

# Desenvolvimento de Sistemas de Software

Trabalho prático  
Grupo:14

Repositório: <https://github.com/LEI-DSS/trabalho-dss-grupo-14>



Diogo Pinto A100551



João Magalhães A100740



Jorge Rodrigues A101758



Mariana Pinto A100756



Rodrigo Gomes A100555

# Índice

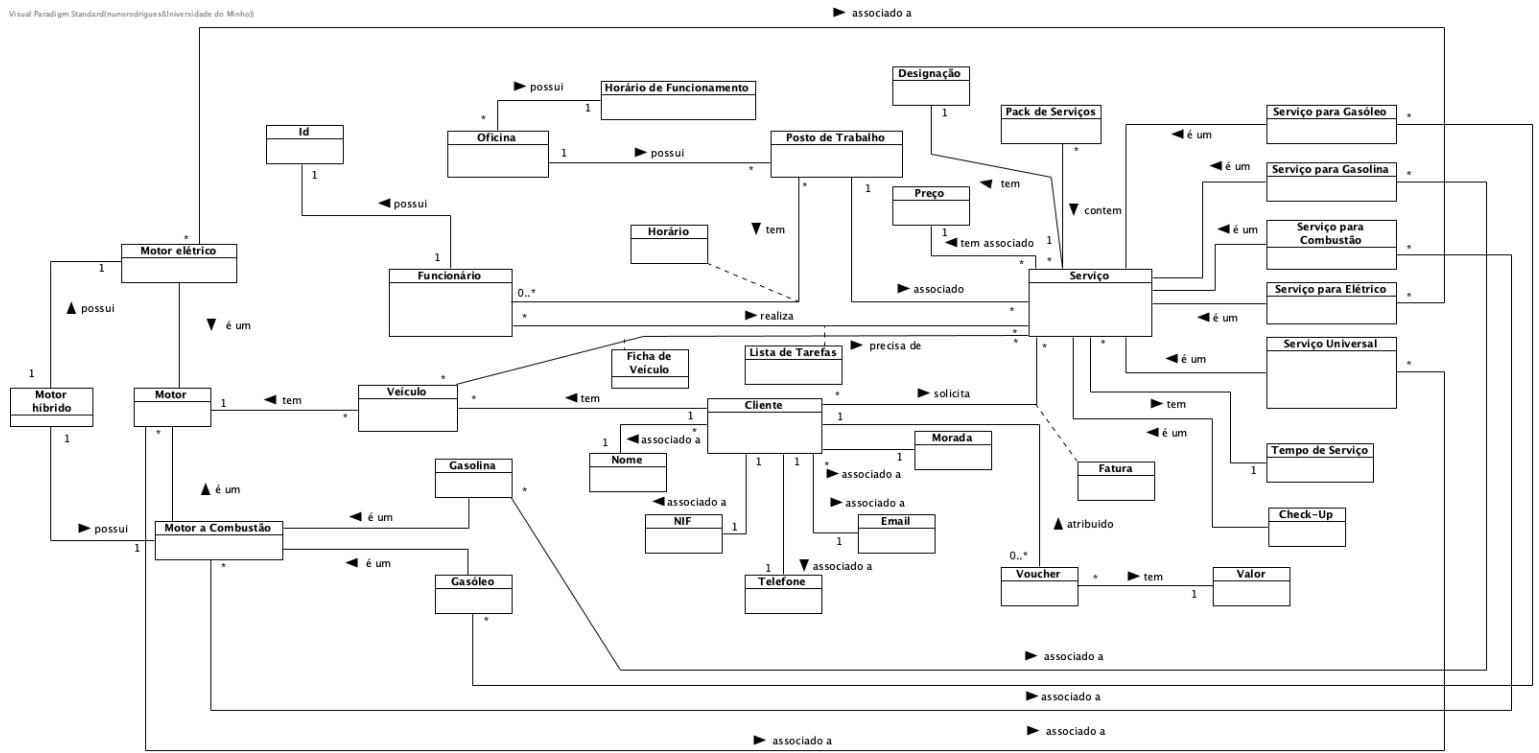
<i>Introdução</i> .....	3
<i>Análise de requisitos</i> .....	4
Modelo de domínio .....	4
Diagrama de Use Cases .....	4
Especificação (apenas os requeridos na adenda) .....	6
Diagramas de Atividade .....	8
<i>Modelação Conceptual da solução</i> .....	9
Diagrama de classes.....	9
Diagramas de Sequência .....	10
<i>Solução Implementada</i> .....	13
Diagrama de componentes .....	13
Diagrama de classes.....	14
Diagrama de Packages .....	14
Diagrama de Sequência .....	14
Diagrama de Instalação (apenas uma possibilidade).....	20
<i>Resultados obtidos</i> .....	21

# Introdução

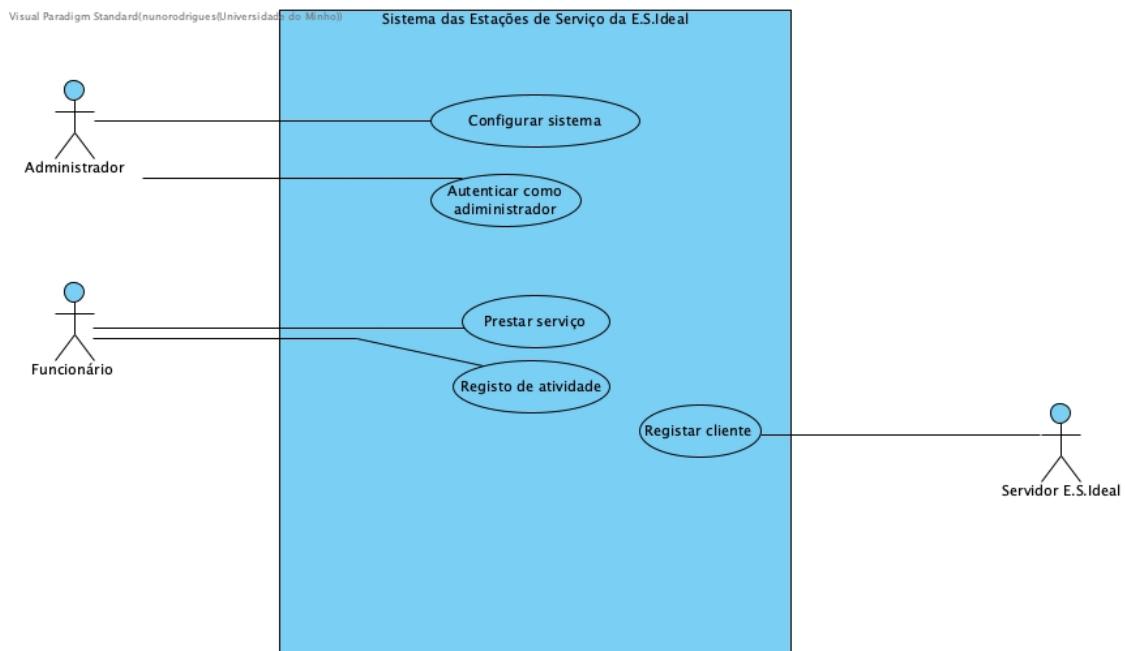
O presente trabalho ambiciona expor o progresso do grupo no desenvolvimento de uma aplicação para a *E.S.Ideal* com o objetivo de gerir o funcionamento das suas estações. Neste documento apenas serão expostos os diagramas resultantes da necessidade de analisar os requisitos, conceber um modelo conceptual e de apresentar a solução final. No final, consta ainda uma secção de resultados obtidos que visa, sumariamente, relatar o que foi desenvolvido em cada fase.

