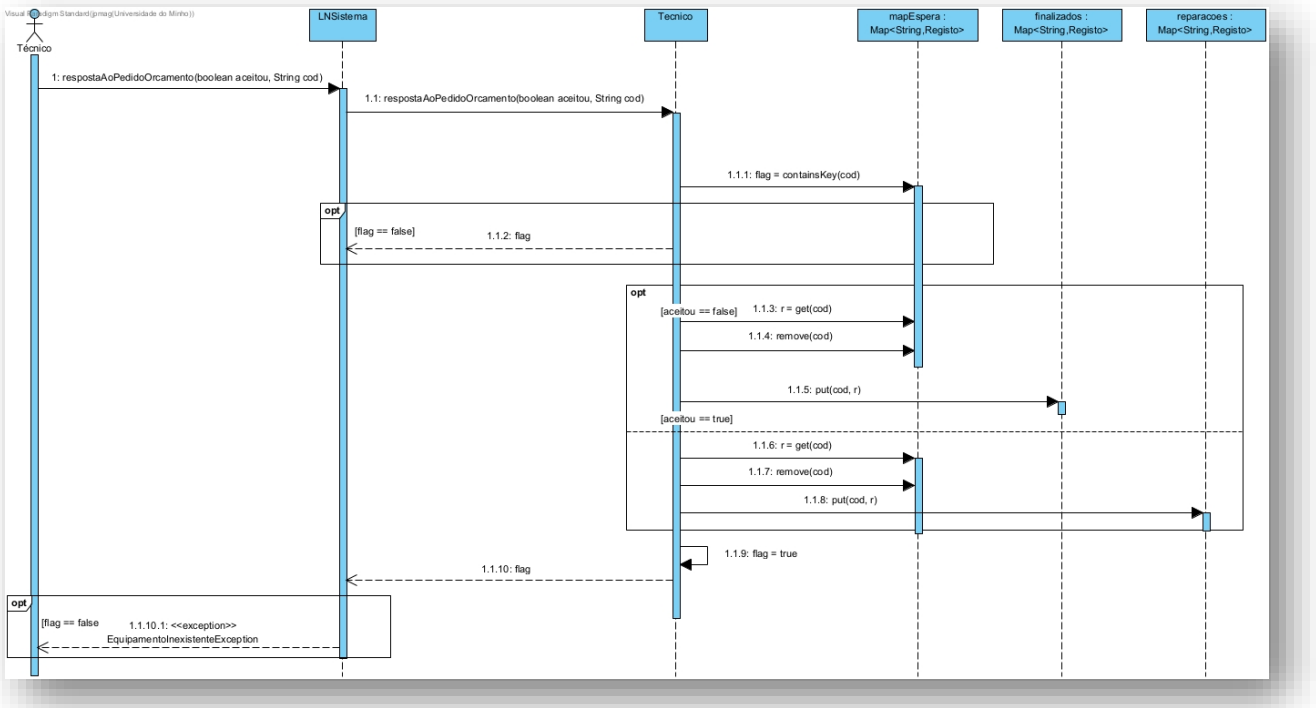

Figura 20: Calcular Orçamento

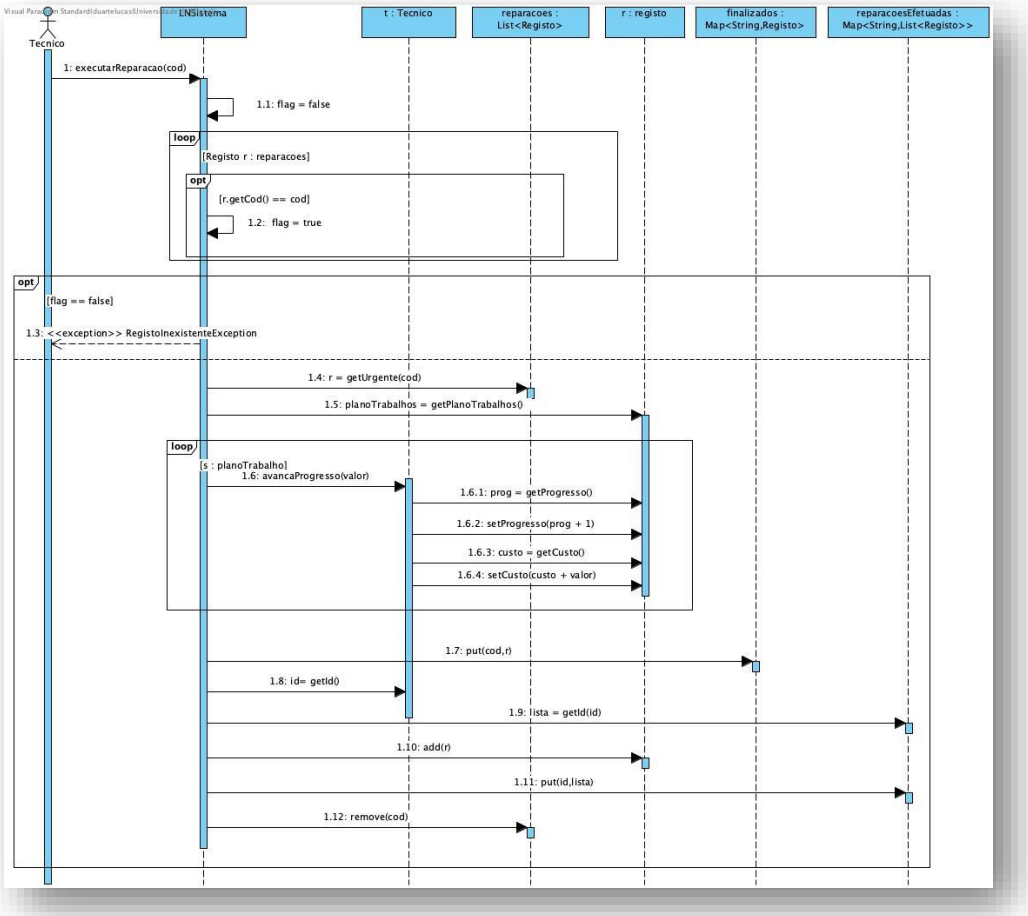

Figura 21: Executar Reparação

