

RELATÓRIO DO TRABALHO PRÁTICO DE DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE

PARKUM

Simulação de controlo e Gestão de um Parque de Estacionamento



Grupo 1



Elementos do grupo (da esquerda para a direita)

Vanessa Catarina de Seabra Campos nº 54801
Andreia Patrícia Matias da Silva nº 56837
Ana Isabel dos Anjos Sampaio nº 54740
Miguel Pinto da Costa nº 54746
Hugo Emanuel da Costa Frade nº 54750

Universidade do Minho
2 de Janeiro de 2011

ÍNDICE

Introdução	1
Captura e análise de requisitos	3
Modelo de Domínio	3
Diagramas de Use Case do PARKUM	5
Actores	5
Especificação textual dos Use Cases de Sistema	8
Entrar no parque (Cliente sem registo):	8
Sair do parque(Cliente sem registo):	9
Devolve Bilhete:	9
Pagar Bilhete:	10
Pagar Multa por atraso:	11
Emitir recibo:	11
Estacionar Veículo:	12
Contactar Funcionário:	12
Desocupar lugar:	12
Sair modo avença:	13
Sair modo automático:	13
Entrar modo avença:	14
Entrar modo automático:	15
Editar Cliente:	16
Registrar Cliente:	17
Remover Cliente:	18
Obter Relatórios Diários.	18
Obter Relatórios de máquinas de pagamento:	19
Obter Relatórios de Facturação	19
Obter tabelas de distribuição de ocupações:	20

Listar Pagamentos Atraso:	20
Receber pagamento de multa por perda de bilhete:	21
Auxiliar Cliente:	21
Receber Pagamento de Multa por Perda	22
Registrar Manutenção	22
Diagramas de Sequência	23
Concepção e Desenvolvimento	41
Identificação dos subsistemas do PARKUM:	41
Refinamento dos diagramas de Use Cases de sistema e dos diagramas de sequência de sistema	42
Subsistema Portal de Entrada	42
Subsistema Portal de Saída	43
Subsistema de Gestão de Clientes	44
Subsistema de Gestão de Relatórios	45
Subsistema PARKUM-CS	45
Subsistema Sensores de Lugar	46
Subsistema Máquina de Pagamento	46
Auxiliar Cliente	47
Contactar Funcionário	47
Desocupar Lugar	48
Devolver Bilhete	48
Editar Cliente	49
Emitir Recibo	50
Entrar modo automático	51
Entrar modo Avença	52
Entrar no Parque	53
Estacionar Veiculo	54
Listar Pagamentos em Atraso	54

Obter Relatórios Diários	55
Obter Relatórios de Facturação	56
Obter Relatórios Máquinas de Pagamento	56
Obter Tabelas de Distribuição	57
Pagar Bilhete com Dinheiro	58
Pagar Bilhete com Cartão Magnético	60
Pagar Multa por Atraso	61
Receber Pagamento de Multa	62
Registrar Cliente	62
Registrar Manutenção	64
Remover Cliente	65
Sair do Parque	66
Sair Modo Automático	67
Sair Modo Avença	68
Diagramas de Actividades	69
Entrar no parque (cliente de bilhete)	70
Entrar no parque (com cartão de avença)	71
Entrar no parque (de modo automático)	72
Estacionar Veículo	73
Pagar Bilhete	74
Registrar cliente	75
Sair do parque (cliente com bilhete)	76
Diagramas Máquinas de Estados	77
Portal de Entrada	77
Portal de Saída	79
Maquinas de Pagamento	81
Sensores de Lugar	82
Diagramas de Classes	83
PortalEntrada	83

PortalSaida	83
Barreira	83
Monitor	83
Bilhete	83
GestaoClientes	83
FichaCliente	83
Pagamento	83
MaquinaPagamento	84
Recibo	84
GestaoRelatorios	84
SensoresEstacionamento	84
PARKUM-CS	84
PainelInformacao e PainelGlobal	84
BaseDados	84
Agrupamento em Packages	85
Acessos	85
Clientes	87
Pagamentos	88
PARKUM-CS	89
SCI: Sistema Central de Informação	90
Implementação	91
Base de dados	91
Interface	91
Tabelas criadas	92
Tabela Clientes	92
Tabela Funcionário	93
Tabela Maquinas	93
Tabela Modo_entrada	93

Tabela Modos_pagamento	94
Tabela Pisos	94
Código Implementado	95
Detecção de Problemas	127
Conclusão	128
Mini-manual de utilização da aplicação	129
Interface PARKUM	129
Gestão de Clientes	130
Listagem de Pagamentos em Atraso	133
Relatórios	134
Manutenção	140
Registrar Pagamentos	141

RELATÓRIO DO TRABALHO PRÁTICO DE DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE

PARKUM

Simulação de controlo e Gestão de um Parque de Estacionamento

Ana Isabel dos Anjos Sampaio nº 54740

Miguel Pinto da Costa nº 54746

Hugo Emanuel da Costa Frade nº 54750

Vanessa Catarina de Seabra Campos nº 54801

Andreia Patrícia Matias da Silva nº 56837

2 de Janeiro de 2011

INTRODUÇÃO

No âmbito da Unidade Curricular de Desenvolvimento de Sistemas de Software foi-nos proposto a realização de um sistema de controlo e Gestão para Parques de Estacionamento - PARKUM.

Com este projecto pretende-se modelar um sistema de um parque de estacionamentos convencional, com várias entradas e várias saídas possíveis, multi-nível, com capacidade finita, e suficientemente moderno para aceitar diversas formas de pagamento, bem como controlo automático, através de sensores, da sua capacidade disponível a cada momento e por nível.

Como tal, este relatório abordará de forma detalhada todos os requisitos que foram necessários satisfazer para a elaboração deste projecto, tendo em conta as fases típicas do Rational Unified Process (RUP).

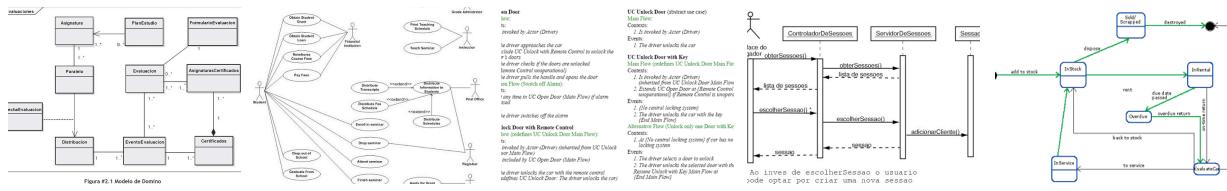
Primeiramente, foi feita a captura e análise de requisitos do PARKUM, onde se especificou o Modelo de Domínio do problema, o desenho dos Diagramas de Use Cases do Sistema, a sua Especificação textual e a elaboração dos Diagramas de Sequência correspondentes aos Use Cases especificados.

Após esta primeira abordagem desenvolveu-se, de seguida, a concepção e desenvolvimento do projecto onde se identificou os diversos subsistemas que garantem a funcionalidade global do PARKUM. De seguida, foram feitos refinamentos dos diagramas de Use Cases, de

todos os diagramas de sequência e foram apresentados diagramas de actividade de Use Cases mais relevantes. Desenvolveu-se o comportamento alguns subsistemas (portais de entrada e saída, dos sensores e da máquina de pagamento) através de diagramas de estados. Foi feito ainda diagramas de classes de todos os subsistemas do PARKUM, que também foram agrupados em *packages*.

Após essa fase de compreensão e estruturação dos dados, passamos para a fase de implementação em JAVA, onde foram desenvolvidos os métodos para criar a aplicação do PARKUM.

Este relatório apresenta uma descrição da estrutura criada para a aplicação, assim como as principais decisões tomadas relativamente a implementação das estruturas de dados escolhidas.



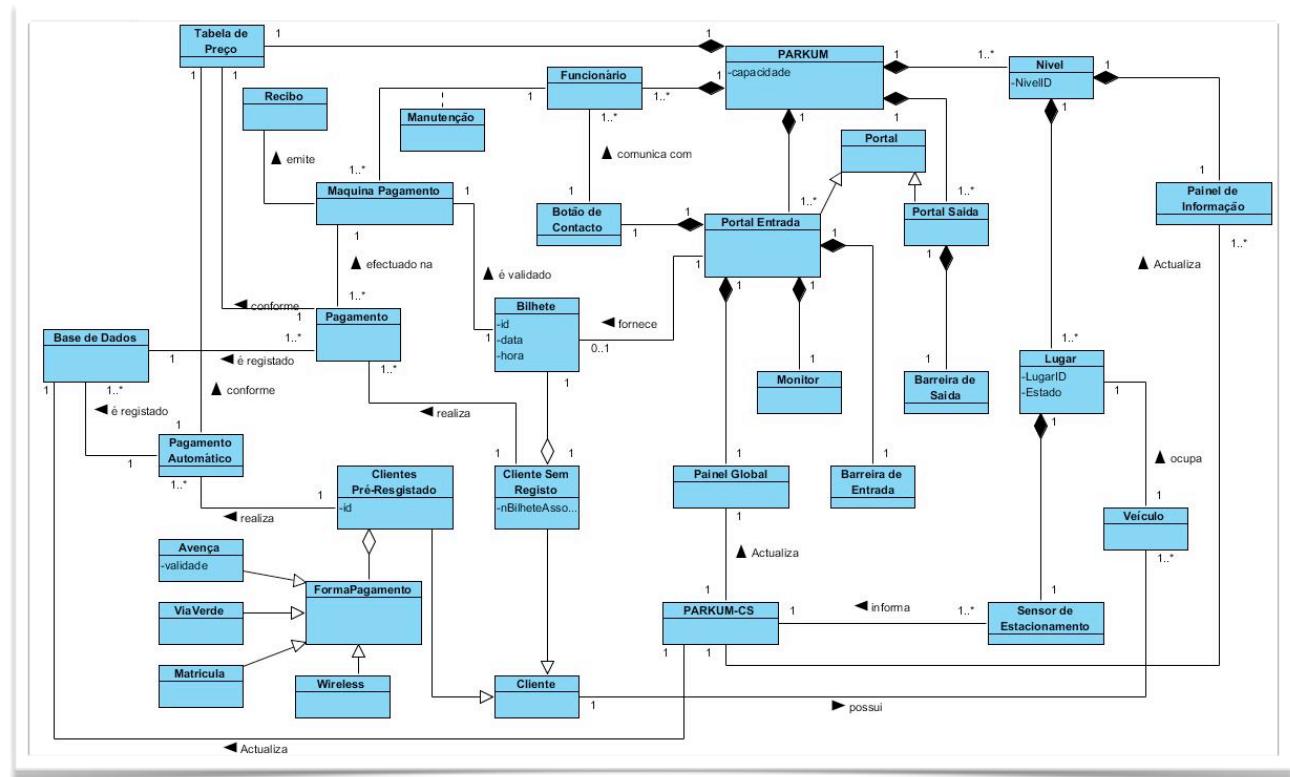
*Exemplo ilustrativo da sequência da realização do trabalho
(atenção estes exemplos não são relativos ao trabalho)*

CAPTURA E ANÁLISE DE REQUISITOS

O sistema PARKUM é um componente único capaz de fornecer todas as funcionalidades requisitadas. Como tal desenvolve-se, primeiramente, o modelo de domínio onde se identificam e relacionam todos os termos relevantes no sistema PARKUM.

MODELO DE DOMÍNIO

Através do modelo de domínio descrevemos a estrutura de informação do sistema PARKUM. O diagrama contém a informação sobre o sistema, nomeadamente no que respeita à identificação das relações e ao papel que as entidades desempenham nessa relação. Por isso, foi necessário relacionar todas as entidades necessárias de forma a explicar o funcionamento geral do sistema PARKUM.



Do modelo de domínio é possível verificar que o PARKUM é um sistema que contém uma relação de composição com o funcionário, o portal de entrada, o portal de saída, nível. Este tipo de agregação está representada com um diamante preto que atribui um significado mais forte à agregação, exemplificando uma dependência directa entre as duas entidades (se a parte deixar de existir, o todo também deixa e vice-versa).

No caso do bilhete e do cliente sem registo, bem como, os clientes pré-registrados e a forma de pagamento, existe também outro tipo de agregação, que está representado com um diamante branco, que evidencia o facto de que um todo é composto por partes.

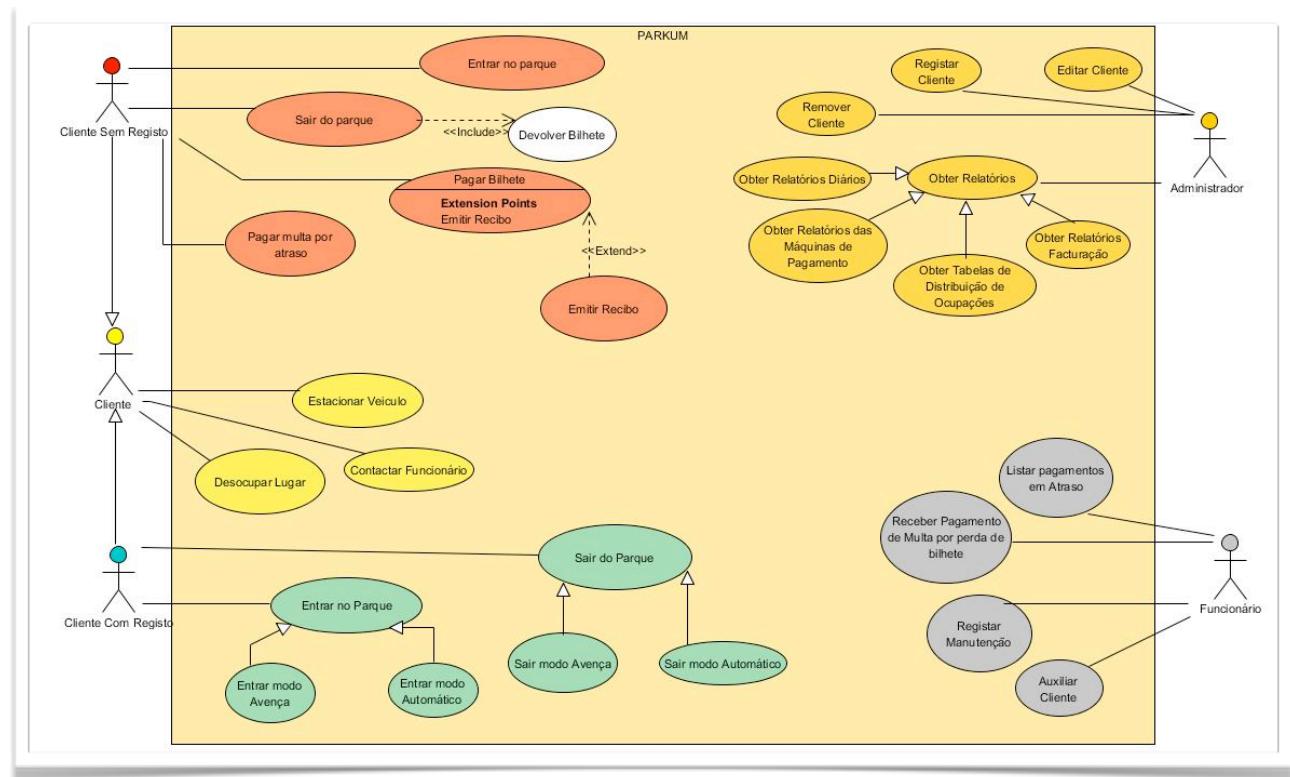
É ainda importante referenciar que as associações existentes são também caracterizadas por possuir uma multiplicidade, que indica quantos objectos participam na relação. A multiplicidade do PARKUM assumiu várias formas sendo de realçar:

- 1..1 - “Um para Um” - obrigatório existir um objecto, temos por exemplo no caso em que:
 - o PARKUM (apenas um sistema) possui (apenas) uma tabela de preços;
 - um portal de entrada tem apenas uma barreira de entrada;
- 1..* - “Um para Muitos” - um ou infinitos objectos da classe:
 - um ou mais sensores
 - um ou mais lugares
 - um ou mais pagamentos
- 0..* - zero ou infinitos objectos da classe:
 - o portal de entrada pode ou não fornecer bilhete conforme o cliente tiver, ou não, registado no sistema.

DIAGRAMAS DE USE CASE DO PARKUM

No diagrama de use case identificou-se as fronteiras do PARKUM, descreveu-se os requisitos (use cases) que devem ser disponibilizados a cada um dos diversos utilizadores (actores), bem como as relações existentes entre eles.

O resultado do levantamento de requisitos encontra-se representado pelo seguinte diagrama de use case:



ACTORES

A primeira tarefa desenvolvida para construir um diagrama de use cases foi a identificação dos actores do PARKUM. Um actor representa uma entidade externa que interage com o sistema.



- Os actores **Cliente Com Registo** e **Cliente Sem Registo** são generalizações do **Cliente**, estes actores são as pessoas que irão interagir com o sistema. Isto é, os clientes são as entidades que estacionam o veículo no parque de estacionamento.
- O **Funcionário** - empregado do PARKUM é a entidade que recebe as notificações de auxílio dos clientes, faz a manutenção da máquina de pagamento, recebe pagamentos de multa por perda de bilhete e verifica, ainda, pagamentos em atraso.
- O **Administrador** - empregado que está encarregue de efectuar a gestão de clientes como por exemplo: criar, editar, remover clientes da base de dados assim como de obter relatórios sobre a actividade do PARKUM.

Após a identificação dos actores, deve-se identificar para cada cada actor os use case com que o PARKUM interage:

- **Cliente:**

- | | |
|---|--|
| <ul style="list-style-type: none"> - Estacionar Veículo; - Contactar Funcionário; | <ul style="list-style-type: none"> - Desocupar Lugar; |
|---|--|

- **Cliente Sem registo:**

- | | |
|---|--|
| <ul style="list-style-type: none"> - Entrar no parque; - Sair do parque; - Devolver Bilhete; | <ul style="list-style-type: none"> -Pagar Bilhete; -Emitir Recibo; -Pagar multa por atraso; |
|---|--|

- **Cliente Com registo:**

- | | |
|---|---|
| <ul style="list-style-type: none"> -Entrar no Parque : -Entrar modo Avença; -Entrar modo Automático; | <ul style="list-style-type: none"> - Sair do Parque: -Entrar modo Avença; -Entrar modo Automático; |
|---|---|

- **Funcionário**

- Listar pagamentos em Atraso;
- Receber Pagamento de Multa por perda de bilhete;
- Registar Manutenção;
- Auxiliar Cliente;

- **Administrador:**

- Obter Relatórios:

- ♦Obter Relatórios Diários;
 - ♦Obter Relatórios das Máquinas de Pagamento;
 - ♦Obter Tabelas de Distribuição de Ocupações;
 - ♦Obter Relatórios Facturação;

- Editar Cliente;

- Registar Cliente;

- Remover Cliente;

A comunicação entre um actor e os use cases é representada por uma linha ou uma seta que indicam direcção da comunicação.

ESPECIFICAÇÃO TEXTUAL DOS USE CASES DE SISTEMA

A descrição dos use case assume a forma de um texto livre ou estruturado, com um conjunto de passos numerados. São textos simples que registam decisões conjuntas.

Resultam de tomadas de decisão conjuntas entre clientes, utilizadores, que fixam funcionalidades do PARKUM.

Por exemplo no Use Case

ENTRAR NO PARQUE (CLIENTE SEM REGISTO):

O cliente entra no Parque pressiona o botão de bilhete , neste instante o sistema verifica a ocupação do parque, regista e emite bilhete , abre a barreira de entrada ,o cliente entra no parque e o sistema regista a entra no parque e fecha a barreira. Poderá ocorrer uma excepção caso se verifique que o parque está cheio.

Super Use Case		
Author	Grupo 1	
Date		
Brief Description		
Preconditions		
Post-conditions	A entrada do cliente é registada	
Flow of Events	Actor Input	System Response
	I Pressiona botão de bilhete	
	2	Verifica ocupação do parque
	3	Regista bilhete
	4	Emite bilhete
	5 Pega no bilhete	
	6	Abre barreira de entrada
	7 Entra no parque	
	8	Fecha a barreira de entrada
	9	Regista entrada
Exception 2a. O Parque Está Cheio	Actor Input	System Response
	I	Informa que o parque está cheio

É importante realçar ainda que nas descrições o actor é a entidade que desencadeia a acção. A pré-condição representa a informação necessária para garantir os parâmetros que o sistema precisa para efectuar a acção. Por sua vez a pós condição indica o resultado da acção. O *flow of events* é o fluxo de sucesso dos eventos, as alternativas indicam outros caminhos para caso de sucesso. As *exceptions* são situações que ocorrem quando ocorre uma falha por parte do cliente ou do sistema.

SAIR DO PARQUE (CLIENTE SEM REGISTO):

Super Use Case		
Author	Grupo I	
Date		
Brief Description		
Preconditions		
Post-conditions	Sistema regista a saída	
Flow of Events	Actor Input	System Response
	I include: Devolver Bilhete	
	2	Verifica se o bilhete está pago
	3	Verifica se não excedeu o tempo de saída
	4	Recolhe bilhete
	5	Abre a barreira de saída
	6 Sai do parque	
	7	Fecha a barreira de saída
	8	Regista saída
Exception 2a: Bilhete Não Está Pago	Actor Input	System Response
	I	Informa que o bilhete não foi pago
	2	Devolve bilhete
3 Pega no bilhete		
Exception 3a: Tempo de Saída Excedeu o Tempo Limite	Actor Input	System Response
	I	Informa que é necessário o pagamento de multa por atraso na saída
	2	Devolve o bilhete
3 Pega no bilhete		

DEVOLVE BILHETE:

Super Use Case	Devolver Bilhete	
Author	Grupo I	
Date		
Brief Description		
Preconditions		
Post-conditions	O bilhete é recolhido pelo portal de saída	
Flow of Events	Actor Input	System Response
	I Introduz bilhete	
	2	Lê o bilhete
	3	Verifica que o bilhete é válido
Exception 3a: Bilhete Inválido	Actor Input	System Response
	I	Informa que o bilhete é inválido
	2	Devolve o bilhete
3 Pega no bilhete		

PAGAR BILHETE:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description	Cliente pretende efectuar pagamento.		
Preconditions			
Post-conditions	Montante presente na máquina aumenta o valor pago pelo cliente.		
Flow of Events	Actor Input	System Response	
	I	Insere bilhete.	
	2		Verifica hora de entrada.
	3		Calcula tempo que permaneceu no parque
	4		Calcula montante a pagar
	5		Informa montante a pagar
	6	Efectua o pagamento.	
	7		Verifica valor inserido
	8		Regista pagamento
	9		Regista hora e data de saída
	10		Pergunta se quer recibo
	II	Não quer Recibo	
	12		Devolve bilhete.
13	Pega bilhete		
Alternative 7a O Valor Inserido Requer Troco	Actor Input	System Response	
	I		O valor inserido é superior ao valor a pagar
	2		Calcula troco
	3		Devolve o troco
4		Volta ao passo 8	
Alternative 11a: Pedir Recibo	Actor Input	System Response	
	I		extend by: Receber recibo
2		Pega no Recibo	

PAGAR MULTA POR ATRASO:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description			
Preconditions	Atraso na saída do cliente		
Post-conditions	A máquina fica com o bilhete		
Flow of Events		Actor Input	System Response
	1	Introduz o bilhete	
	2		Lê o bilhete
	3		Calcula o tempo que permaneceu a mais no parque
	4		Calcula montante da multa
	5		Informa qual o montante a pagar
	6	Efectua pagamento	
	7		Regista pagamento no bilhete
	8		Regista nova hora de saída
	9		Confirma pagamento
	10		Devolve o bilhete

EMITIR RECIBO:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description			
Preconditions	Cliente efectuou pagamento do bilhete		
Post-conditions	Recibo é emitido		
Flow of Events		Actor Input	System Response
	1	Pede recibo	
	2		Escreve dados no recibo
	3		Emite recibo
Exception 2a: Impossível Emitir Recibo		Actor Input	System Response
	1		Informa erro

ESTACIONAR VEÍCULO:

Super Use Case		
Author	Grupo 1	
Date		
Brief Description	Cliente estaciona veículo num lugar livre.	
Preconditions		
Post-conditions	N.º de veículos no piso aumenta; N.º de lugares livres diminui; A cor do sensor passa a vermelha	
	Actor Input	System Response
1	Procura lugar livre	
2	Estaciona veículo	
3		Actualiza sensor
4		Informa estacionamento
5		Actualiza painel de Informação
6		Actualiza painel global

CONTACTAR FUNCIONÁRIO:

Super Use Case		
Author	Grupo 1	
Date		
Brief Description	Cliente contacta funcionário.	
Preconditions		
Post-conditions	Funcionário é notificado que foi chamado pelo cliente.	
	Actor Input	System Response
1	Carrega no botão do funcionário	
2		Envia mensagem de chamada do funcionário
3		Informa sucesso da operação

DESOCUPAR LUGAR:

Super Use Case		
Author	Grupo 1	
Date		
Brief Description		
Preconditions		
Post-conditions	N.º de veículos no piso aumenta; N.º de lugares livres aumenta; A cor do sensor é verde	
	Actor Input	System Response
1	Retira veículo	
2		Actualiza sensor de estacionamento
3		Informa desocupação
4		Decrementa o n.º de lugares ocupados
5		Actualiza painel de informação
6		Actualiza painel global

S A I R M O D O A V E N Ç A :

Super Use Case	Sair do Parque		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions	Sistema possui um novo registo de saída		
Flow of Events	Actor Input	System Response	
	1	Insere cartão de avença	
	2		Lê cartão de avença
	3		Verifica validade do cartão
	4		Devolve cartão
	5	Pega no cartão	
	6		Valida saída
	7		Abre a barreira de saída
	8	Sai do parque	
	9		Fechá a barreira de saída
10		Regista saída	
Exception 3a: Cartão Inválido	Actor Input	System Response	
	1		Informa que o cartão é inválido
	2		Devolve cartão de avença
3	Pega no cartão de avença		

S A I R M O D O A U T O M Á T I C O :

Super Use Case	Sair do Parque		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions			
Flow of Events	Actor Input	System Response	
	1	Aproxima-se do portal	
	2		Deteta o dispositivo
	3		Levanta a barreira de saída
	4	Sai do parque	
	5		Fechá a barreira de saída
	6		Regista a saída
Exception 2a: Não Existe o Dispositivo no Sistema	Actor Input	System Response	
	I		Informa que o dispositivo não existe

ENTRAR MODO AVENÇA:

Super Use Case	Entrar no Parque	
Author	Grupo 1	
Date		
Brief Description		
Preconditions		
Post-conditions	Sistema possui um novo registo de entrada	
Flow of Events	Actor Input	System Response
	1 Insere o cartão de avença	Lê número de cliente
	2	Verifica validade do cartão
	3	Verifica ocupação no parque
	4	Verifica se existem pagamentos em atraso
	5	Devolve cartão de avença
	6	Pega cartão de avença
	7	Valida entrada
	8	Abre a barreira de entrada
	9	Entra no parque
	10	Fechá a barreira de entrada
	11	Regista dados de cliente e da entrada
	Actor Input	System Response
Exception 3a: Cartão Inválido	1	Informa que o cartão é inválido
	2	Devolve cartão de avença
	3	Pega no cartão de avença
Exception 4a: O Parque Está Cheio	Actor Input	System Response
	1	Informa que o parque está cheio
	2	Devolve cartão de avença
	3	Pega no cartão de avença
Exception 5a: Pagamentos Em Atraso	Actor Input	System Response
	1	Informa que o pagamento não está em dia
	2	Devolve cartão de avença
	3	Pega no cartão de avença

ENTRAR MODO AUTOMÁTICO:

Super Use Case	Entrar no Parque		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions	Sistema possui um novo registo de entrada		
Flow of Events	Actor Input	System Response	
	I	Aproxima-se do portal de entrada	
	2		Detecta o dispositivo de entrada do cliente
	3		Verifica se a identificação do dispositivo existe
	4		Verifica que o parque não está cheio
	5		Regista dados de clientes e a entrada
	6		Informa que a entrada é autorizada
	7		Abre a barreira
	8	Entra no parque	
	9		Fecha a barreira
Exception 3a: Identificação Não Existe	Actor Input	System Response	
I		Informa que o dispositivo não existe	
Exception 4a: O Parque Está Cheio	Actor Input	System Response	
I		Informa que o parque está cheio	

EDITAR CLIENTE:

Super Use Case			
Author	Grupor		
Date			
Brief Description			
Preconditions			
Post-conditions	Ficha de cliente alterada		
Flow of Events		Actor Input	System Response
	1	Selecciona alteração de clientes	
	2		Pede identificação do cliente
	3	Insere dados do cliente	
	4		Lê de dados de cliente
	5		Verifica se o cliente existe
	6		Mostra ficha de cliente
	7	Altera dados	
	8		Lê dados alterados
	9	Submete dados	
	10		Verifica validade dos dados
	11		Actualiza ficha de cliente
	12		Informa sucesso da operação
Exception 4a: Cliente Não Existe		Actor Input	System Response
	1		Informa que o cliente não existe
Exception 8a: Dados Inválidos		Actor Input	System Response
	1		Informa que os dados são inválidos
	2		Cancela operação

REGISTAR CLIENTE:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions	Novo Cliente Registado		
Flow of Events	Actor Input	System Response	
	I	Cria nova ficha de cliente	
	2		Pede para inserir dados do cliente
	3	Insere dados dos cliente	
	4		Lê dados do cliente
	5		Verifica dados de cliente
	6		Pede para inserir dados do veículo
	7	Insere dados do veiculo	
	8		Lê dados do veículos
	9		Pede a forma de pagamento
	10	Insere a forma de pagamento	
	II		Lê forma de pagamento
	I2		Regista o cliente no sistema
	I3		Informa sucesso da operação
Exception 4a O Cliente Já Existe	Actor Input	System Response	
	I		Informa que o cliente já existe
2		Cancela operação	
Alternative 9a: Forma de Pagamento por Avença	Actor Input	System Response	
	I		Escreve dados no cartão de avença
2		Emite cartão de avença	

REMOVER CLIENTE:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description	Remover cliente registrado		
Preconditions			
Post-conditions	Cliente removido		
Flow of Events	Actor Input	System Response	
	1	Pede para remover cliente	
	2		Pede dados do cliente
	3	Insere dados do cliente	
	4		Lê dados do cliente
	5		Verifica dados do cliente
	6	Confirma que quer remover o cliente	
	7		Remove cliente da base de dados
	8		Informa sucesso da operação
Excepção 5a: Informação do Cliente Inválida	Actor Input	System Response	
	1		Informa que os dados são inválidos
	2		Cancela operação

OBTER RELATÓRIOS DIÁRIOS.