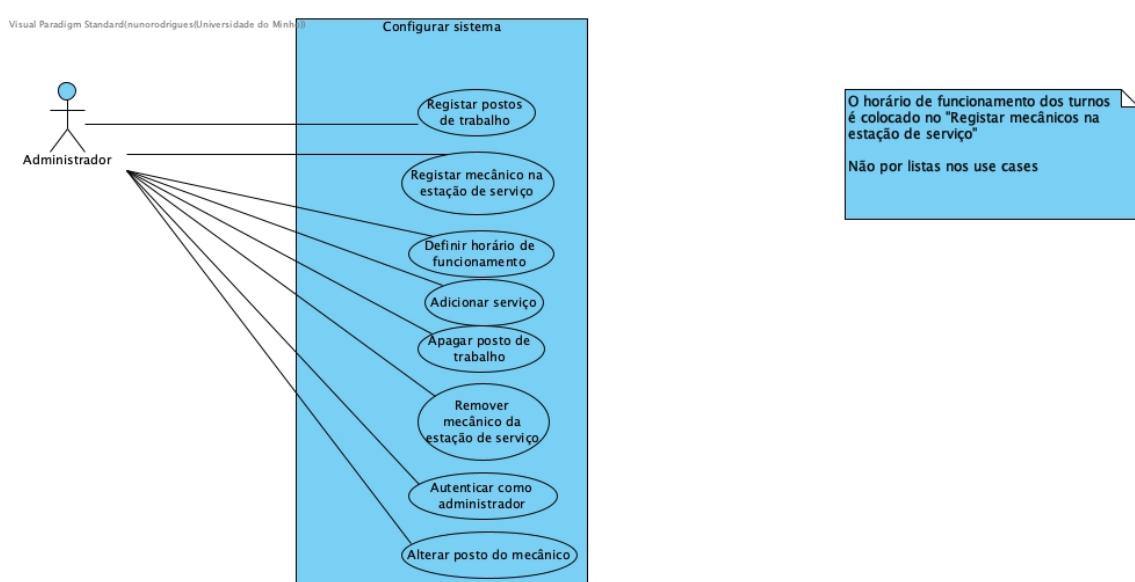
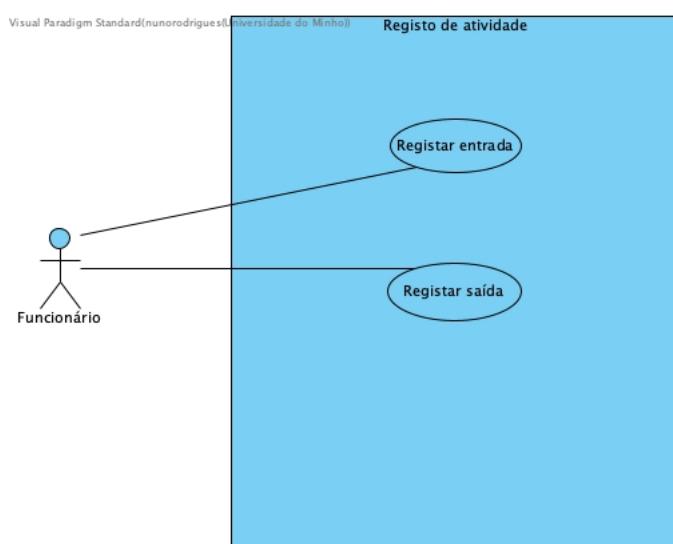
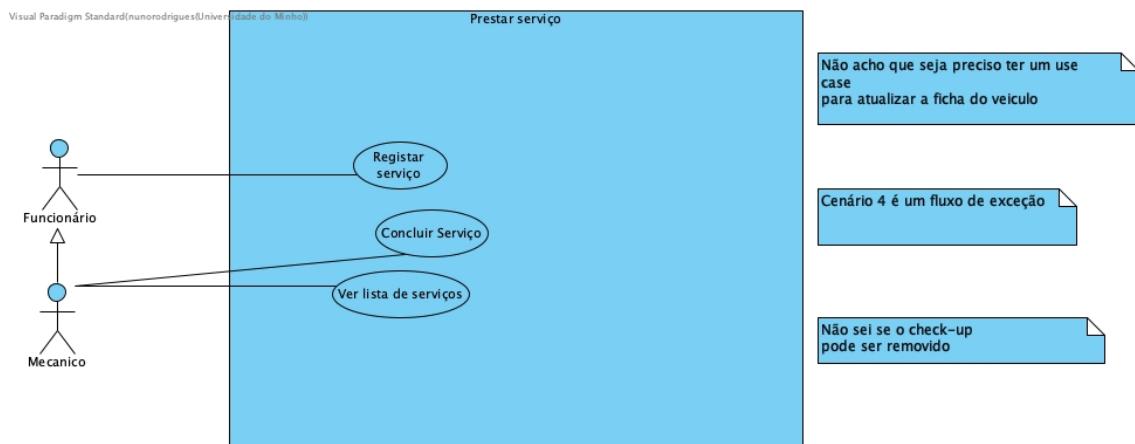
# Análise de requisitos

## Modelo de domínio



## Diagrama de Use Cases





Especificação (apenas os requeridos na adenda)

**USE CASE:** **Registar serviço**

**Descrição:** Funcionário regista um serviço

**Cenários:** 5 e 3

**PRÉ-CONDIÇÃO:** Funcionário dar entrada no sistema e ter competências para realizar o serviço

**PÓS-CONDIÇÃO:** O serviço é registado na ficha do veículo e no posto

**FLUXO NORMAL:**

1. O funcionário insere o serviço requisitado a matrícula do veículo e o NIF do cliente.
2. O sistema atualiza a ficha do veículo associado ao serviço.
3. Sistema atribui a um posto.

**FLUXO DE EXCEÇÃO:** [Não há disponibilidade] (passo 2)

- 3.1 Sistema determina que não será capaz de terminar o serviço até ao fim do dia.

**USE CASE:** **Concluir serviço**

**Descrição:** Mecânico realiza um serviço

**Cenários:** 5 e 3

**PRÉ-CONDIÇÃO:** O mecânico dar entrada no sistema com o cartão de funcionário

**PÓS-CONDIÇÃO:** O serviço é concluído e a ficha do veículo é atualizada

**FLUXO NORMAL:**

1. O mecânico insere o id do serviço, a matrícula do veículo e o NIF do cliente.
2. O sistema atualiza a ficha do veículo associado ao serviço.
3. Sistema atribui a um posto.

**FLUXO ALTERNATIVO:** [Serviço Checkup] (passo 3)

- 3.1 Mecânico lista os ids dos serviços que precisam de ser realizados no veículo.
- 3.2 Mecânico indica a matrícula do veículo.
- 3.3 O sistema atualiza a ficha do veículo associado ao serviço.
- 3.4 Regressa a 3.

**FLUXO ALTERNATIVO:** [Ficha do Veículo vazia] (passo 3)

3.1 Sistema notifica o mecânico que a ficha está vazia.

**FLUXO ALTERNATIVO:** [Ficha do Veículo vazia] (passo 3)

- 3.1.1 Mecânico insere a necessidade de notificar cliente.
- 3.1.2 Sistema notifica o mecânico que a ficha está vazia e envia SMS ao Cliente.

**USE CASE:** **Ver a lista de serviços**

**Descrição:** Um mecânico vê uma lista de serviços no sistema

**Cenários:** 5

**Pré-Condição:** O mecânico dar entrada no sistema com o cartão de funcionário

**Pós-Condição:** O sistema disponibiliza a lista de serviços para o determinado utilizador

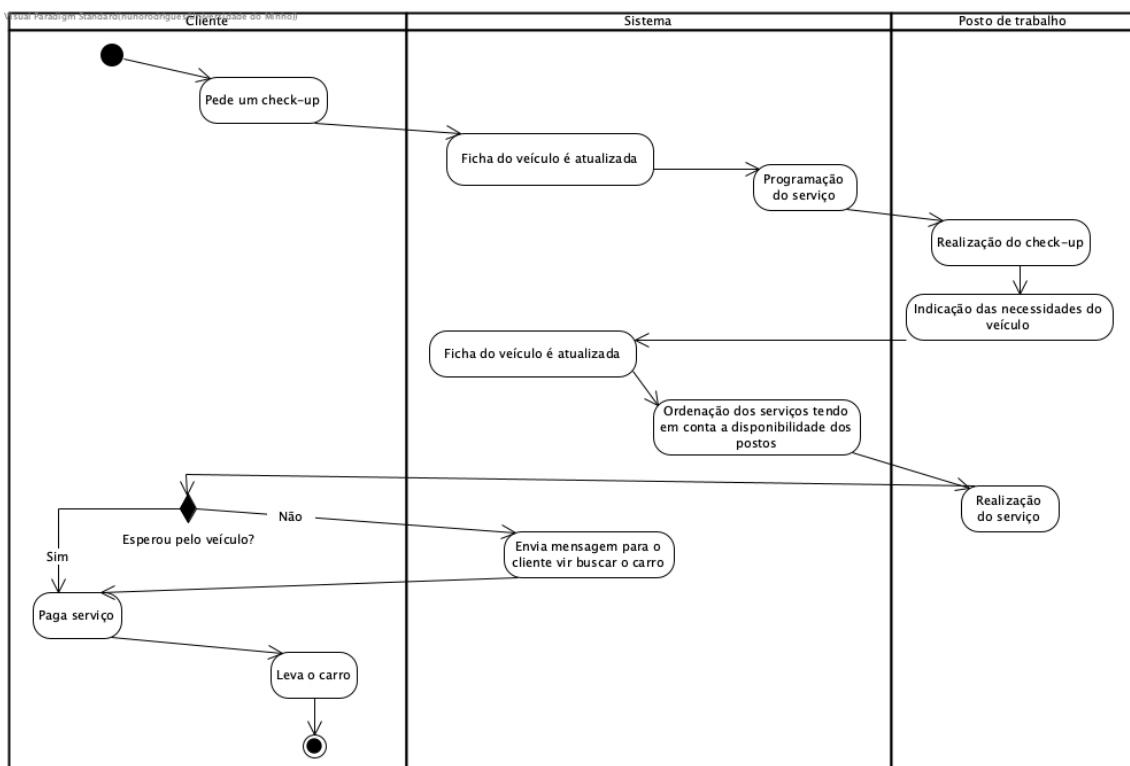
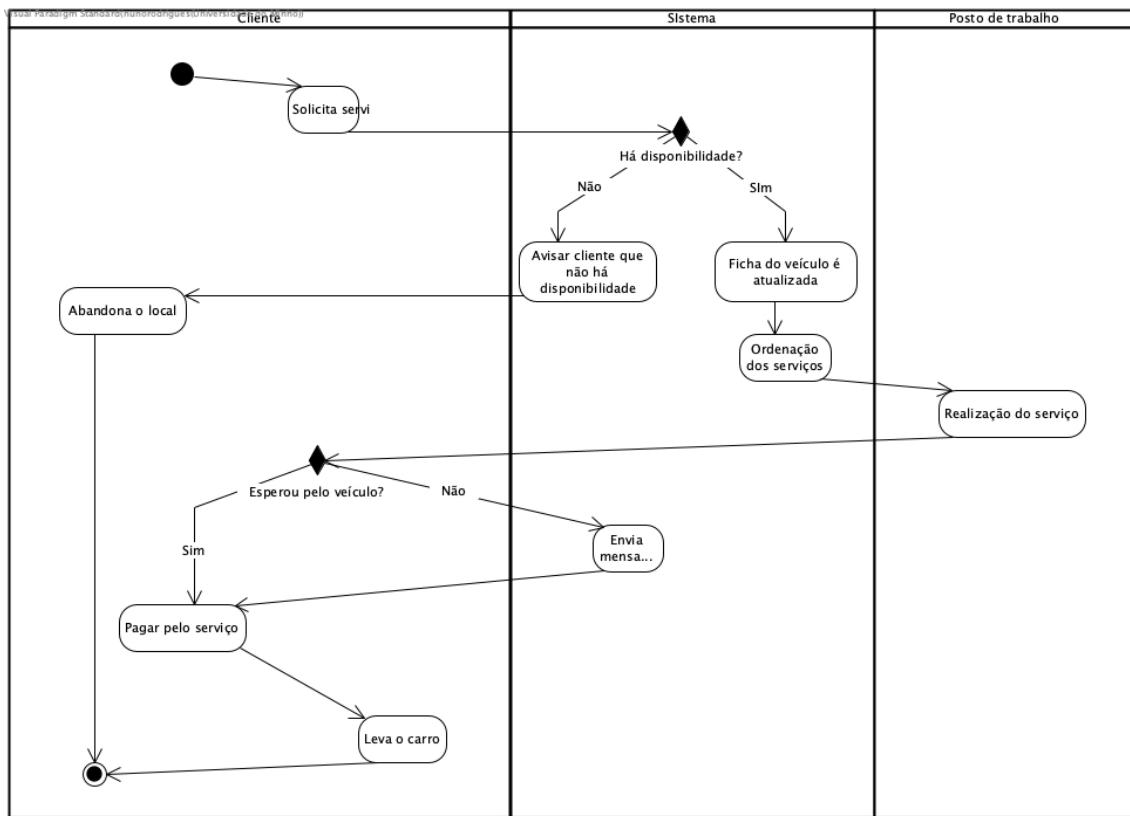
**Fluxo Normal:**