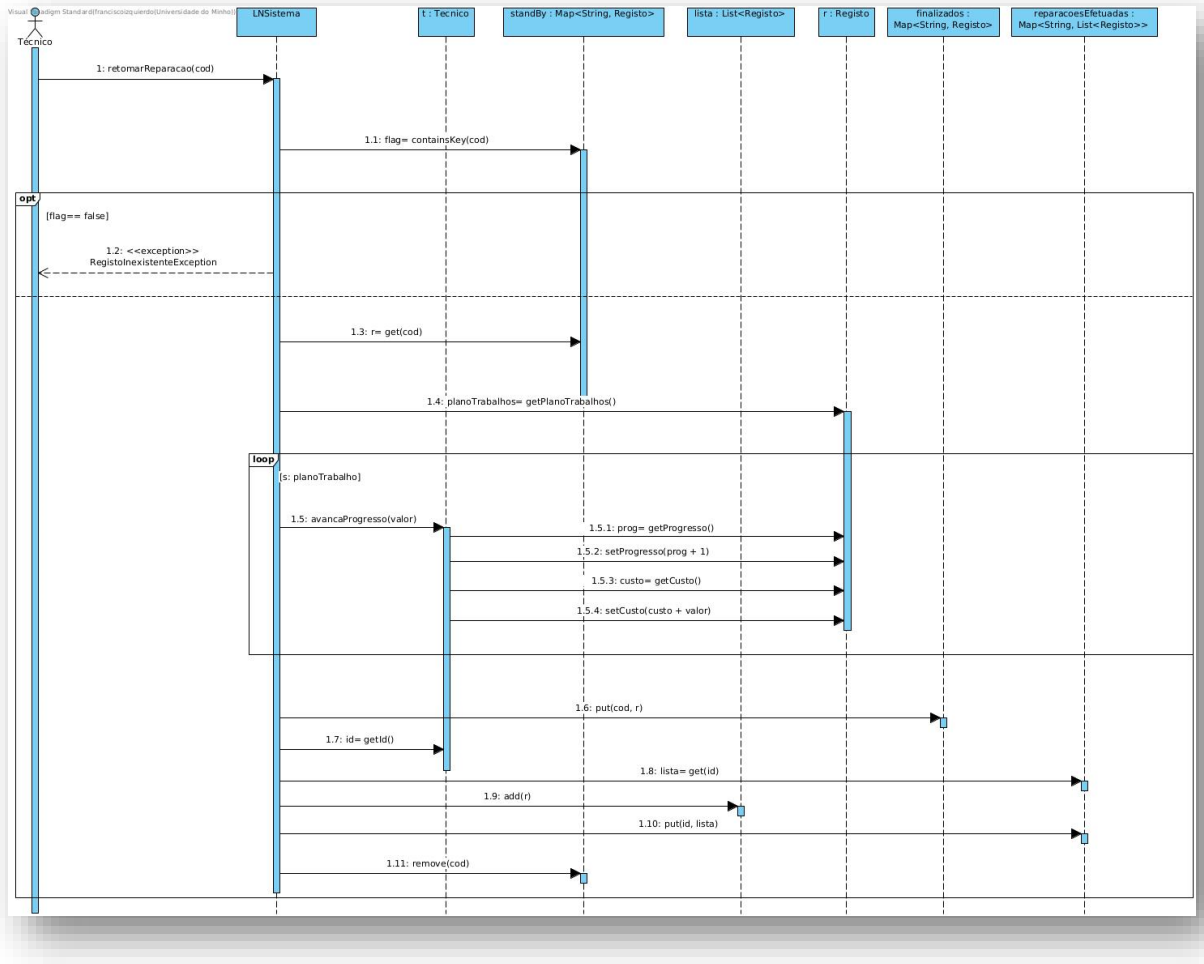

Figura 22: Retomar Reparação

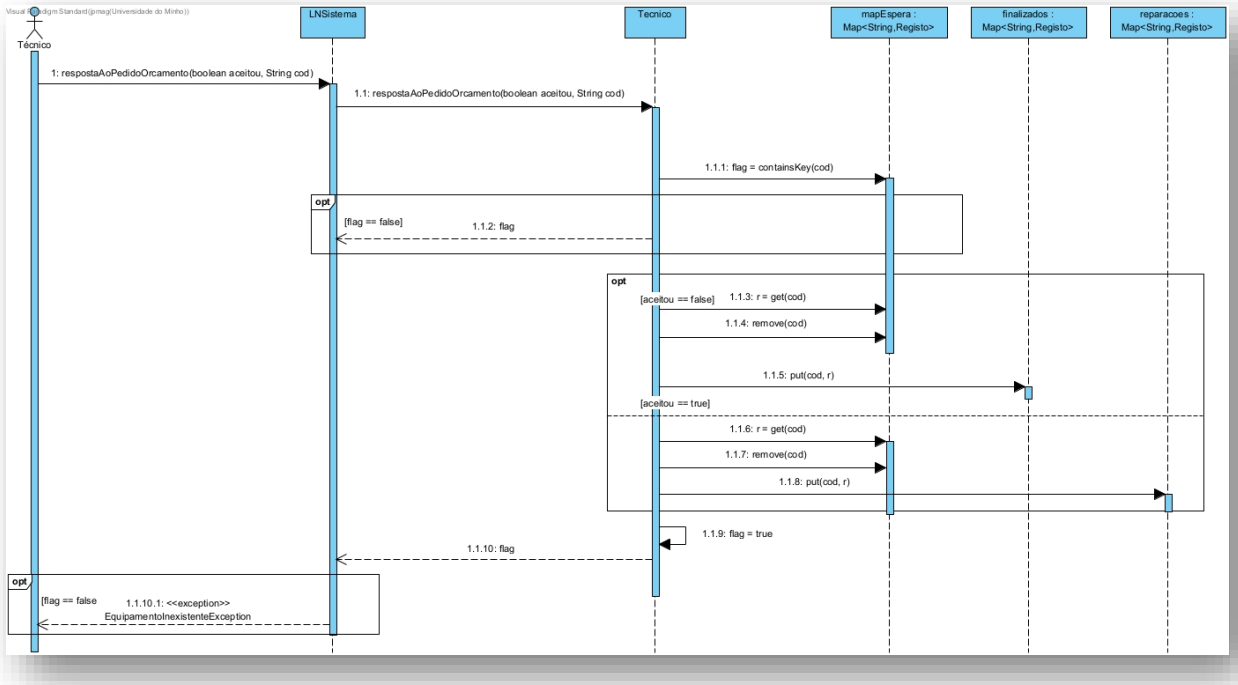