Super Use Case	Obter Relatórios		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions			
Flow of Events	Actor Input	System Response	
	1	Pede tipo de relatório diário	
	2		Pede dados de pesquisa
	3	Insere os dados	
	4		Lê os dados
	5		Pesquisa dados
	6		Gera relatório
	7		Apresenta relatório

O B T E R R E L A T Ó R I O S D E M Á Q U I N A S D E P A G A M E N T O :

Super Use Case	Obter Relatórios		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions			
Flow of Events	Actor Input	System Response	
	1	Pede relatório	
	2		Pede dados da máquina
	3	Insere dados da máquina	
	4		Lê os dados
	5		Recolhe dados
	6		Gera relatório
	7		Apresenta relatório

O B T E R R E L A T Ó R I O S D E F A C T U R A Ç Ã O

Super Use Case	Obter Relatórios		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions			
Flow of Events	Actor Input	System Response	
	1	Pede relatório	
	2		Pede dados de pesquisa
	3	Insere dados	
	4		Lê os dados
	5		Pesquisa os dados
	6		Gera relatório
	7		Apresenta relatório

OBTER TABELAS DE DISTRIBUIÇÃO DE OCUPAÇÕES:

Super Use Case	Obter Relatórios		
Author	Grupo 1		
Date			
Brief Description			
Preconditions			
Post-conditions			
Flow of Events		Actor Input	System Response
	1	Pede relatório com tabelas de distribuição de ocupações	
	2		Pede nível e hora
	3	Insere o nível e a hora	
	4		Lê os dados
	5		Recolhe os dados
	6		Gera relatório
	7		Apresenta relatório

LISTAR PAGAMENTOS ATRASO:

Super Use Case			
Author	Grupo 1		
Date			
Brief Description	Funcionário lista todos os pagamentos em atraso.		
Preconditions			
Post-conditions			
Flow of Events		Actor Input	System Response
	1	Pede lista de pagamentos em atraso	
	2		Procura pagamentos em atraso
	3		Apresenta lista de pagamentos em atraso.
Exception 2a: Não Existem Pagamentos em Atraso	Actor Input	System Response	
	I	Informa que não existem pagamentos em atraso	

R E C E B E R P A G A M E N T O D E M U L T A P O R P E R D A D E B I-L H E T E :

Super Use Case		
Author	Grupo 1	
Date		
Brief Description		
Preconditions	O cliente perdeu o bilhete	
Post-conditions	O cliente tem permissão para sair do parque	
	Actor Input	System Response
1	Recebe pedido de pagamento de multa	
2		Pede dados do cliente
3	Insere dados do cliente	
4		Calcula montante a pagar
5	Recebe pagamento	
6	Regista pagamento no sistema	
7		Actualiza o estado do pagamento
8		Emite recibo
9		Informa sucesso da operação

A U X I L I A R C L I E N T E :

Super Use Case		
Author	Grupo 1	
Date		
Brief Description		
Preconditions	Cliente solicitou ajuda do Funcionário	
Post-conditions	Funcionário sabe qual é o portal ao auxilio	
	Actor Input	System Response
1	Recebe Notificação	
2		Verifica qual o portal de entrada a auxiliar
3		Indica qual o portal a auxiliar

RECEBER PAGAMENTO DE MULTA POR PERDA

Super Use Case		
Author	Grupo 1	
Date		
Brief Description		
Preconditions	O cliente perdeu o bilhete	
Post-conditions	O cliente tem permissão para sair do parque	
Flow of Events	Actor Input	System Response
	1 Recebe pedido de pagamento de multa	Pede dados do cliente
	2	
	3 Insere dados do cliente	
	4	Calcula montante a pagar
	5 Recebe pagamento	
	6 Regista pagamento no sistema	
	7	Actualiza o estado do pagamento
	8	Emite recibo
	9	Informa sucesso da operação

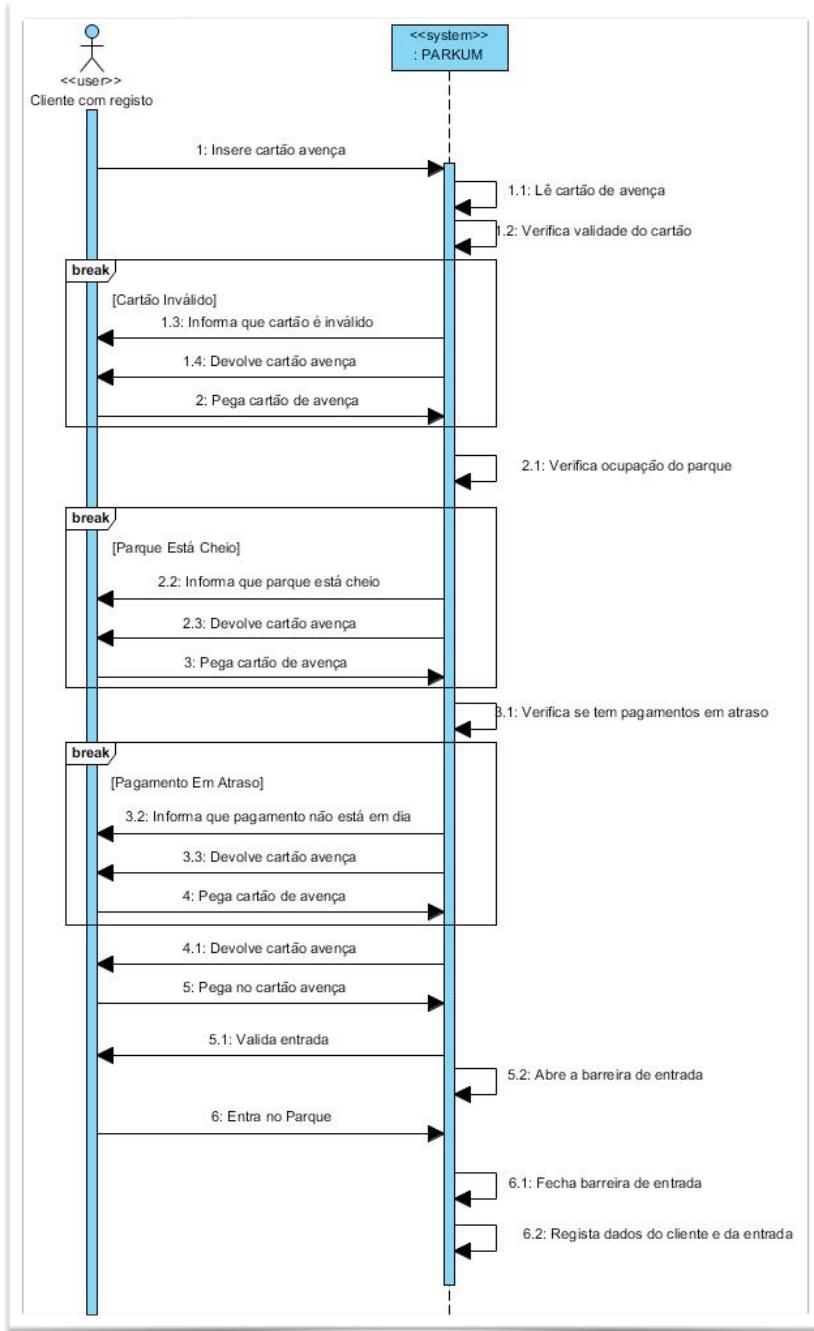
REGISTAR MANUTENÇÃO

Super Use Case		
Author	Grupo 1	
Date		
Brief Description	Funcionário faz a manutenção da máquina.	
Preconditions		
Post-conditions	Manutenção registada	
Flow of Events	Actor Input	System Response
	1 Pede para registar manutenção	
	2	Pede dados do funcionário
	3 Insere dados do funcionário	
	4	Lê os dados
	5	Verifica a validade dos dados
	6	Pede dados da máquina
	7 Insere dados da máquina	
	8	Lê dados
	9	Verifica se a máquina existe no sistema
	10	Regista manutenção
II Informa sucesso da operação		
Exception 4a: Dados Inválidos	Actor Input	System Response
1	Informa que os dados são inválidos	
2	Cancela operação	
Exception 7a: Máquina Não Existe	Actor Input	System Response
1	Informa que o número da máquina que inseriu não existe	
2	Cancela operação	

DIAGRAMAS DE SEQUÊNCIA

Os diagramas de sequência foram construídos com base nos diagramas de Use Case desenvolvidos. Estes diagramas representam a sequência dos processos do programa que envolve o sistema PARKUM. Como tal, os diagramas de sequência expõe todos os métodos existentes nas diversas classes, que implementam o PARKUM. A partir destes diagramas, temos a percepção de como interagem os objectos do sistema entre si, isto é, da sequência de processos efectuação perante cada acção do sistema.

Diagrama de Sequência do Use-Case Entrar modo Avença

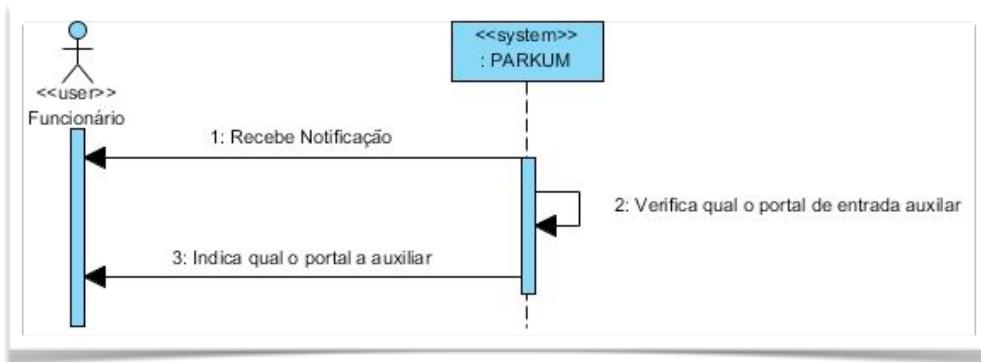


Neste diagrama de sequência, que representa o use case “Entra no Parque Modo Avenida”, verificamos que o actor Cliente com Registo interage várias vezes com sistema.

Primeiramente, o cliente quando entra no parque insere o cartão de avenida, esta mensagem é representada por uma seta horizontal de forma a comunicar com o sistema PARKUM. Na mensagem é enviado a forma como o sistema e o autor intervêm no parque de estacionamento. Seguidamente, e após o cliente tirar o bilhete, o sistema lê o cartão de avenida e verifica a validade do cartão. Se o cartão não for válido esta excepção é representada num fragmento de interacção designado por *break*. Caso o cartão seja válido o sistema irá verificar se o parque está cheio. Se o parque estiver cheio será, novamente, representado no diagrama uma nova excepção. Seguidamente o sistema irá verificar se o cliente tem os pagamentos em dia , outra excepção poderá acontecer se o cliente não possuir os pagamentos em dia o sistema.

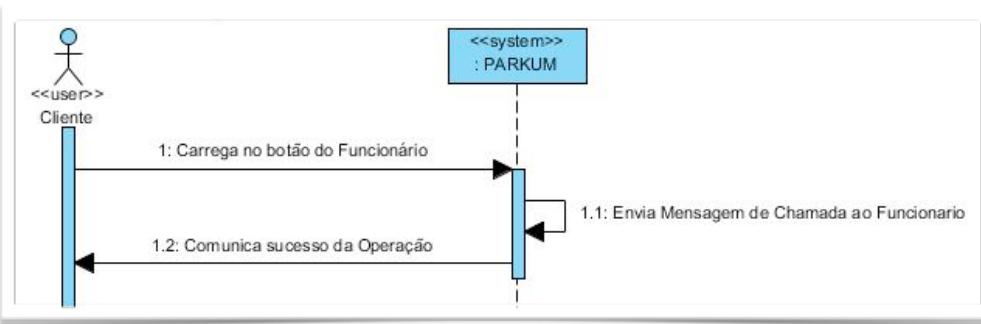
Se nenhuma excepção ocorrer, o sistema devolve o cartão de avenida, o cliente recolhe o cartão, valida a entrada, e abre a barreira de entrada. Entretanto, o cliente entra no parque, o fecha a barreira e o sistema regista os dados do cliente e da entrada.

Diagrama de Sequência do Use-Case Auxiliar Cliente



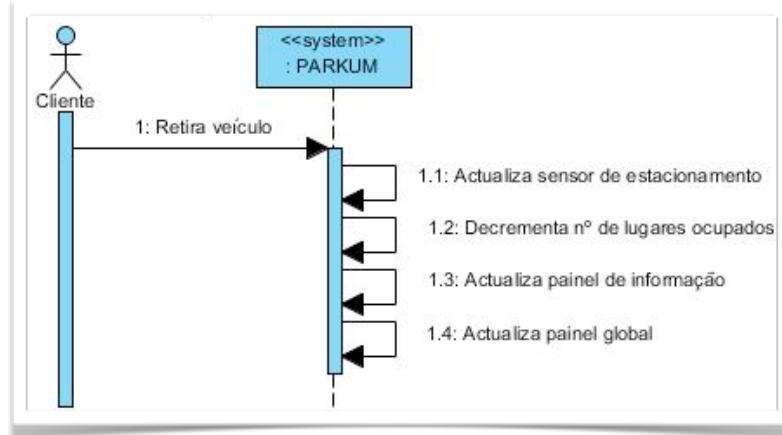
Este diagrama, representa o use case “Auxiliar Cliente”. Verificamos que o funcionário recebe uma notificação do sistema PARKUM, a informar que o cliente precisa de auxilio no portal de entrada. De seguida, o sistema verifica qual o portal a auxiliar, e indica o funcionário.

Diagrama de Sequência do Use-Case Contactar Funcionário



Quando o cliente pretende contactar o funcionário, o sistema envia uma mensagem de chamada ao funcionário.

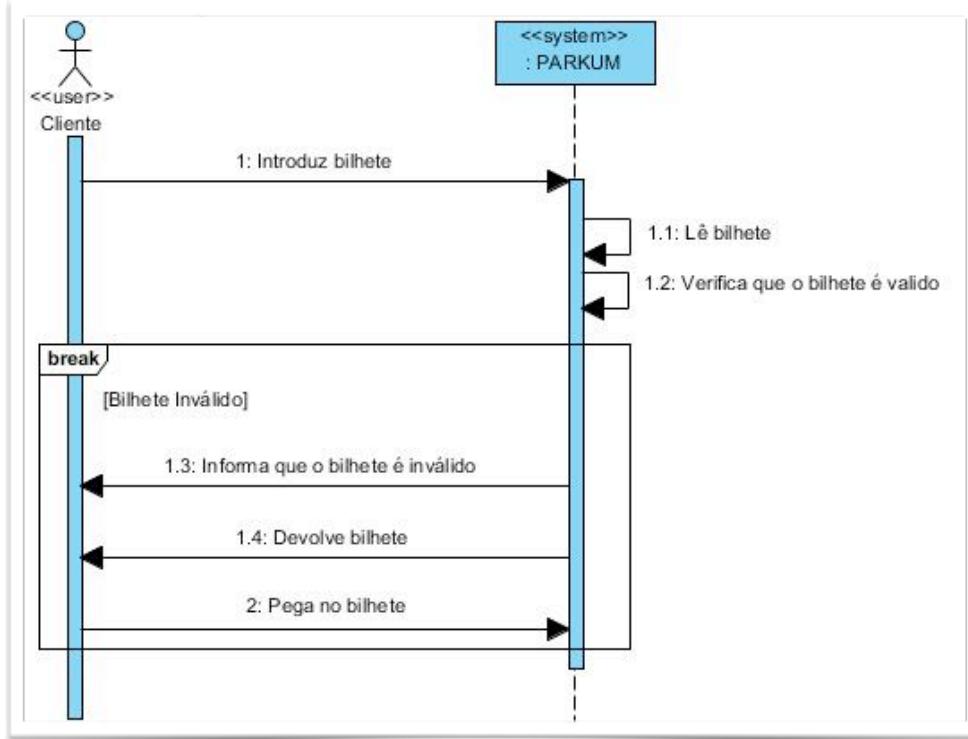
Diagrama de Sequência do Use-Case Desocupar Lugar



Este diagrama representa a acção do cliente quando desocupa um lugar de estacionamento.

Primeiramente, quando o cliente retira o veículo, o sistema PARKUM actualiza a cor do sensor para verde. De seguida, o sistema volta a decrementar o número de lugares ocupados e actualiza a o painel de informação e o painel global.

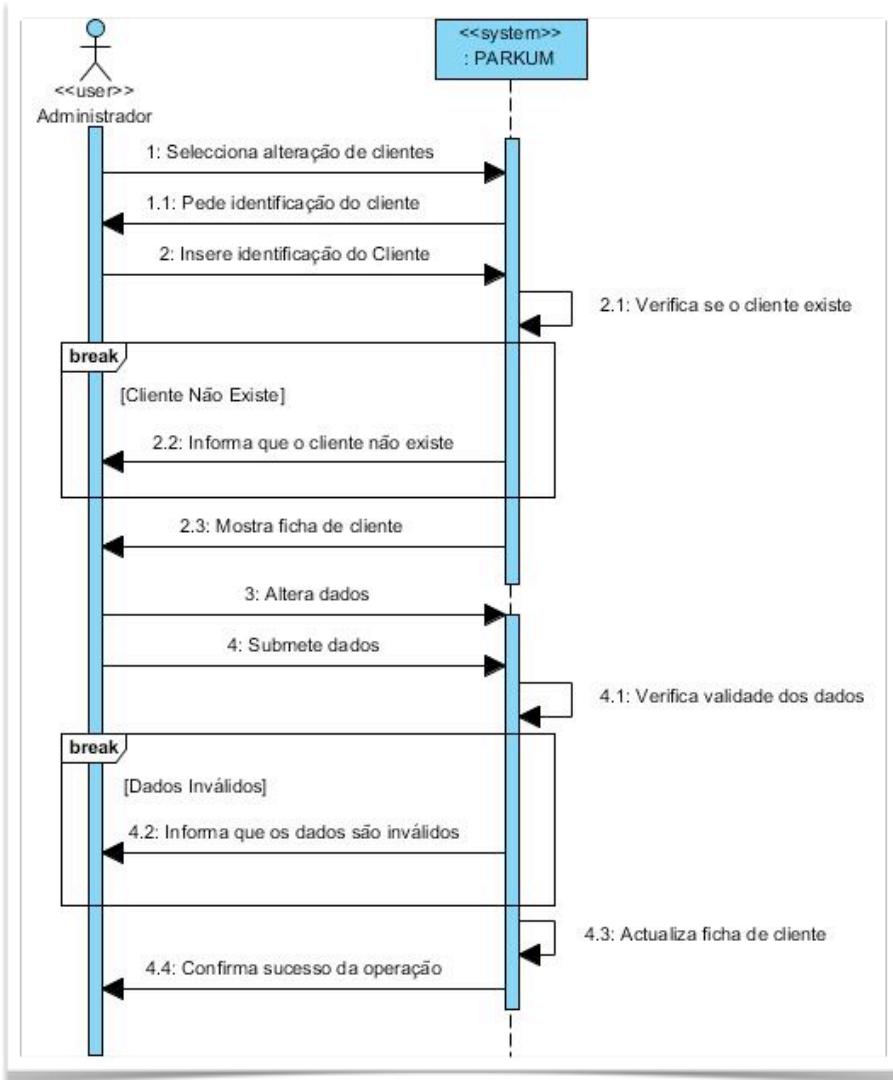
Diagrama de Sequência do Use-Case Devolver Bilhete



O cliente introduz o bilhete, o sistema lê o bilhete e de seguida verifica se é válido.

Aqui poderá ocorrer um excepção, representada no diagrama através de um *break*, caso o bilhete seja inválido. Neste caso, o sistema informa que o bilhete é inválido, devolve o bilhete e o cliente pega no bilhete.

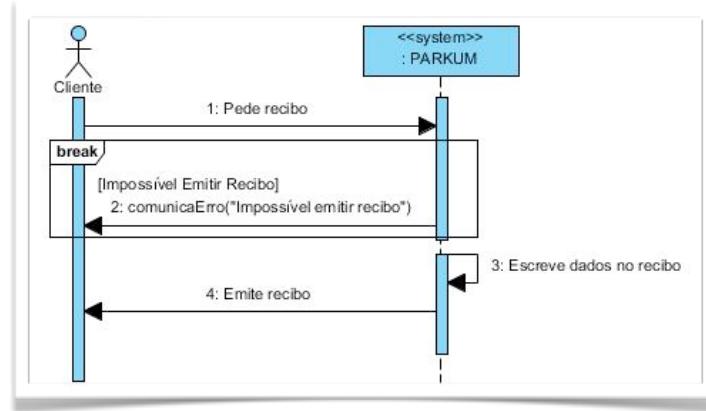
Diagrama de Sequência do Use-Case Editar Cliente



No sistema, o administrador do sistema selecciona a opção de alteração de cliente. O sistema pede a identificação do cliente que se pretende alterar. O administrador, insere a identificação do cliente. O sistema verifica se o cliente existe. Caso o administrador tenha colocado um nome de um cliente que não existe, existe uma excepção, que indica que o cliente não existe no sistema. No entanto, caso exista de facto o cliente, o sistema apresenta a ficha do cliente. O administrador altera os dados que pretende, submete os dados e, por último, verifica se os dados são válidos. Aqui poderá ocorrer, novamente, uma excepção se os dados do cliente não

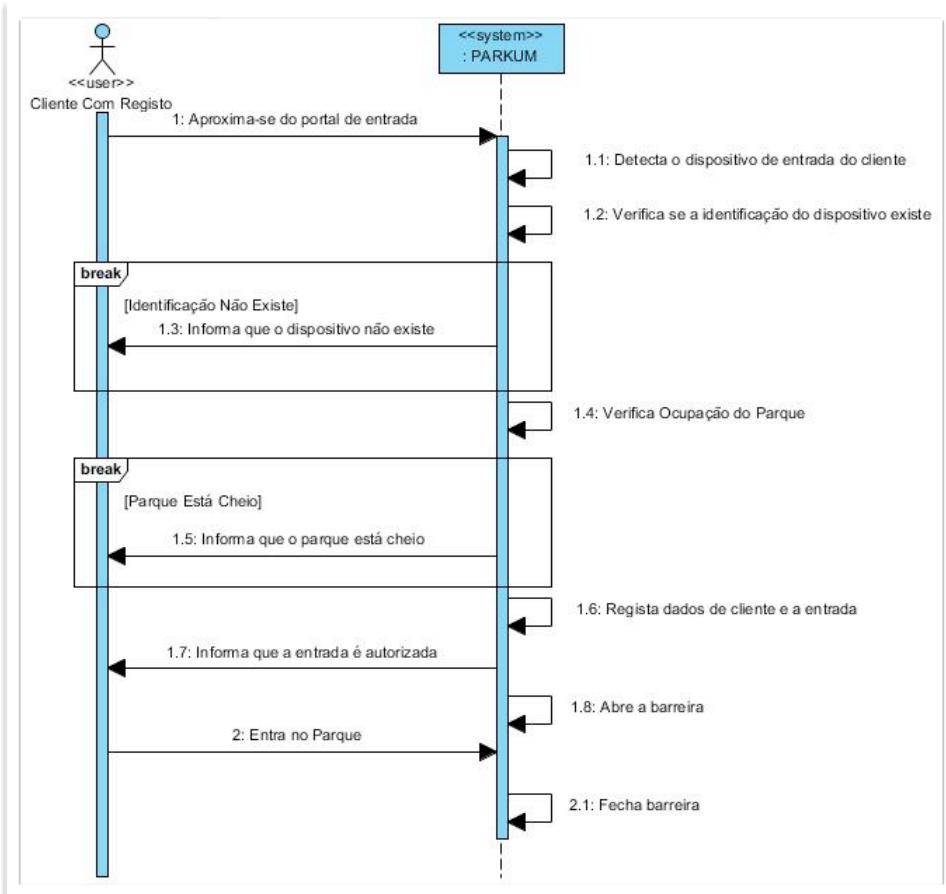
forem válidos. De seguida, é actualiza a ficha do cliente. O sistema informa o funcionário que operação foi realizada com sucesso.

Diagrama de Sequência do Use-Case Emitir Recibo



O cliente pede recibo ao sistema. O sistema escreve dados do recibo e emite o recibo ao cliente. Caso ocorra algum problema em emitir o recibo, o sistema informa o erro ao cliente.

Diagrama de Sequência do Use-Case Entrar Modo Automático

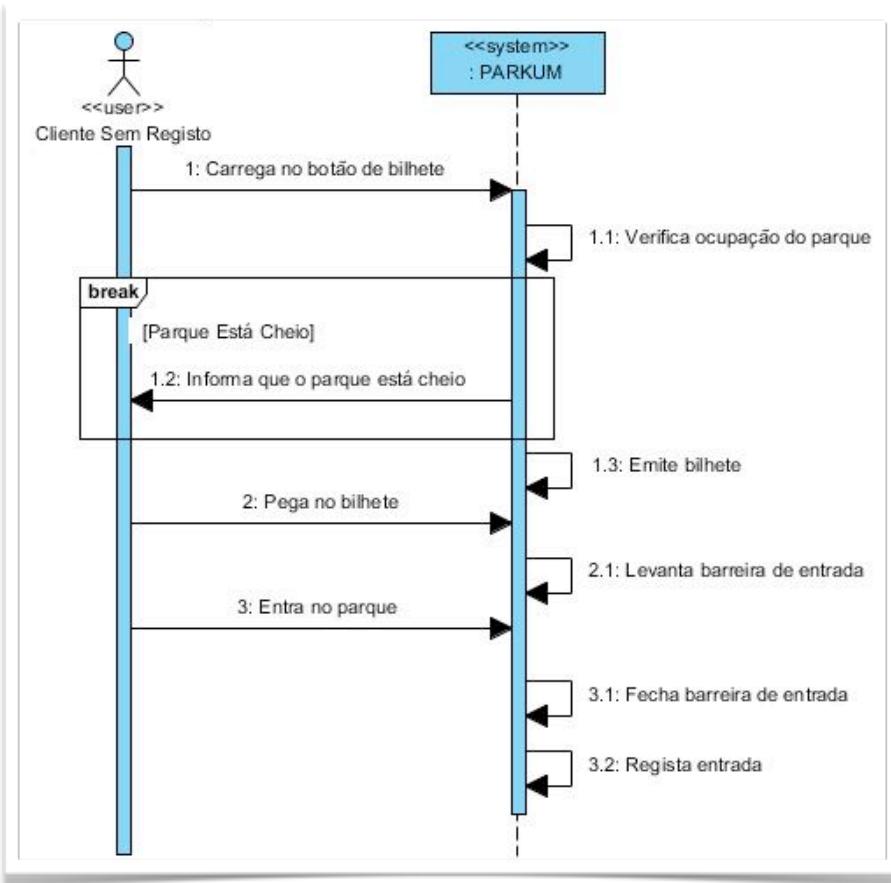


Quando o cliente com registo aproxima-se do portal de entrada, o sistema PARKUM detecta qual o dispositivo de entrada que o cliente possui e verifica se existe a identificação do dispositivo no sistema. De seguida, o sistema verifica se o parque está cheio. O PARKUM informa que a entrada é autorizada e abre a barreira. O cliente entra no parque o sistema fecha a barreira e regista os dados da entrada cliente.

Poderá ocorrer uma excepção caso o sistema não detecte a identificação do dispositivo.

E também caso o parque esteja cheio. Nestas situações o sistema informa o cliente das excepções.

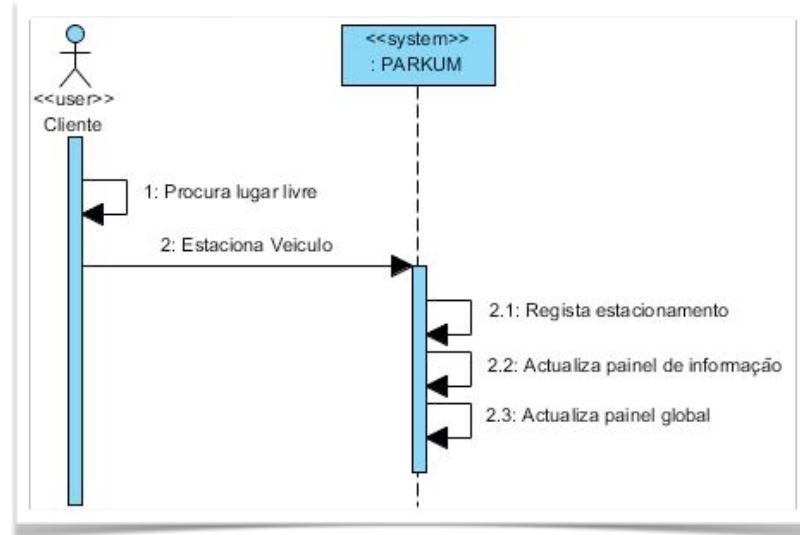
Diagrama de Sequência do Use-Case Entrar no parque



Neste diagrama verificamos que o cliente sem registo carrega no botão de bilhetes. Seguidamente, o PARKUM verifica a ocupação do parque. O cliente é informado de que a sua entrada é autorizada, e o sistema abre a barreira de entrada. O cliente entre no parque e o PARKUM fecha a barreira.. É registado os dados do cliente e da entrada, no sistema.