1. Mecânico indica o seu id e o id do seu posto de trabalho.
2. O sistema disponibiliza a lista de serviços.

**Fluxo de Exceção:** [Não há disponibilidade] (passo 2)

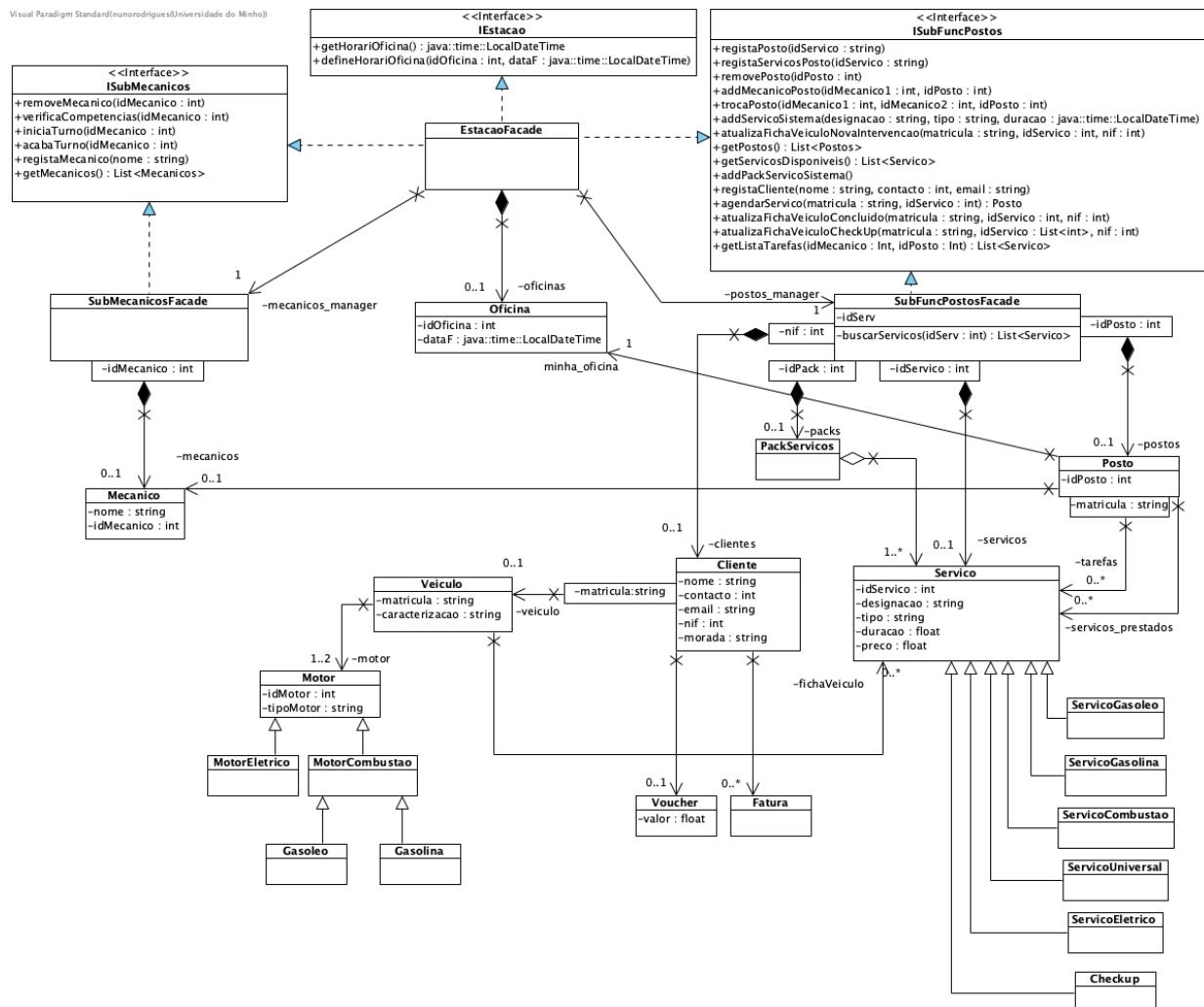
- 3.1. O sistema verifica e envia um alerta ao utilizador, indicando-o que não tem uma lista de serviços associada.

## Diagramas de Atividade



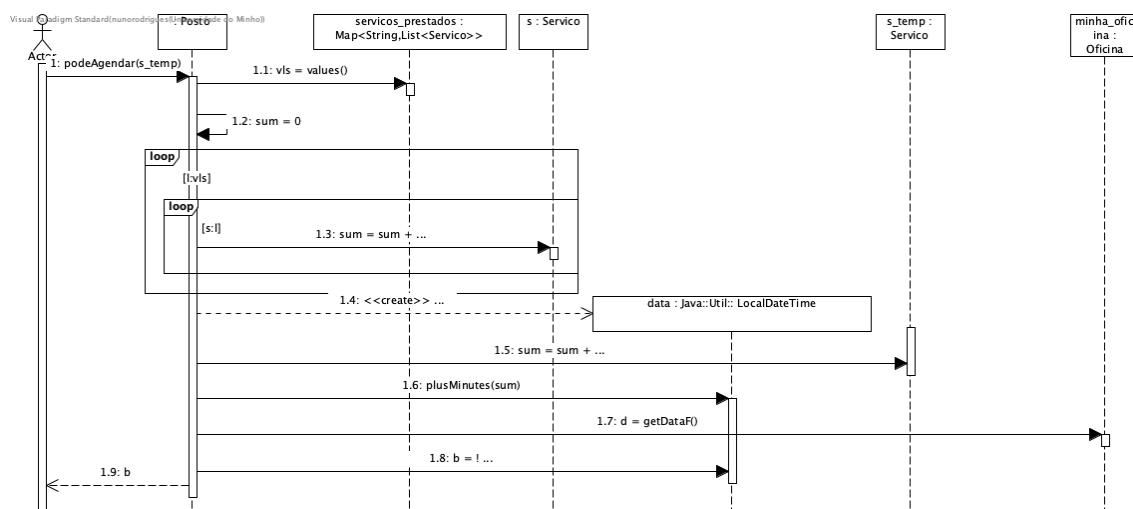
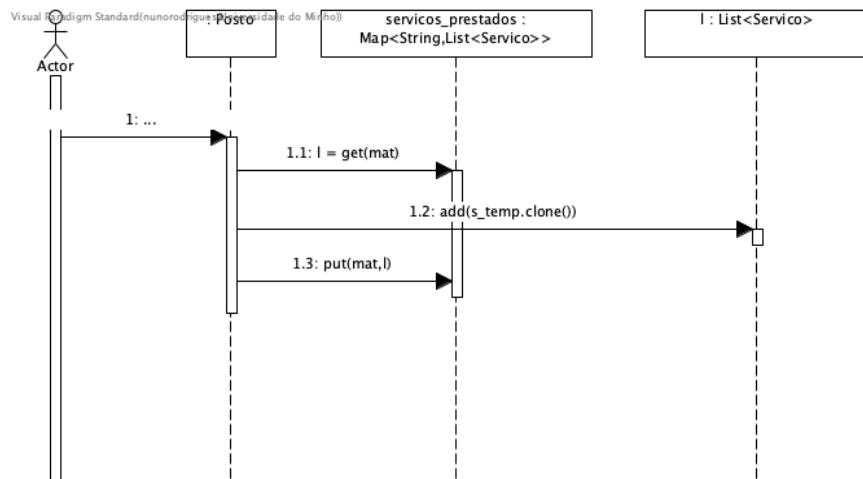
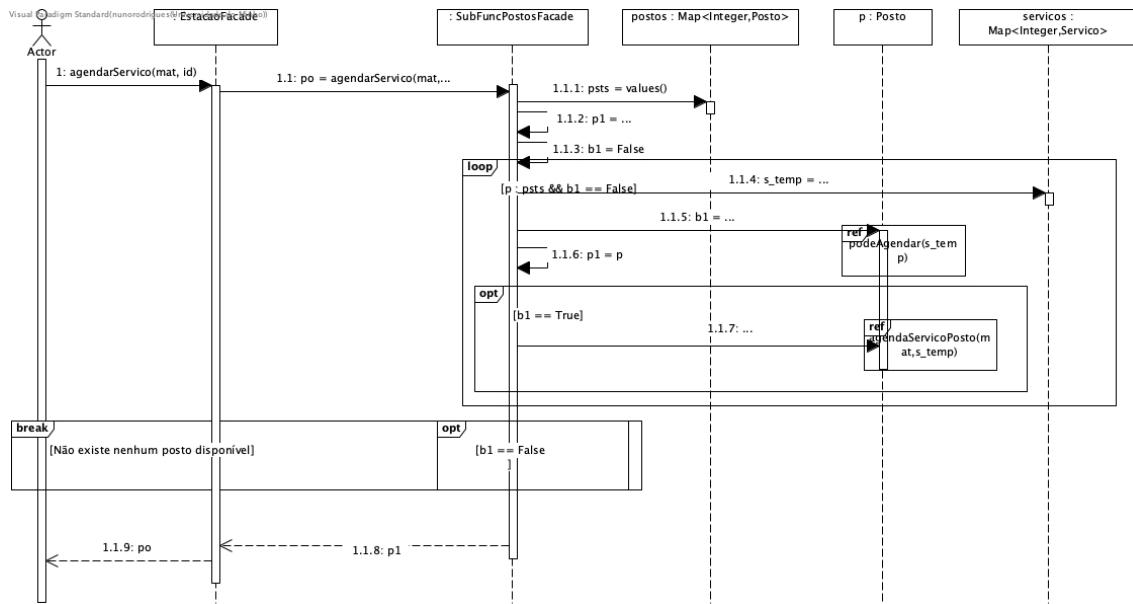
# Modelação Conceptual da solução

