Universidade do Minho

Departamento de Informática

Desenvolvimento de Sistemas de Software

Gestão de oficinas da E.S.Ideal

Grupo 39 - Repositório do trabalho

Gonçalo Brandão

Maya Gomes

Henrique Pereira

Luís Caetano

Simão Antunes

Gonçalo Araújo Brandão A100663
Maya Gomes A100822
Henrique Morais Pereira A100831
Luís de Castro Rodrigues Caetano A100893
Simão Pedro Ferreira AntunesA100597

Conteúdos

1 Modelo de domínio

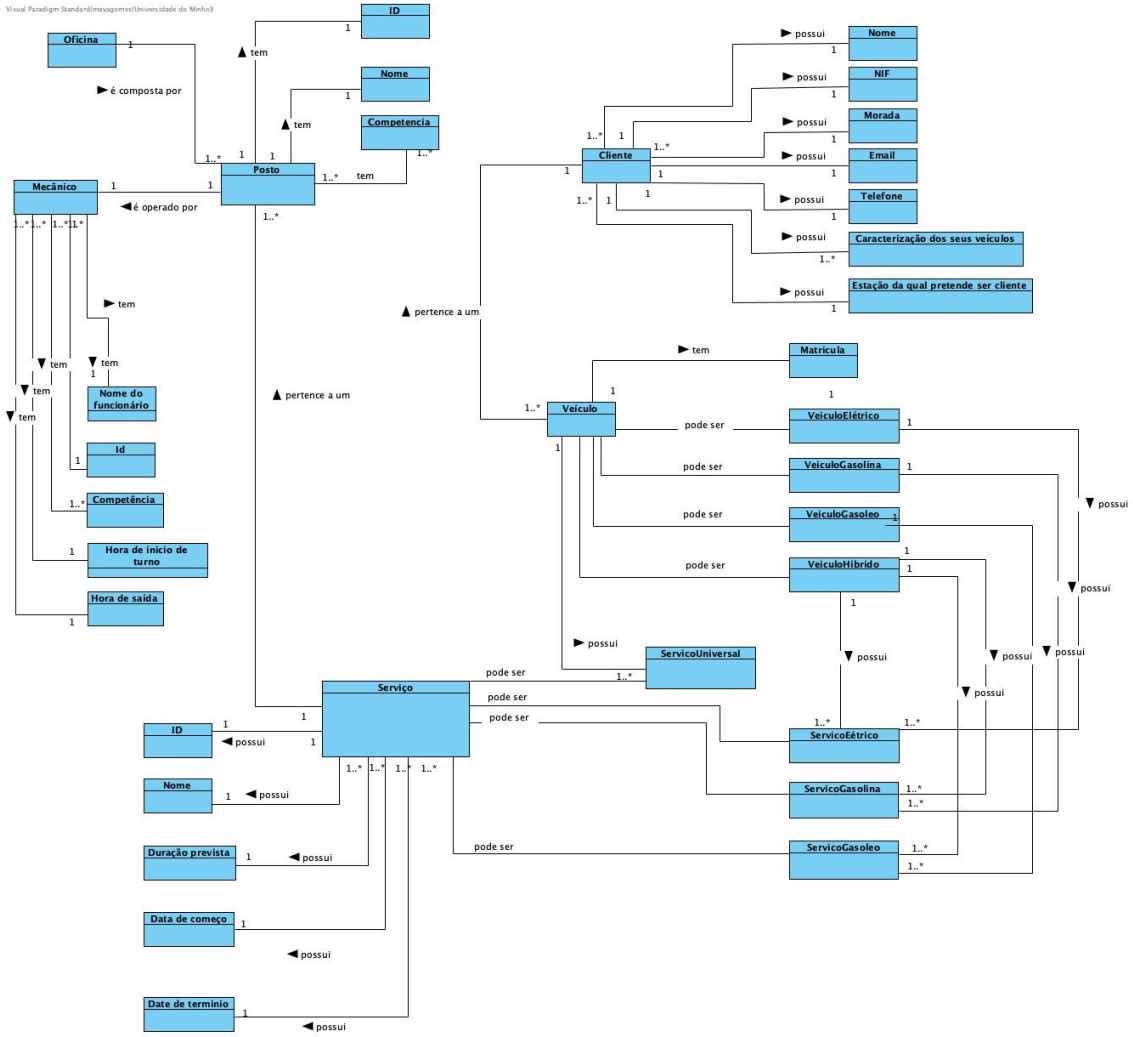


Figura 1: Modelo de Domínio.

2 Modelo de Use Case

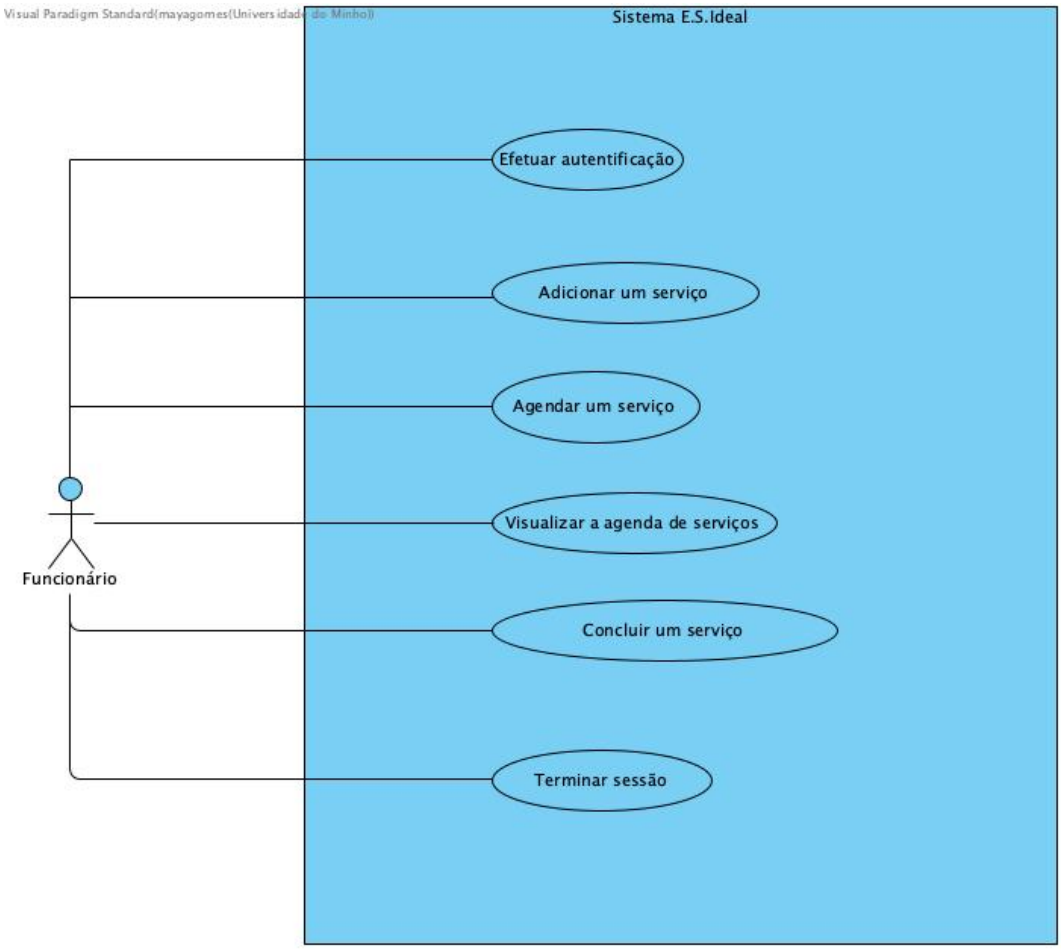


Figura 2: Modelo de Use Case.