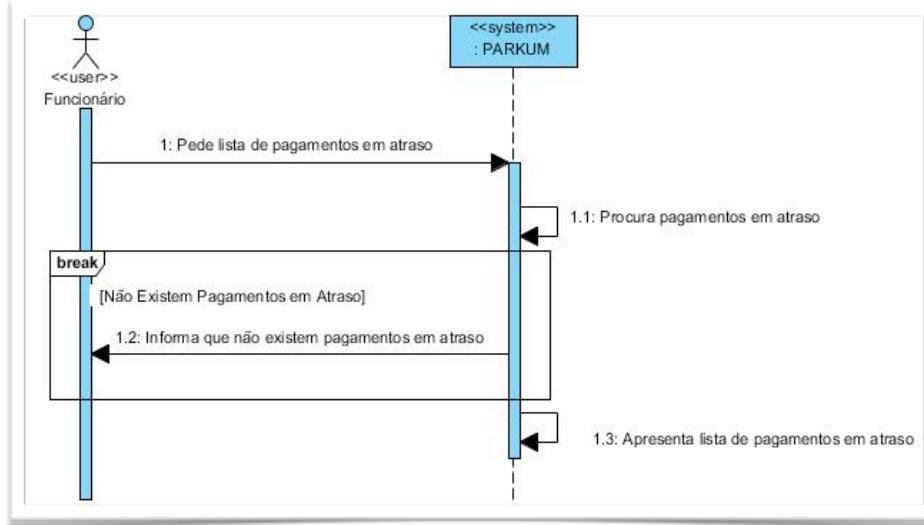
Poderá ocorrer uma excepção caso o sistema não detecte a identificação do dispositivo, e também caso o parque esteja cheio. Nestas situações o sistema informa o cliente das excepções.

Diagrama de Sequência do Use-Case Estacionar Veiculo



O cliente depois de entrar no parque procura um lugar livre. Após efectuar o estacionamento, o sistema regista o estacionamento, actualiza o painel de informação e o painel global.

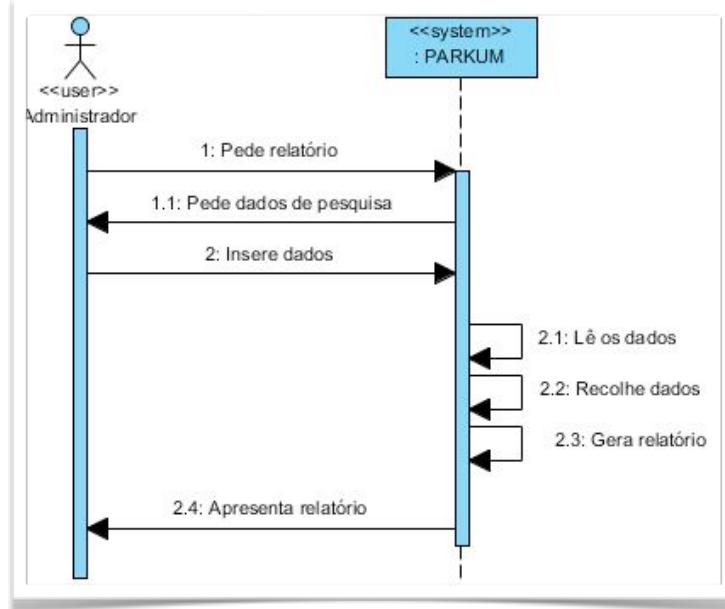
Diagrama de Sequência do Use-Case Listar Pagamentos em Atraso



Neste diagrama verifica-se que o funcionário pede ao sistema a lista de pagamentos em atraso. O sistema procura os pagamentos em atraso, e de seguida apresenta a lista dos cliente com pagamentos em atraso.

Poderá ocorrer uma exceção caso não exista nenhum pagamento em atraso.

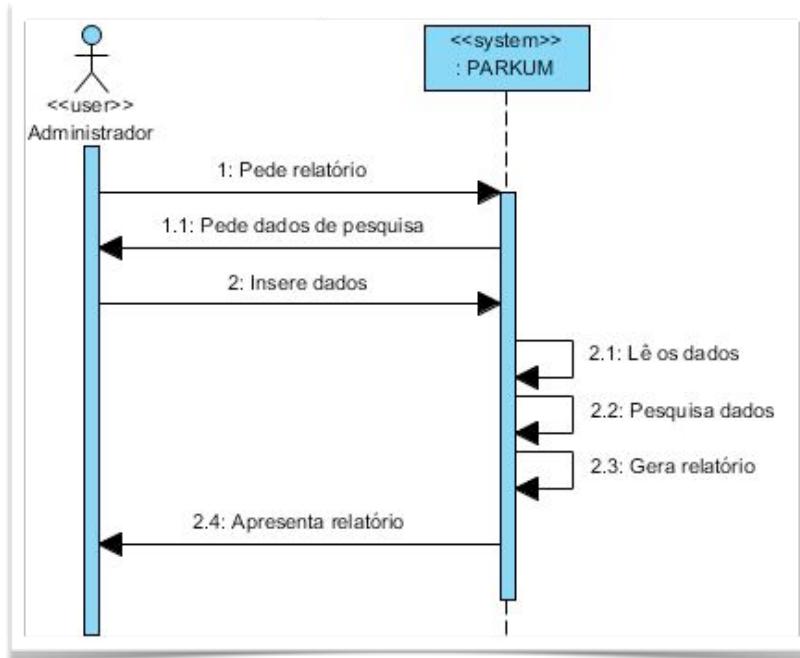
Diagrama de Sequência do Use-Case Obter Relatórios Diários



Este diagrama descreve o modo de obter relatórios diários.

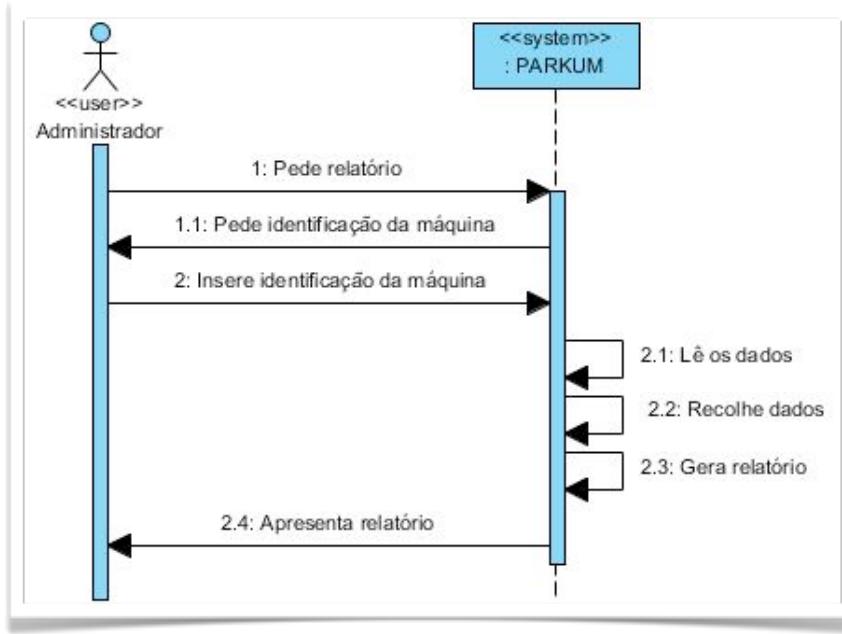
Primeiramente, o funcionário pede o tipo de relatório diário ao sistema. O sistema pede os dados de pesquisa, o administrador insere os dados de pesquisa. De seguida o sistema lê os dados, recolhe os dados, gera o relatório, e por fim apresenta relatório ao funcionário.

Diagrama de Sequência do Use-Case Obter Relatórios Facturação



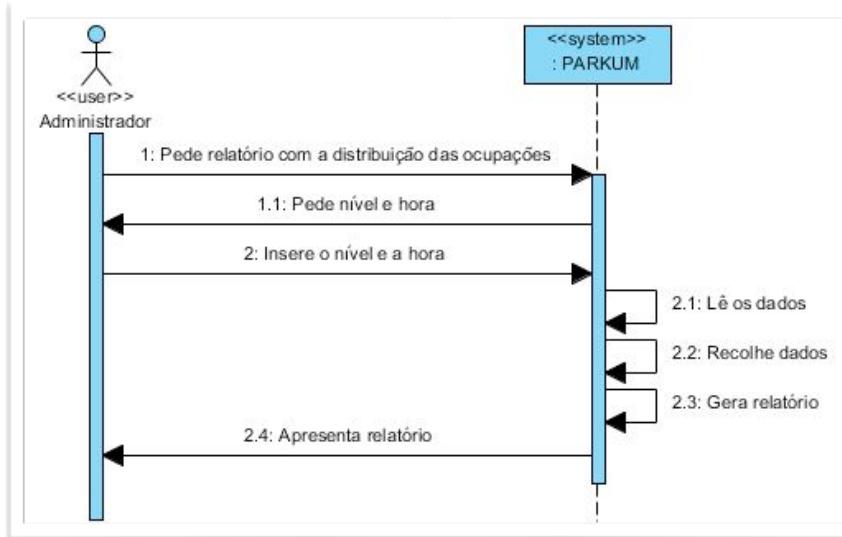
O funcionário pede para emitir relatório de facturação ao sistema. O sistema pede os dados de pesquisa, o administrador insere os dados de pesquisa. De seguida o sistema lê os dados, recolhe os dados, gera o relatório, e por fim apresenta relatório ao funcionário.

Diagrama de Sequência do Use-Case Obter Relatórios Máquinas de Pagamento



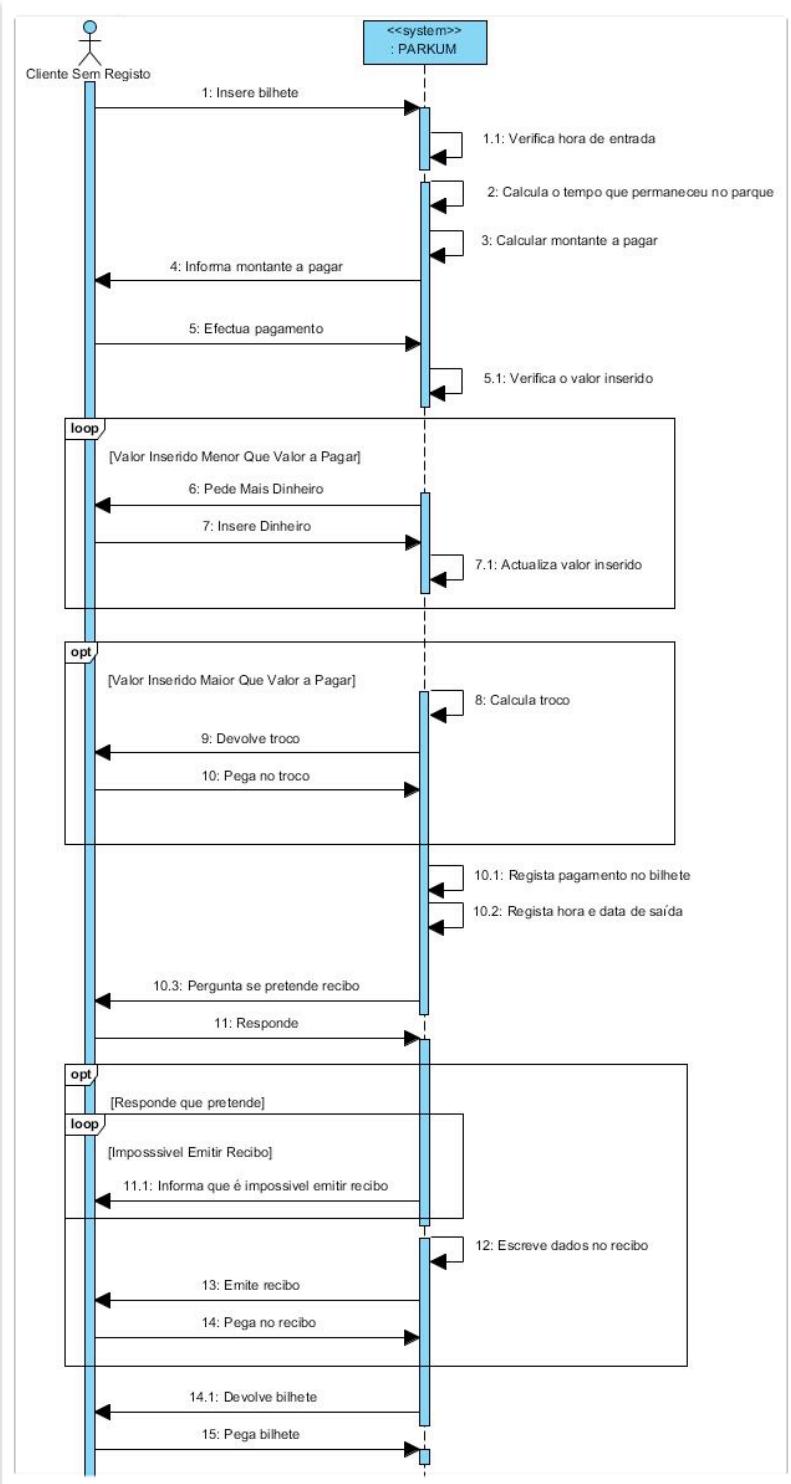
Esta acção inicia-se quando o funcionário pede relatório das máquinas de pagamento, ao sistema. O sistema pede ao funcionário a identificação da máquina que se pretende emitir relatório. De seguida o sistema lê os dados, recolhe os dados, gera o relatório, e por fim apresenta-o ao funcionário.

Diagrama de Sequência do Use-Case Obter Tabelas de Distribuições



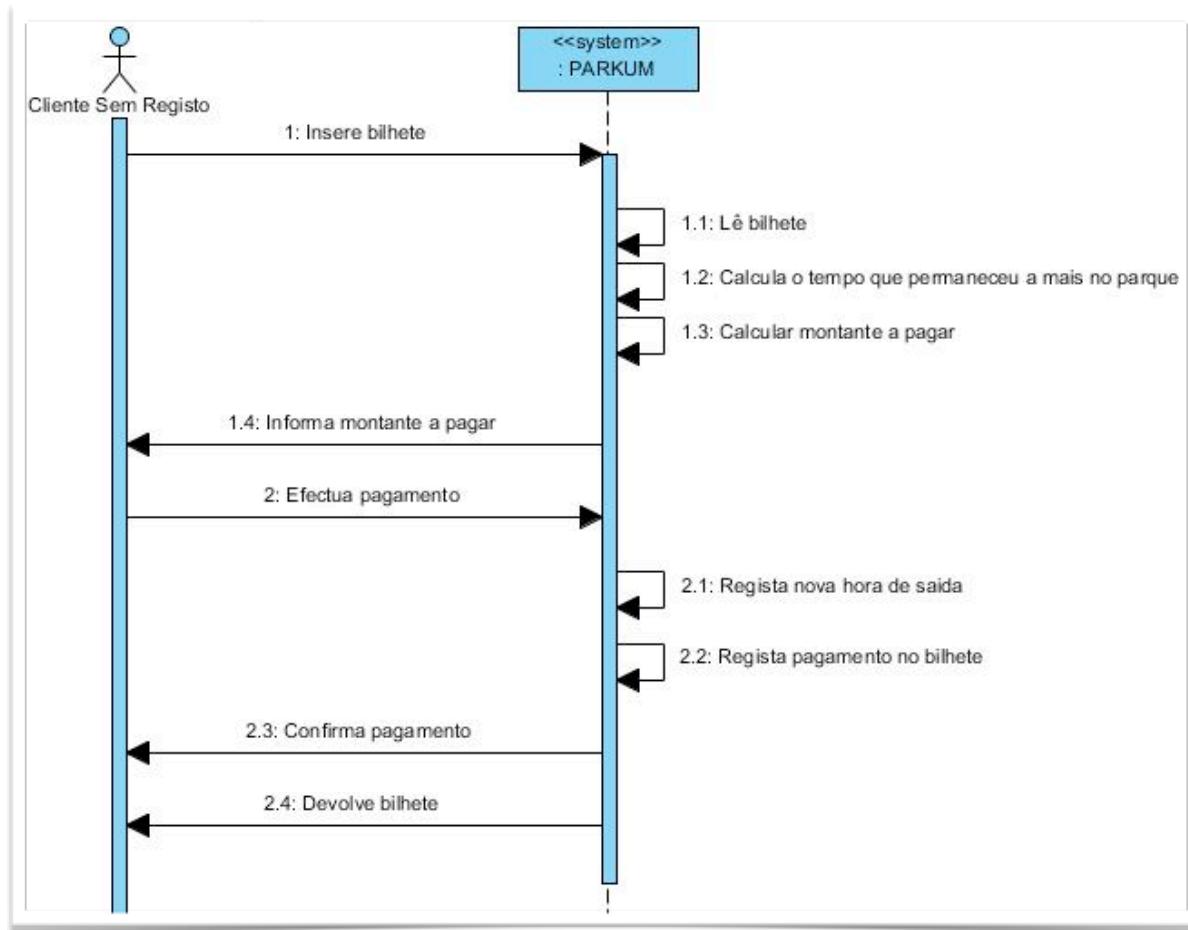
Inicialmente, o funcionário informa o sistema que pretende um relatório com as distribuições das ocupações do sistema. O sistema pede ao funcionário o nível e a hora que pretende e o funcionário insere os dados. O sistema lê os dados, recolhe os dados, gera relatório, e apresenta-o ao funcionário.

Diagrama de Sequência do Use-Case Pagar Bilhete



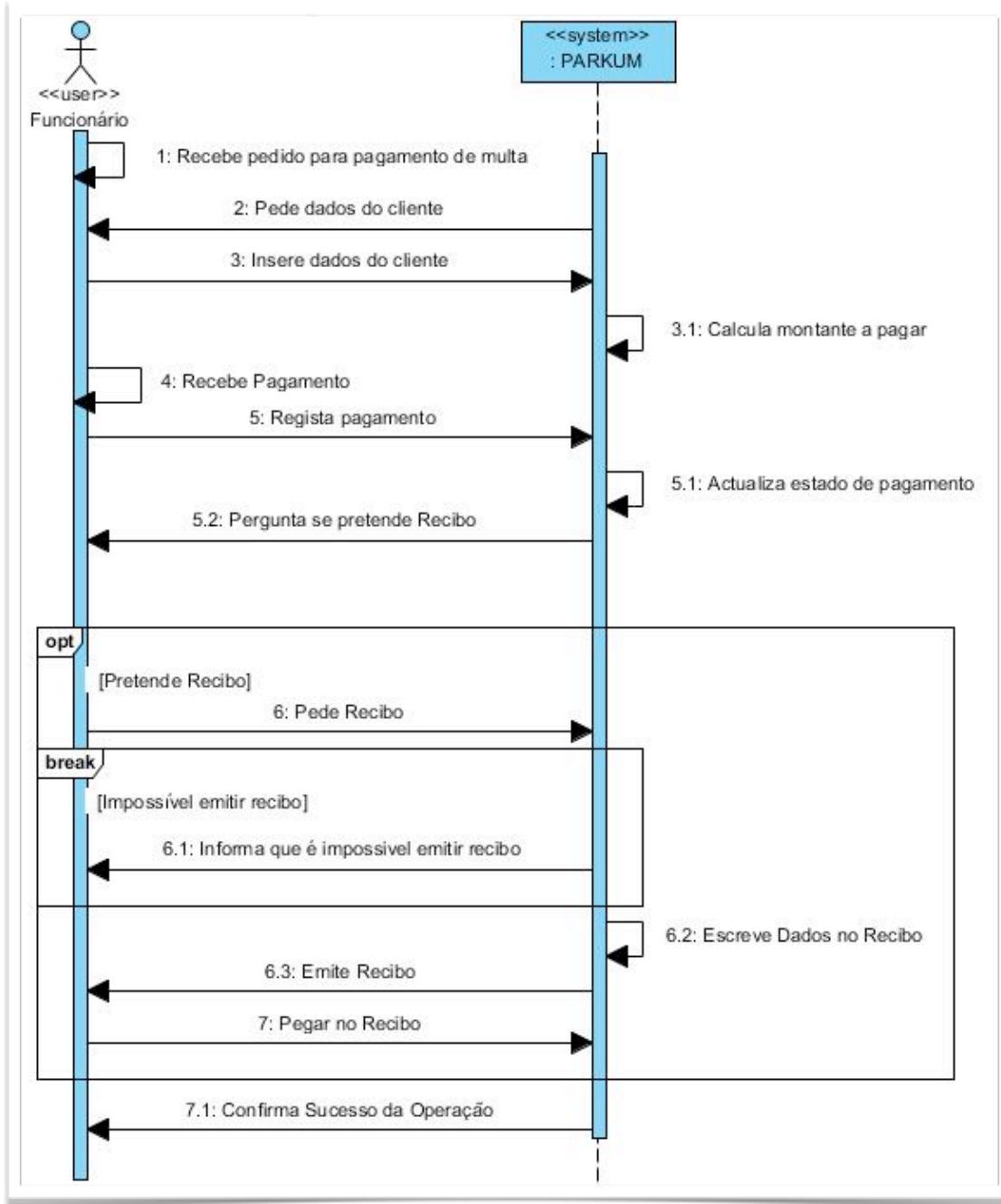
Relativamente ao diagrama de pagar bilhete, pode-se verificar que começa com o cliente sem registo a inserir o bilhete no sistema. O PARKUM verifica a hora de entrada, calcula o tempo que o cliente permaneceu no parque e calcula o montante a pagar. De seguida, o sistema informa ao cliente o valor do montante a pagar. O cliente efectua o pagamento, e o sistema verifica o valor inserido. Note-se agora, que enquanto o valor inserido for menor que o valor a pagar, o sistema entra em ciclo (*loop*), de modo a o cliente inserir o montante correcto. Caso o montante inserido for maior que o valor a pagar o sistema calcula o troco, devolve o troco ao cliente. Depois do cliente efectuar o pagamento o sistema regista o pagamento no bilhete, a hora e a data de saída. Por conseguinte, o sistema notifica o cliente se pretende recibo do pagamento. Se o cliente pretender o recibo, o sistema escreve os dados no recibo e depois emite o mesmo. De seguida, o sistema devolve o bilhete. E o cliente recolhe o bilhete já pago.

Diagrama de Sequência do Use-Case Pagar Multa por Atraso



Caso o cliente necessite de pagar multa por atraso terá de inserir o bilhete no sistema. O sistema lê o bilhete, calcula o tempo que permaneceu a mais no parque, e de seguida calcula o montante a pagar. De imediato, o sistema informa o cliente do valor do montante a pagar, e o cliente efectua o pagamento. O PARKUM, regista nova hora de saída, e regista o pagamento no bilhete. O cliente confirma o valor a pagar e o sistema devolve o bilhete.

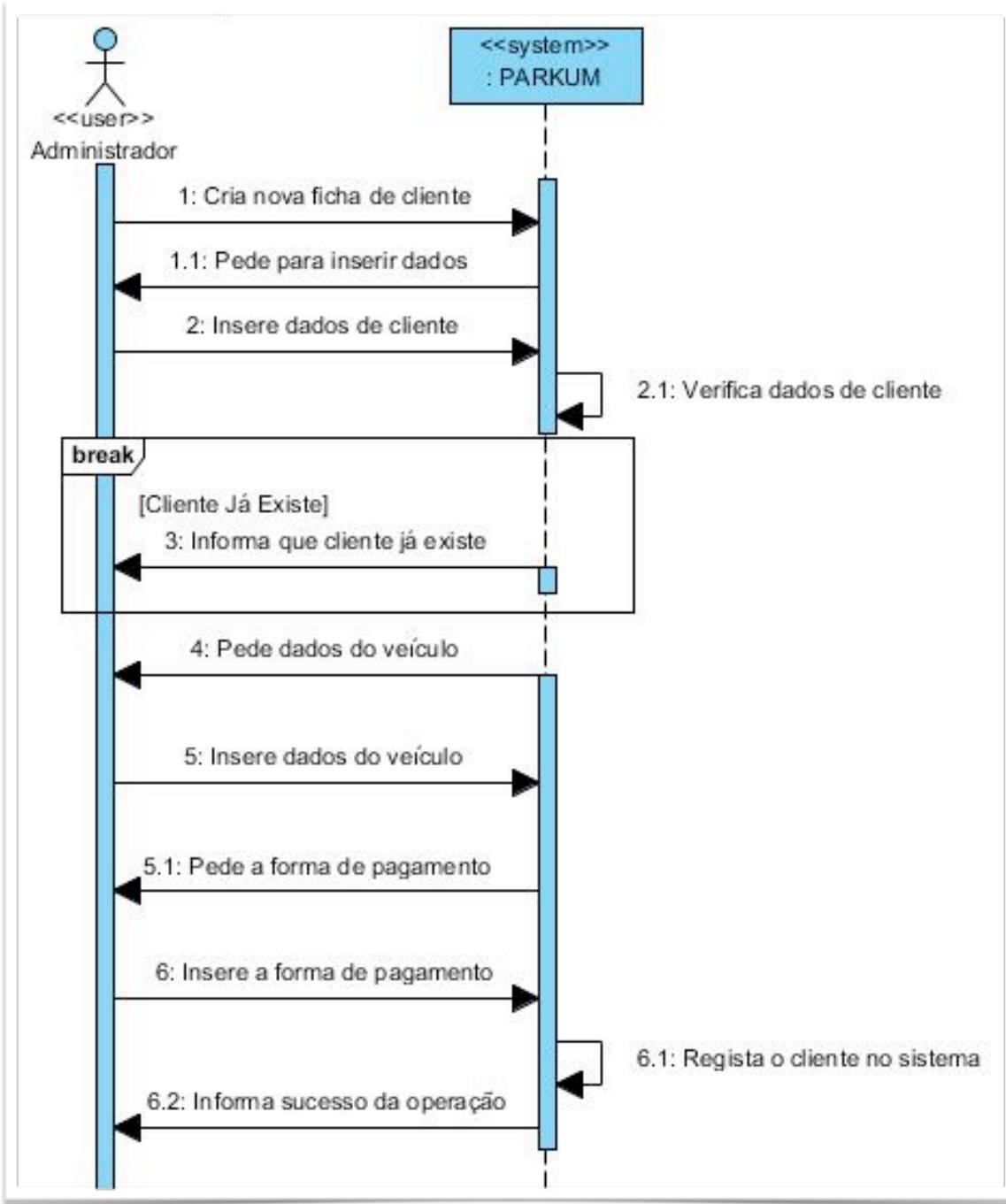
Diagrama de Sequência do Use-Case Receber Pagamento de Multa



O funcionário é notificado a receber pagamento de multa. O PARKUM pede os dados do cliente, o funcionário insere os dados do cliente. O sistema calcula o valor do montante a pagar.

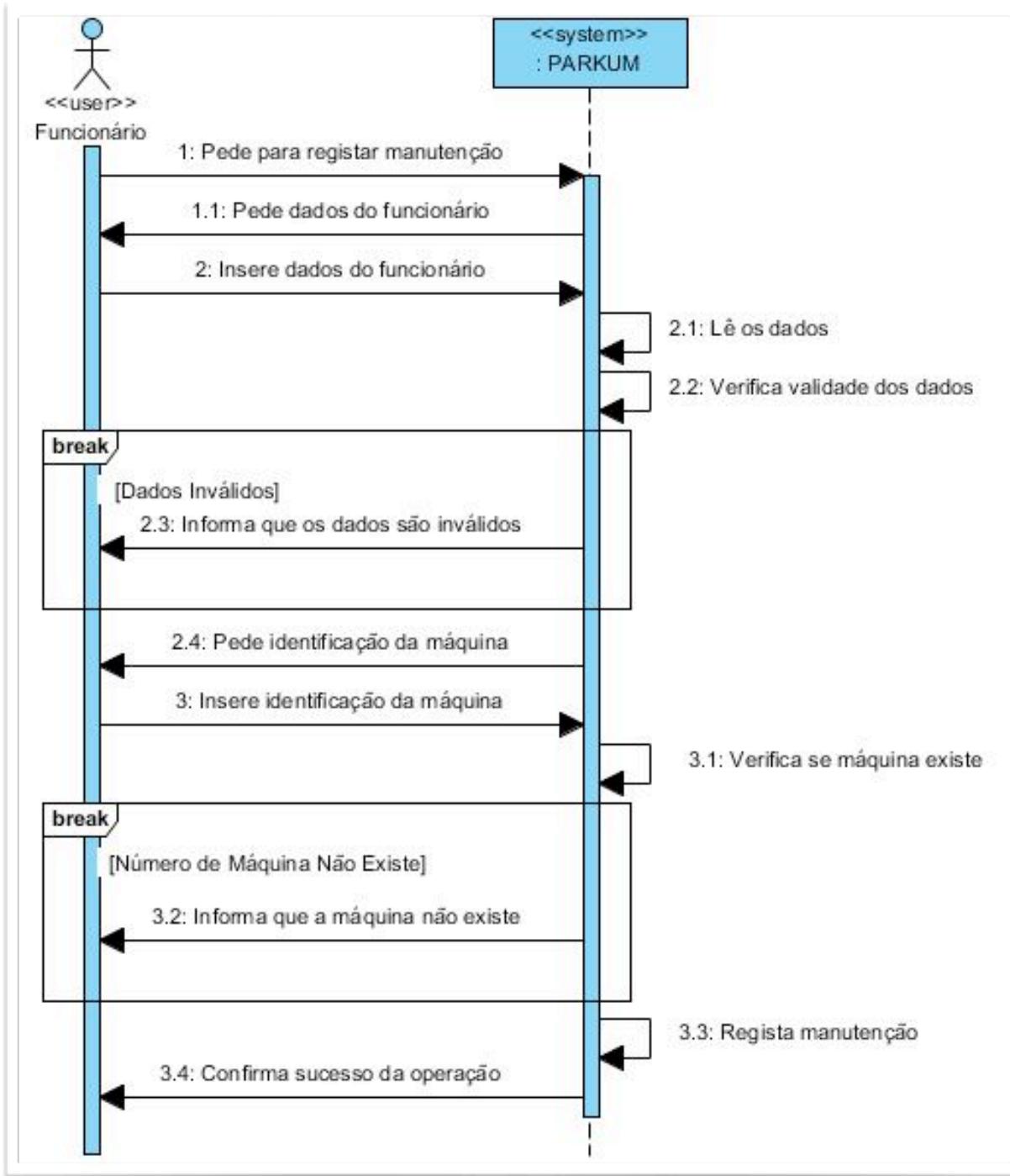
O funcionário após receber o pagamento, regista o pagamento da multa no sistema. O sistema actualiza o estado de pagamento, e informa o cliente se pretende recibo. Se o cliente quiser recibo, o sistema escreve os dados no recibo, de seguida emite o recibo. O cliente pega no recibo, e o sistema confirma sucesso da operação.