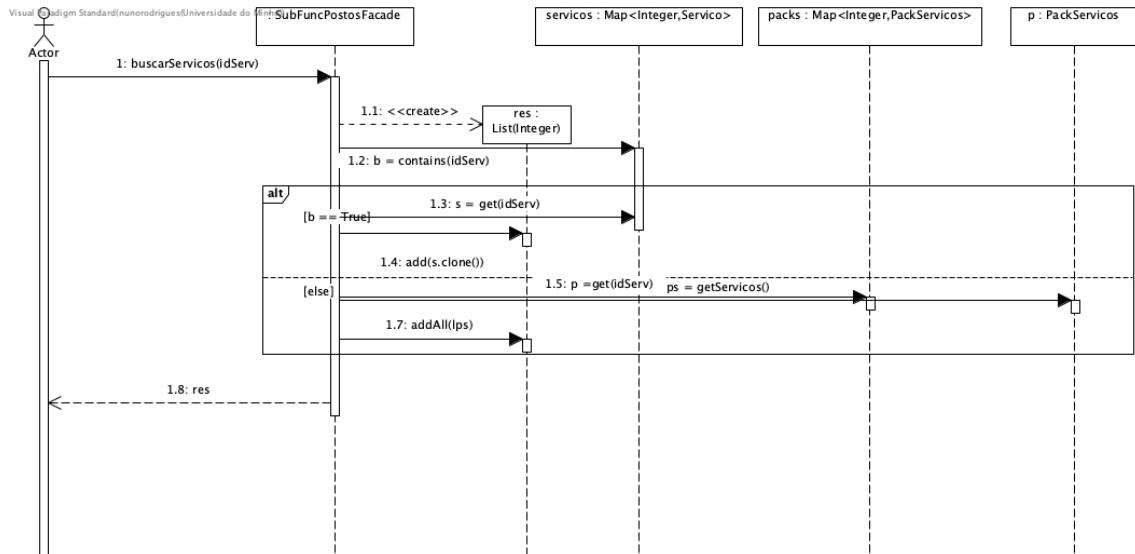
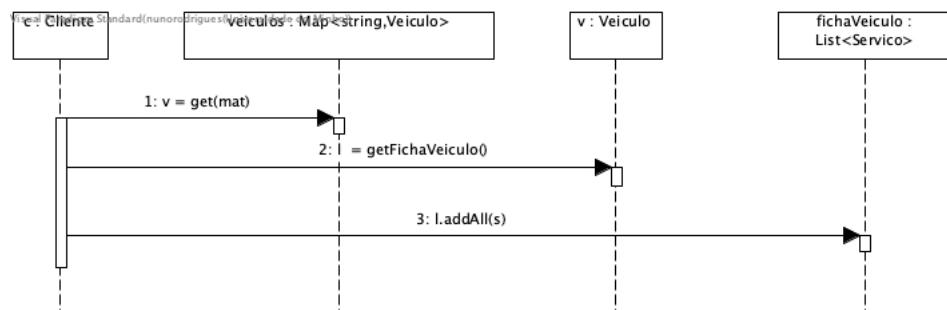
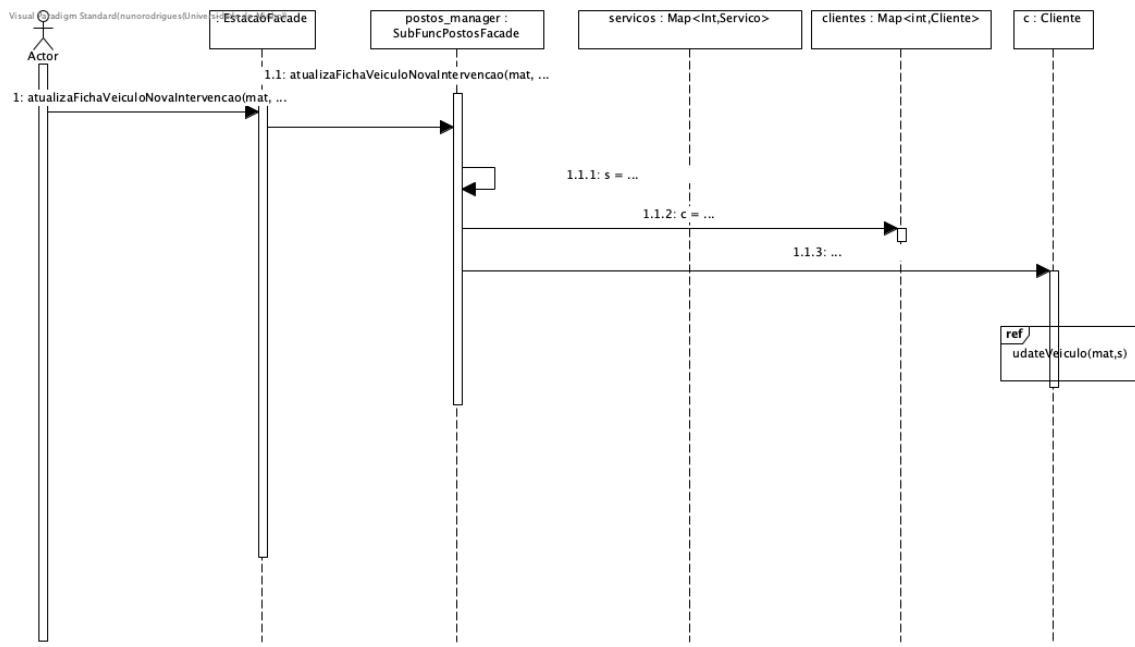
## Diagrama de classes