3 Especificações de Use Cases

Seguem dois exemplos da especificação dos nossos Uses Cases. Os restantes Uses Cases localizam-se em /Modelos/USES CASES + API.xlsx.

3.1 Adicionar um Serviço

			1. Dividir os fluxos	2. Identificar responsabilidade da LN	3. definir API (identificar operações)	4. identificar subsistemas (agrupar operações)
USE CASE:		Adicionar um serviço.				
DESCRIÇÃO:		O funcionário adiciona um serviço ao sistema.				
CENÁRIOS:		Todos ou quase				
PRÉ-CONDIÇÃO:		O funcionario estar autenticado e o cliente ter preenchido a ficha do veículo com as diversas informações.				
PÓS-CONDIÇÃO:		O sistema fica com o registo do serviço necessário para o veículo.				
FLUXO NORMAL:						
	1.	Mecânico fornece o NIF do proprietário do veículo ao sistema	1			
	2.	O sistema verifica que o NIF é válido	1	verificar se o NIF inserido existe no sistema	loginFichaCliente(String idCliente)	subUtilizador
	3.	O funcionário escolhe a opção de "pedir um serviço "	2			
	4.	O sistema pede a matrícula do veículo	2			
	5.	O funcionário introduz a matrícula do veículo	2			
	6.	O sistema verifica se a matrícula é válida e pertence ao cliente em questão	2			
	7.	O funcionário insere os dados necessários sobre o serviço pedido pelo cliente	3			
	8.	O sistema atualiza a ficha do veículo com o novo serviço inserido	3	o sistema cria um novo serviço	criarServico(String idCliente, Posto posto, String matricula, String nomeServico)	SubServico
	9.	O sistema agenda quando vai ser realizado o serviço	3	agendar o serviço	procuraHorario(Servico servico, Cliente cliente, List<LocalDateTime> horariosNegados)	SubServico
FLUXO ALTERNATIVO	(1)	[O funcionário não consegue atualizar a ficha do veículo] (passo 3)				
	3.1.	O sistema deve notificar o erro e permitir que o funcionário tente novamente ou entre em contato com o administrador.	2			
FLUXO DE EXCEÇÃO	(2)	[O nif do cliente não é válido] (passo 2)				
	2.1.	Informa o cliente que a atualização não é possível porque o NIF dado não existe ou porque o cliente não tem conta	1			
FLUXO DE EXCEÇÃO	(3)	[A matrícula do veículo do cliente não é válida] (passo 6)				
	6.1.	Informa o cliente que a atualização não é possível porque a matrícula dada não existe no sistema ou é de outro	2			

Figura 3: Adicionar um Serviço.

3.2 Agendar um Serviço.

			1. Dividir os fluxos	em	2. Identificar responsabilidades da LN	3. definir API (identificar operações)	4. identificar subsistem operações)
USE CASE:		Agendar um serviço.					
DESCRIÇÃO:		O serviço pedido pelo cliente é atribuído a um posto (e um mecanico) num determinado dia e hora					
CENÁRIOS:		Todos ou quase					
PRÉ-CONDIÇÃO:		O cliente vai á oficina pedir um serviço					
PÓS-CONDIÇÃO:		O sistema fica com o registo do serviço a ser realizado e a sua respetiva data bem como o mecânico atribuído					
FLUXO NORMAL:							
	1.	O Funcionário escolhe a opção mais adequada com base no pedido do cliente.	1				
	2.	O Pedido é atribuído ao posto adequado	1				
	3.	O sistema percorre os horários disponíveis no posto em questão e escolhe um horario com duração adequada para o serviço em questão	1				
	4.	O sistema sugere o horário ao cliente	2		atribuir um dia e uma hora para a realização do serviço	procuraHorario(Servico servico, Cliente cliente, List<LocalDateTime> horariosNegados)	SubServico
	5.	O cliente confirma o horário	2				
	6.	O horário é atribuido e o serviço é registado	2		o cliente confirma o horário se este for adequado	confirmaAgendamento(Cliente cliente, Servico servico, List<LocalDateTime> horariosNeg	SubServico
	(1)	[O funcionário não consegue atualizar a ficha do veículo] (passo 3)					
FLUXO ALTERNATIVO	3.1.	O sistema deve notificar o erro e permitir que o funcionário tente novamente ou entre em contato com o administrador.	1				
	(1)	[O cliente não pode comparecer no horário atribuído)] (passo 7)					
FLUXO ALTERNATIVO	7.1.	O sistema deve agendar novamente o pedido atribuindo um horário diferente do anteriormente escolhido.	3				
	(2)	[código de utente não existe] (passo 2)					
FLUXO DE EXCEÇÃO	2.1.	Informa o cliente que a atualização não é possível porque o NIF dado não existe ou porque o cliente não tem conta	1				

Figura 4: Agendar um Serviço.

4 Lógica de negócio (LN)

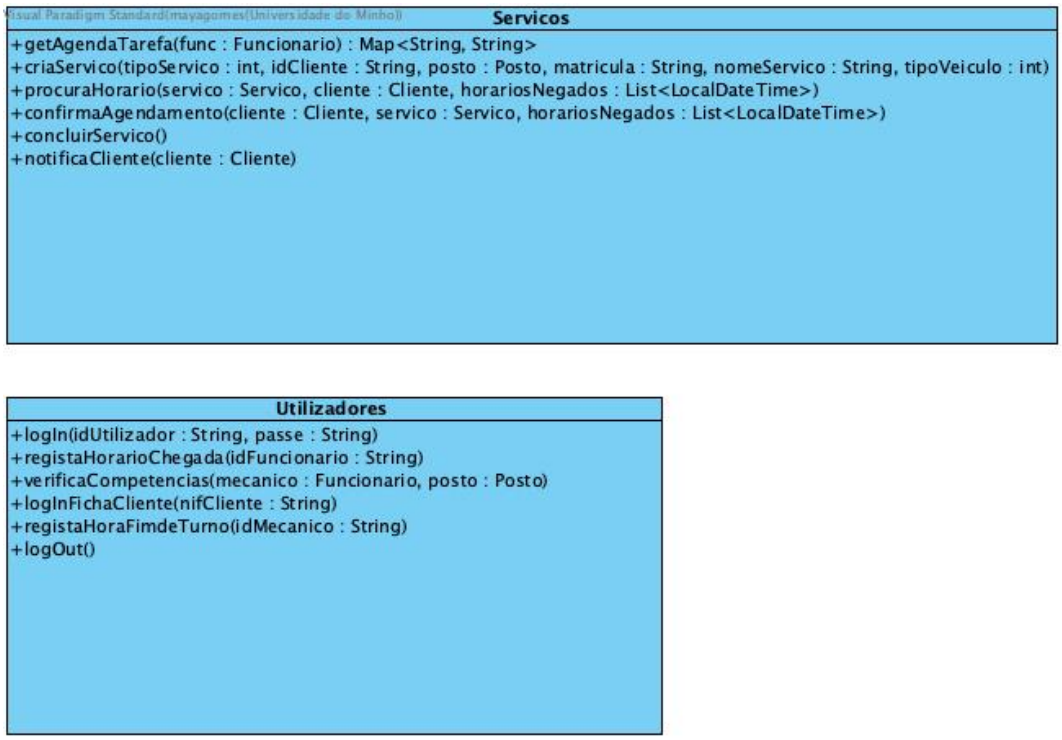


Figura 5: Lógica de negócio (LN).