Figura 23: Confirmar Orçamento

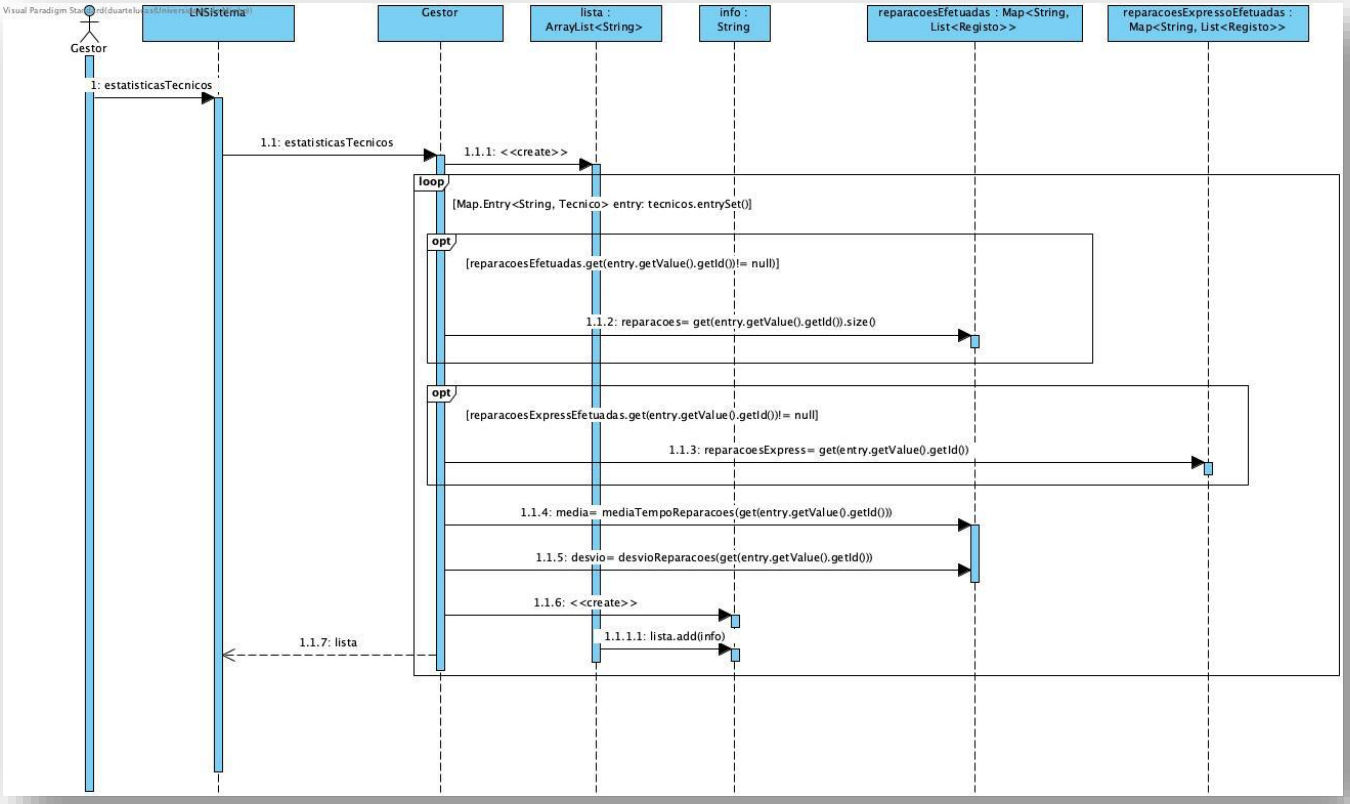

Figura 24: Consulta Estatísticas Técnicos