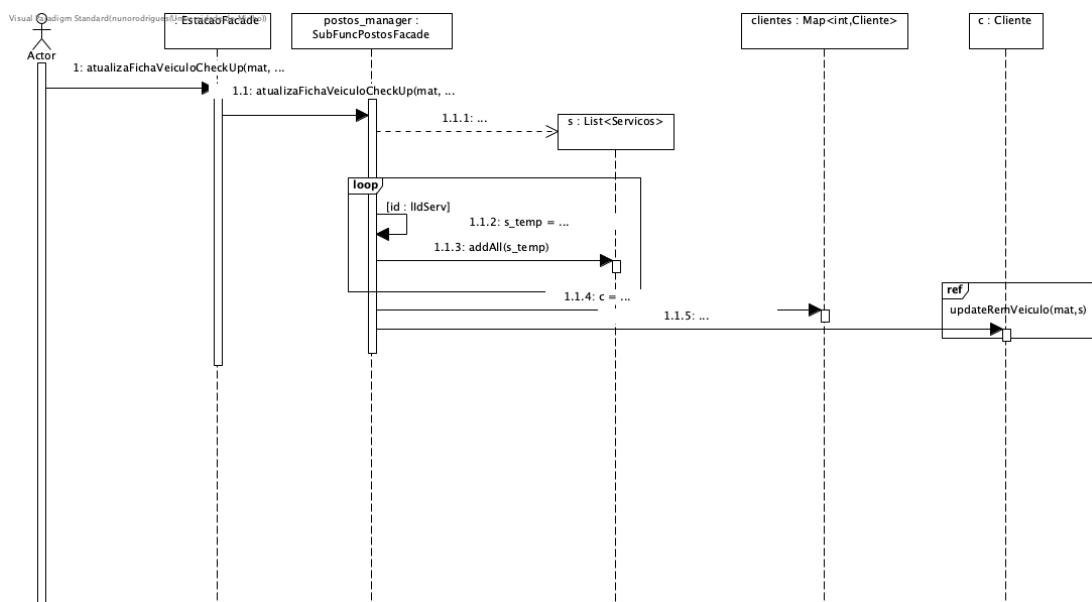
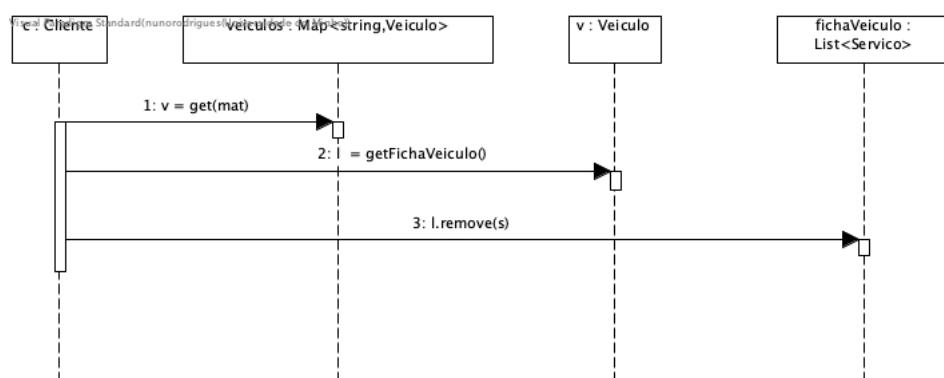
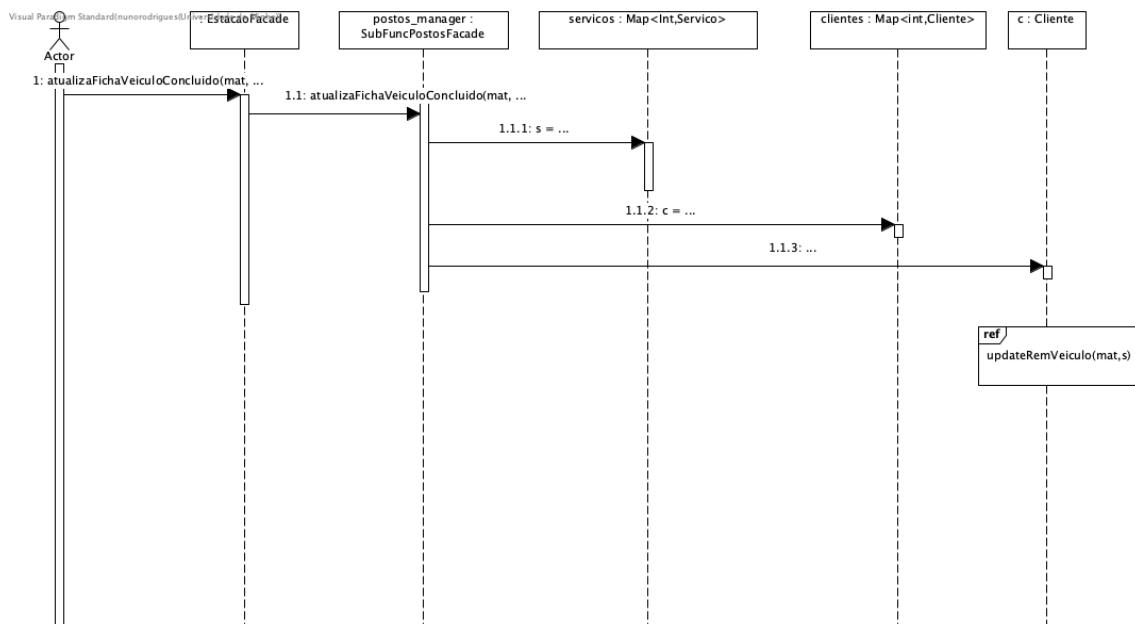
## Diagramas de Sequência

(Use case ->Registrar Serviço)

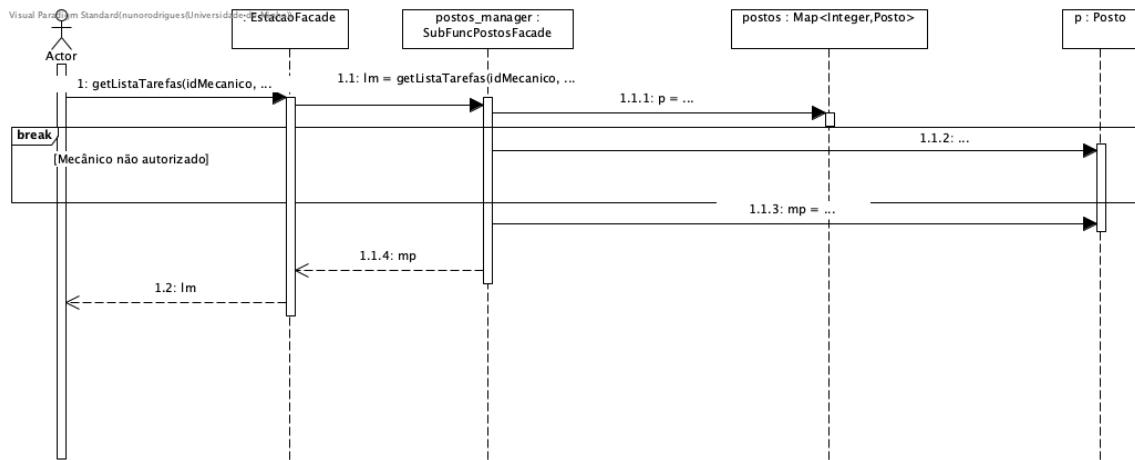




(Use case ->Concluir Serviço)

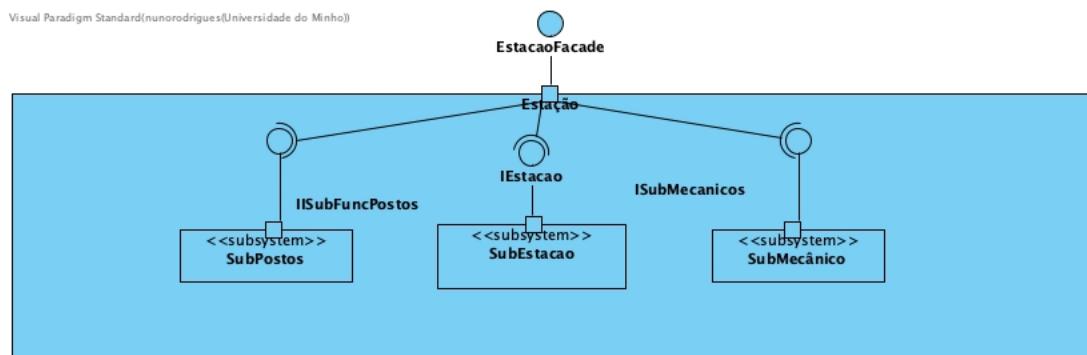


(Use case ->Ver a lista de Serviços)