5 Diagrama de componentes

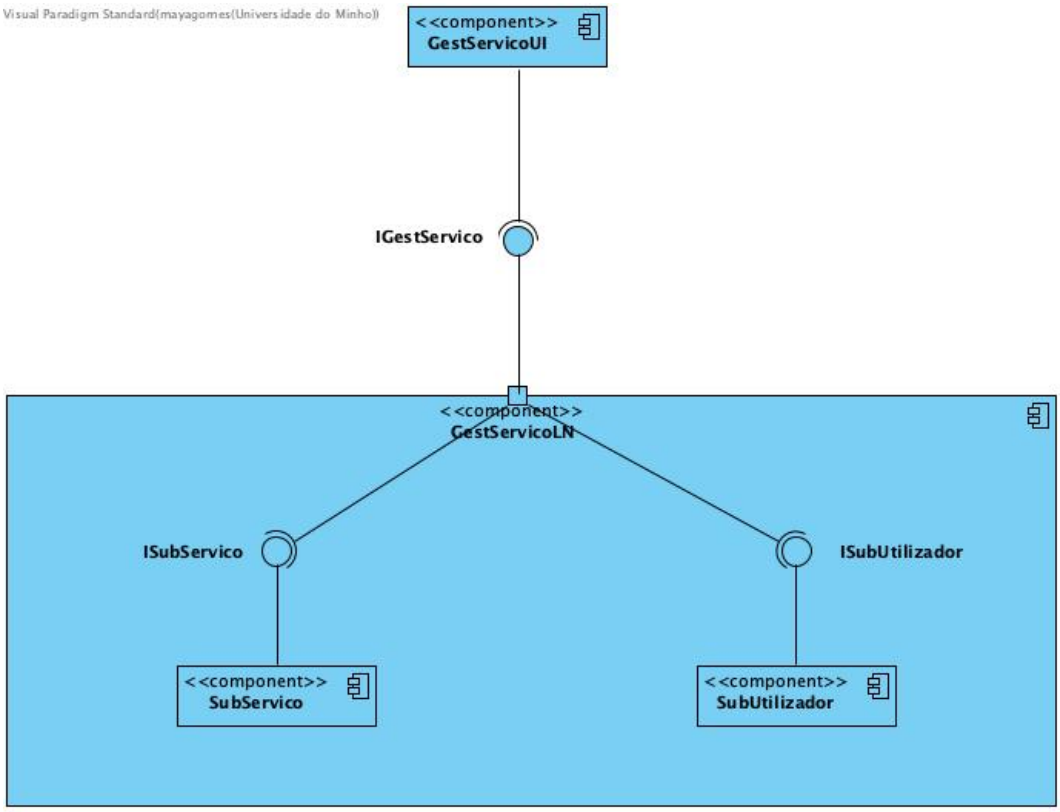


Figura 6: Diagrama de componentes.

6 Diagrama de classes

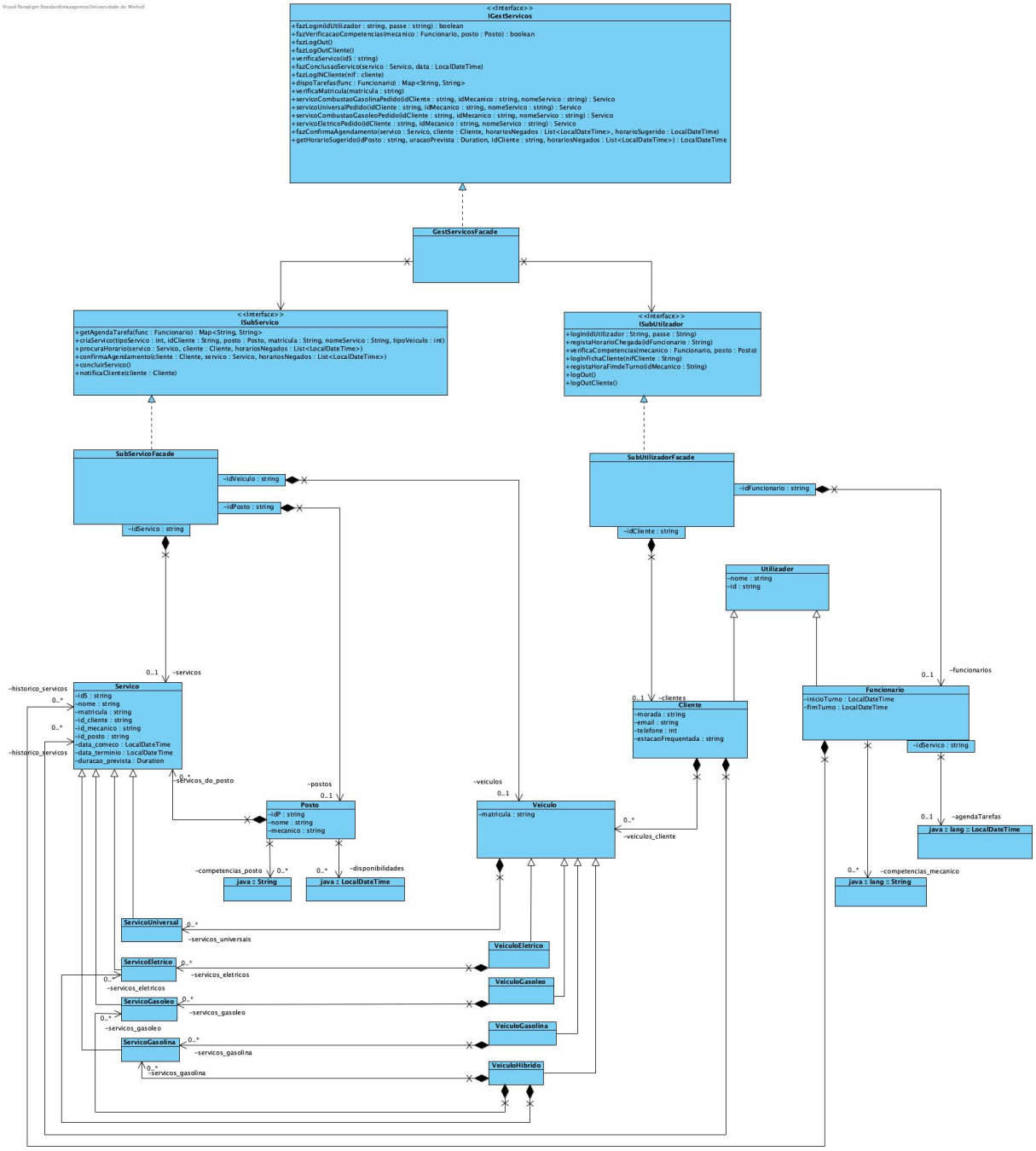


Figura 7: Diagrama de classe.