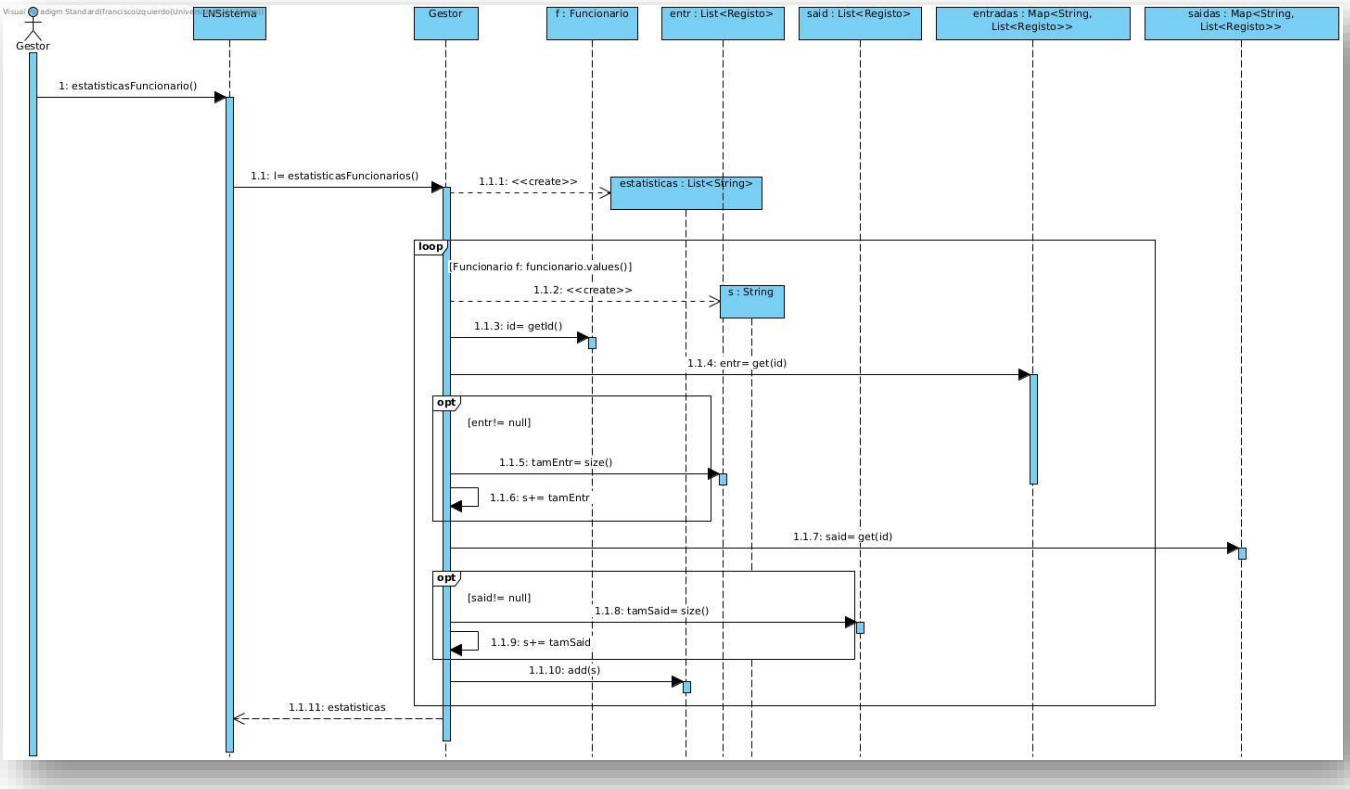

Figura 25: Consulta Estatísticas Funcionários

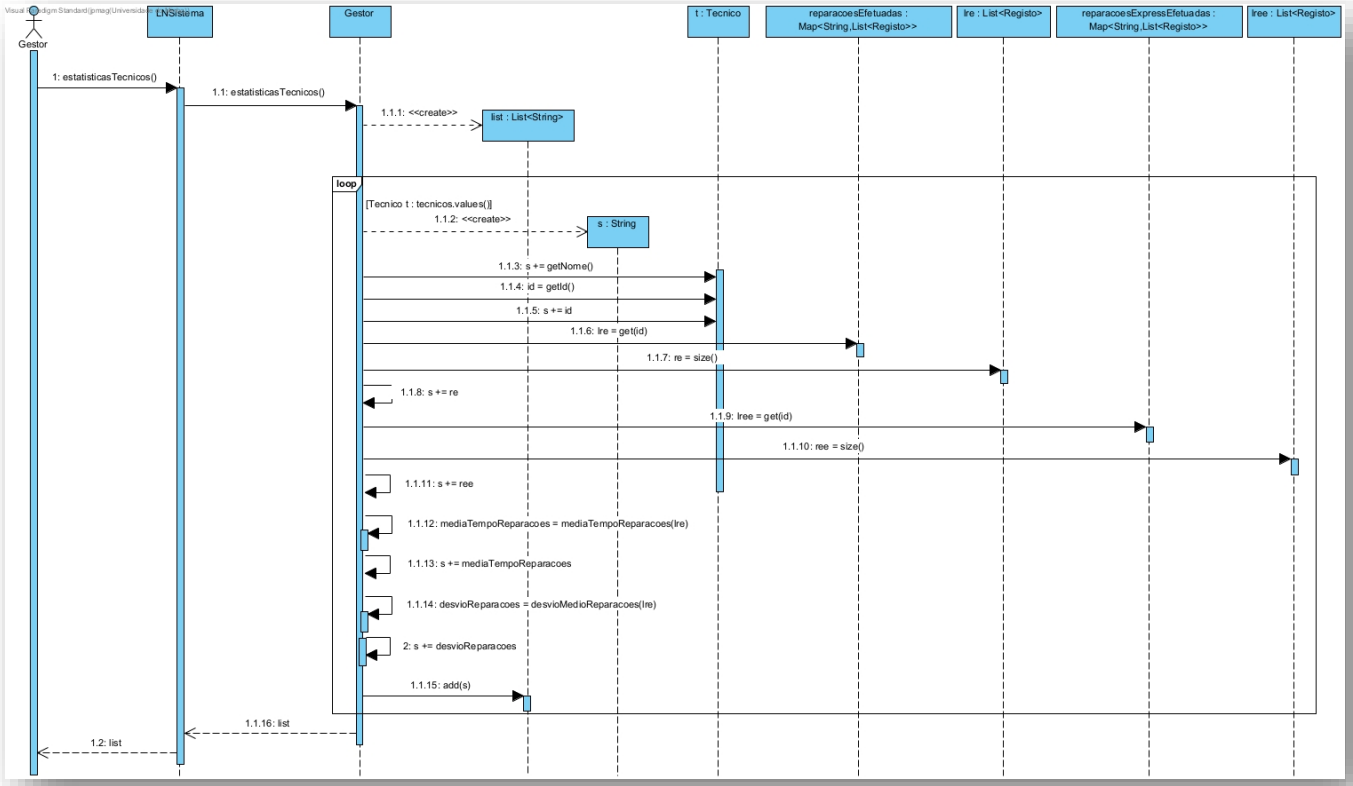