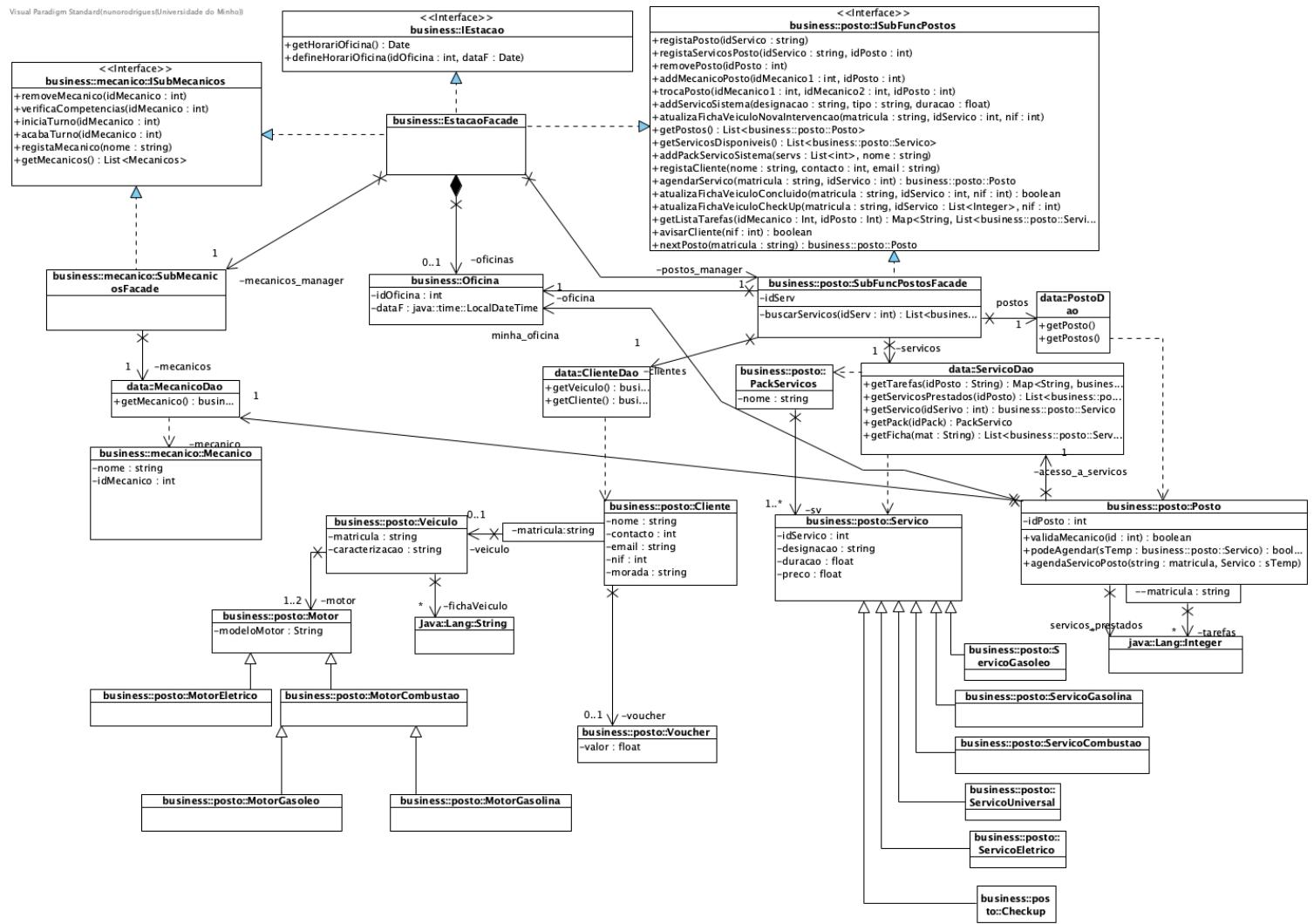


## Solução Implementada

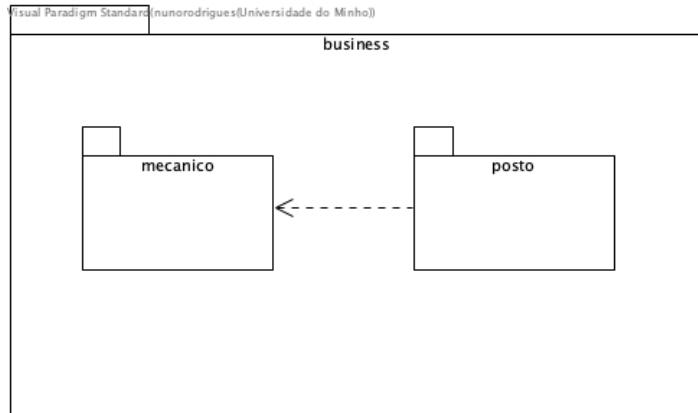
Diagrama de componentes



## Diagrama de classes

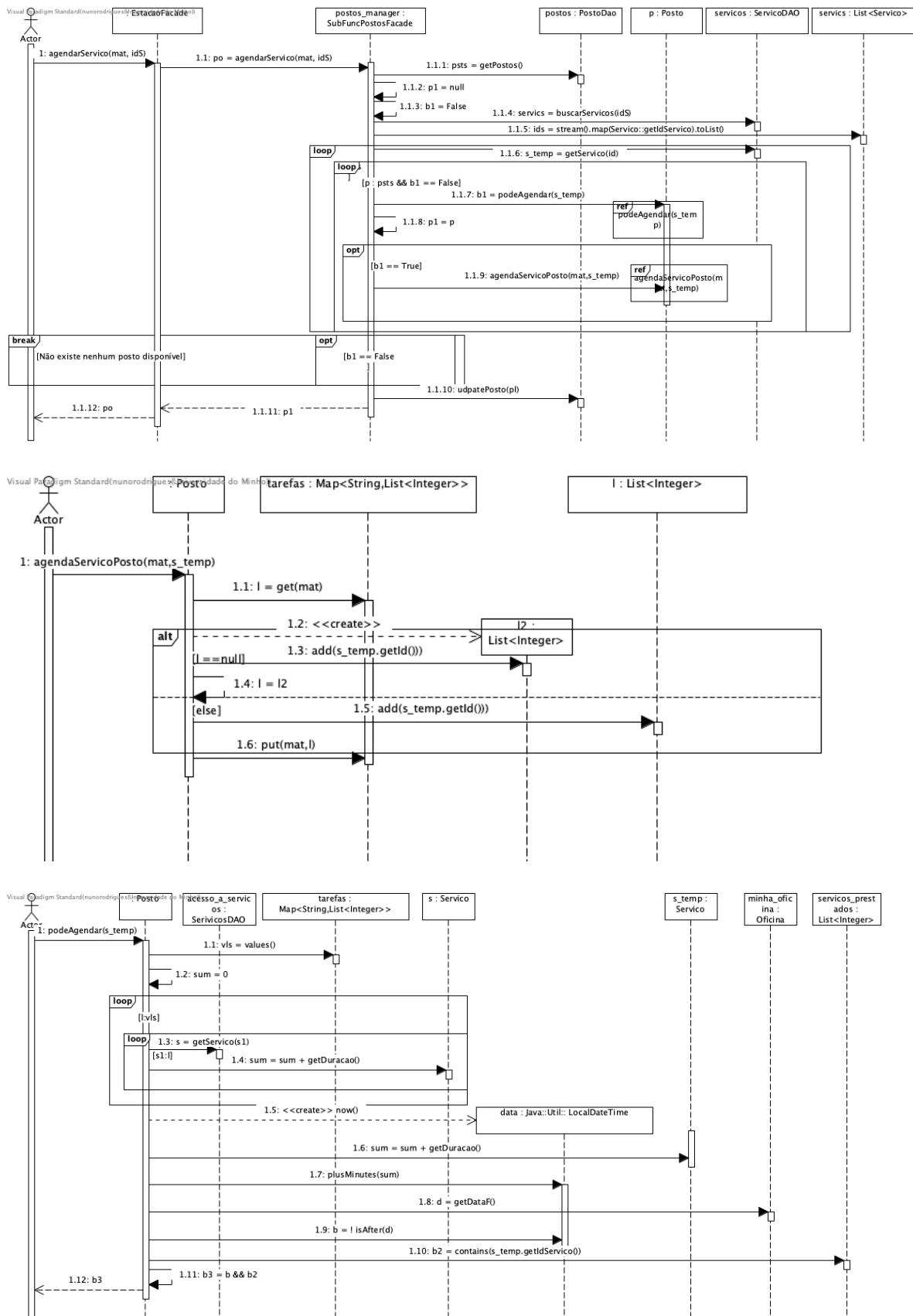


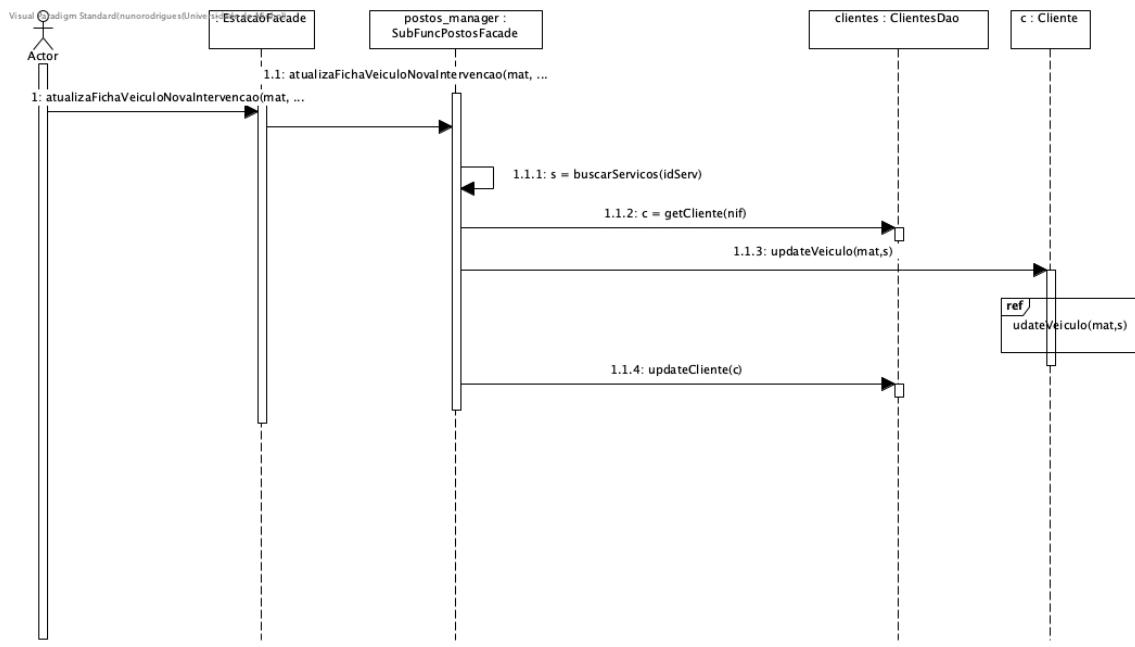