Diagrama de Sequência do Use-Case Registar Cliente



Neste diagrama de sequência é possível verificar que o administrador do sistema pretende criar uma nova ficha de cliente. O sistema pede ao administrador para inserir os dados do cliente. Depois de inserir os dados o sistema verifica a validade dos dados e pede os dados do veículo. De seguida, o sistema pede a forma de pagamento. O administrador após especificar o modo de pagamento do cliente, regista o cliente no sistema e informa o administrador do sucesso da operação.

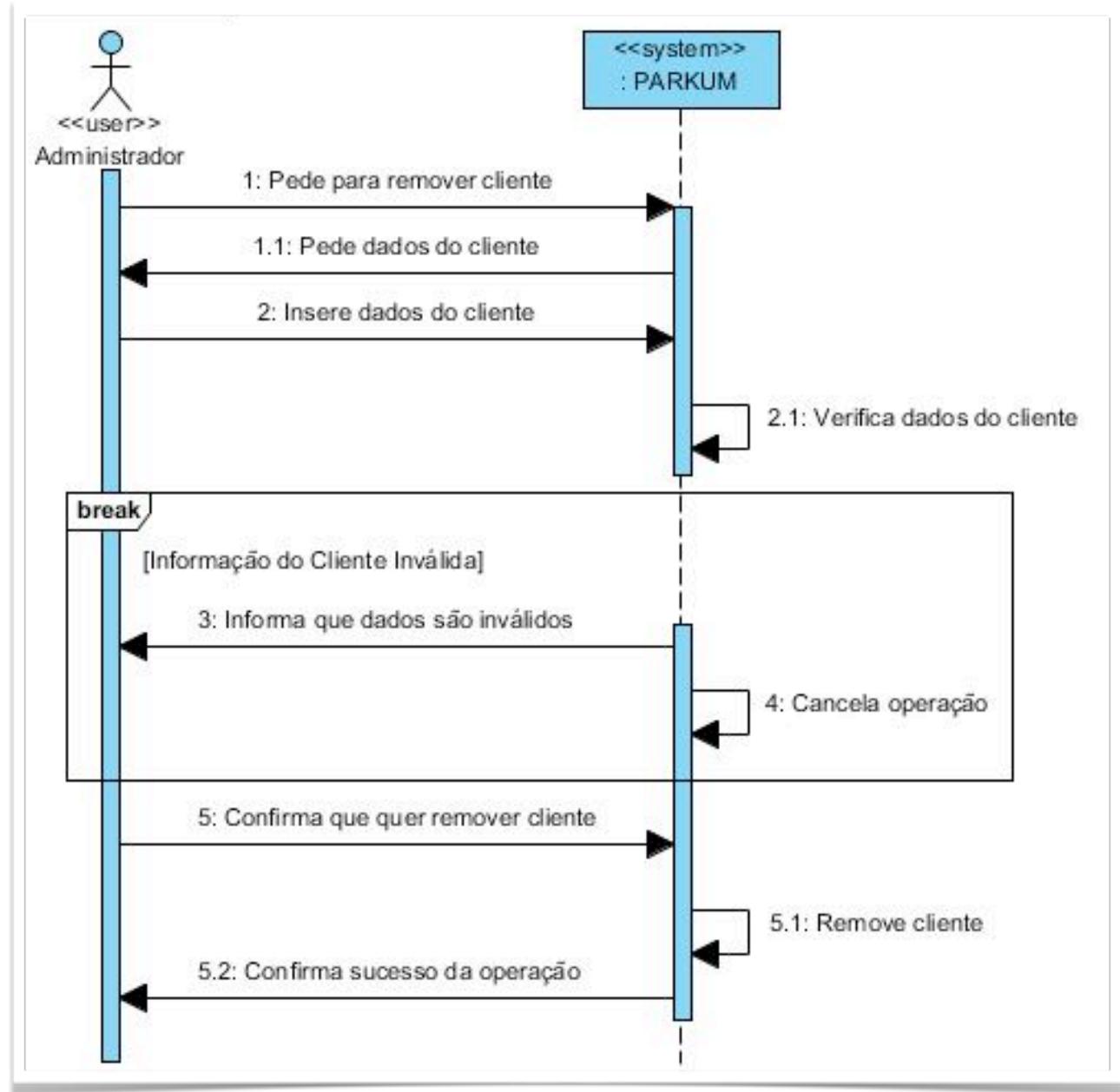
Diagrama de Sequência do Use-Case Registar Manutenção



Primeiramente, o funcionário pede ao sistema para registrar manutenção da máquina de pagamento. O sistema pede ao funcionário para inserir os seus dados no sistema. O PARKUM lê os dados e verifica a sua validade. Por conseguinte, o sistema pede a identificação da máquina e o administrador insere a identificação da máquina que pretende registar a manutenção. O sistema verifica se a máquina existe e regista a sua manutenção, informando o funcionário o sucesso da operação.

É ainda importante realçar que se eventualmente os dados inseridos forem inválidos ou o número da máquina inserido pelo funcionário não existir, o sistema retorna a operação com uma mensagem de erro.

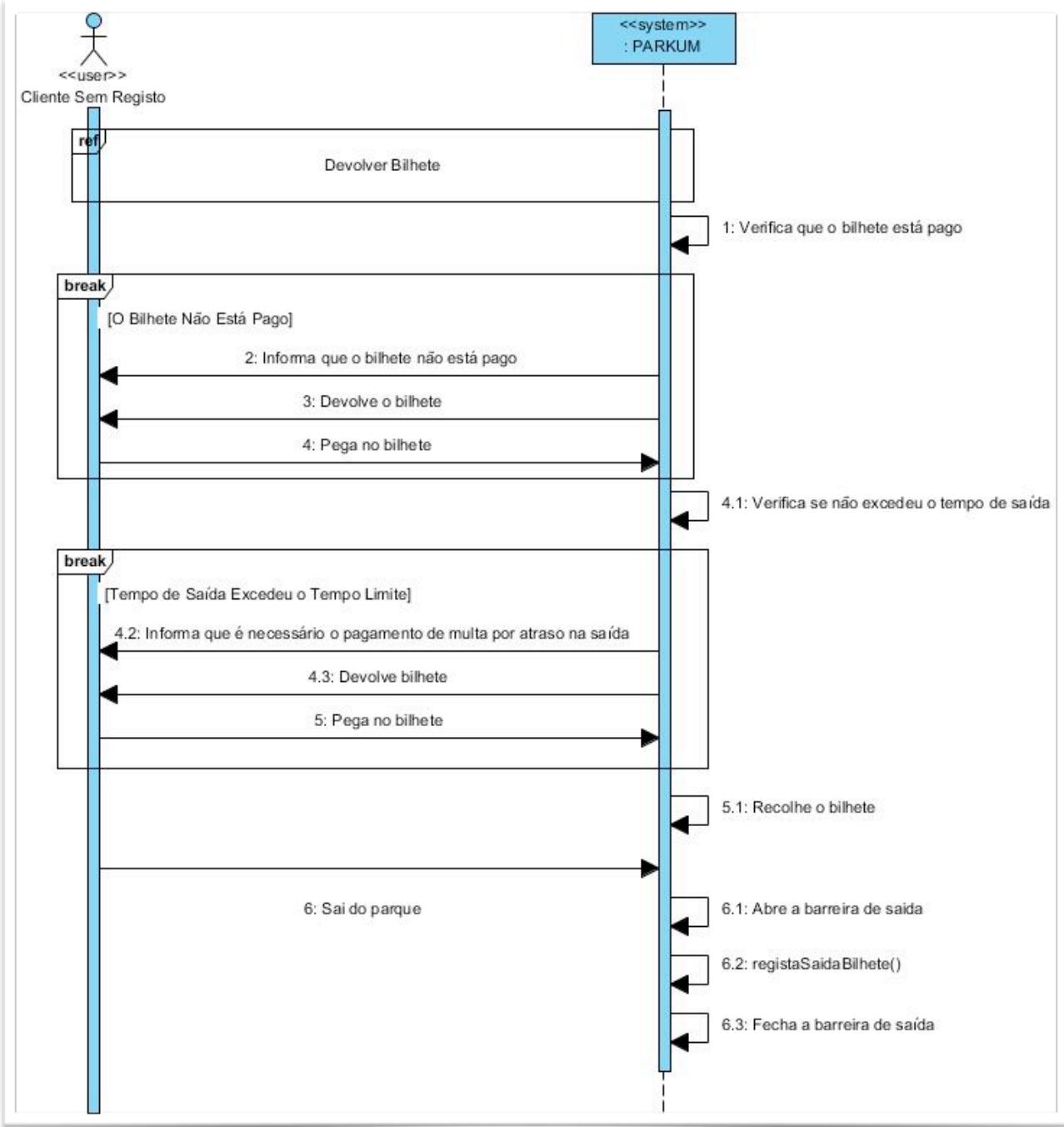
Diagrama de Sequência do Use-Case Remover Cliente



O administrador informa o sistema que pretende remover um cliente do sistema. O PARKUM pede os dados do cliente. Depois do administrador ter inserido a informação do cliente o sistema verifica os dados. O administrador confirma que pretende remover o cliente. Por fim o sistema remove o cliente e confirma o sucesso de operação.

Poderá ocorrer um *break* acaso a informação do cliente seja inválida.

Diagrama de Sequência do Use-Case Sair do Parque

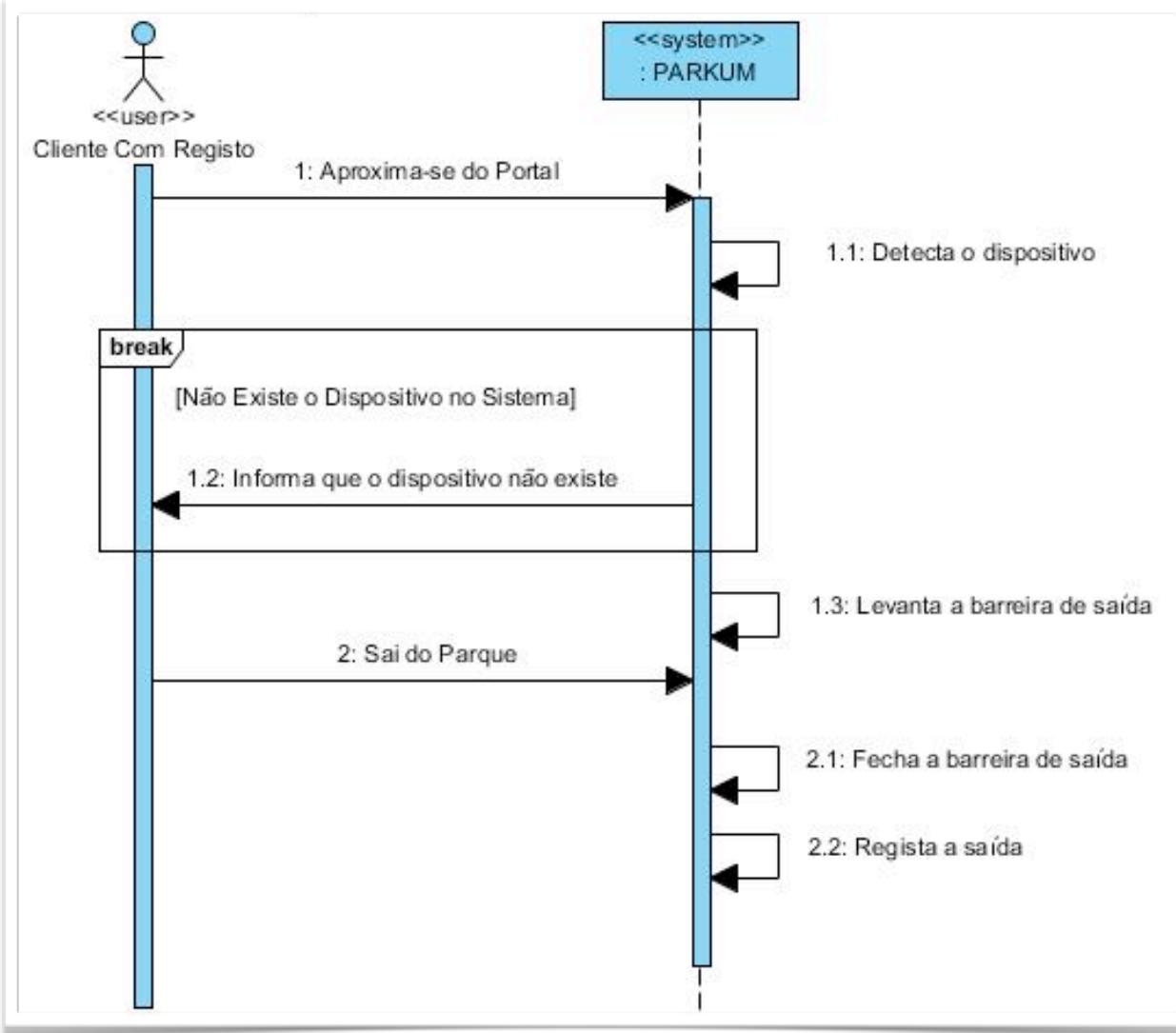


Neste diagrama de sequência verificamos que esta acção é executada fazendo referência a outro diagrama de sequência: devolver bilhete. Seguidamente, o sistema verifica se o bilhete está pago, e se o cliente não excedeu o tempo limite de saída. Se não ocorrer nenhuma exceção o sistema recolhe o bilhete, e regista a saída. O cliente sai do parque, o sistema abre a barreira de saída. O sistema fecha a barreira e regista a saída do cliente no parque.

Caso se verifique que o bilhete que o cliente introduziu no sistema não esteja pago, acontecerá um *break* e o sistema informa que o bilhete não foi pago e devolve o bilhete ao cliente.

Por sua vez, caso se verifique que o tempo de saída ultrapassou o tempo limite, o PARKUM informa que é necessário o pagamento de multa por atraso na saída. O sistema devolve o bilhete ao cliente.

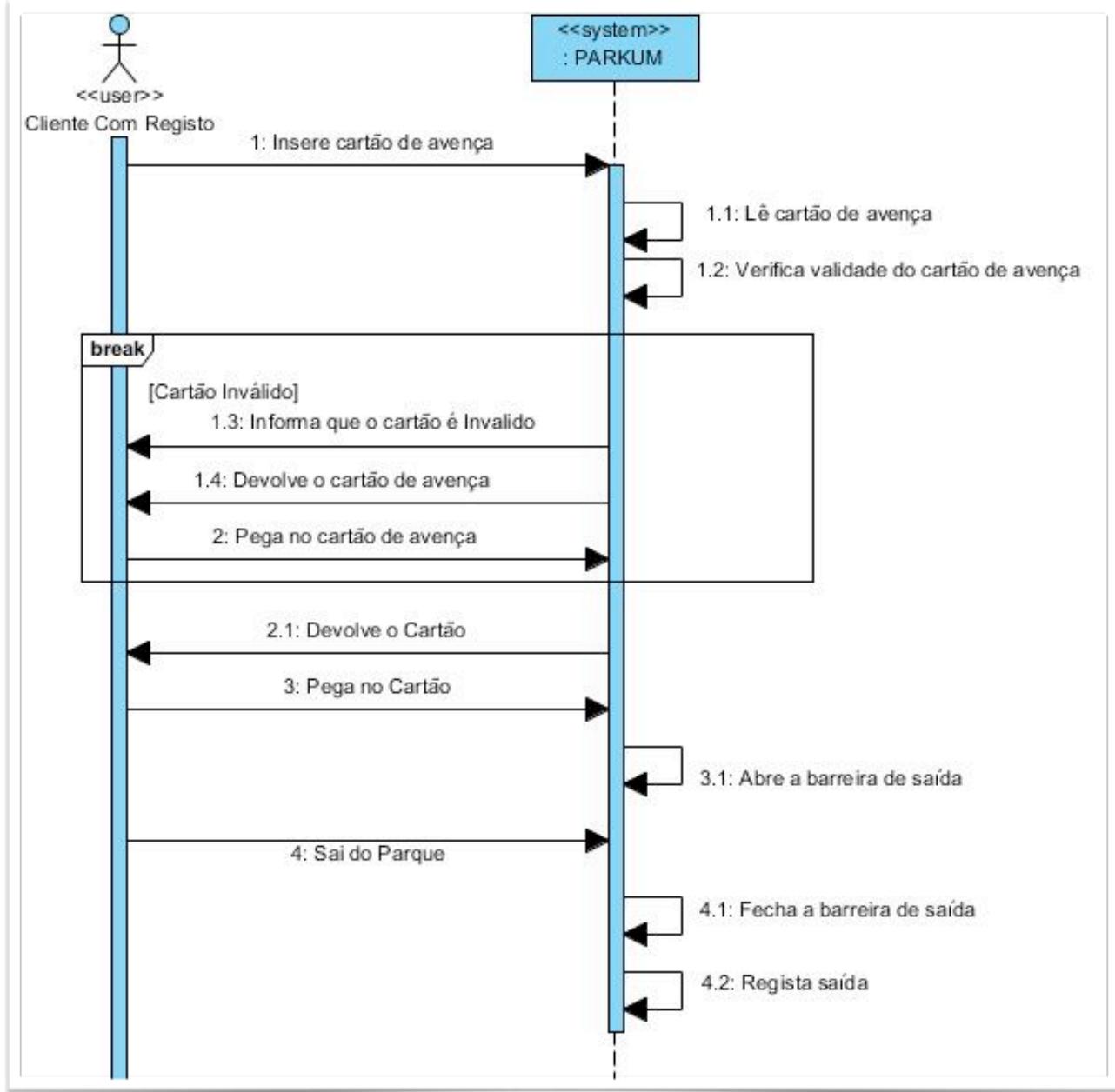
Diagrama de Sequência do Use-Case Sair Modo Automático



Este diagrama de sequência inicia-se quando o cliente com registo aproxima-se do portal de entrada e o sistema detecta o dispositivo automático existente no veículo. O sistema abre a barreira de saída, o cliente sai do parque. De seguida, o sistema regista a saída do cliente no parque e fecha a barreira.

Poderá ocorrer um excepção quando o cliente não possui dispositivo automático.

Diagrama de Sequência do Use-Case Remover Cliente



Como se pode verificar neste diagrama de sequência inicia-se quando o actor, o cliente sem registo, insere o cartão de avenida no sistema. O sistema lê o cartão e verifica a sua validade. O sistema devolve o bilhete ao cliente. Depois de o cliente pegar no cartão de avenida, o sistema abre a barreira de saída. O cliente sai do parque, o sistema fecha a barreira de saída e registra a saída do cliente.

Note-se que poderá ocorrer um *break* caso o sistema detecte que o cartão é inválido. Assim, após informar ao cliente da não validade do cartão os sistema devolve o bilhete ao cliente.

CONCEPÇÃO E DESENVOLVIMENTO

Após esta fase de análise de requisitos, é necessário desenvolver alguns pontos já identificados acima, de modo a obter uma informação mais detalhada de todo o projecto.

IDENTIFICAÇÃO DOS SUBSISTEMAS DO PARKUM:

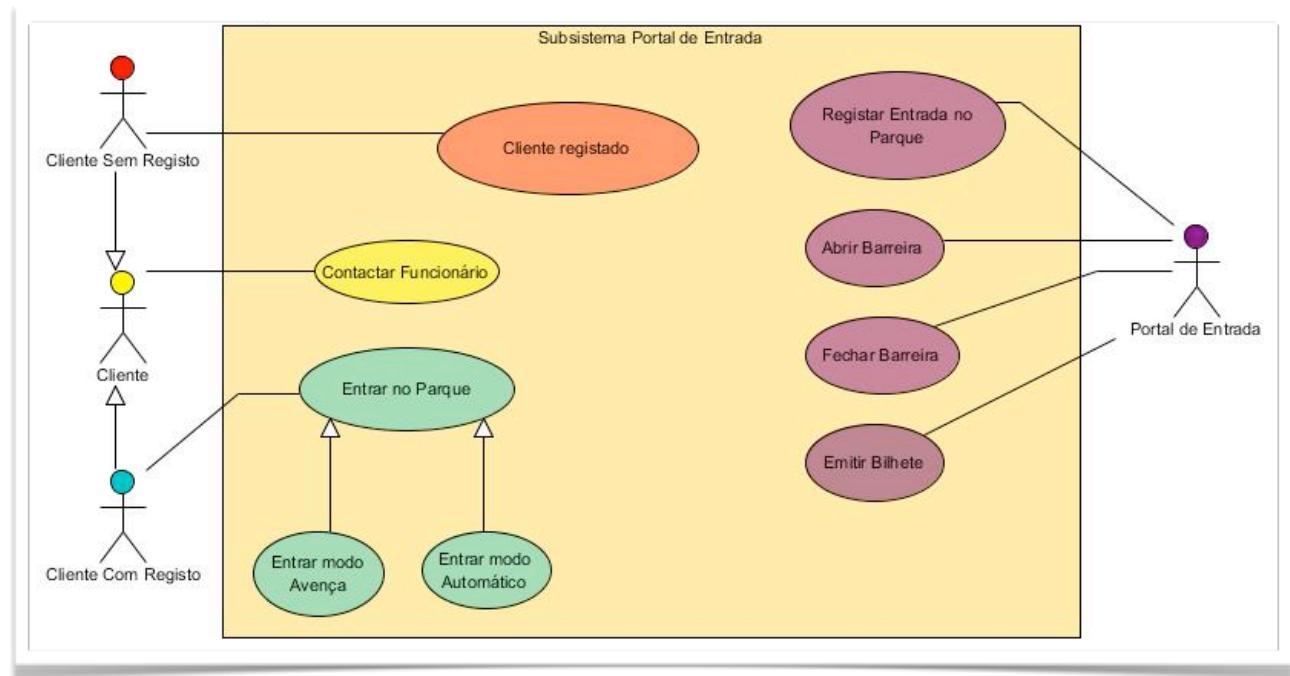
O PARKUM pode ser dividido em diversos subsistemas, que no seu conjunto garantem a funcionalidade global do sistema. Identificamos os seguintes subsistemas:

- Portal de Entrada: responsável pelo controlo de entradas.
- Portal de Saída: responsável pelo controlo de saídas.
- Máquina de Pagamento: responsável pelo processo de pagamento para clientes de bilhete.
- Gestão de Clientes: responsável por gerir todos os clientes registados o parque.
- Gestão de Relatório: responsável por gerar os relatórios sobre toda a informação relacionada com o funcionamento do parque.
- Base de Dados: suporte de toda a informação do parque.
- PARKUM-CS: sistema de contagem para o controlo de lugares livres.
- Sensores de lugar: responsável pela identificação de lugares livres.

REFINAMENTO DOS DIAGRAMAS DE USE CASES DE SISTEMA E DOS DIAGRAMAS DE SEQUÊNCIA DE SISTEMA

Após a identificação dos diversos subsistemas associados ao PARKUM, surgiu a necessidade de refinar os diagramas anteriormente feitos. Deste modo, efectuamos o refinamento dos diagramas de Use Cases de Sistema, de acordo com cada subsistema:

S U B S I S T E M A P O R T A L D E E N T R A D A

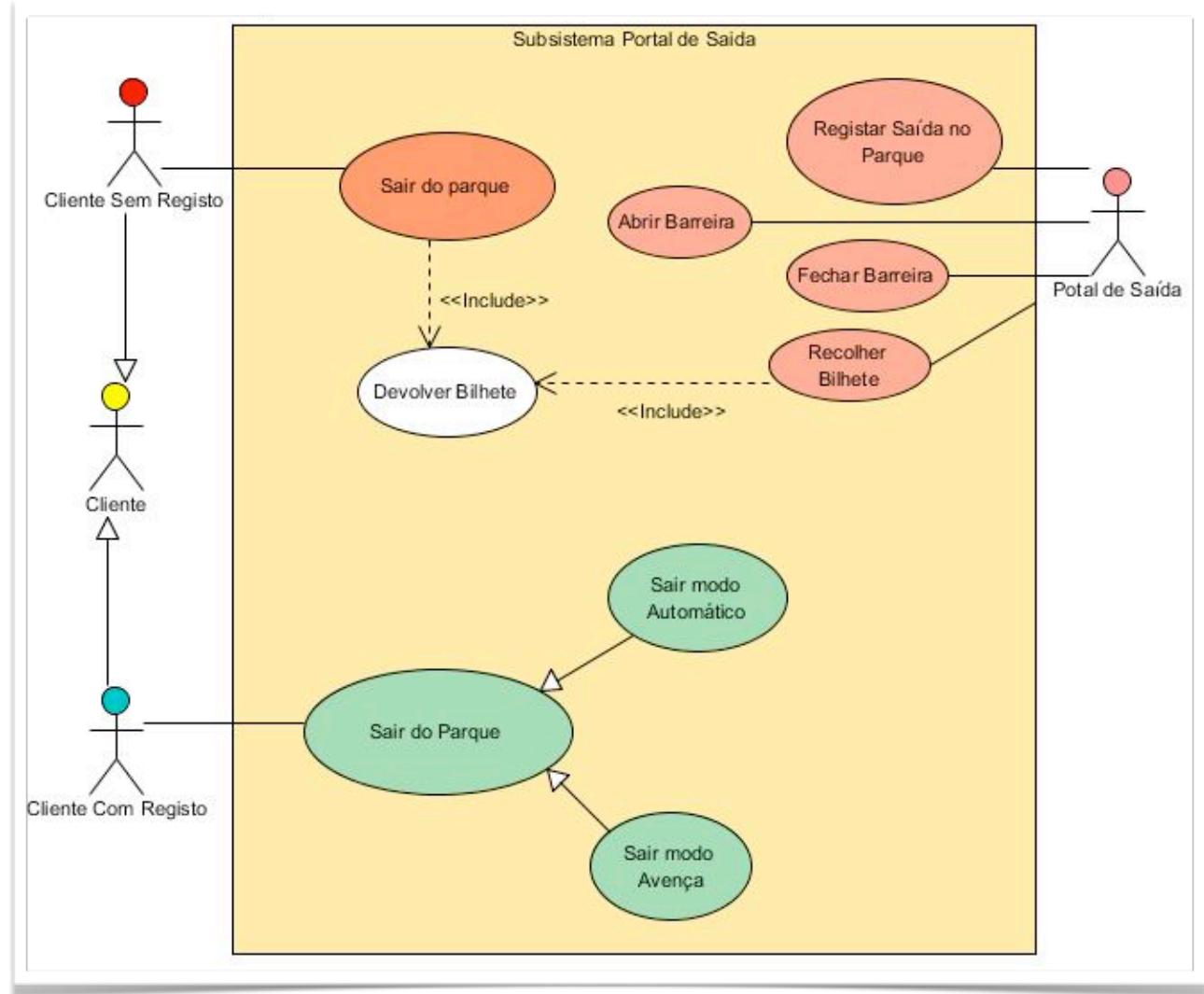


No diagrama em cima estão descritas todas as acções realizadas no subsistema portal de entrada. É neste portal, que o cliente pode contactar o funcionário caso necessite de auxilio na entrada do parque. É importante realçar o facto de existirem modos distintos de entrar no parque, dependendo do facto do cliente já ter feito, ou não registo prévio, no parque de estacionamento.

No caso do cliente sem registo apenas é executado o use case: Entrar no parque. Relativamente ao cliente com registo a sua entrada no parque poderá ser feita por modo de avenida ou por entrada automática, dependendo da forma de pagamento que o cliente escolher.

Há ainda diversas acções desempenhadas pelo portal de entrada aquando da entrada de um veículo no parque: registrar entrada no parque, abrir a barreira de entrada, fechar a barreira e emitir bilhete.