7 Diagrama de classes com DAO

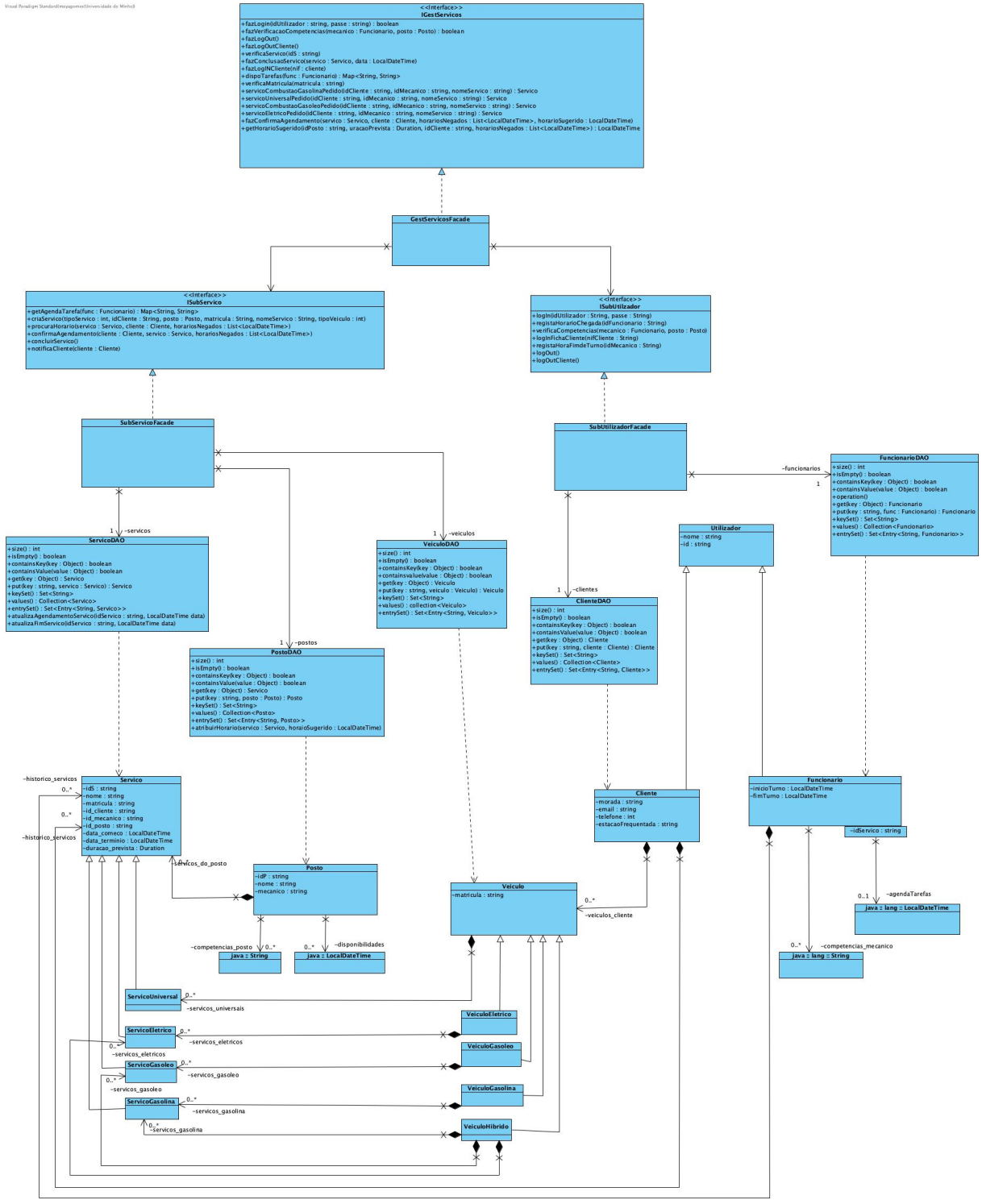


Figura 8: Diagrama de classe com DAO.

8 Base de Dados (mysql)

8.1 Modelo conceptual

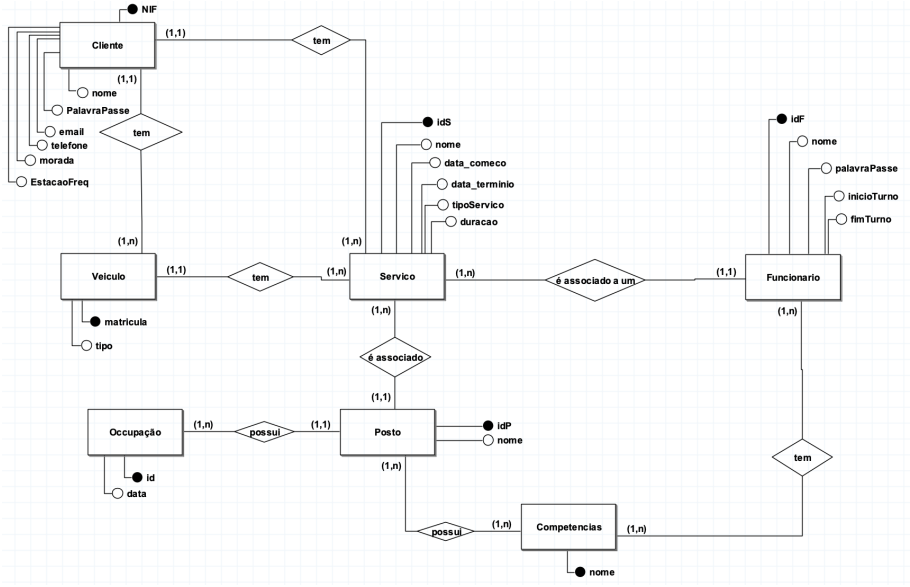


Figura 9: Modelo conceptual da base de dados.

8.2 Modelo lógico

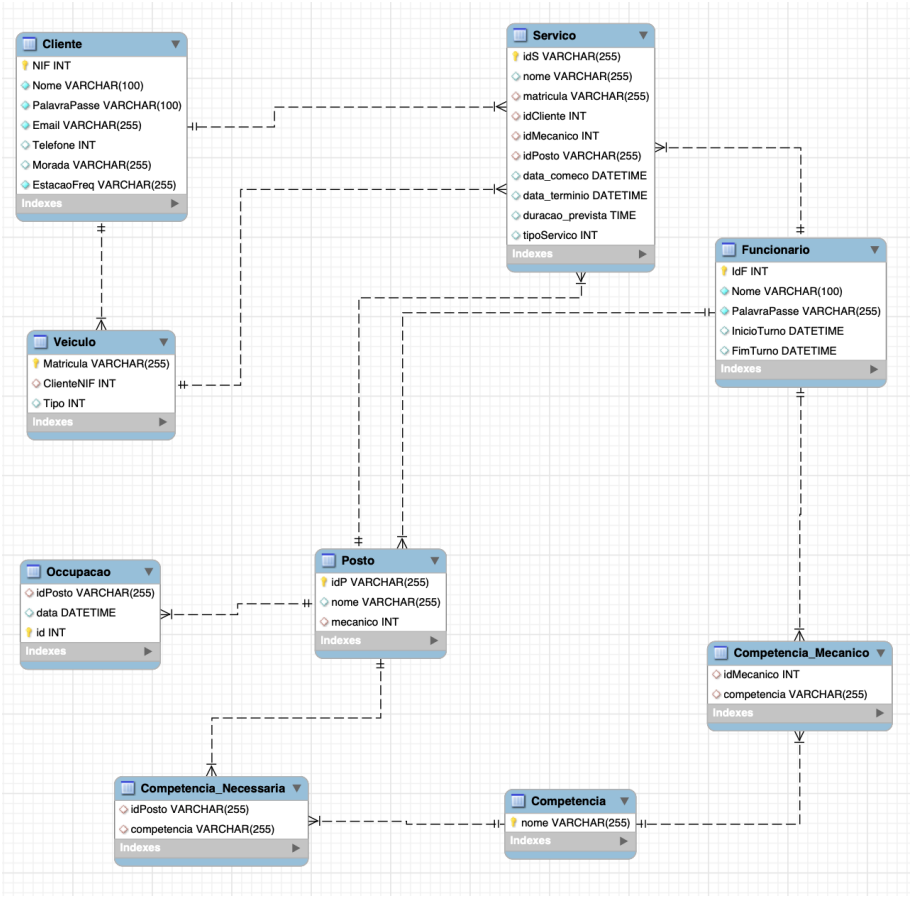
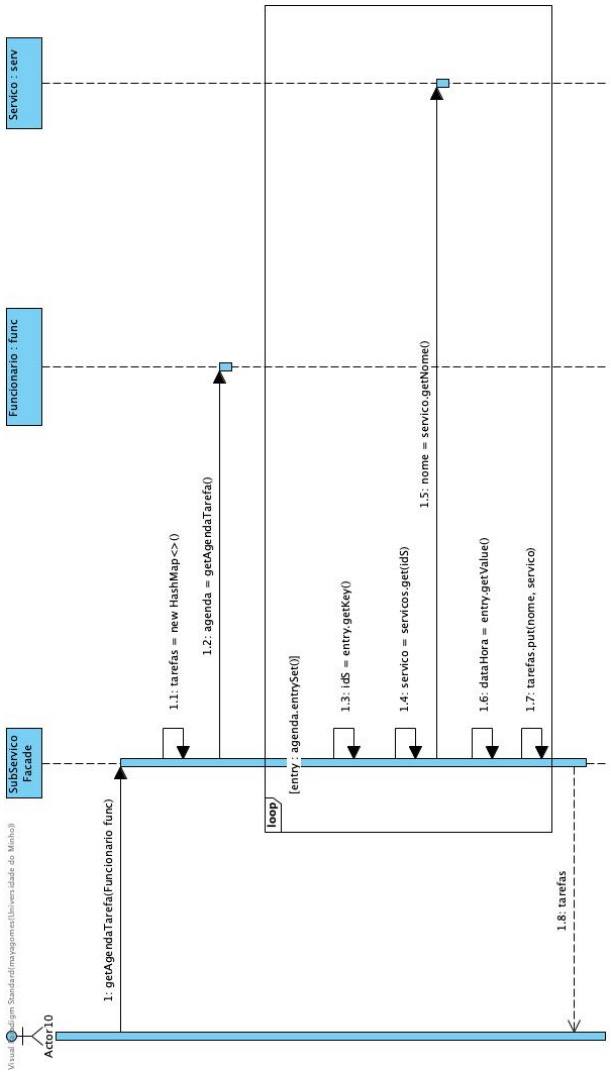