## Diagrama de Packages

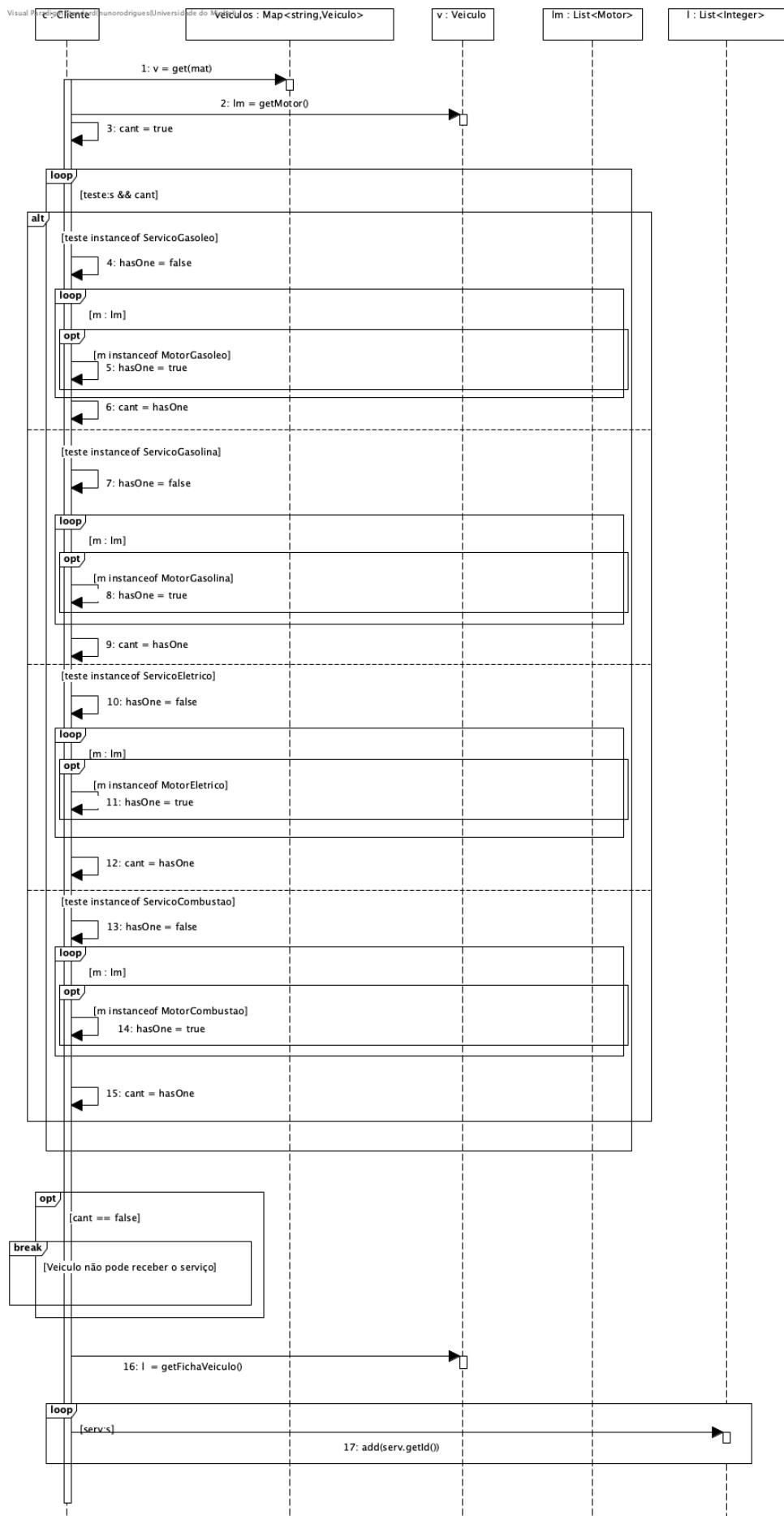


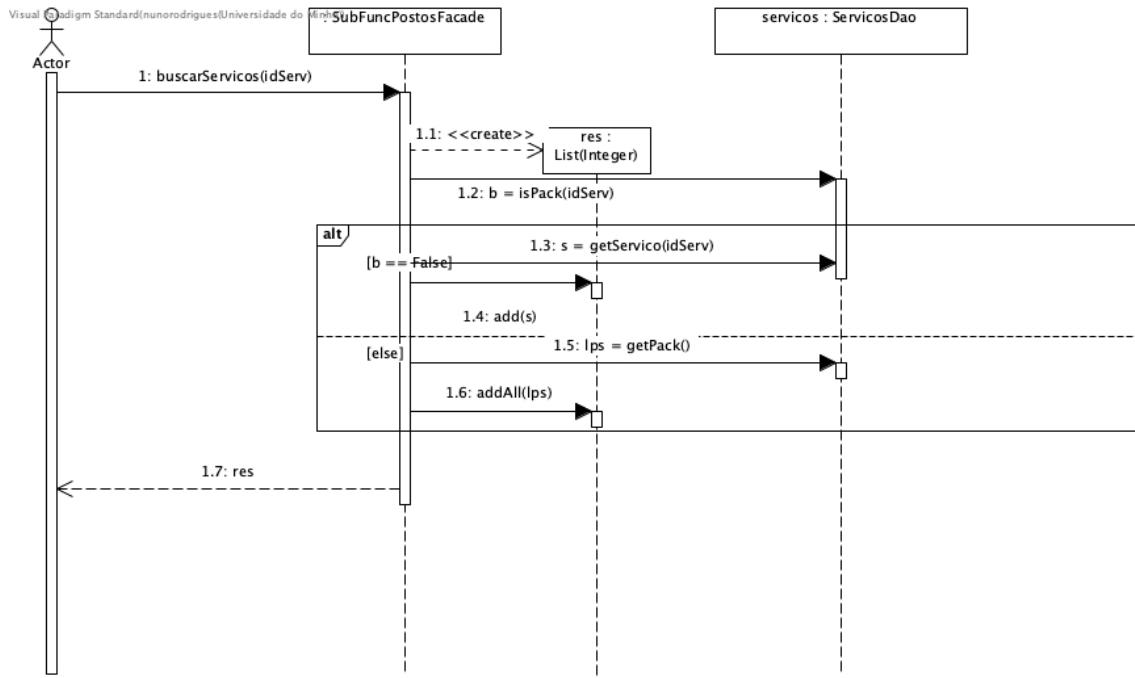
## Diagrama de Sequência

(Use case ->Registrar Serviço)

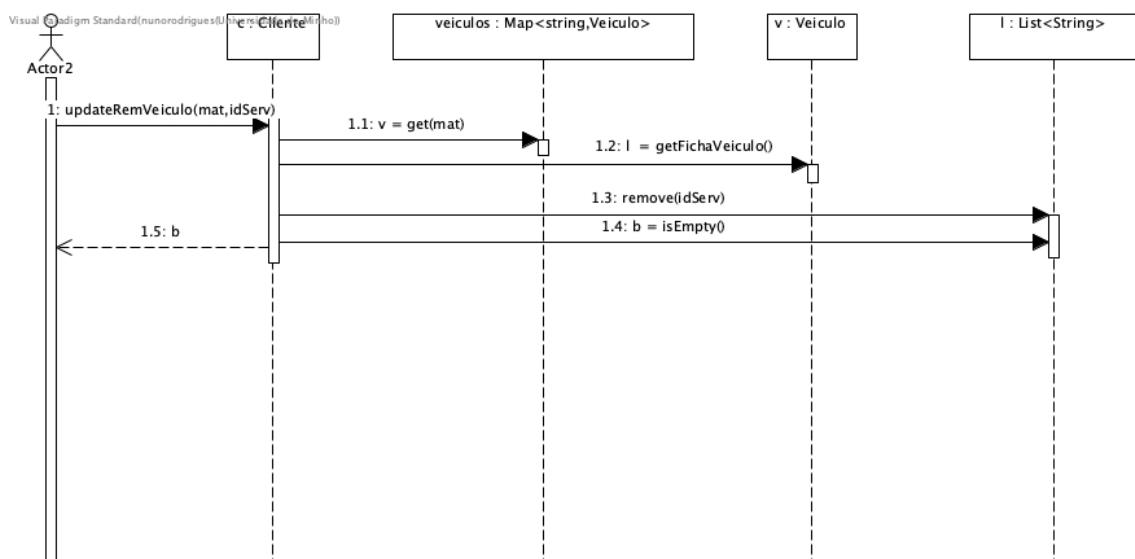
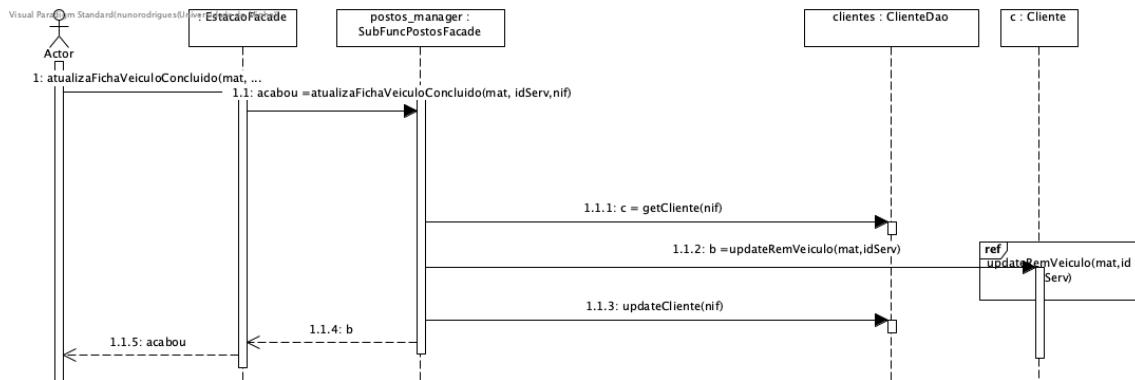