SUBSISTEMA PORTAL DE SAÍDA



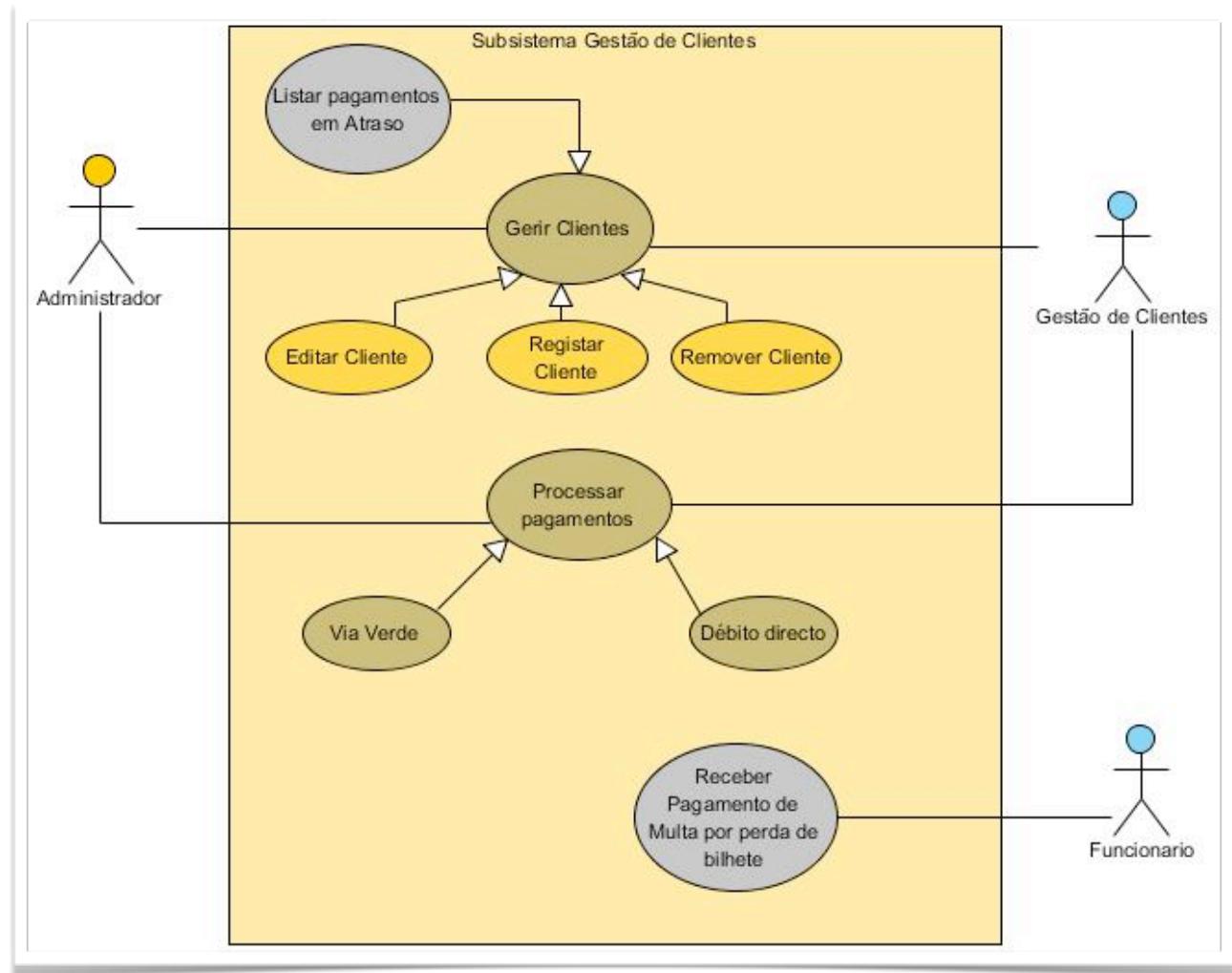
Como se pode verificar existe, no caso do cliente sem registo, uma relação entre use cases. Essa relação é representada através de um *include*. A relação *include* significa que um determinado use case utiliza ou inclui a funcionalidade disponibilizada num outro use case.

Nesta situação quando o cliente sem registo pretende sair do parque é necessário, primeiramente, devolver o bilhete, por essa razão, existe um *include* do use case devolver bilhete. Note-se ainda, que também é necessário esse *include* para que a acção remover bilhete por parte do portal de saída seja realizado.

No caso do cliente com registo os use cases sair modo automático e sair modo de avenida são generalizações do sair do parque, dependendo da forma de pagamento do cliente.

Há ainda acções realizados pelo portal de saída quando o cliente pretende sair do parque: abrir barreira, fechar barreira, registar saída no parque.

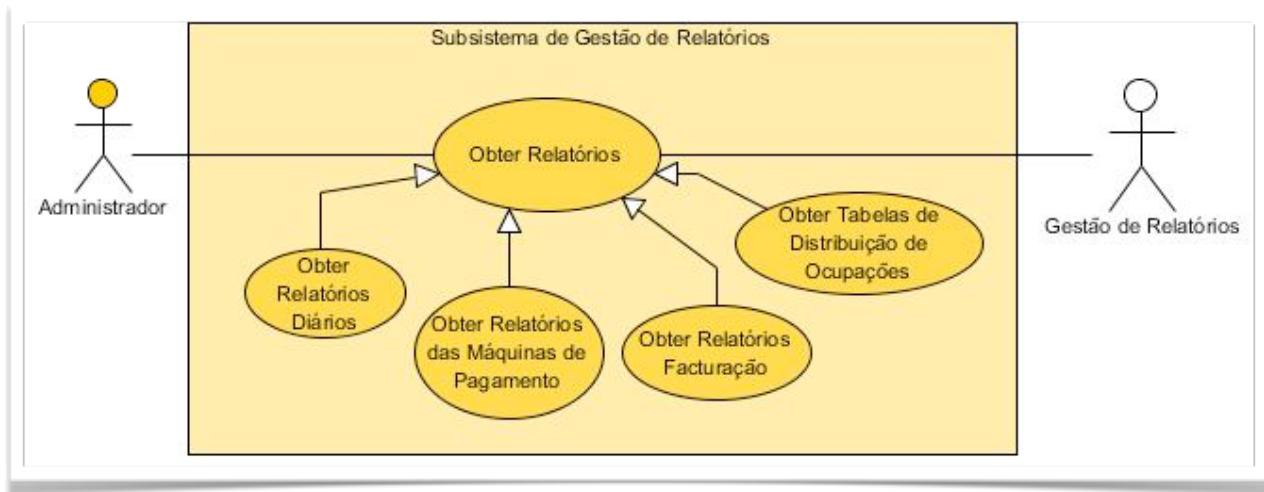
SUBSISTEMA DE GESTÃO DE CLIENTES



Neste subsistema estão representadas todas os use case realizados pela gestão de clientes. Como se pode verificar o use case ‘gerir clientes’ é uma generalização de listar pagamentos em atraso, editar cliente, registrar cliente e remover cliente. Do mesmo modo, pode-se verificar que o use case processar pagamentos apenas é feito ou relativamente aos clientes que são utilizadores de via verde, ou os de débito directo.

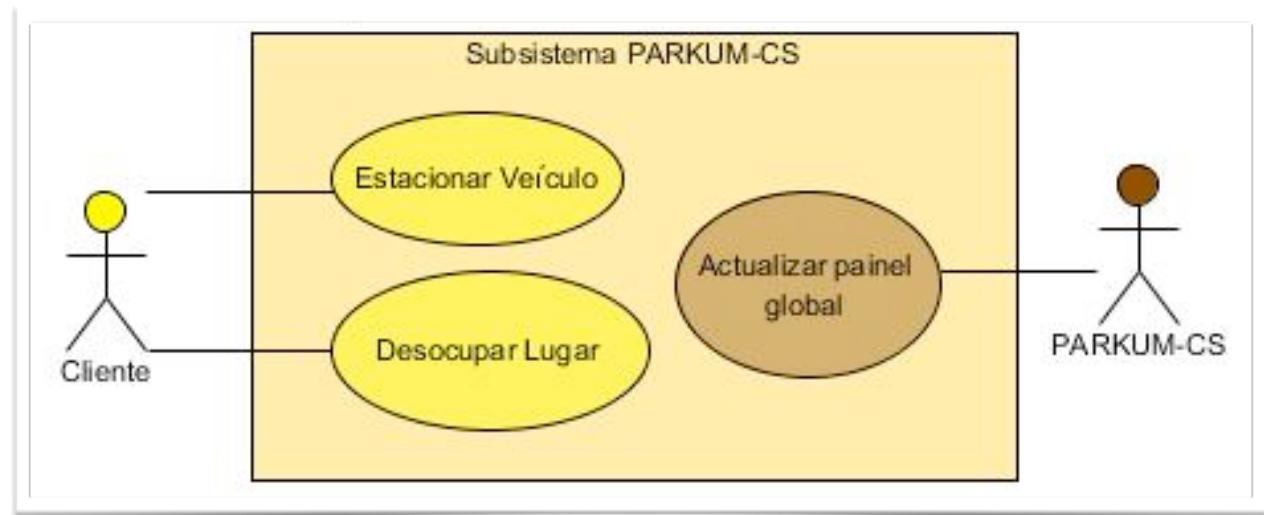
Neste subsistema o funcionário pode ainda receber o pagamento de multa por perda de bilhete.

SUBSISTEMA DE GESTÃO DE RELATÓRIOS



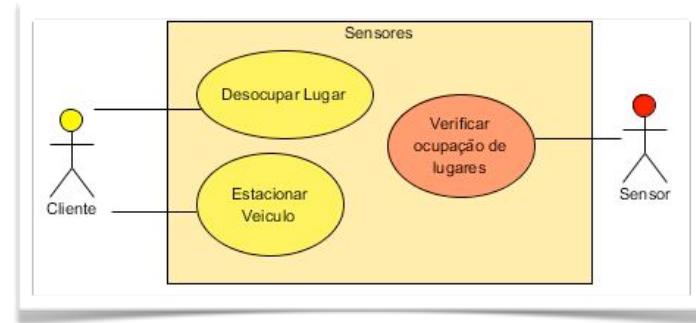
A gestão de relatórios contém toda a informação relacionada com o funcionamento do parque. Deste modo, o administrador poderá optar por gerir os seguintes tipo de relatórios: relatórios diários, relatórios das máquinas de pagamento, relatórios de facturação e tabelas de ocupação.

SUBSISTEMA PARKUM-CS



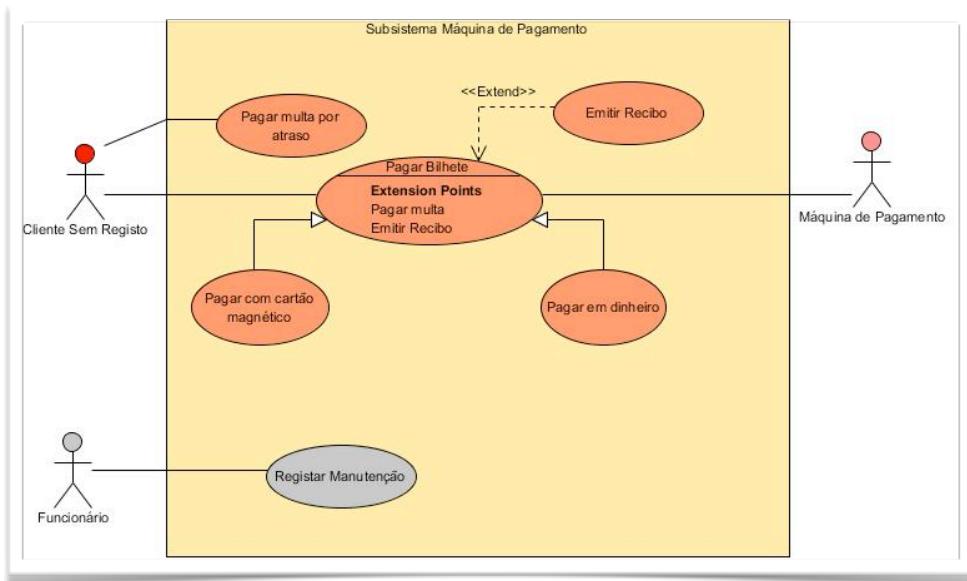
Este subsistema é o sistema responsável pela contagem das viaturas que entram e saem de cada nível. Esta informação é dada pelos sensores de estacionamento quando um cliente ocupa ou desocupa um lugar no parque. Por isso ele tem a responsabilidade de incrementar e decrementar o número de estacionamentos efectuados no PARKUM.

S U B S I S T E M A D E S E N S O R E S D E L U G A R



O subsistema de sensor de lugar é a entidade responsável por activar a cor do sensor quando o cliente ocupa ou desocupa o lugar de estacionamento. Quando o sensor detecta um lugar ocupado o sensor altera a cor para vermelho. Se, eventualmente, o cliente pretender desocupar o lugar a cor do sensor muda para vermelho.

S U B S I S T E M A M Á Q U I N A D E P A G A M E N T O



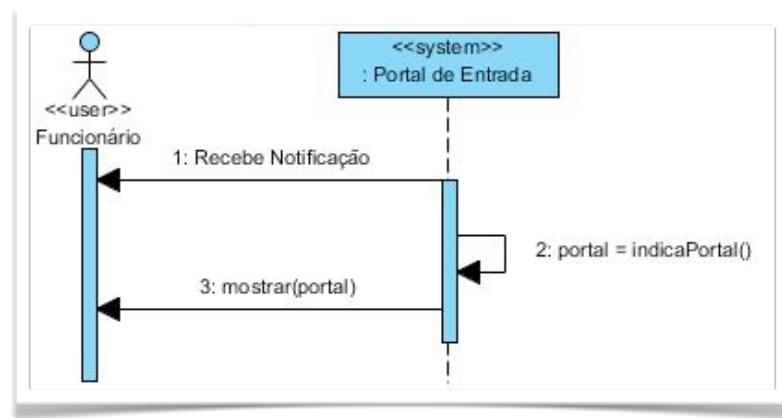
O subsistema máquina de pagamento é a entidade responsável por efectuar o pagamento, do bilhete do cliente sem registo. Neste subsistema o cliente paga o bilhete de estacionamento, onde poderá optar pelo tipo de pagamento que pretende fazer, isto é, pagar em dinheiro ou em multibanco. Caso o cliente pretenda recibo do pagamento, será invocado outro use case, que esta representado pelo *extend* designado por emitir recibo. O funcionário intervém na máquina de pagamento para efectuar a sua manutenção.

Após a refinação dos use case, dos subsistemas do PARKUM foi desenvolvido o refinamento de todos os diagramas de sequência, de modo a incluir os subsistemas identificados.

Relativamente aos diagramas de sequência refinados é importante realçar que incluem mais *LifeLines* do que os inicialmente descritos, sem a refinação. Isso deve-se ao facto de que nestes casos são especificados, concretamente, qual a parte do sistema do PARKUM que fica encarregue de determinada operação. Nesta fase, as mensagens enviadas, nas setas, estão especificados os métodos que são utilizados para descrever uma determinada acção.

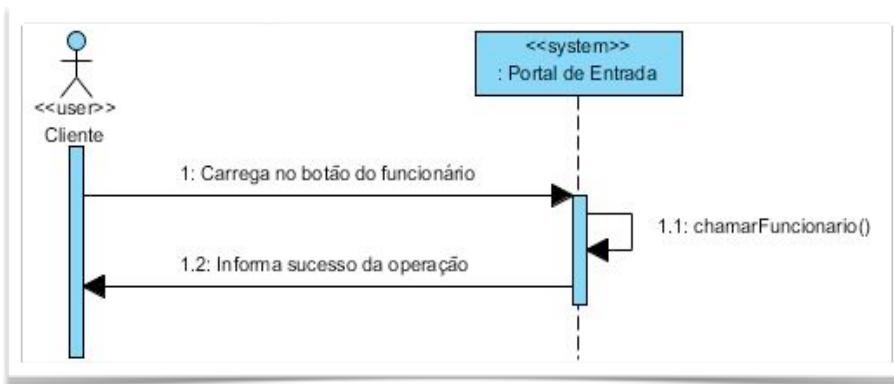
Os diagramas abaixo mostram-nos como é feita a comunicação entre os actores e os subsistemas, subdivididos nas respectivas *LifeLines* e com todos os métodos que lhes tão subjacentes de uma forma simples e detalhada.

AUXILIAR CLIENTE



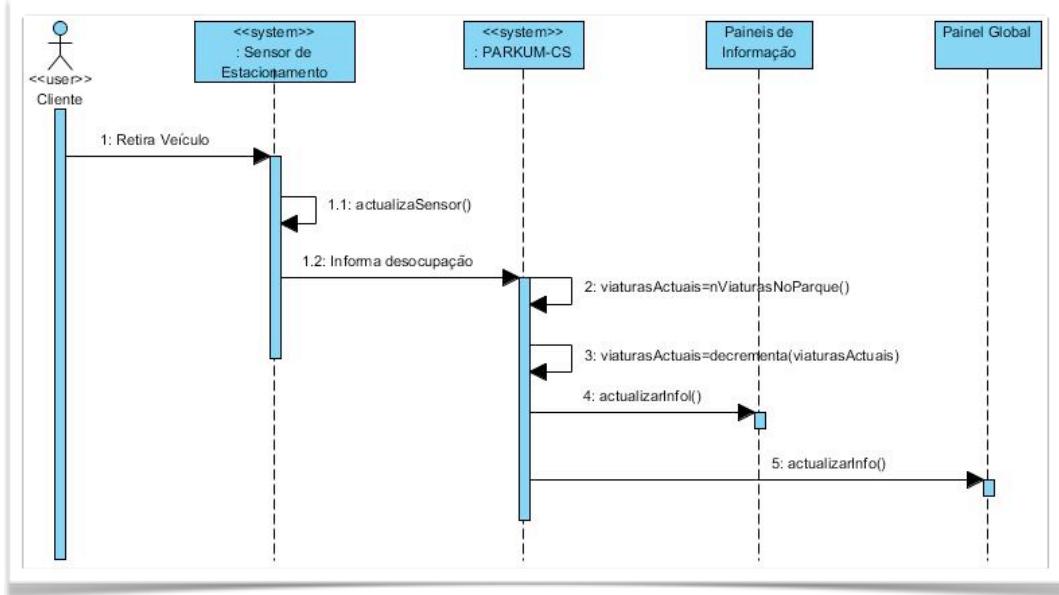
O funcionário é notificado pelo subsistema portal de entrada que o cliente pressionou o botão de auxilio. Seguidamente, o subsistema verifica qual o portal de entrada que foi notificado. De seguida, implementa um método *indicarPortal()*, de forma a informar o funcionário.

CONTACTAR FUNCIONÁRIO



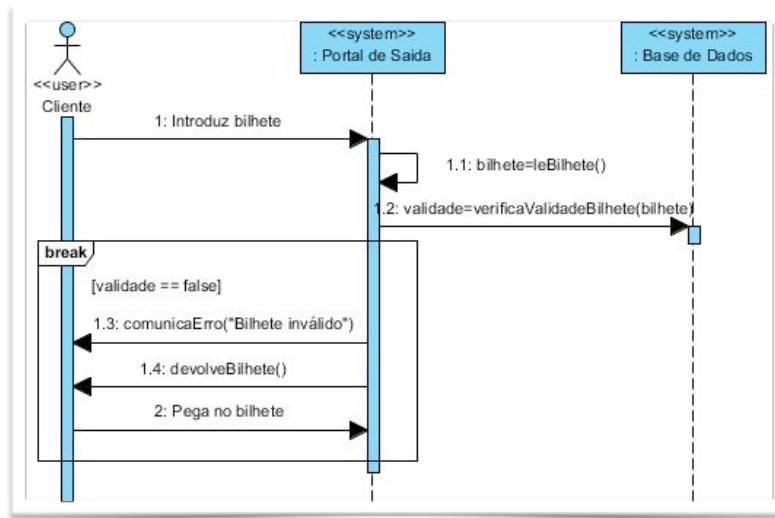
O que o diagrama acima representa a interacção entre o cliente e o subsistema de entrada. Aqui o cliente pressiona o botão para contactar funcionário, o sistema chama o funcionário e seguidamente informa o sucesso da acção ao cliente.

D E S O C U P A R L U G A R



Relativamente ao diagrama de desocupar lugar este inicia-se quando o cliente retira o veículo do estacionamento. O sensor de estacionamento detecta a desocupação do lugar e faz actualização da sua cor, com o método *actualizasensor()*. De seguida, o sensor envia esta informação o PARKUM-CS, que por sua vez verifica quantos veículos estão no parque com o método *nViaturasNoParque()* e depois decrementa esse número com o método *decrementa()*. E por fim é feita a actualização ao painel global e aos painéis de informação.

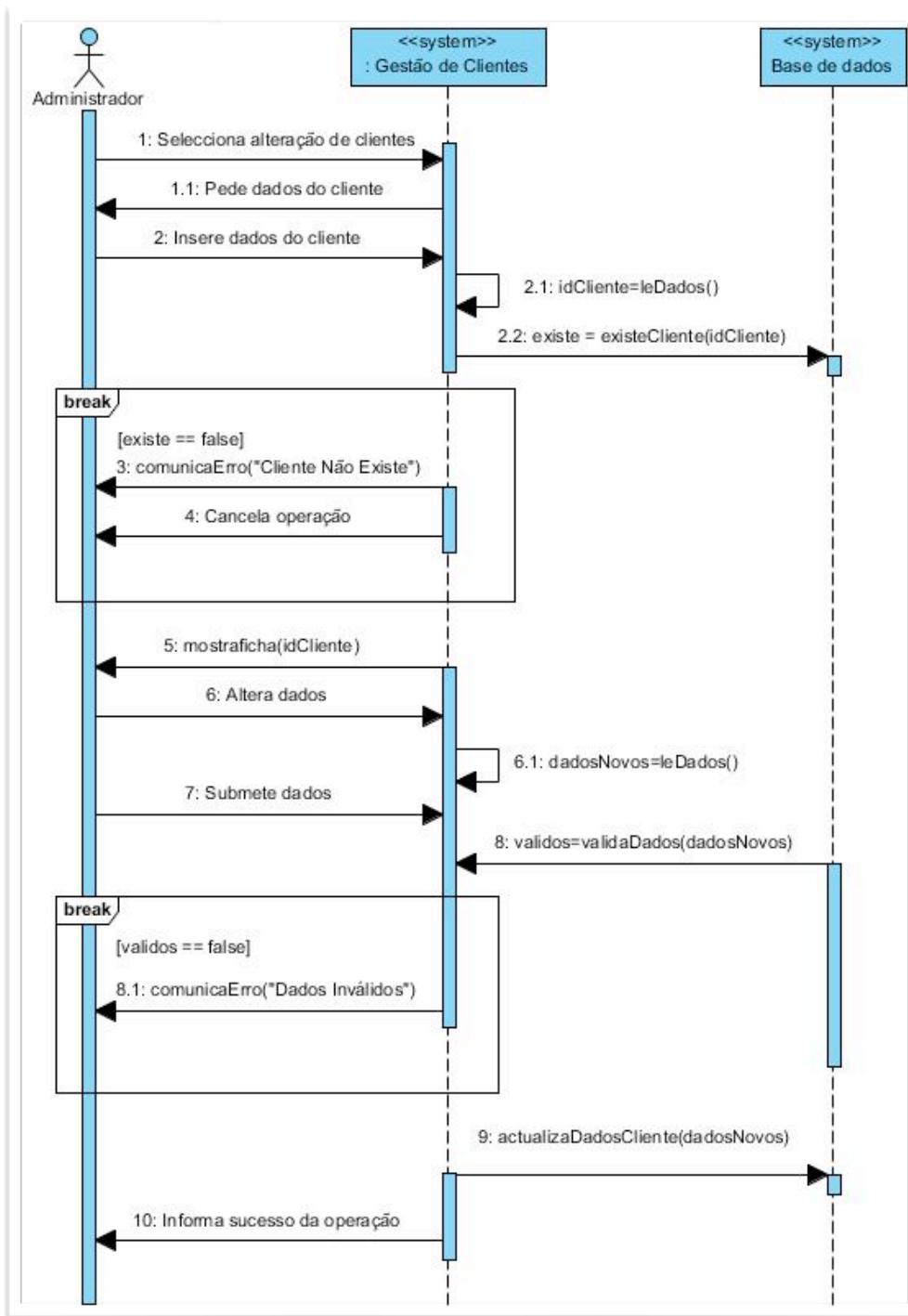
D E V O L V E R B I L H E T E



O diagrama acima descreve o devolver bilhete do cliente sem registo. Primeiramente, o cliente insere o bilhete no portal de saída. O portal de saída lê o bilhete com o método *lebilhete-*

te0. O sistema pergunta à base de dados se o bilhete é válido. Caso o bilhete seja verificado como válido, o portal de saída fica com o bilhete. Caso contrário, ocorre uma exceção, o portal de saída informa o cliente que bilhete não válido, com método *devolvebilhete()* o portal devolve o bilhete ao cliente, que de seguida retira o bilhete.

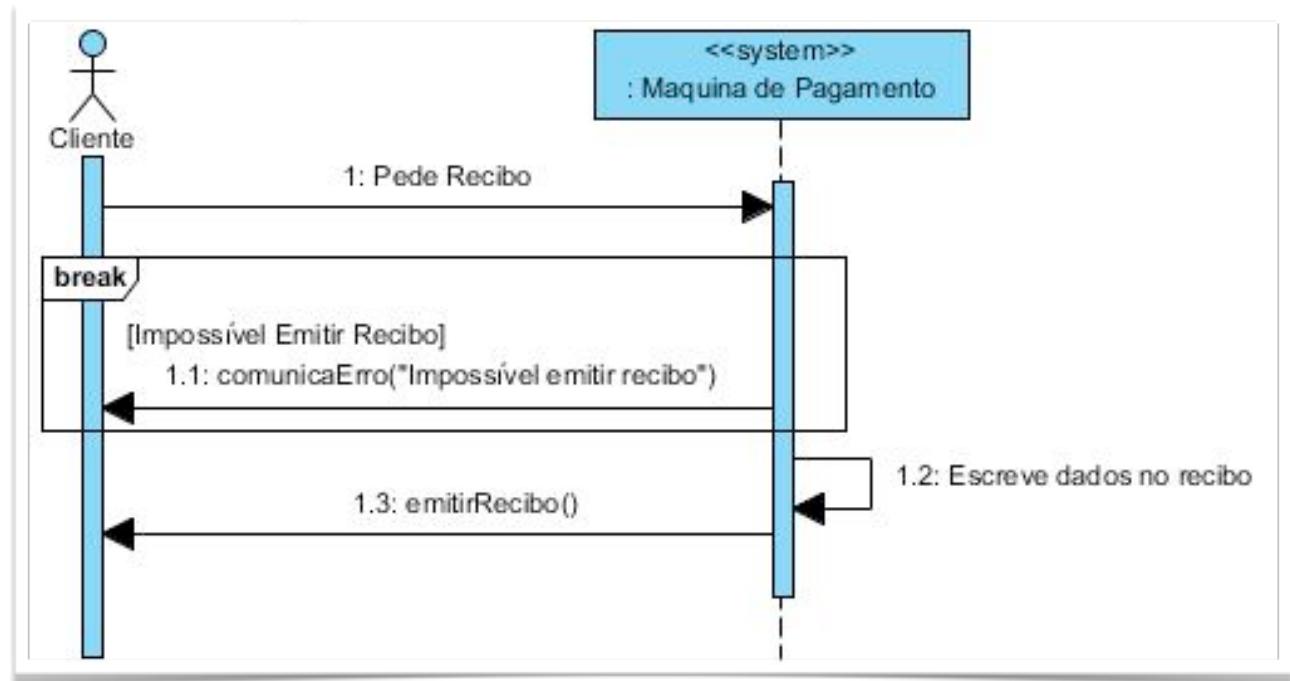
EDITAR CLIENTE



Neste diagrama de sequência é possível verificar que o administrador selecciona a opção de alteração de clientes, no subsistema de gestão de clientes. Aqui o administrador pede os dados do cliente, e de seguida o portal de entrada lê os dados através do método *leDados()*. A base de dados verifica se existe registo do cliente, caso exista a gestão de clientes mostra os dados ao administrador. O administrador altera os dados que pretende. De imediato, gestão de clientes lê esses dados com o método *leDados()*. Quando o administrador submete os dados, a gestão de clientes verifica se são válidos. De seguida são actualizados os dados do cliente com o método *actualizaDadosCliente()*. E o subsistema informa o administrador do sucesso da acção. Se os dados não forem válidos comunica o erro através do método *comunicaErro()*.