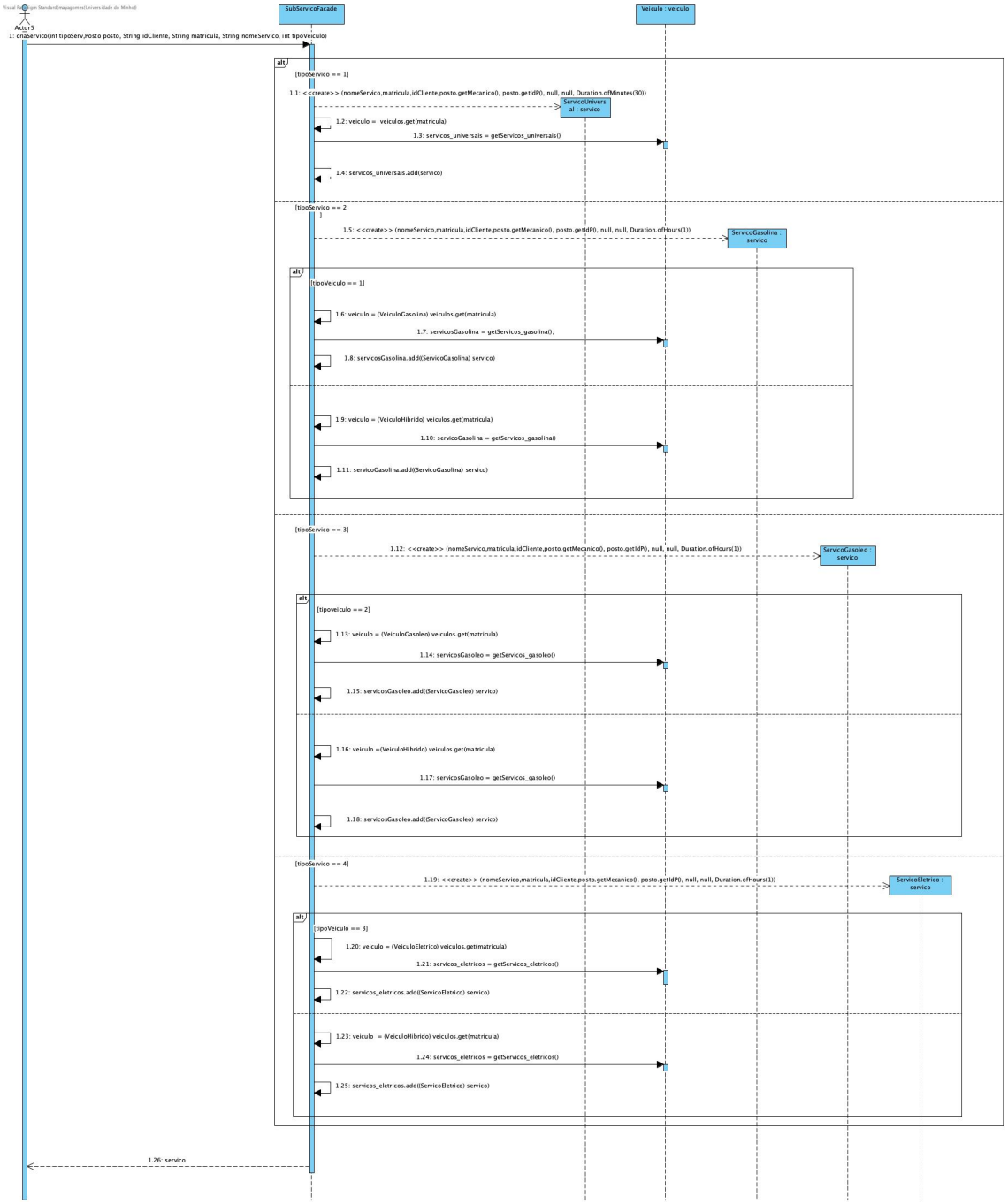
Figura 10: Modelo lógico da base de dados.