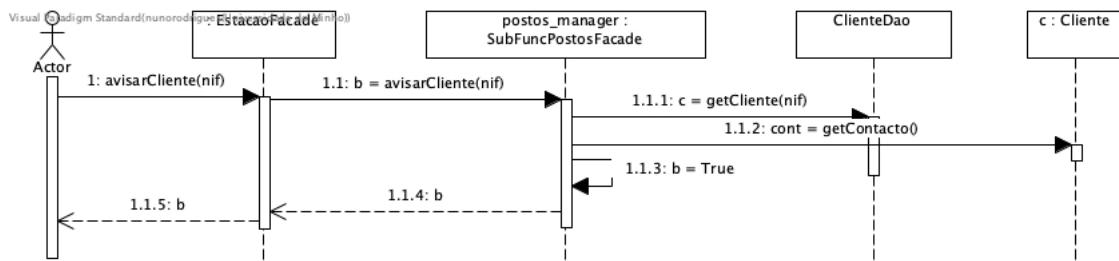
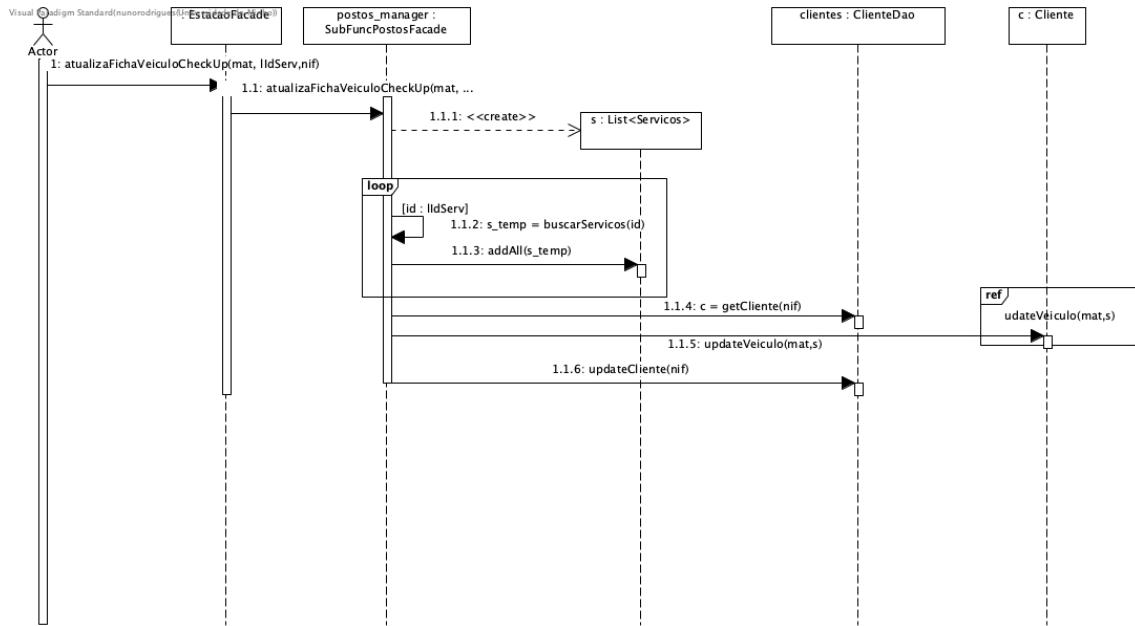






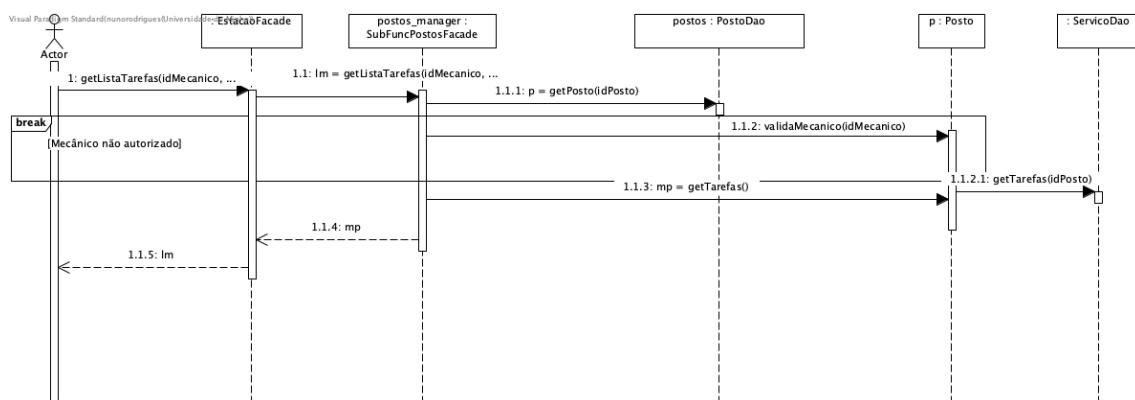
(Use case ->Concluir Serviço)



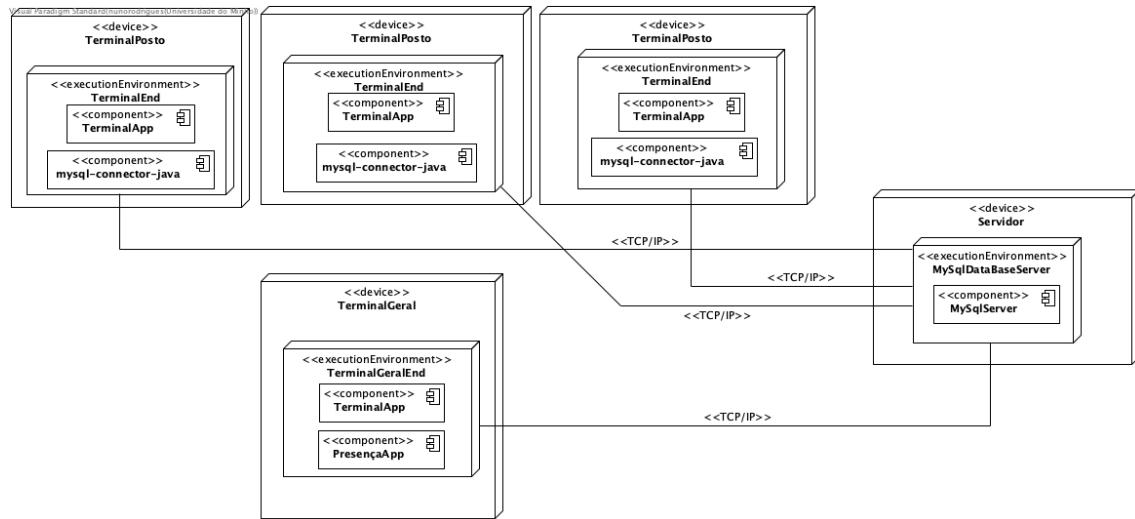


A lógica de enviar um sms depende de APIs externas, por isso apenas fingimos que enviamos retornando um bool

(Use case ->Ver a lista de Serviços)



## Diagrama de Instalação (apenas uma possibilidade)



## Resultados obtidos

Depois de seguir um desenvolvimento em cascata e de utilizar os diferentes diagramas da UML, conseguimos obter uma implementação possível da qual estamos satisfeitos. Nesta secção vamos, sumariamente, mencionar os resultados obtidos para cada fase e perceber no que é que ajudou os diferentes diagramas.

Primeiramente, o modelo de domínio foi feito sobre um diagrama de classes e serviu de base para todo o progresso. O mesmo se pode dizer do diagrama de Use Cases, onde fica bastante claro, inclusive na implementação de interfaces de utilizador, como deve ser feita a interação com o sistema. Desta forma, achamos que é algo bastante preciso e necessário para a conceção de uma aplicação. No caso dos diagramas de Atividade, pressupondo que os casos de uso estão devidamente especificados, não é a ferramenta mais útil. Achamos que é uma maneira rápida de perceber o desenrolar das interações com o sistema, mas não bate os casos de uso.

Na modelação conceptual, utilizamos um diagrama de classes assim como vários diagramas de sequência para representar com um alto nível de abstração o futuro programa. O diagrama de classes foi novamente bastante útil para entender como as classes iriam interagir entre si e os respetivos dados que seriam precisos. Os diagramas de sequência, construídos com base no diagrama anterior, serviram para perceber que de facto é possível concretizar as ações necessárias de manipulação de dados para funcionar o programa. Contudo, achamos que a semântica do diagrama de sequência é um pouco desnecessariamente cabulosa, isto é, não é difícil de entender, e é uma boa forma de documentar o funcionamento de uma operação, mas requer passos extras ao que um programador tipicamente faz quando está a escrever código. Ainda possui a desvantagem de não dar para testar, o que muitas vezes (como no nosso caso), permite que cheguem alguns problemas de lógica à implementação. O diagrama de componentes foi baste útil e serviu como primeiro passo para chegar a um diagrama de classes. Este foi facilmente obtido através dos casos de uso especificados.

Por fim, na implementação, adaptamos alguns dos diagramas existentes de modo a suportar persistência de dados. De modo geral, este passo foi agilizado graças à documentação que obtivemos da secção anterior. Novamente o diagrama de classes provou ser muito útil para o grupo. A adaptação dos diagramas de sequência também foi rápida e não comprometeu a legibilidade dos mesmos. Além disso, surge agora o diagrama de packages, que com pouco esforço ajuda a salientar as camadas e divisões da aplicação, mostrando a nossa *api* e como ela será usada. A nível de componentes surge também agora a camada de dados, onde a nossa lógica de negócios interage. Finalmente, completamos com o diagrama de instalação, que mostra uma possível forma de implementar o nosso programa.