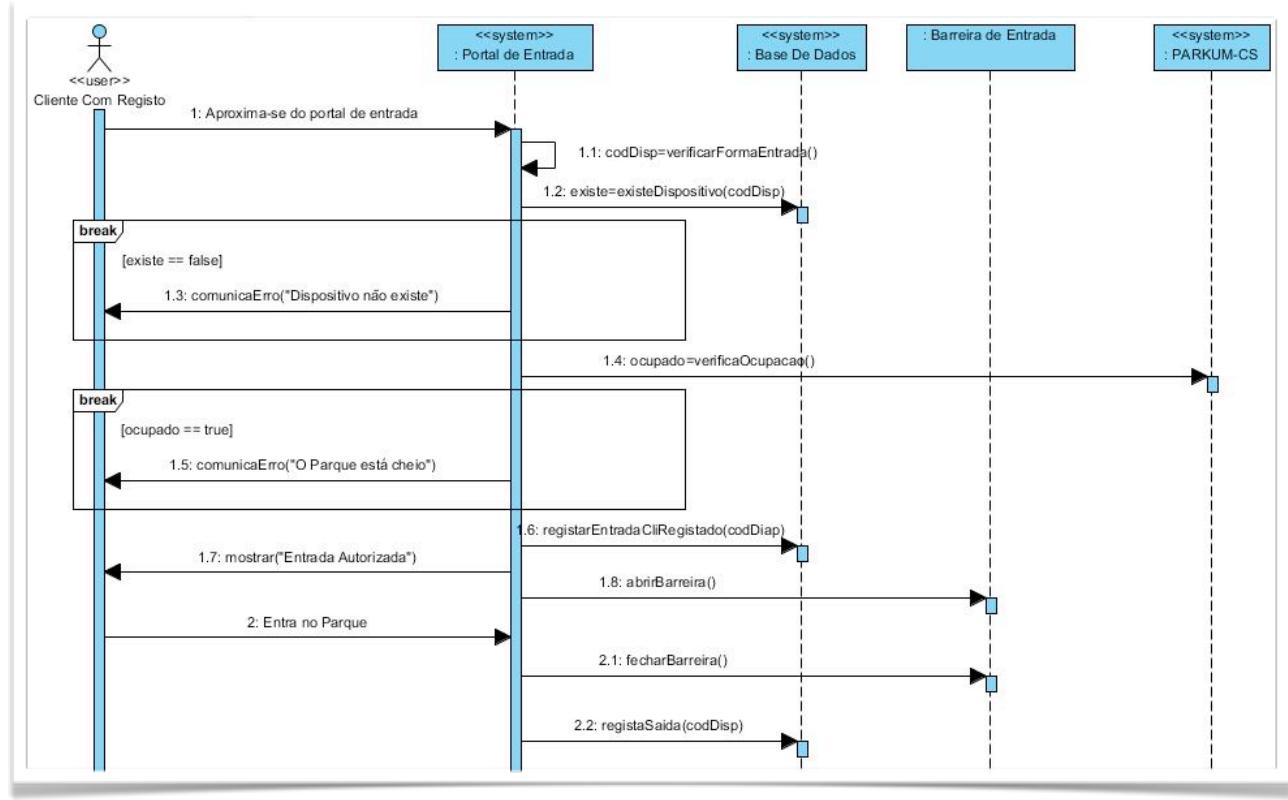
Caso não se verifique nenhum registo do cliente na base de dados, o administrador é informado do erro através do método *comunicarErro()* e a operação é cancelada.

E M I T I R R E C I B O



Esta acção começa com o cliente a pedir o recibo a máquina de pagamento. A máquina, regista dados e emite recibo através do método *emitirRecibo()*. Caso a máquina não consiga emitir o recibo informa o cliente do erro através do método *comunicaErro()*.

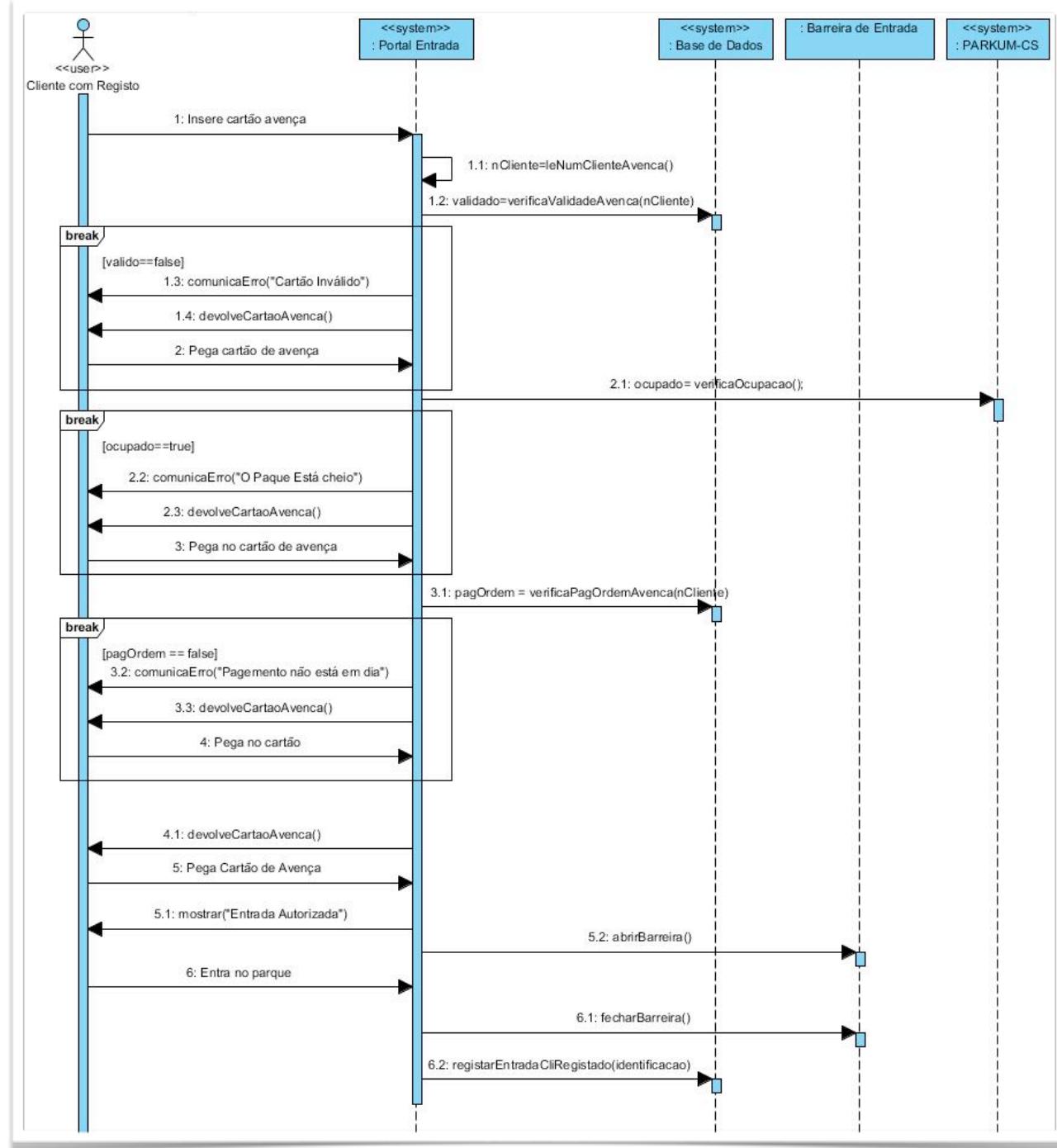
ENTRAR MODO AUTOMÁTICO



O diagrama acima representa o modo de entrada automática. O cliente aproxima-se do parque de estacionamento e o portal de entrada o verifica qual o dispositivo utilizado pelo cliente no modo de entrada, através do método `verificarFormaEntrada()`. O portal de entrada, por sua vez, verifica na base de dados se o dispositivo detectado está registado no sistema, utilizando o método `existeDispositivo()`. Seguidamente, o portal comunica com o PARKUM-CS de modo a verificar a ocupação do parque, através do método `verificaOcupacao()`. Caso o parque tenha lugar livres, o portal regista a entrada na base de dados através do método `registrarEntrada()`. Por conseguinte, o portal informa o cliente que a entrada foi autorizada utilizando o método `mostrar()`. Com o método `abrirbarreira()`, a barreira de entrada abre e o cliente entra no parque. Depois do cliente entrar no parque a barreira é fechada com método `fecharbarreira()` a regista a entrada do cliente.

Se eventualmente o parque estiver cheio, ou o dispositivo não estiver registado no sistema esta informação é dada ao cliente através do método `comunicaErro()`.

ENTRAR MODO AVENÇA

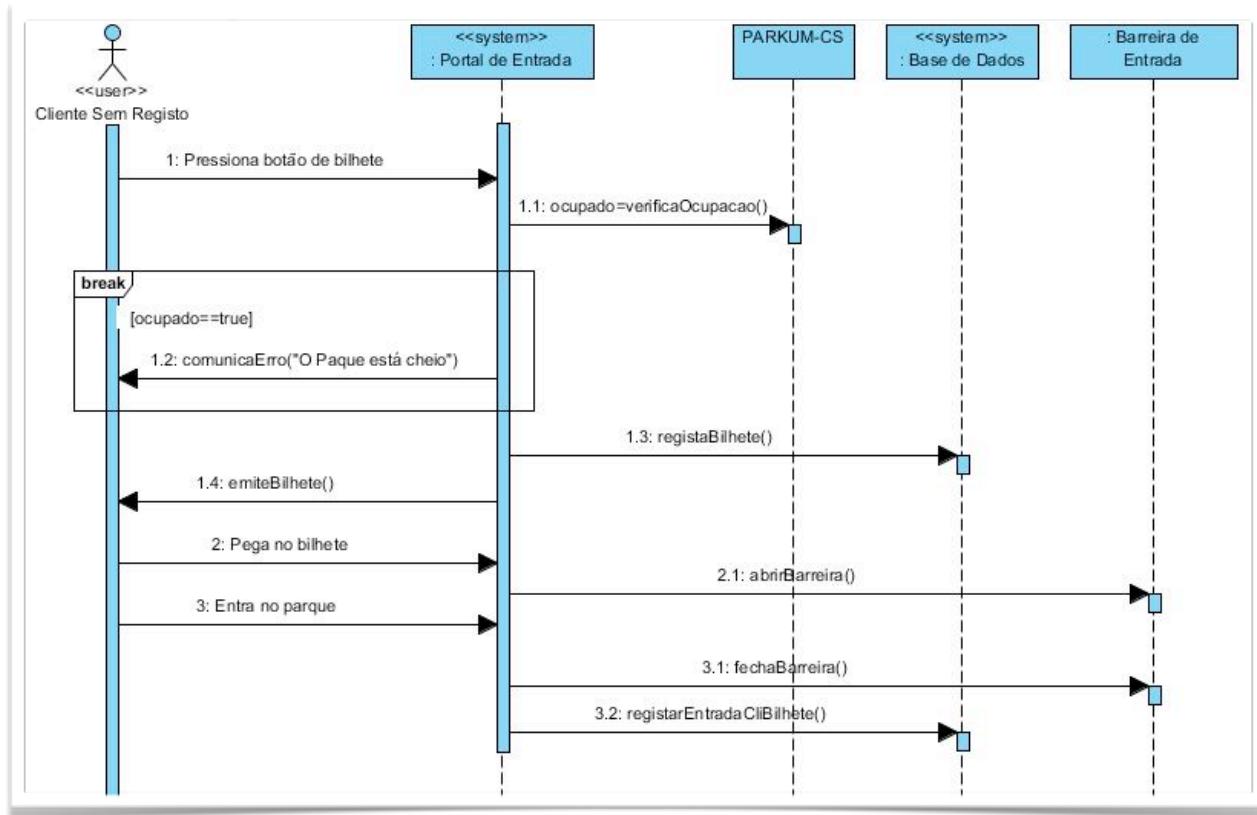


Quando um cliente entra no parque de estacionamento por modo de avença, introduz o cartão de avença no portal de entrada. Nesta acção, o portal lê o número do cliente do cartão através do método `leNumClienteAvenca()`. De imediato é verificado na base de dados se o número de cliente está válido, através do método `verificaValidadeAvenca()`. Sendo o cartão válido, é verificado, de seguida, a ocupação do parque no subsistema PARKUM-CS, através do método `verificaOcupacao()`. Para verificar se o pagamento do cartão de avença está em dia é utilizado o

método `verificaPagOrdemAvenca()`. Depois disto é devolvido o cartão, `devolveCartaoAvenca()`. O portal de entrada informa que é possível a entrada através do método `mostrar()`, a barreira abre com o método `abrirbarreira()`. O cliente entra no parque o portal de entrada informa a barreira para fechar com o método `fecharbarreira()`. Por último e, o portal de entrada regista a entrada na base de dados, com o método `registarEntradaCliRegistado(identificacao)`.

Na situação em que o cartão de avença está inválido, o parque está cheio, ou o pagamento está em atraso o portal de entrada comunica o respectivo erro, devolve o cartão através do método `devolvecartaoAvenca()` e o cliente pega no cartão.

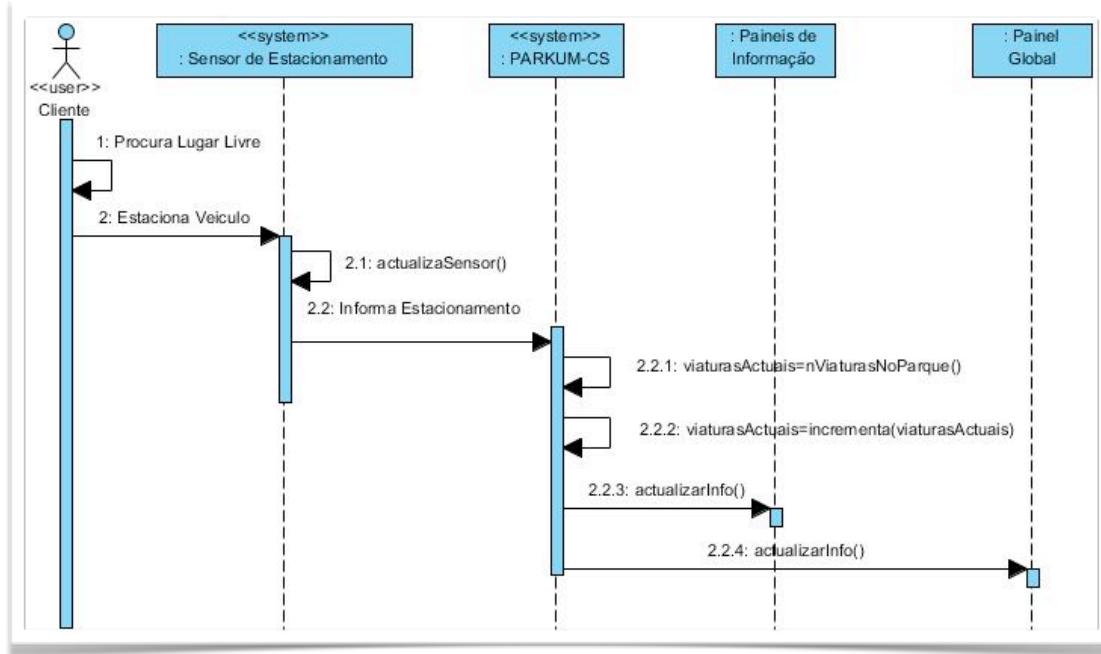
ENTRAR NO PARQUE



No diagrama acima o cliente começa por pressionar o botão do bilhete no portal de entrada. O portal verifica se o parque está cheio, ao comunicar com o PARKUM-CS através do método `verificaOcupacao()`. Se o parque não estiver cheio, o portal emite o bilhete através do método `emitirbilhete()`. Quando o cliente retira o bilhete, o portal informa a barreira para abrir através do método `abrirbarreira()`. O cliente entra no parque. O portal de entrada fecha a barreira através do método `fecharbarreira()` e o portal de entrada regista dados do bilhete na base de dados através do método `registaEntradaCliBilhete()`.

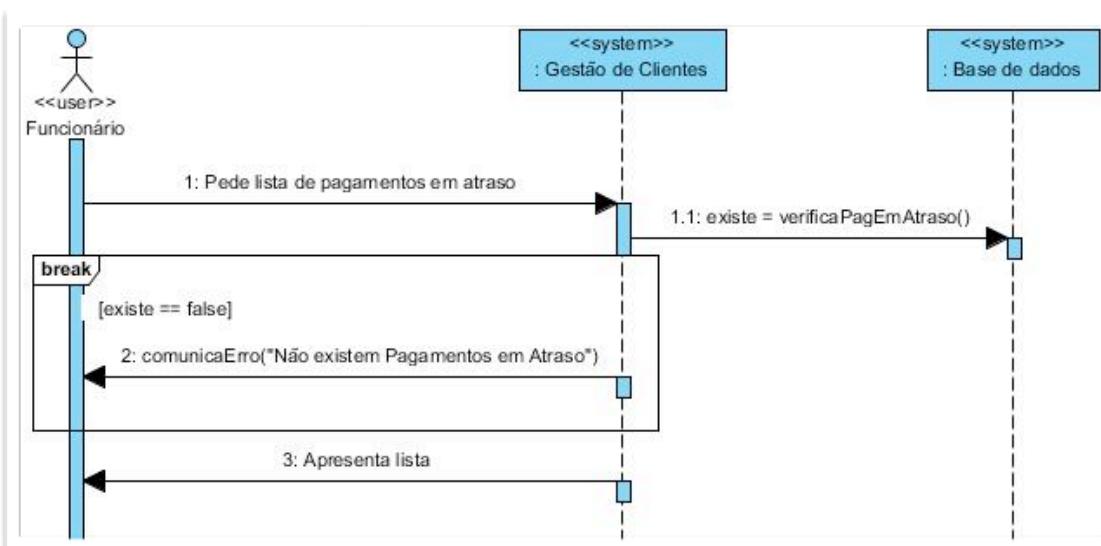
Caso o parque esteja cheio o portal de entrada avisa o cliente através do método `comunicaErro()`.

ESTACIONAR VEICULO



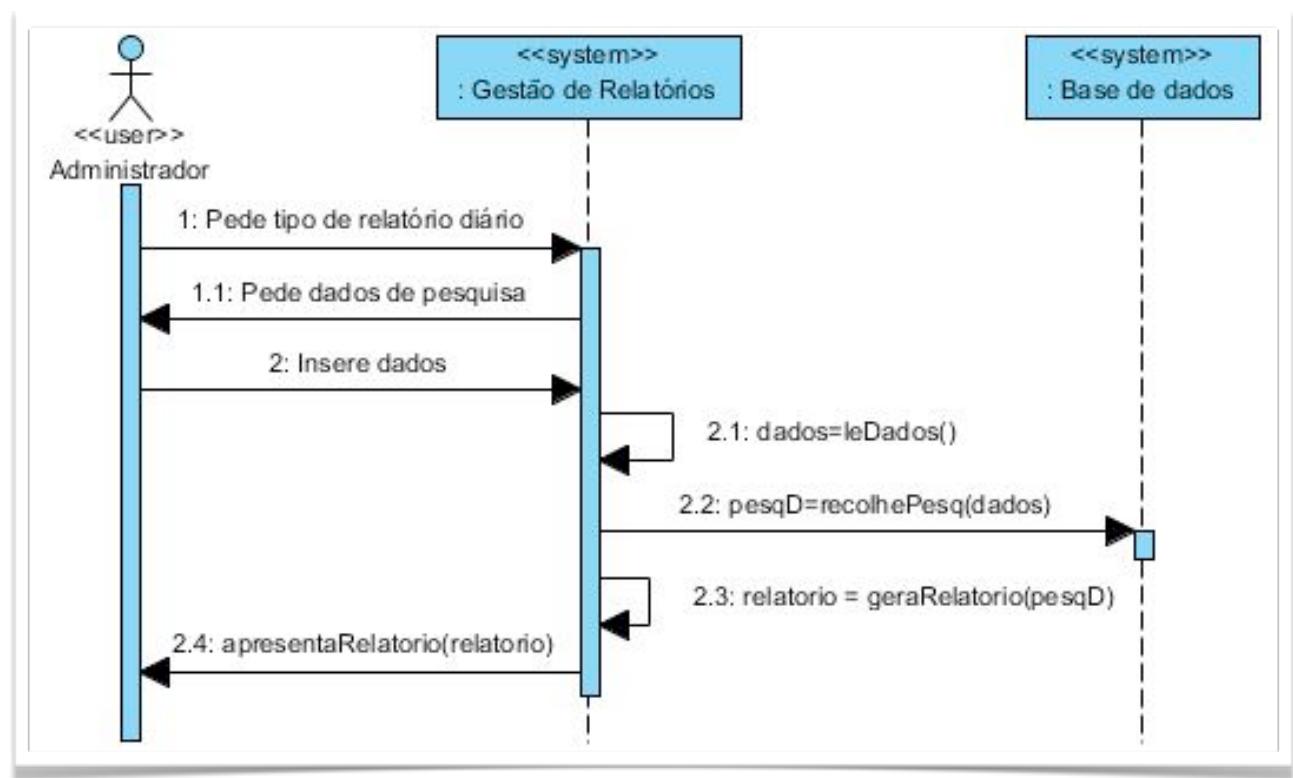
Quando o cliente estaciona o veículo no parque, o sensor de estacionamento detecta o estacionamento do veículo e altera a sua cor com o método *actualizaSensor()*. Seguidamente o sensor informa o registo de um estacionamento ao PARKUM-CS e este mesmo calcula o número de veículos no parque através do método *nViaturasNoParque()*. De seguida, incrementa o número de veículos de parque e actualiza os painéis de informação e o painel global com o método *actualizarInfo()*.

LISTAR PAGAMENTOS EM ATRASO



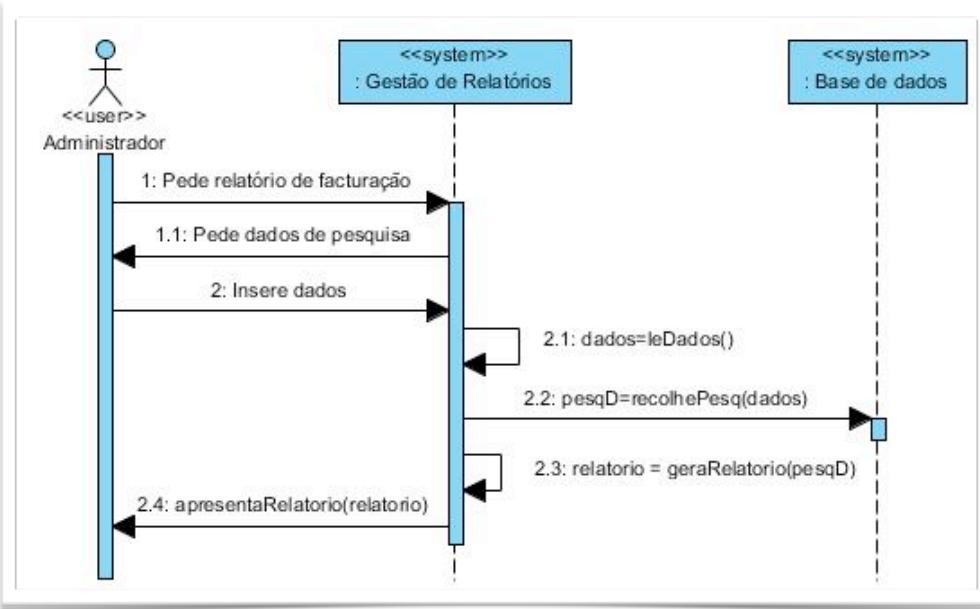
Neste diagrama de sequência é possível verificar que o funcionário pede ao subsistema de gestão de pagamentos, a lista de pagamentos em atraso. Por conseguinte, é verificado na base de dados os pagamentos em atraso e através do método *verificaPagEmAtraso()*. Caso se verifique clientes com pagamentos em atraso, o subsistema de gestão de pagamentos devolve esse informação ao funcionário. Se, eventualmente, não existirem pagamentos em atraso a gestão de pagamentos informa o erro pelo método *comunicaErro()*.

O B T E R R E L A T Ó R I O S D I Á R I O S



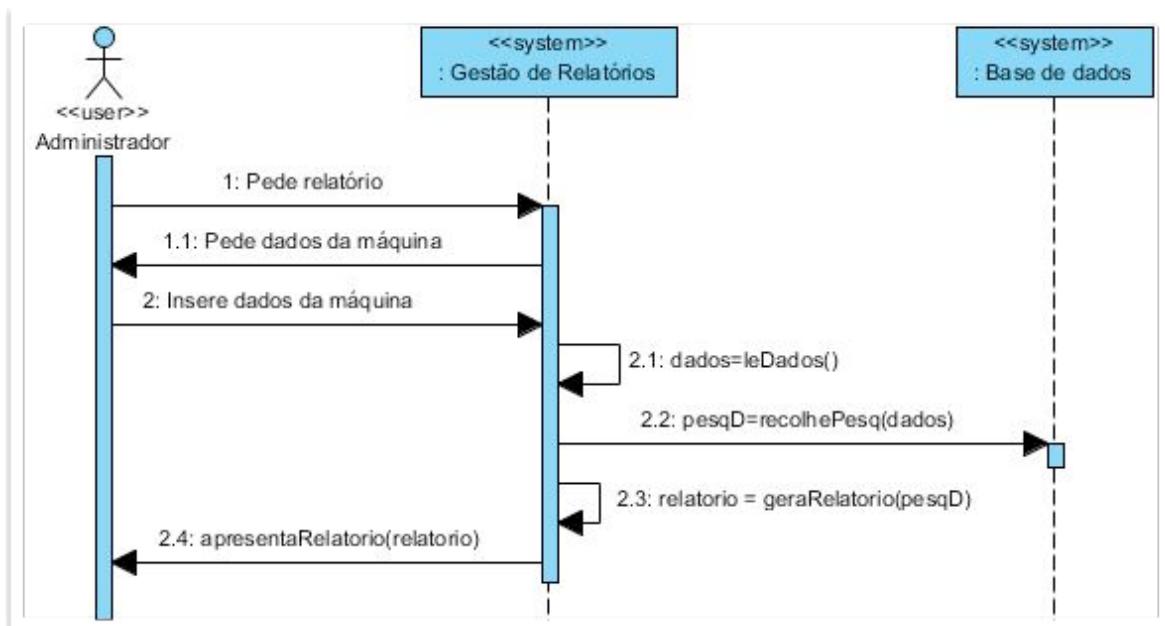
O administrador informa o subsistema de gestão de relatórios que pretende um relatório diário. O sistema pede os dados de pesquisa, o administrador insere dados do cliente e o sistema de gestão lê os dados através do método *leDados()*. De seguida, o sistema recolhe os dados na base de dados, e gera o relatório com o método *geraRelatorio()*. Por fim, apresenta-o ao administrador através do método *apresentaRelatorio()*.

OBTER RELATÓRIOS DE FACTURAÇÃO



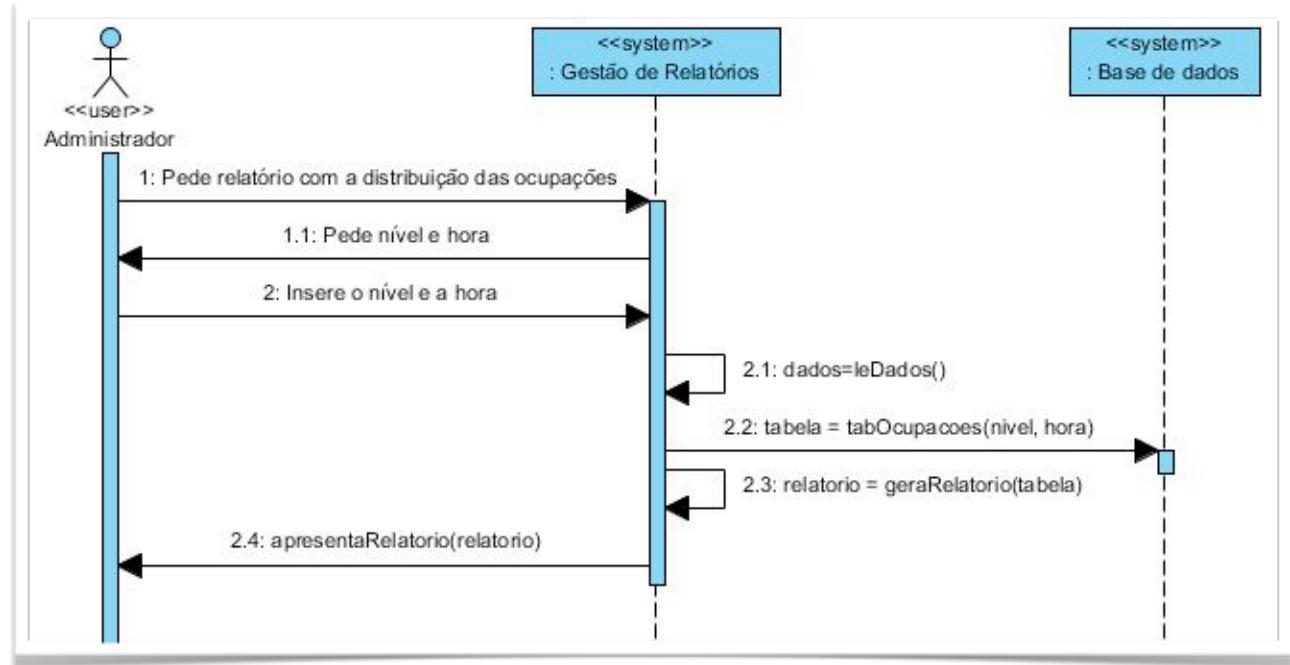
O administrador informa o subsistema de gestão de relatórios que pretende um relatório de facturação. O sistema pede os dados de pesquisa, o administrador insere dados do cliente e o sistema de gestão lê os dados através do método *leDados()*. De seguida, o sistema recolhe os dados na base de dados, e gera o relatório com o método *geraRelatorio()*. Por fim, apresenta-o ao administrador através do método *apresentaRelatorio()*.