9 Diagramas de sequencia

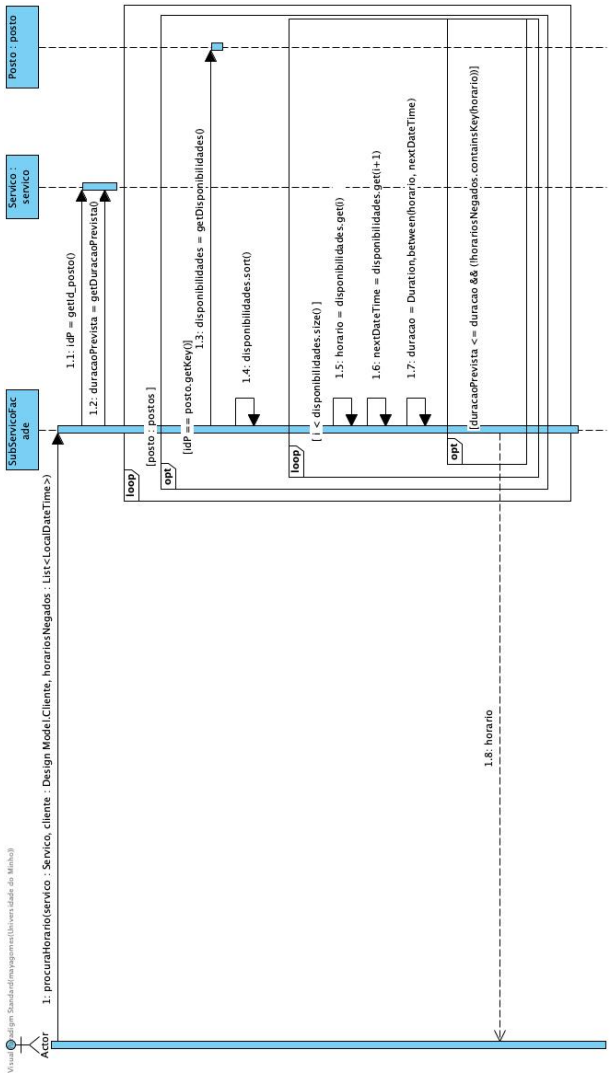
9.1 Agenda de tarefas do mecânico



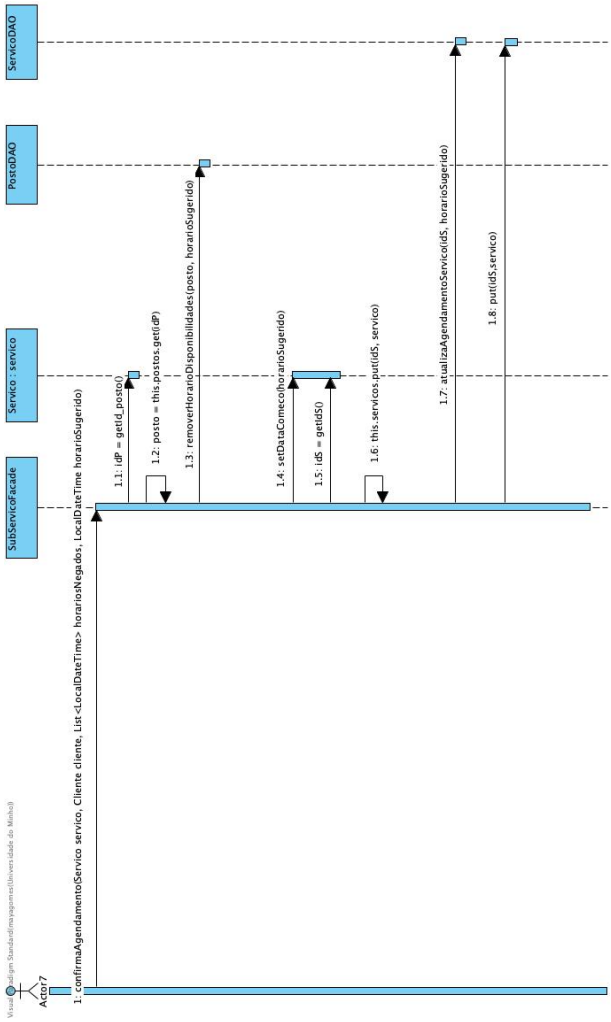
9.2 Criar um novo serviço



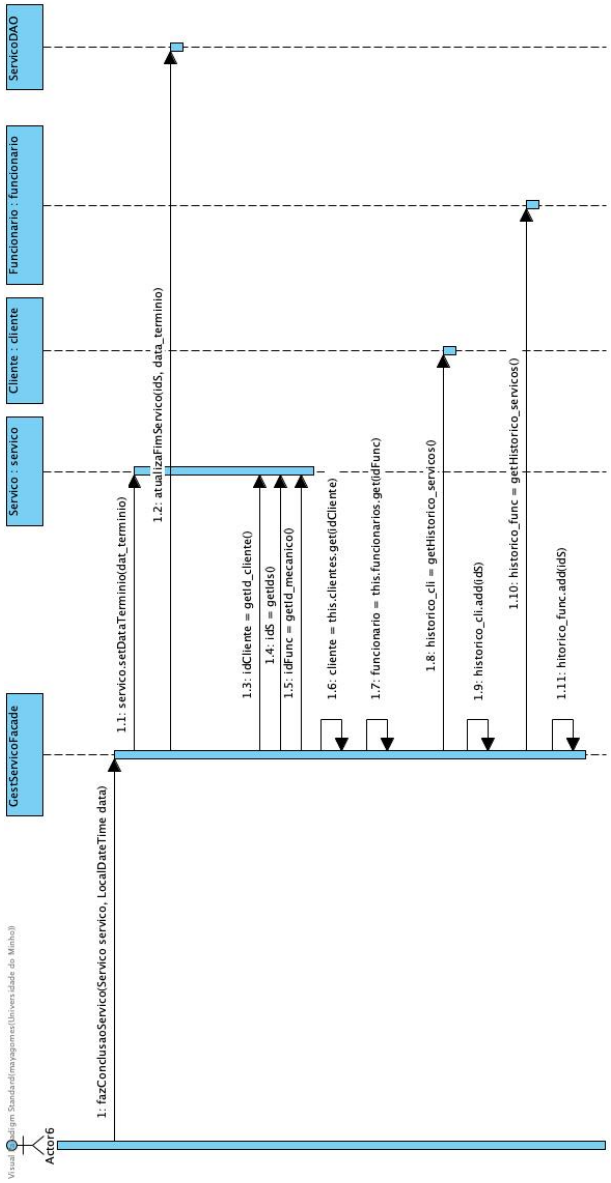
9.3 Procurar um horário para agendamento