Figura 26: Consulta Lista de Reparações

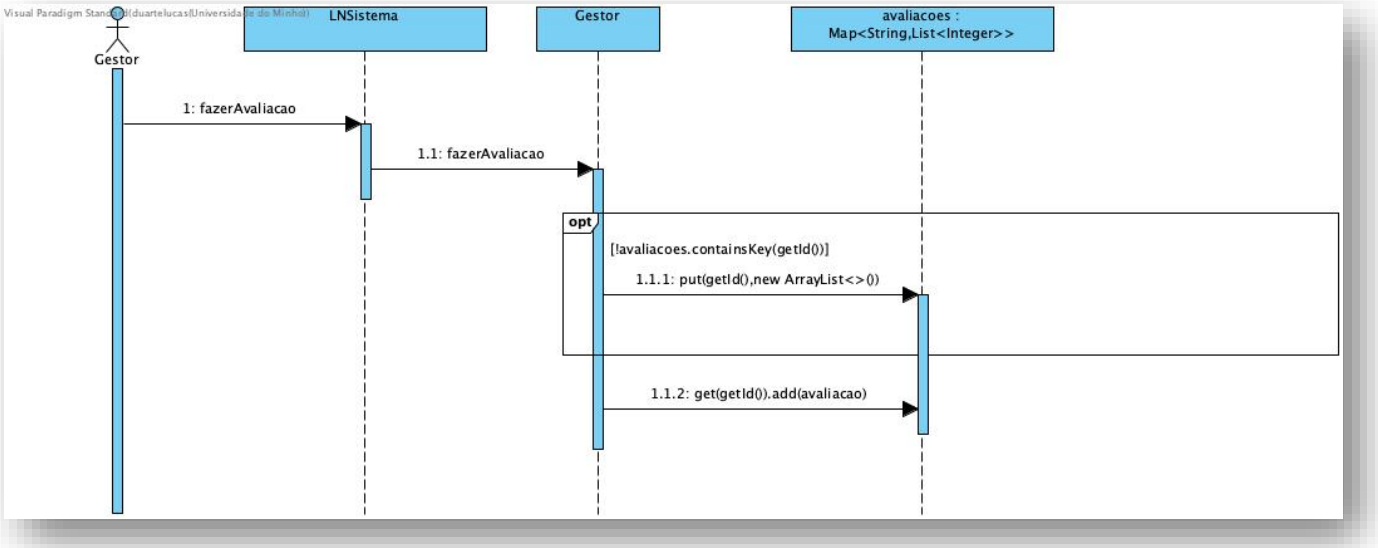

Figura 27: Avaliar o Centro

Implementação

Funcionalidades:

- Menu Inicial:
  - Autenticar Funcionário
  - Autenticar Técnico
  - Autenticar Gestor

- Menu do Funcionário:
  - Registar serviço expresso ™
  - Pedido de Orçamento
  - Levantar Equipamento

- Menu do Técnico:
  - Calcular orçamento
  - Executar reparação
  - Retomar reparação
  - Confirmar orçamentos

- Menu do Gestor:
  - Consultar estatísticas dos técnicos
  - Consultar estatísticas dos funcionários
  - Consultar lista de reparações
  - Atribuir avaliação ao centro
  - Ver avaliações

Classe Utilizador:

Esta classe é abstrata e contém a informação base de qualquer tipo Utilizador(id, Nome, Palavra-Passe), assim como os seus métodos necessários(gets e sets).

Classe Funcionário:

A classe funcionário estende a classe Utlizador, usando as variáveis da classe Uilizador. Possui os seus respetivos métodos e necessários (gets e sets). Esta classe regista um pedido de orçamento, um serviço express e por fim uma entrega ao cliente e o respetivo pagamento do equipamento.

Classe Técnico:

A classe técnica estende a classe Utlizador, usando as variáveis da classe Uilizador. Possui os seus respetivos métodos e necessários(gets e sets). É nesta classe onde se realiza um plano de trabalho a um pedido de orçamento, um orçamento e onde se atribui um tempo ao pedido mais antigo armazenado.

Classe Gestor:

A classe Gestor estende a classe Utlizador, usando as variáveis da classe Uilizador. Possui os seus respetivos métodos e necessários(gets e sets). Nesta classe, permitimos adicionar um funcionário ao centro de reparação, como também permite adicionar um técnico ao centro de reparação. Conseguimos devolver a informação relevante de todos os funcionáriose ainda permite ao gestor fazer uma avaliação do centro.

Classe Registo:

Nesta classe guardamos o registo de dois tipos de equipamentos, os equipamentos com um pedido de avaria “normal” e outro com um pedido de uma avaria express. Nesta mesma classe geramos o código de cada avaria, associado ao registo.

Classe LNSistema:

Esta classe é de extrema relevância na arquitetura usada, o padrão Facade, uma vez que estabelece a ligação entre a interface do utilizador e a lógica de negócio do sistema. Assim, permite que a interação efetuada pelo utilizador proveniente da classe Menu tenha uma

resposta de volta, sendo processada nesta classe, havendo uma fachada para com o utilizador que não sabe o teor do processamento que é feito internamente.

Classe SystemInfo:

Esta classe tem como objetivo armazenar a informação relevante ao funcionamento do sistema, sendo uma classe estática e que permite aquando da execução do sistema, guardar informação dinâmica. Assim, o propósito desta classe é o de uma base de dados, que guarde a informação de forma concisa e legível.