OBTER RELATÓRIOS MÁQUINAS DE PAGAMENTO



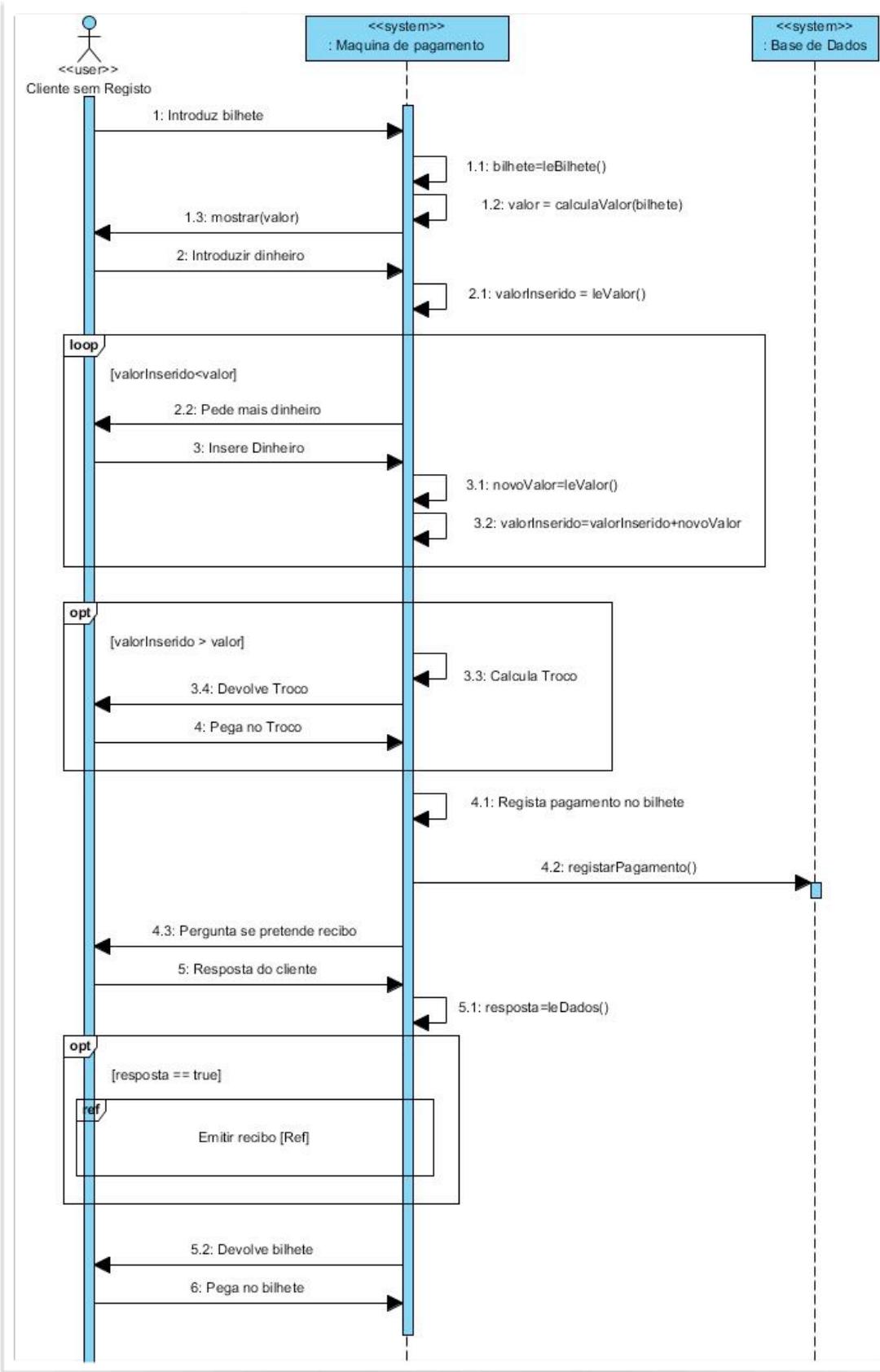
O administrador informa o subsistema de gestão de relatórios que pretende um relatório da máquina de pagamento. O sistema pede os dados de pesquisa, o administrador insere dados do cliente e o sistema de gestão lê os dados através do método *leDados()*. De seguida, o sistema recolhe os dados na base de dados, e gera o relatório com o método *geraRelatorio()*. Por fim, apresenta-o ao administrador através do método *apresentaRelatorio()*.

O B T E R T A B E L A S D E D I S T R I B U I Ç Ã O



O administrador informa o subsistema de gestão de relatórios que pretende um relatório de tabelas de distribuição de ocupações. O sistema de relatórios pede nível e hora de pesquisa, o administrador insere o nível e a hora e o sistema de gestão lê os dados através do método *leDados()*. De seguida, o sistema recolhe os dados na base de dados, e gera o relatório com o método *geraRelatorio()*. Por fim, apresenta-o ao administrador através do método *apresentaRelatorio()*.

PAGAR BILHETE COM DINHEIRO

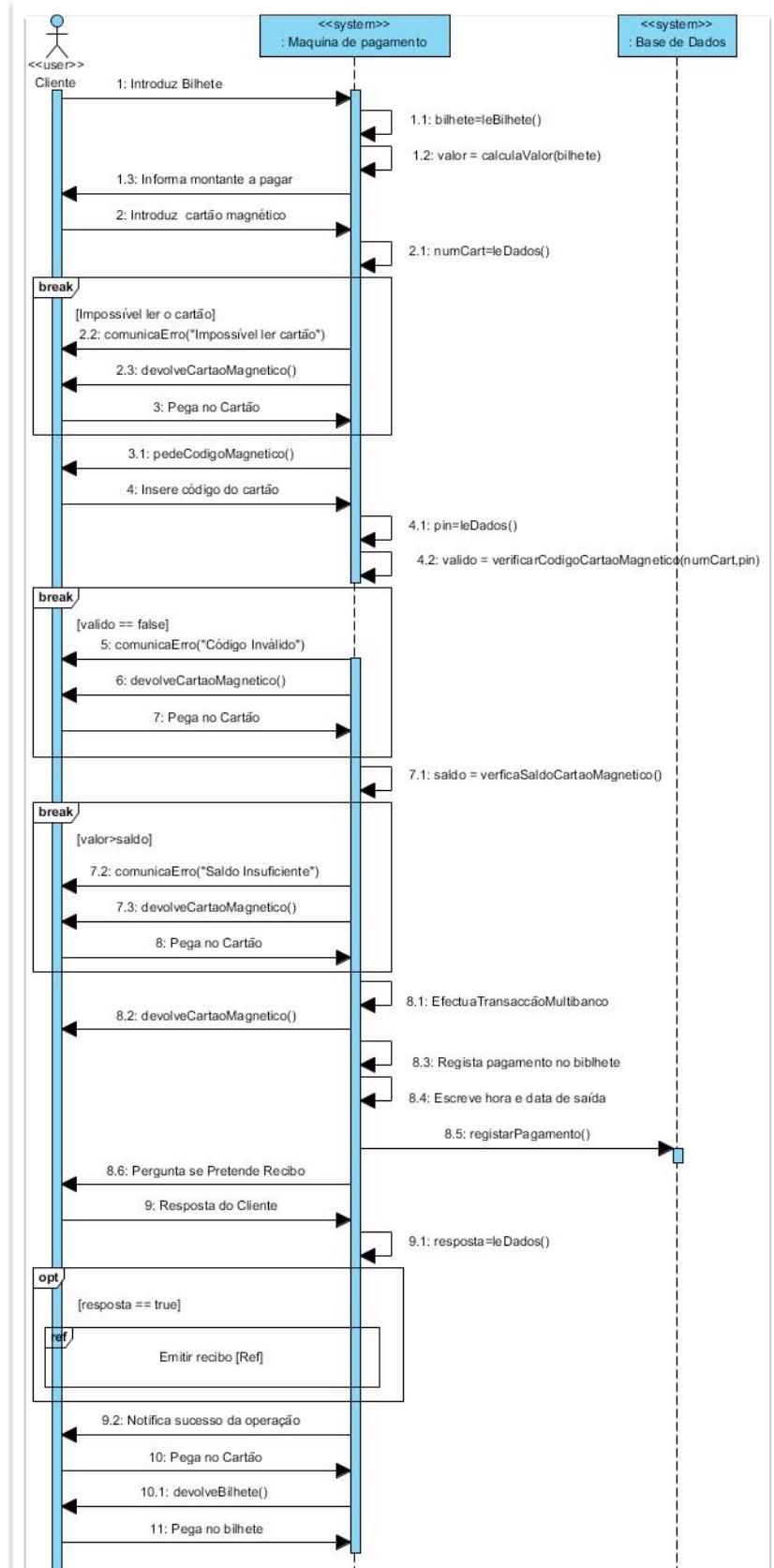


Nesta acção o cliente começa por inserir o bilhete na máquina de pagamento. A máquina recorre ao método *leBilhete()* para ler o bilhete. De seguida, calcula o valor a pagar com o método *calculaValor()* e mostra o valor ao cliente. O cliente insere o dinheiro, enquanto o valor inserido for menor que o pedido a máquina informa o cliente que é necessário mais dinheiro. Se o valor inserido for superior ao requerido a máquina de pagamento calcula o troco através do método *calculaTroco()*, e devolve o respectivo troco. A máquina guarda os dados do bilhete e envia os dados do bilhete para a base de dados através do método *registarPagamento()*. Seguidamente, questiona o cliente se quer recibo. Se, eventualmente, o cliente não quiser recibo a máquina de pagamento devolve o bilhete. Caso contrário, se o cliente quiser recibo é executada a acção emitir recibo e devolvido o bilhete.

PAGAR BILHETE COM CARTÃO MAGNÉTICO

O cliente começa por inserir o bilhete na máquina de pagamento. A máquina de pagamento lê o bilhete e calcula o valor a pagar com os métodos *leBilhete()* e *calculaValor()*, respectivamente. Depois, informa o cliente do montante a pagar. O cliente insere o cartão magnético na máquina de pagamento, a máquina lê os dados do cartão utilizando o método *leDados()*. Se a máquina conseguir ler o cartão magnético é pedido o código do cartão ao cliente que o insere. A máquina de pagamento lê o pin através do método *leDados()* e verifica se é válido o código, pelo método *verificarCodigoCartaoMagnetico()*. Seguidamente é verificado se existe saldo suficiente para a efectuar transacção através do método *verificaSaldoCartaoMagnetico()*. Caso exista saldo suficiente é efectuada a transacção, e também é registado o pagamento no bilhete. No bilhete é ainda escrita a hora de entrada e de saída e estes dados são registados na base de dados.

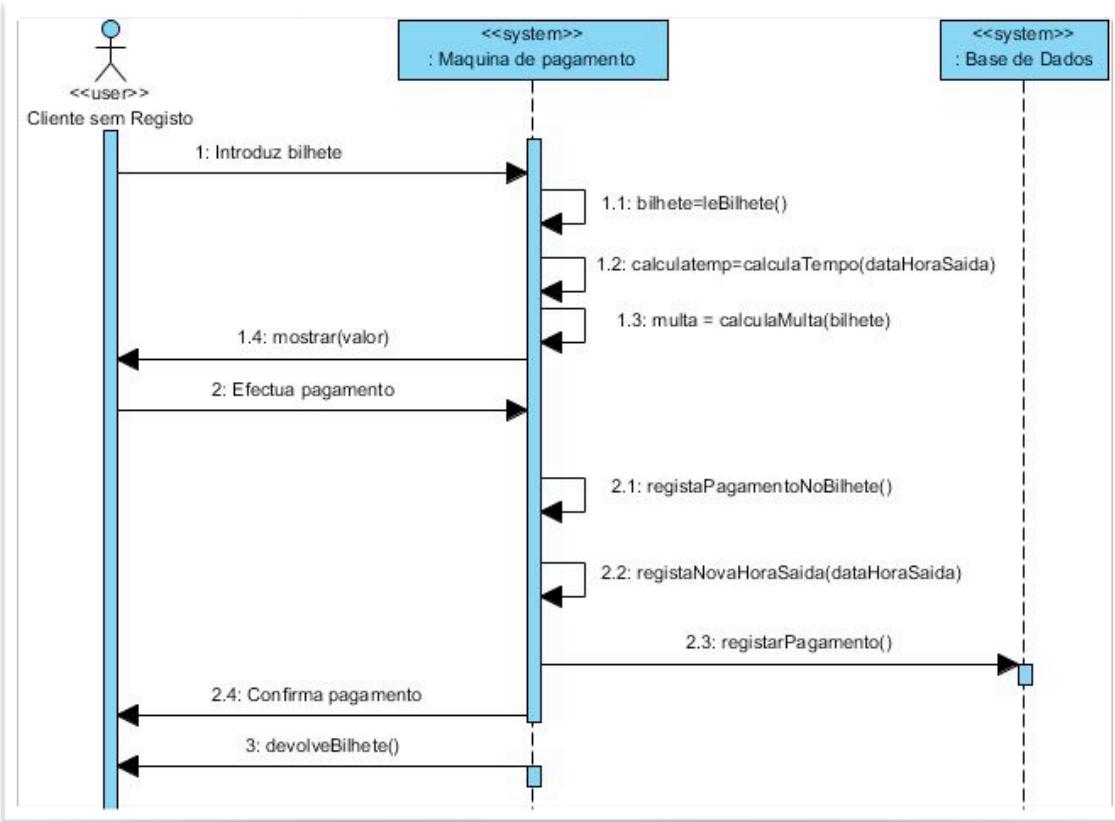
Após o pagamento o cliente pode optar por recibo. A máquina informa o cliente



do sucesso do pagamento, devolvendo cartão magnético, pelo método *devolveCartaoMagnetico()*. O cliente retira o cartão e depois a máquina devolve o bilhete, com o método *devolveBilhete()*.

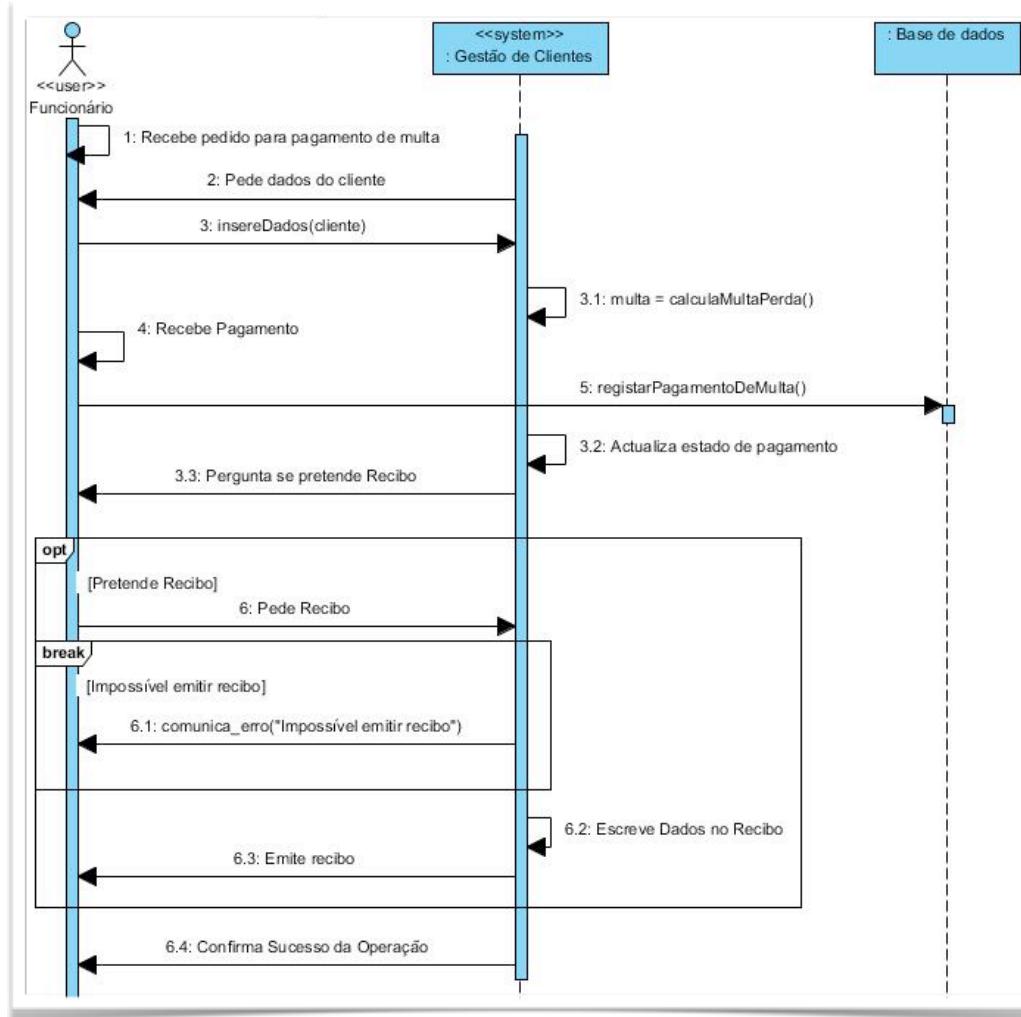
Se não for possível ler o cartão, o pin for inválido ou o saldo for insuficiente o cliente é informado do erro através do método *comunicaErro()* é devolvido o cartão magnético através do método *devolveCartaoMagnetico()*.

P A G A R M U L T A P O R A T R A S O



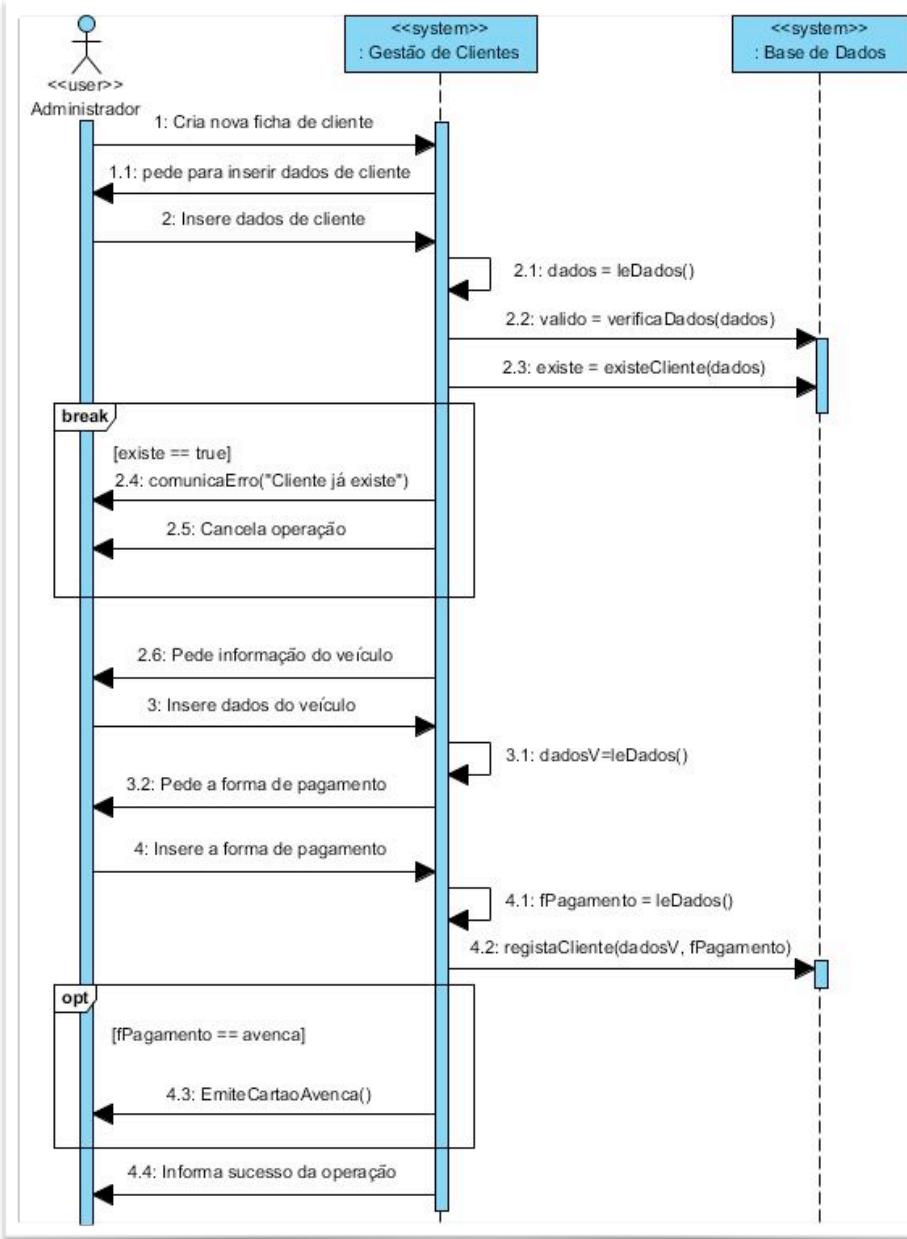
Neste diagrama o cliente insere o bilhete na máquina de pagamento. A máquina lê o bilhete, calcula o tempo ultrapassado após a último pagamento do bilhete, pelo método *calculaTempo()*. Pelo método *calculaMulta()* é calculado o valor da multa e informado o cliente desse mesmo valor. O cliente paga o valor da multa e é registado no bilhete o pagamento, pelo método *registaPagamentoNoBilhete()*. No bilhete é também registada a nova hora de saída no bilhete pelo método *registaNovaHoraSaida()*. A máquina de pagamento envia os dados à base de dados que regista os dados do bilhete através do método *registarPagamento()*. Por fim é confirmado o pagamento, e devolvido o bilhete com o método *devolveBilhete()*.

RECEBER PAGAMENTO DE MULTA



Nesta acção o funcionário recebe o pedido para pagamento de multa. O subsistema de gestão de clientes pede ao funcionário os dados do cliente. Insere os dados do cliente pelo método `insereDados()`, é calculado o valor da multa através do método `calculaMultapPerda()`. O funcionário recebe o pagamento da multa e regista o pagamento na base de dados através do método `registarPagamentoMultta()`. A gestão de clientes actualiza o estado de pagamento e questiona o funcionário se é necessário recibo, se for necessário é executada a acção emitir recibo. De seguida, o subsistema de pagamentos informa o sucesso da operação.

REGISTAR CLIENTE

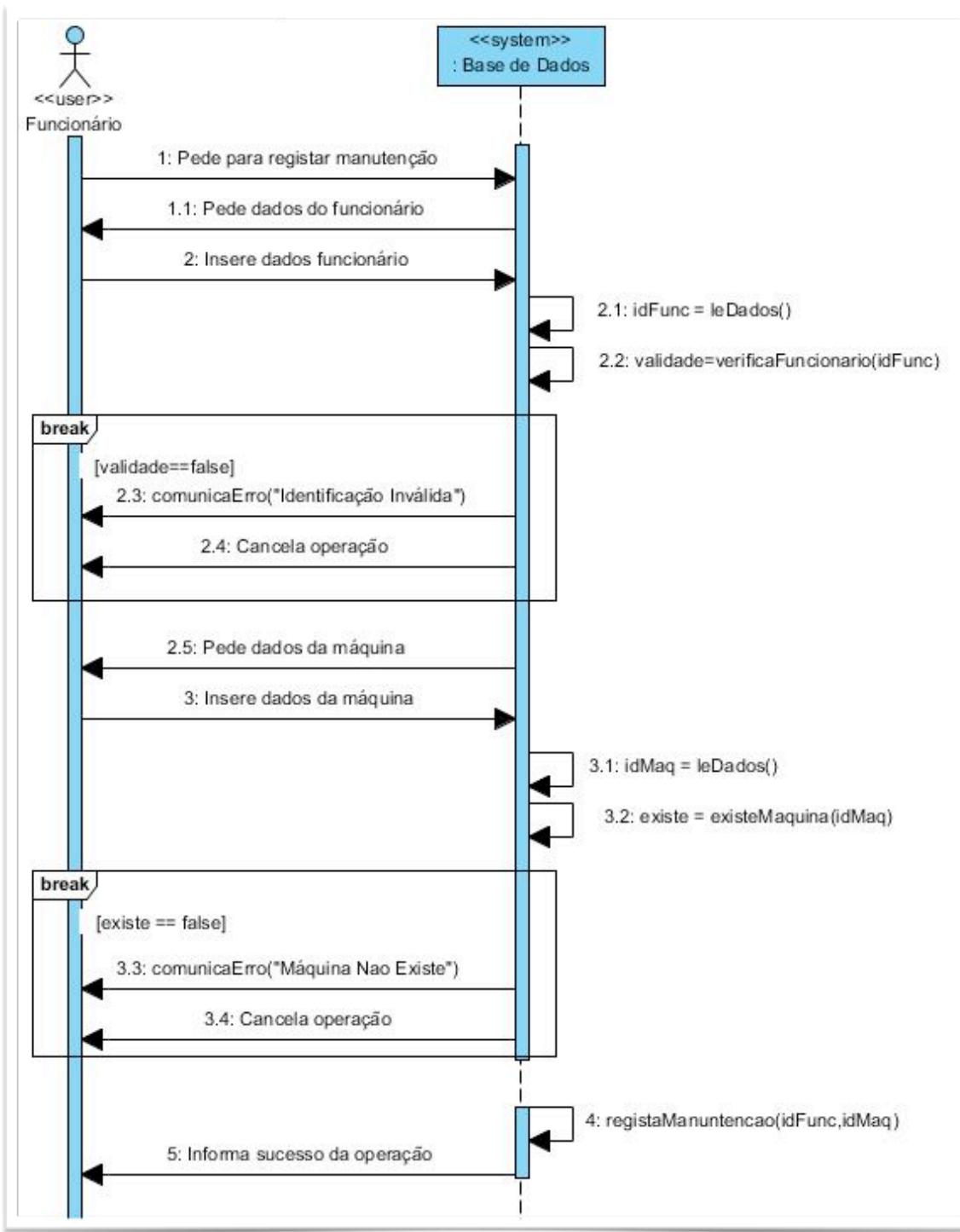


culo, o funcionário insere dados do veículo, a gestão de pagamentos lê os dados e pede a forma do pagamento. Estes dados são guardados na base de dados os dados através do método *registaCliente(dadosV,fPagamento)*.

Note-se ainda que quando o cliente escolhe o modo de avença como forma de pagamento, a gestão de clientes emite, de imediato o cartão de avença .

A acção registar cliente inicia-se quando o administrador pede ao subsistema de gestão de clientes, que pretende a criar a ficha de um novo cliente. Como tal, o administrador insere os dados do cliente. A gestão de clientes lê os dados do cliente através do método *leDados()*. De seguida é verificado se os dados são válidos e já não existem os dados na base de dados através dos métodos *verificaDados(dados)* e *existeCliente(dados)*, respectivamente. Caso já exista registo do cliente na base de dados, o administrador é avisado do erro pelo método *comunicaErro()*, e a operação é cancelada. Caso não exista o cliente é pedido os dados do veí-

REGISTAR MANUTENÇÃO

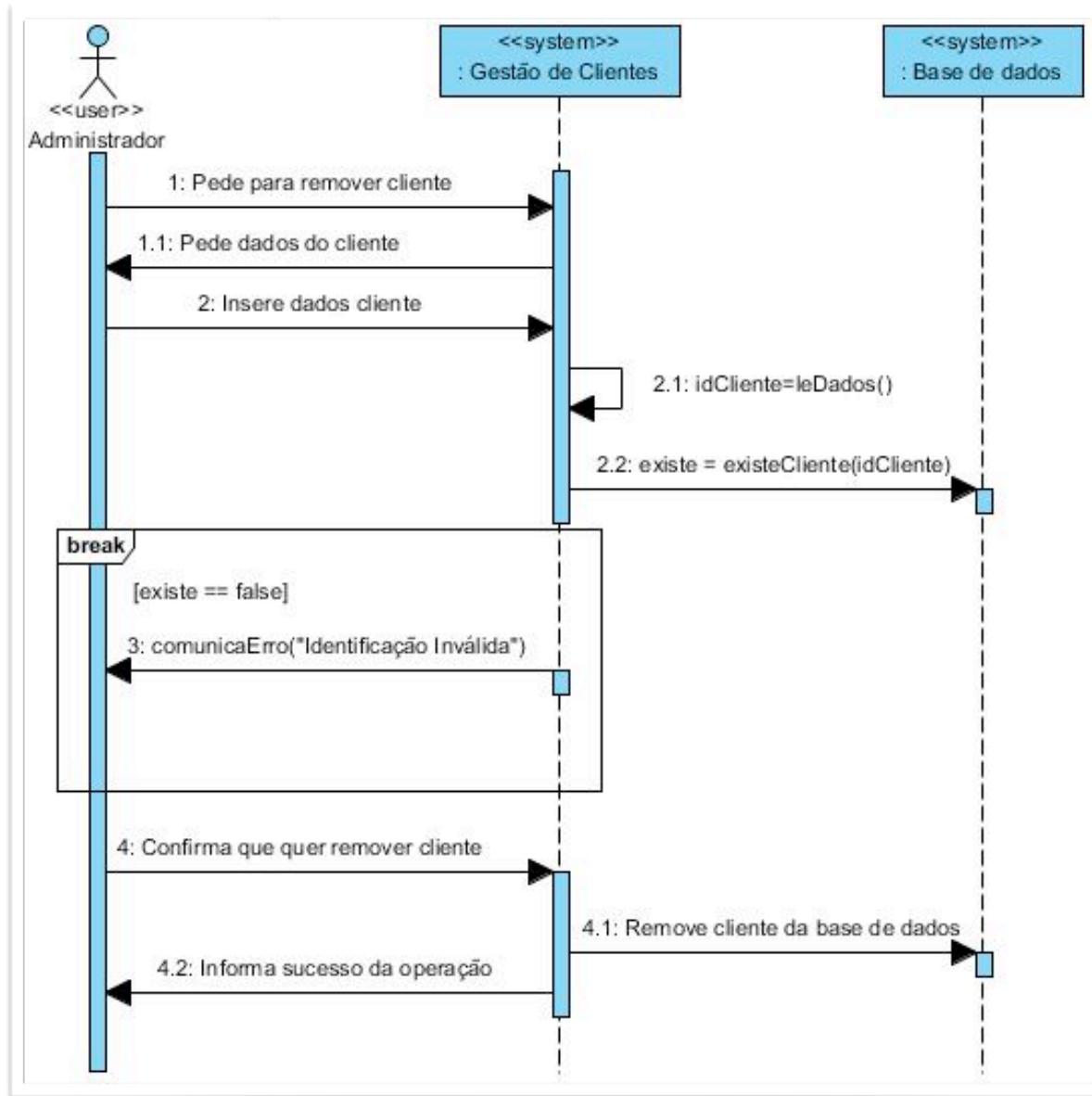


Nesta acção, o funcionário pede á base de dados para registrar manutenção da máquina de pagamento. A base de dados pede os dados do funcionário. Depois do funcionário escrever os seus dados é feita a leitura e a verificação desses dados com os métodos: *leDados()* e *verificaFuncionario()*, respectivamente. Caso se verifique que os dados do funcionário são válidos, o funcionário indica o número da máquina que se pretende fazer a manutenção. A base de dados

verifica, através do método `existeMaquina()`, se a máquina existe no sistema. E de seguida é registada a manutenção com o método `registaManutencao()`. O funcionário é informado do sucesso da operação.