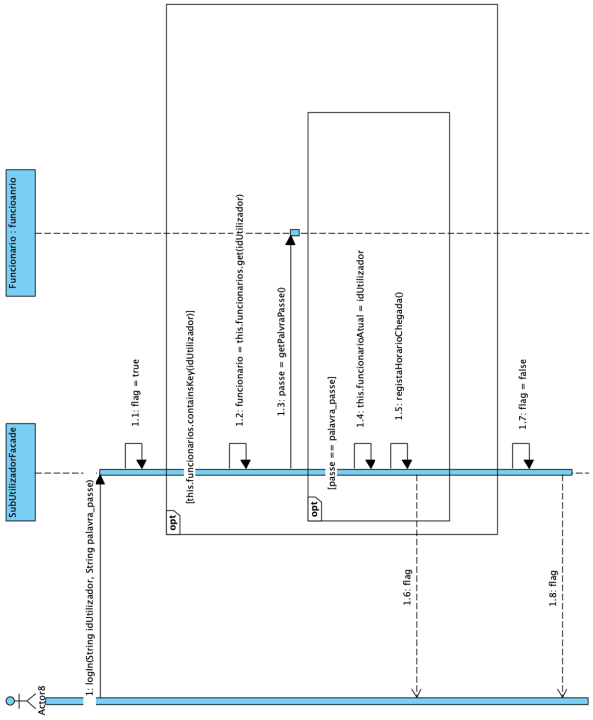
9.4 Confirmar um agendamento



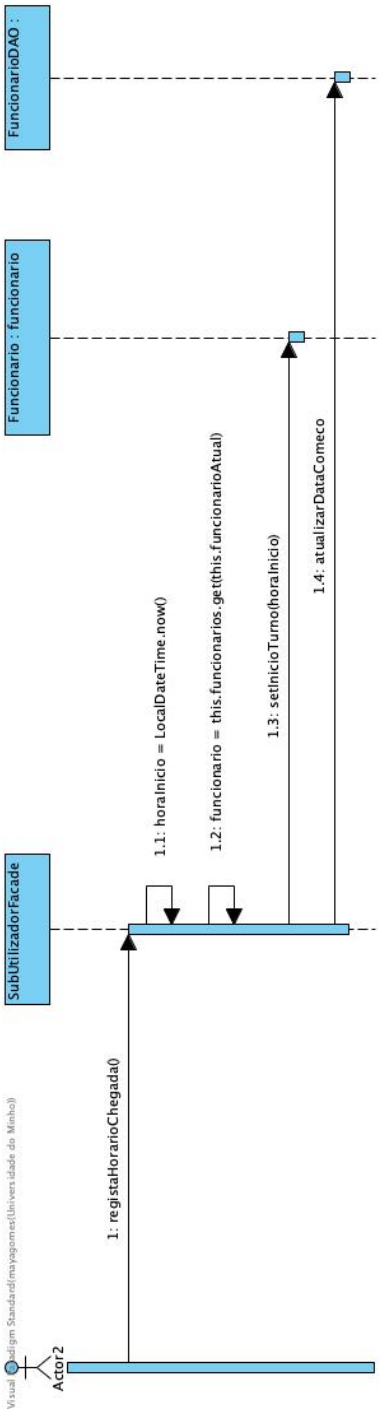
9.5 Concluir um serviço



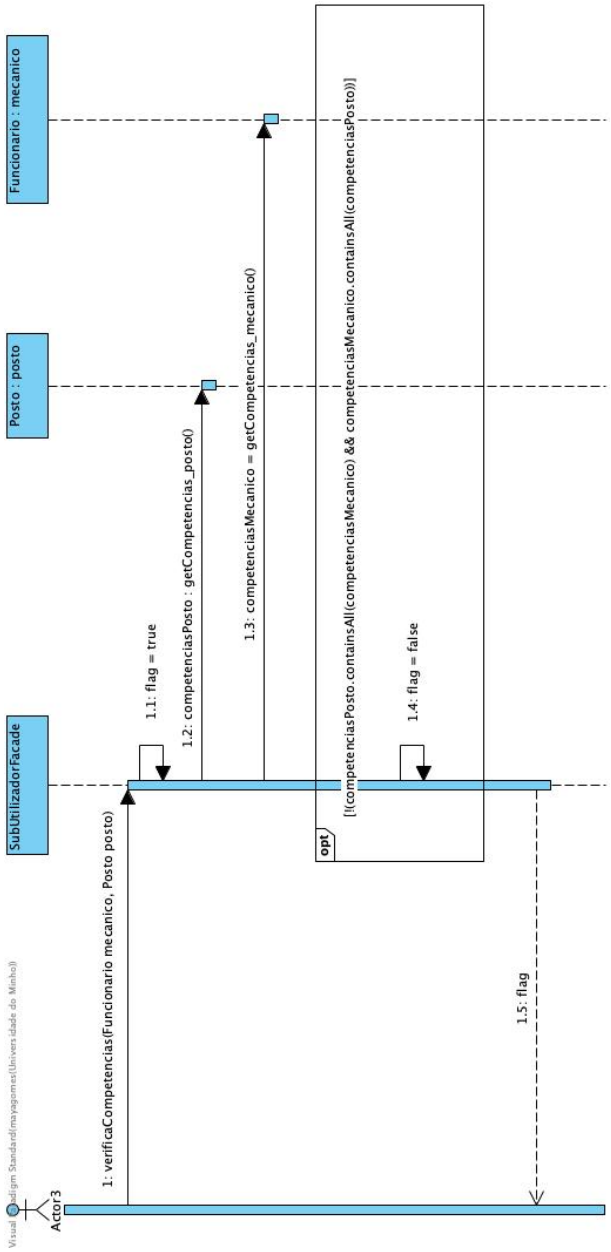
9.6 logIn


Funcionario : funcioanrio
SubUtilizadorFacade
Actor8
1: loginString idUtilizador, String palavra_passe)
1.1: flag = true
[this.funcionarios.containsKey(idUtilizador)]
1.2: funcionario = this.funcionarios.get(idUtilizador)
1.3: passe = getPalvraPasse()
[passe == palavra_passe]
1.4: this.funcionarioAtual = idUtilizador
1.5: registaHorarioChegada()
1.6: flag
1.7: flag = false
1.8: flag
opt
opt

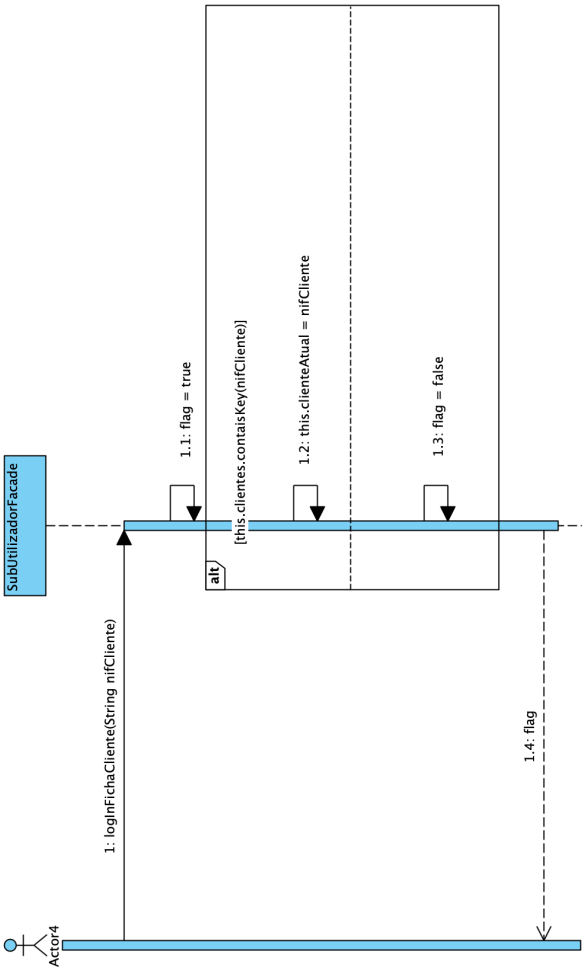
9.7 Registar o horário de chegada ao turno



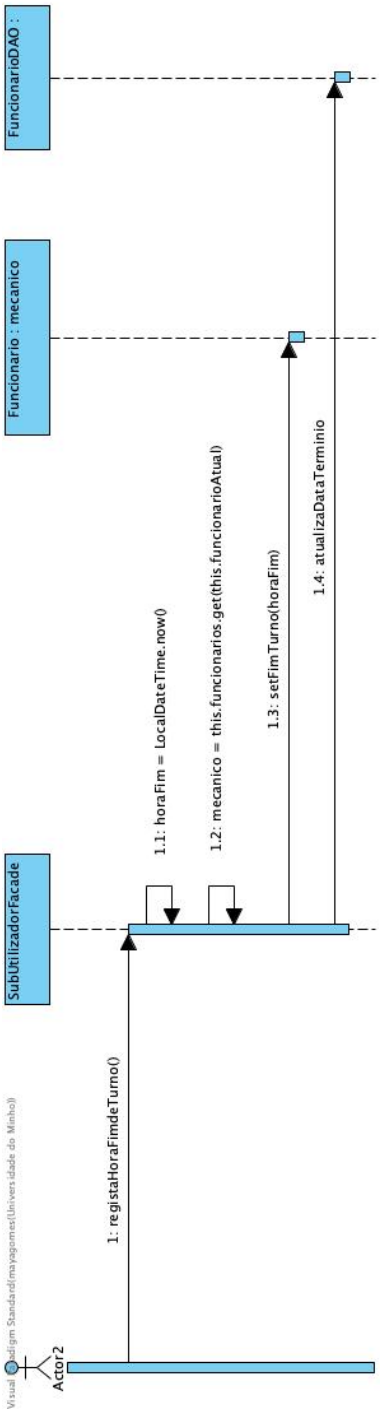
9.8 Verificar as competências do mecânico para o posto