- **Diagrama de Classe**

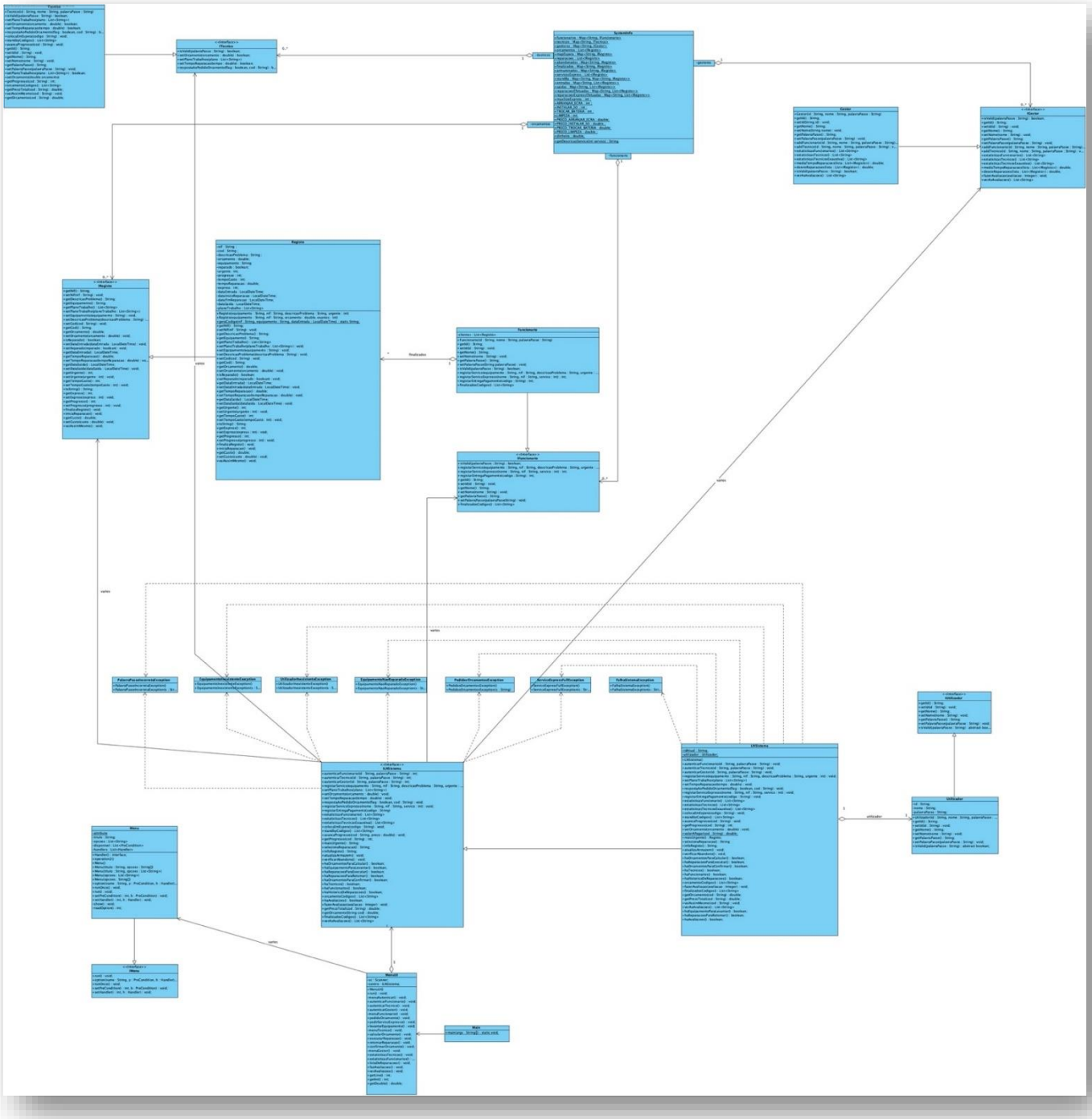

Figura 28: Diagrama de Classes

- **Diagrama de Máquina de Estado**

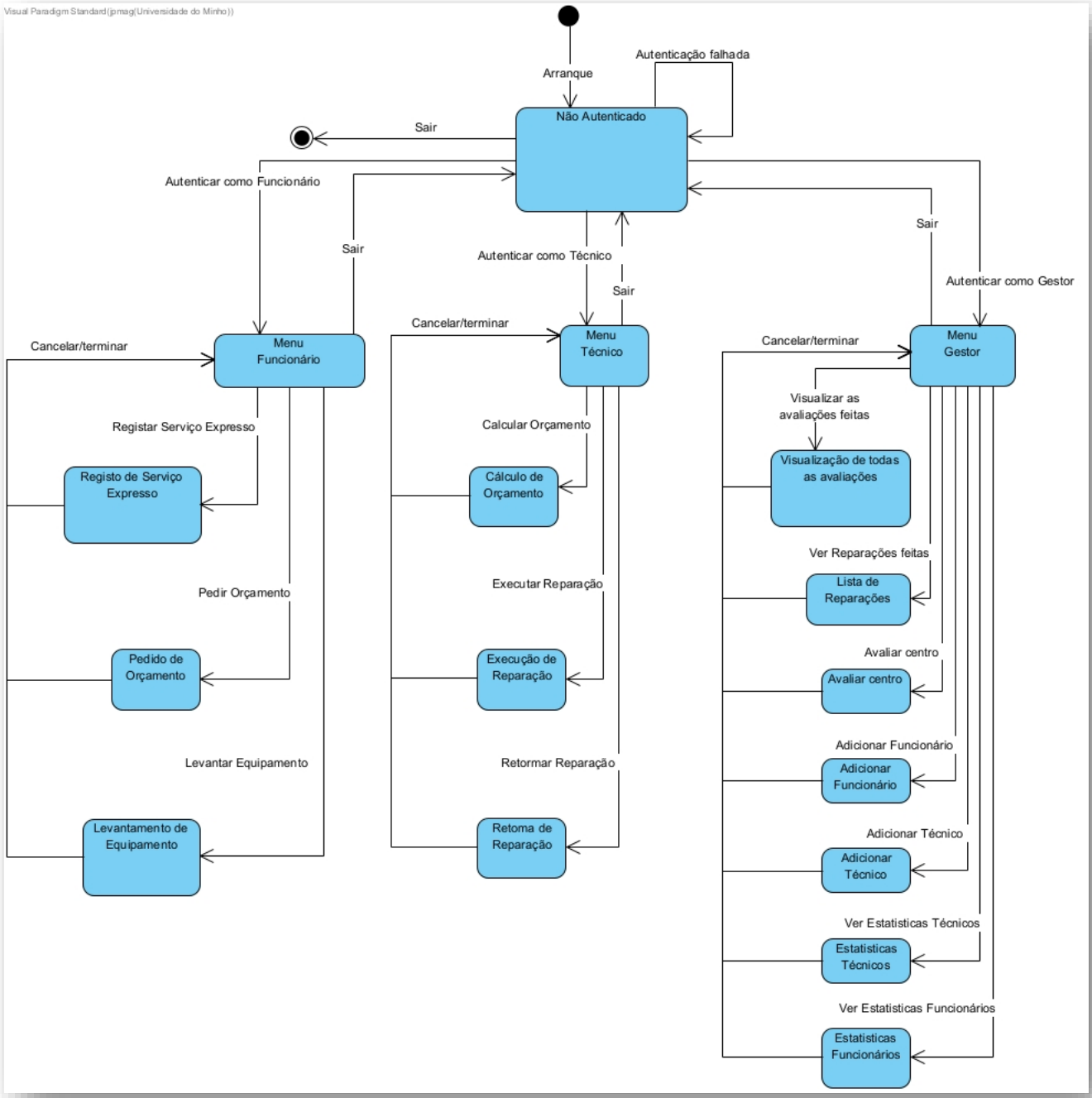

Figura 29: Diagrama de Máquina de Estado

- **Diagrama de Package**

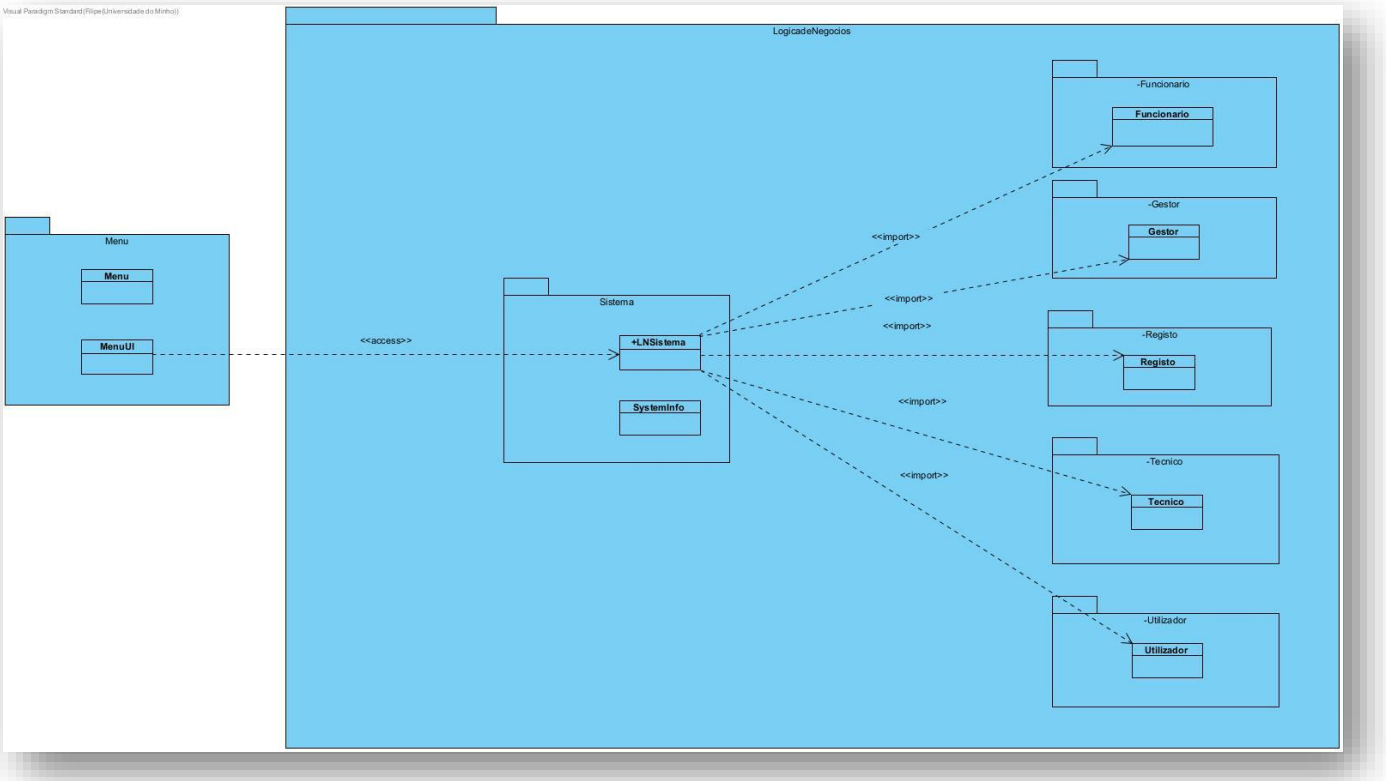

Figura 30: Diagrama de Package

Conclusão/Resultados Obtidos

Este trabalho revelou-se uma forma muito eficaz de consolidar e pôr em prática todo o conhecimento obtido na disciplina.

Enquanto grupo, tínhamos como objetivo fazer um planeamento que refletisse da forma mais aproximada possível aquela que seria a implementação final do código, isto é, gostaríamos que todos os diagramas e use cases fossem claros e bem estruturados de maneira que se um programador os seguisse estritamente acabasse por obter um código não só funcional, mas capaz de cumprir todos os requisitos esperados do trabalho. Isto acabou por se revelar um pouco mais complicado do que tínhamos em mente, algo que apenas se tornou claro na fase de implementação. De modo a tentar corrigir alguns destes erros vimo-nos obrigados a fazer um considerável número de alterações nos nossos diagramas.

Com isto o trabalho mostrou que para a implementação de um sistema de software coerente e conciso é preciso planear e estruturar devidamente todos os passos que irão levar à implementação física do sistema objetivo. Assim, o desenvolvimento de um sistema de software mostrou ser uma mais-valia uma vez que nos permitiu retificar e repensar toda a planificação a ser ponderada em trabalhos futuros do mesmo teor.