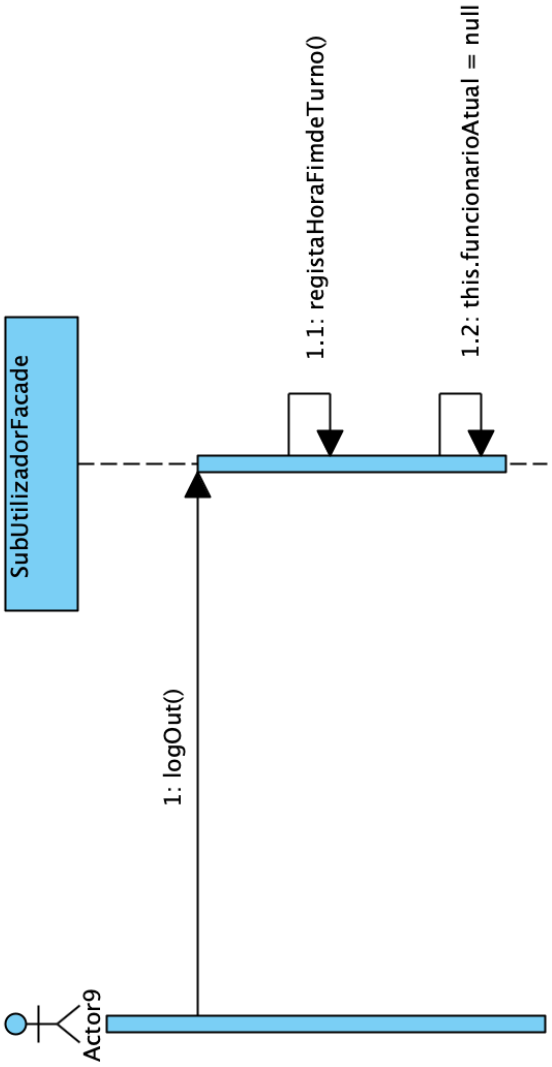
9.9 Login na ficha de um cliente



9.10 Registar o horário de fim do turno



9.11 Logout


SubUtilizadorFacade
1.1: registaHoraFimdeTurno()
1.2: this.funcionarioAtual = null
1: logOut()
Actor9

10 Diagrama de package

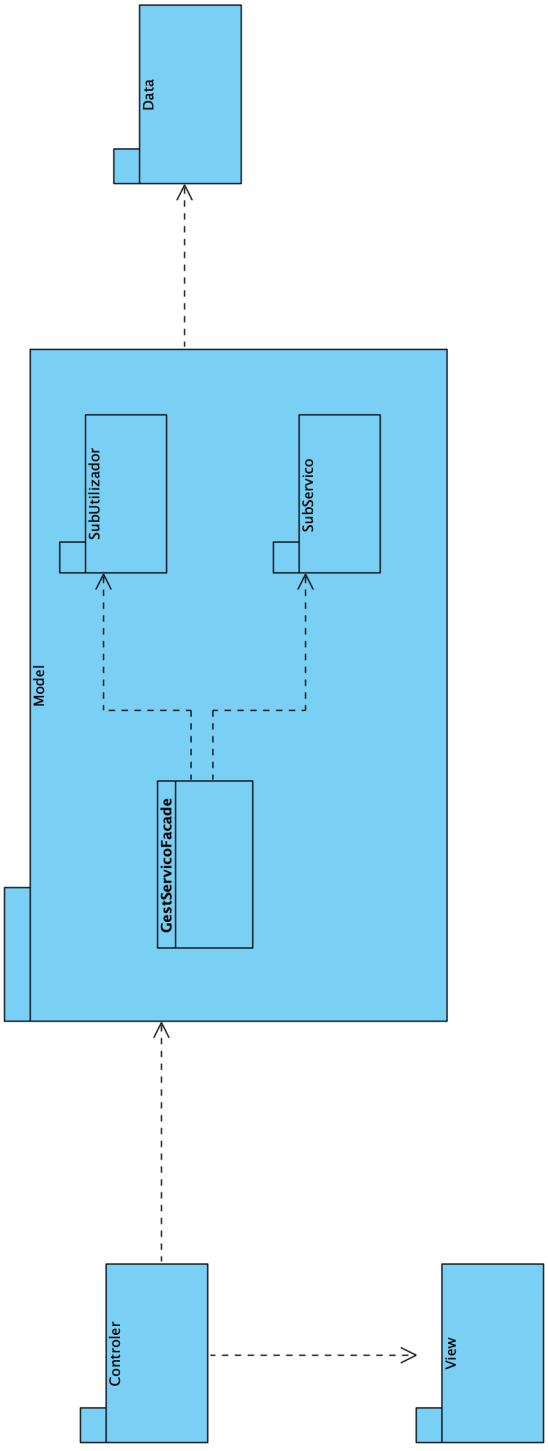


Figura 11: Diagrama de package

11 Descrição dos resultados obtidos

Ao longo do desenvolvimento do nosso sistema de gerenciamento de serviços adotamos diversas abordagens escolhendo por fim a referida anteriormente. O nosso objetivo principal foi criar um ambiente intuitivo e eficaz para gerenciar os serviços dos quatro tipos distintos de veículos: combustão a gasolina, gasóleo, elétricos e híbridos.

Começando pelo diagrama de domínio, esse proporcionou-nos uma visão clara da estrutura do sistema, identificando entidades-chave como Veículo, Cliente, Funcionário e Serviço. No entanto, surgiram dúvidas na parte da divisão dos serviços nomeadamente para os veículos híbridos. Conseguimos porém estabelecer uma boa base para a modelagem da base de dados e as interações entre as diferentes partes do sistema.

Os nossos uses cases, foram fundamentais para organizarmos as funcionalidades essenciais do nosso sistema. Assim conseguimos começar a visualizar um pouco melhor o desenvolvimento das operações do nosso programa. Nesta fase, também surgiram dúvidas sendo necessário restruturar o diagrama diversas vezes.

A especificação dos uses cases e definição dos API's prolongou esta definição dos métodos que iríamos futuramente implementar e organizou a divisão dos mesmos em dois subsistemas: um subsistema para os utilizadores e outro para toda a parte que toca aos serviços. Essa divisão permitiu uma melhor organização e modularidade, para além de ter outras vantagens como a facilidade de manutenção.

Os diagramas de classe (sem e depois com os DAO's) foram elaborados considerando os principais objetos do sistema e seguíndo as ideias que se tinham desenvolvido nos diagramas anteriores, nomeadamente no diagrama de domínio. Nesta fase, foram definidas as três classes Facade que iríamos usar mais a frente. Assim, a SubServicoFacade representa a parte do subsistema dos serviços, a SubUtilizadorFacade a parte dos utilizadores e a GestServicoFacade é responsável por interligar as duas facades anteriores. De seguida, a definição dos DAOs facilitou a interação com a base de dados, garantindo uma separação eficiente entre a lógica de negócios e a persistência dos dados. Para a base de dados, também foram elaborados o diagrama conceptual e o modelo lógico.

Os diagramas de sequência forneceram uma representação visual das interações entre as classes do sistema durante métodos cruciais do nosso programa. Esses diagramas foram essenciais para garantir uma compreensão clara dos métodos que pretendíamos desenvolver tal como facilitar as suas implementações.

A implementação em Java e a integração com o MySQL (onde criamos a nossa base de dados) foram realizadas de forma a garantir desempenho e escalabilidade.

Concluíndo, os diagramas elaborados desempenharam um papel crucial na compreensão e desenvolvimento do nosso trabalho. Tal como referido anteriormente, surgiram diversas dúvidas, nomeadamente na parte dos veículos e serviços no que toca a hieraquia. Também foi difícil para nós percebermos como queríamos dividir o trabalho, tanto na parte dos subsistemas como depois no código. No entanto, o grupo conseguiu ultrapassar essas dificuldades e consideramos ter cumprido com os objetivos.