Caso se verifique que os dados do funcionário ou da máquina não são válidos o funcionário é informado do respectivo erro pelo método `comunicaErro()` e a operação é cancelada.

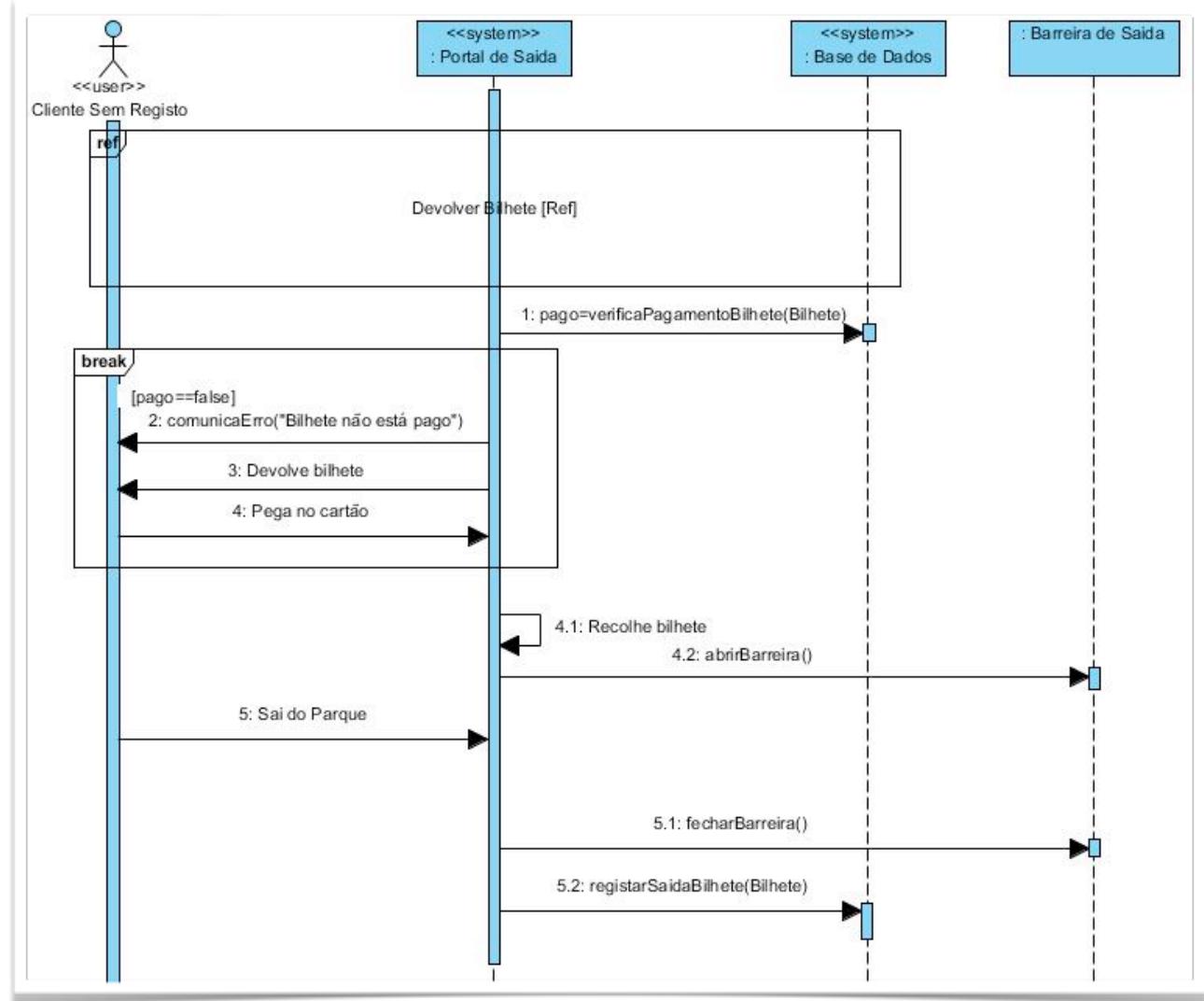
REMOVER CLIENTE



O administrador indica ao subsistema de Gestão de Clientes que pretende remover um cliente do sistema. De seguida, o administrador escreve os dados do cliente que pretende eli-

minar. O subsistema lê os dados, através do método *leDados()*, e o método *existeCliente(idCliente)* verifica se o cliente que se pretende remover existe na base de dados. De seguida, o administrador confirma a opção de remover cliente, e a base de dados fica actualizada. O sistema notifica o administrador que a opção ocorreu com sucesso. Caso o cliente não exista na gestão de clientes o administrador é informada através do método *comunicaErro()*.

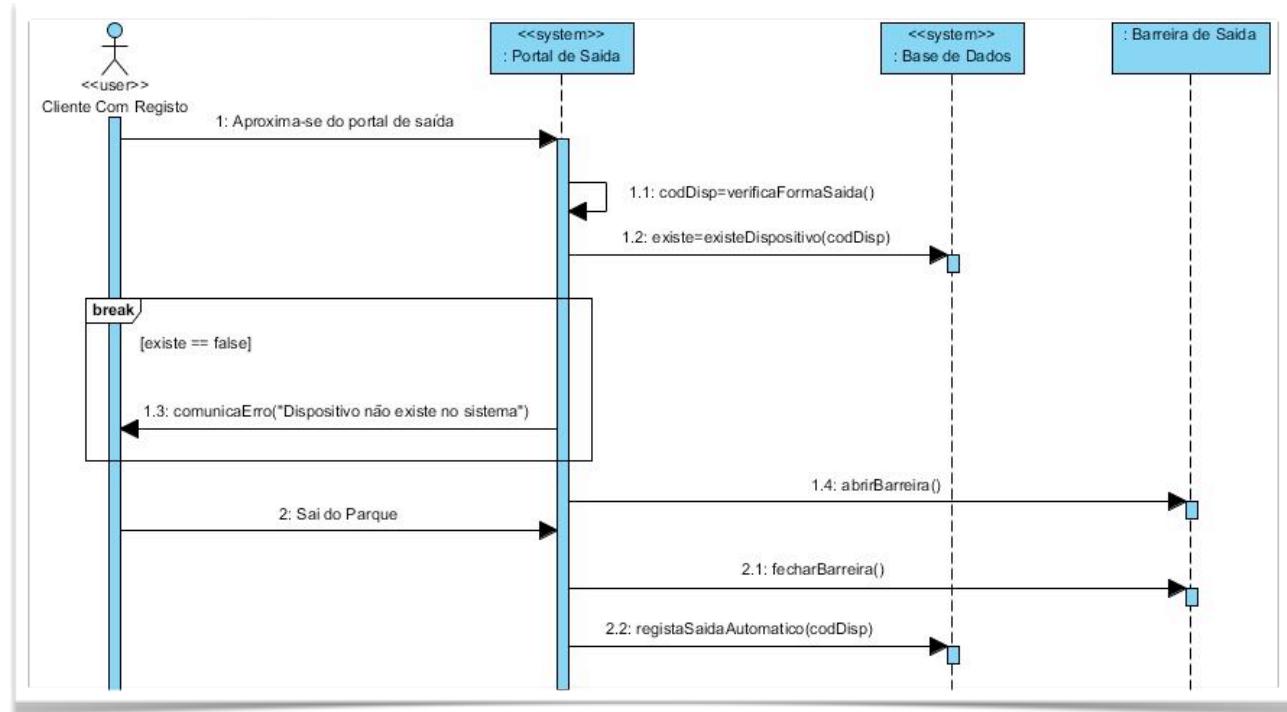
S A I R D O P A R Q U E



Quando um cliente sem registo pretende sair do parque, tem, primeiramente, que devolver o bilhete no portal de entrada, esta acção é executada fazendo referência a outro diagrama de sequência: devolver bilhete. De seguida, o portal de saída verifica se o bilhete está pago, comunicando com a na base de dados através do método *verificaPagamentoBilhete()*. Se o bilhete estiver pago o portal de saída fica com o bilhete e informa a barreira para abrir através do método *abrirBarreira()*. Após o cliente sair do parque fecha a barreira, *fecharbarreira()* , o portal de saída regista a saída na base de dados através do método *registaSaidaBilhete()* .

Caso o bilhete não esteja pago o cliente é avisado, pelo portal de saída e é-lhe devolvido o bilhete .

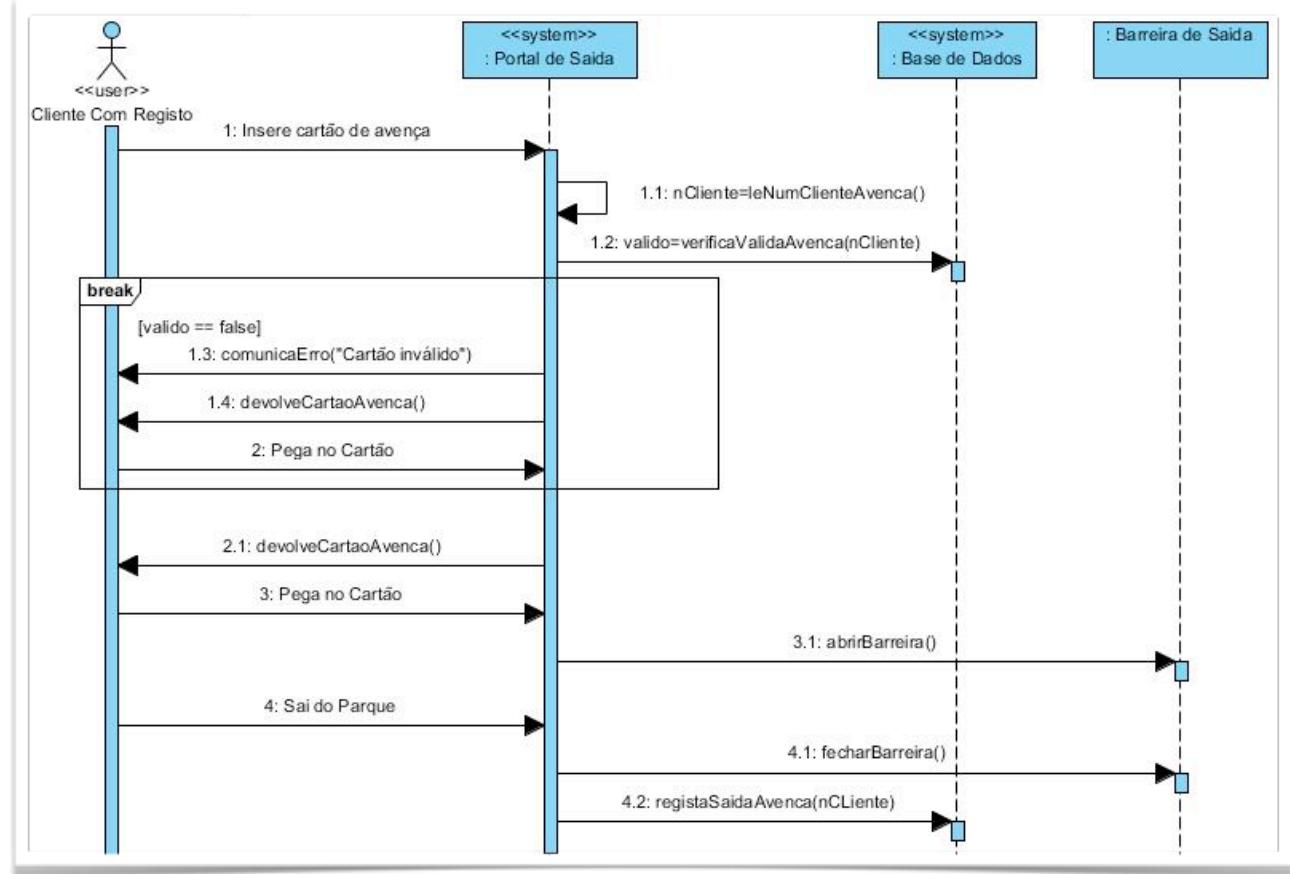
S A I R M O D O A U T O M Á T I C O



Esta operação inicia-se quando o cliente aproxima-se do portal de saída. O portal verifica a forma automática de saída, através do método, *verificarFormaSaída()*. De seguida detecta se o dispositivo automático está registado na base de dados, através do método *existeDispositivo()*. De seguida o portal de saída abre a barreira, *abrirbarreira()*. O cliente sai do parque, barreira fecha através do método *fecharBarreira()*. De seguida, o portal regista a saída com o método *registaSaídaAutomático()*.

Se o dispositivo automático não existir o portal de saída informa o erro ao cliente.

SAIR MODO AVENÇA



Nesta acção o cliente insere o cartão de avença no portal de saída. O portal lê o número do cliente do cartão e verifica se o cartão é válido com os métodos `leNumClienteAvenca()` e `verificaValidaAvenca()`, respectivamente. O cartão de avença, é devolvido ao cliente pelo método `devolveCartaoAvenca()`. A barreira é informada pelo portal de saída através do método `abrirBarreira()` para abrir. De seguida, a barreira é informada pelo portal de saída para fechar, pelo método `fecharBarreira()`. Por último, é registada a saída do cliente, na base de dados, pelo método `registaSaidaAvenca()`.

Caso o cartão de avença seja inválido o cliente é informado do erro no portal de saída, e é devolvido o cartão de avença ao cliente.

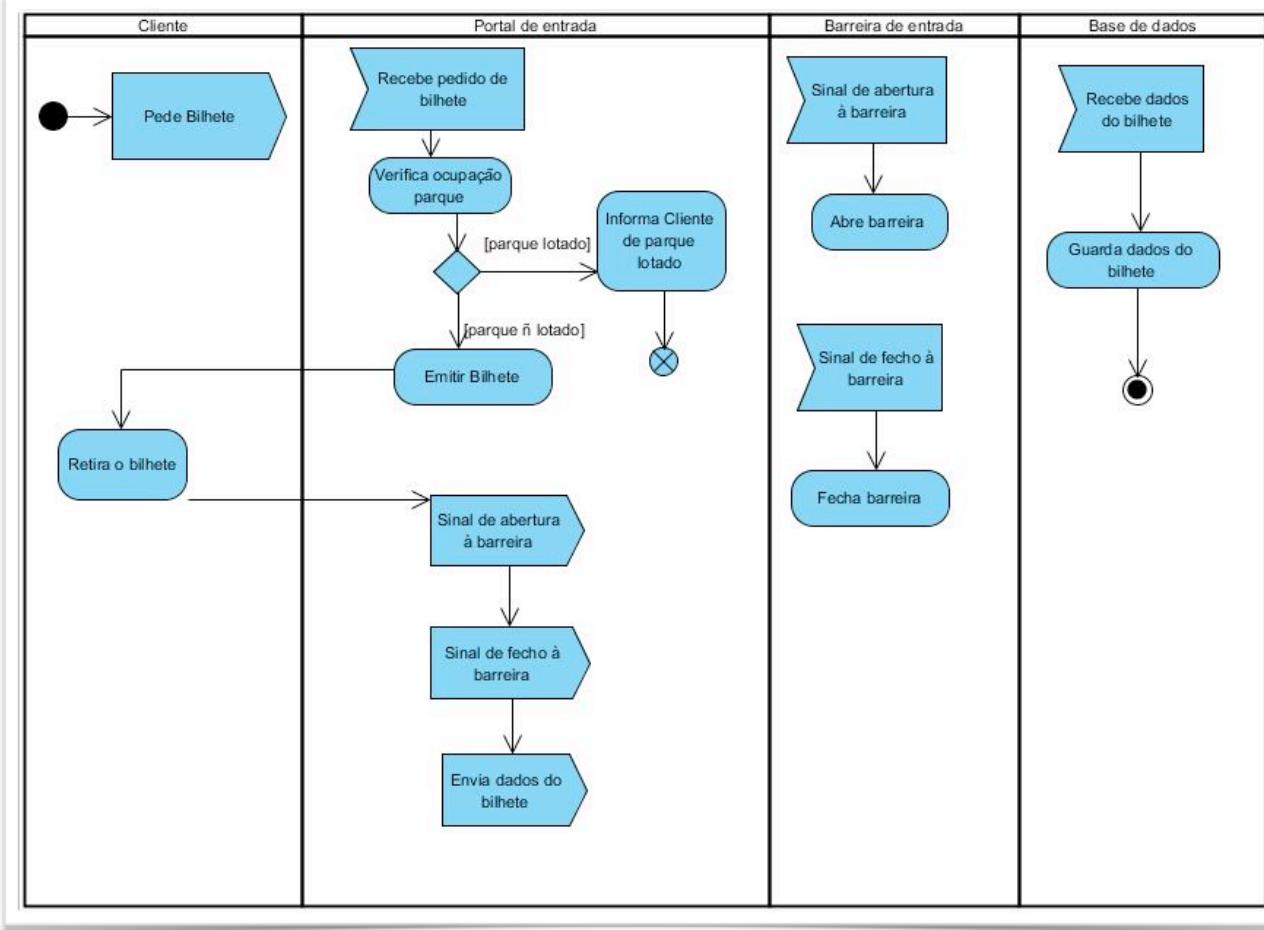
DIAGRAMAS DE ACTIVIDADES

Um diagrama de actividades decompõe uma actividade em sub-actividades (actividades de mais baixo nível), podendo chegar a acções atómicas, com fluxo de controlo sequencial ou concorrente entre sub-actividades.

Como tal foram desenvolvidos os diagramas de actividades para cada Use Case relevante, sendo estes:

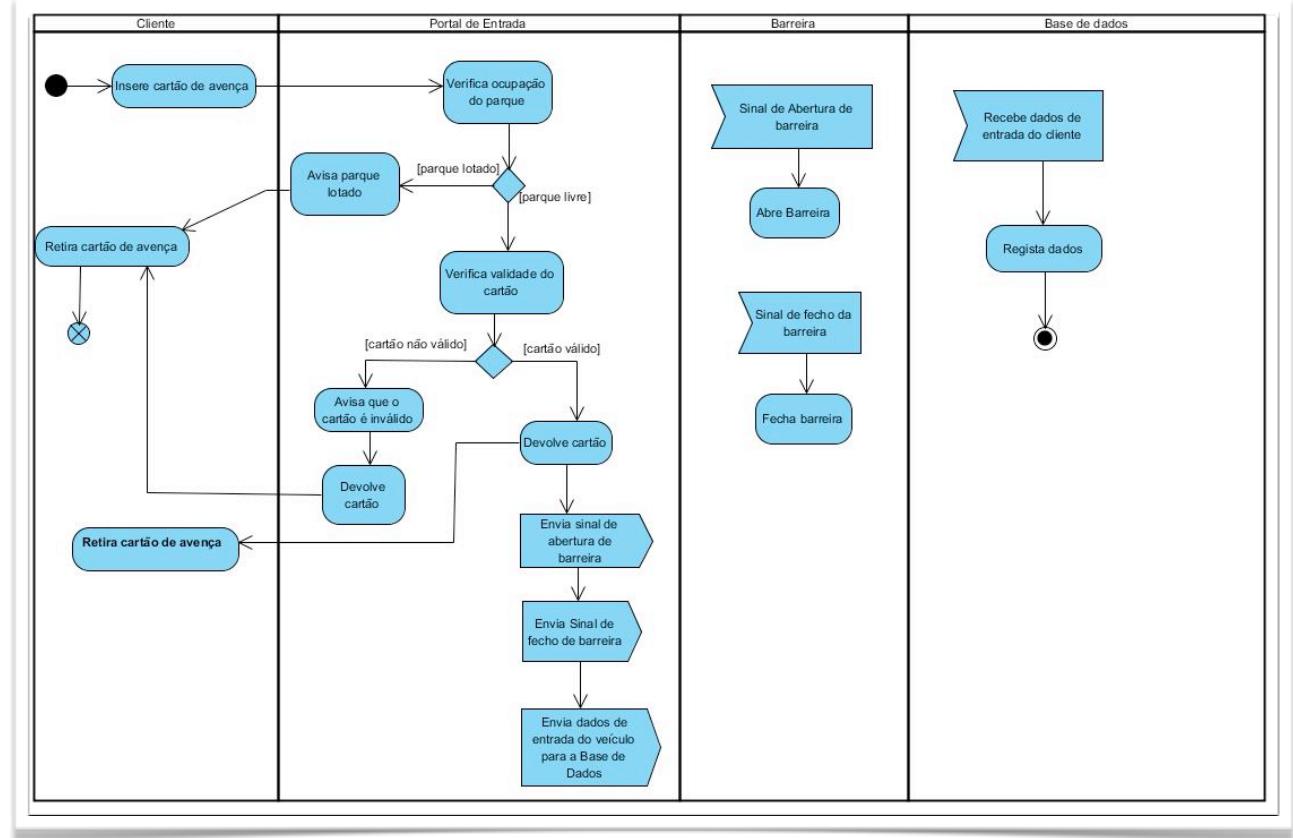
- Entrar no parque (cliente de bilhete)
- Entrar no parque (com cartão de avença)
- Entrar no parque (de modo automático)
- Estacionar Veículo
- Pagar Bilhete
- Registar cliente
- Sair do parque (cliente com bilhete)

ENTRAR NO PARQUE (CLIENTE DE BILHETE)



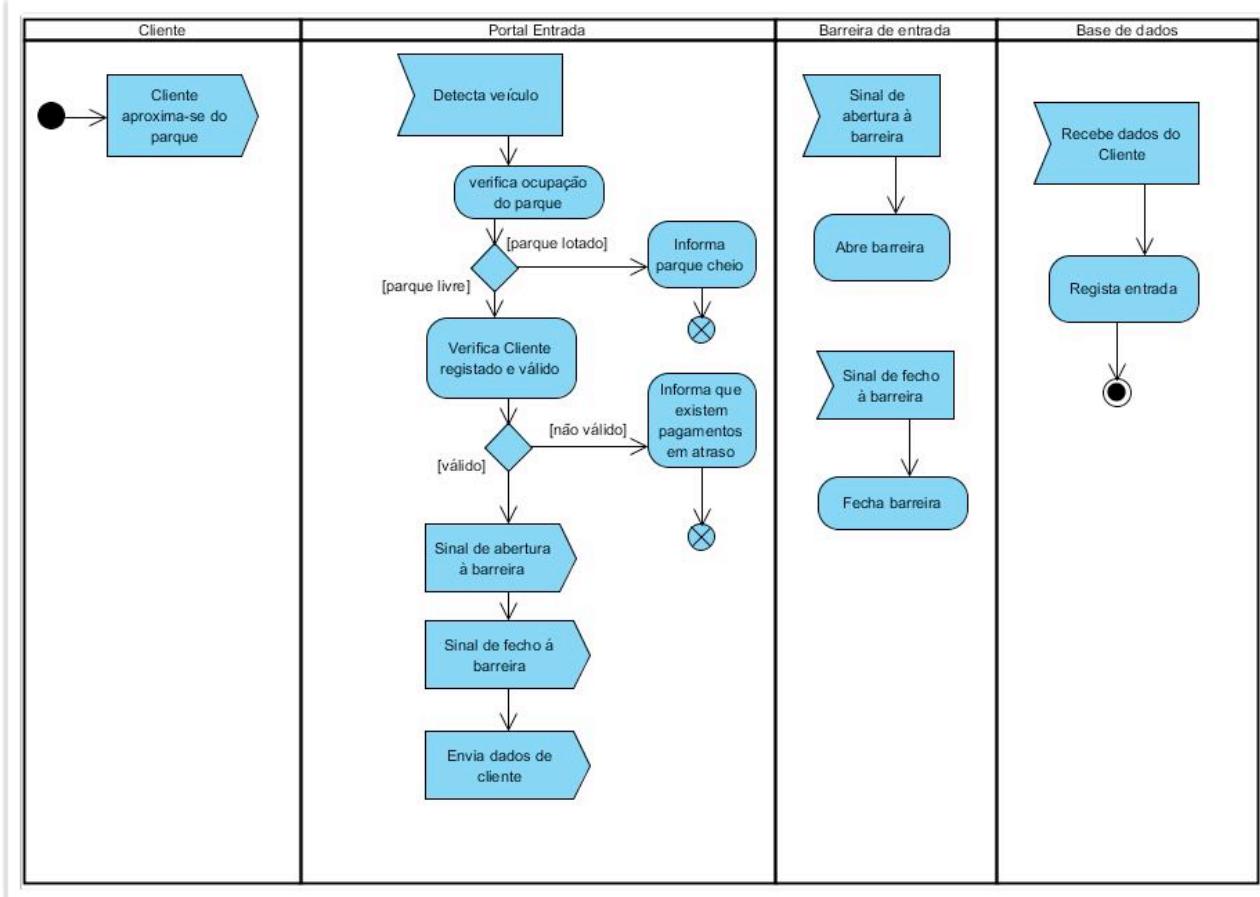
Este diagrama de actividade inicia-se com o envio de um sinal de pedido de bilhete, por parte do cliente. O portal de entrada recebe o pedido, verifica se o parque está cheio e se estiver cheio informa o cliente. Se ainda tiver lugar livres, o portal de entrada envia dados do bilhete para a base de dados e esta guarda os dados do bilhete. Seguidamente, o portal de entrada emite o bilhete e o cliente retira o bilhete. O portal de entrada envia o sinal de abertura à barreira, e quando esta recebe o sinal, abre. Após a entrada no veículo no parque é enviado o sinal à barreira para fechar, por parte do portal de entrada.

ENTRAR NO PARQUE (COM CARTÃO DE AVENÇA)



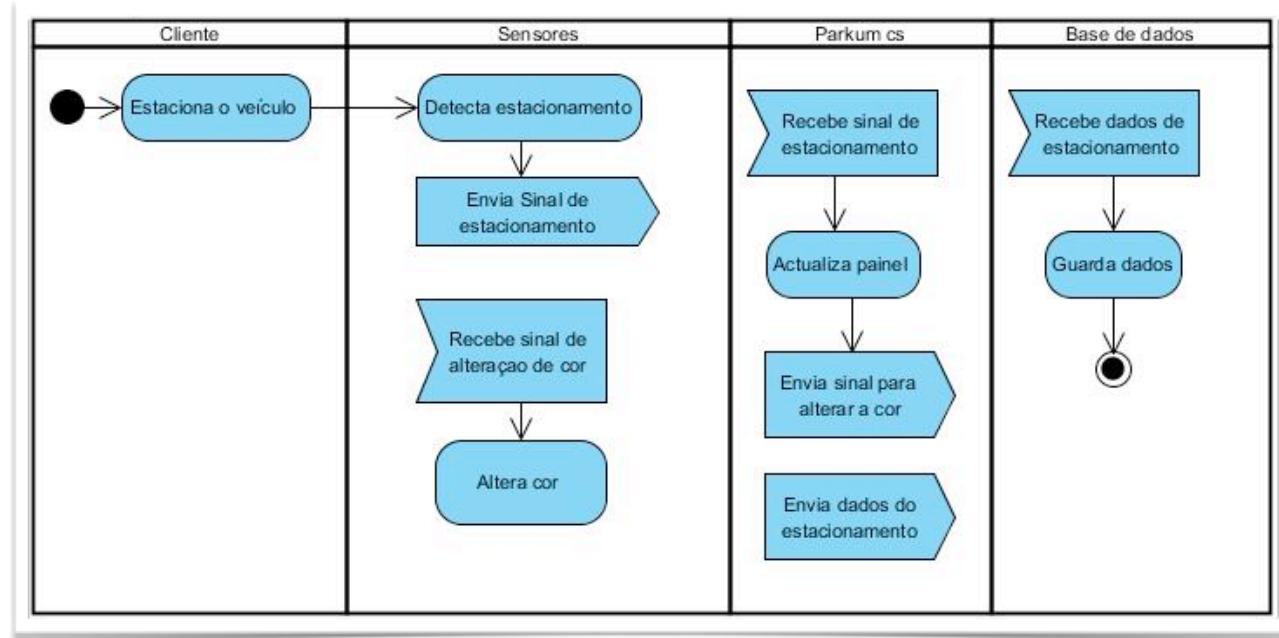
Esta acção inicia-se quando o cliente insere o cartão de avença no portal de entrada. Seguidamente o portal de entrada verifica se o parque está cheio. Se o estiver cheio informa o cliente, e devolve o cartão. Caso contrário, o portal de entrada verifica se o cartão é válido, e caso não seja, o portal devolve o cartão ao cliente. Se o cartão for válido é devolvido o cartão, enviando um sinal de abertura à barreira pelo portal de entrada. A barreira recebe o sinal e abre a barreira, seguidamente, é enviado o sinal de para fechar a barreira os, e então, os dados de entrada do veículo são enviados à base de dados e após a recepção são registados.

ENTRAR NO PARQUE (DE MODO AUTOMÁTICO)



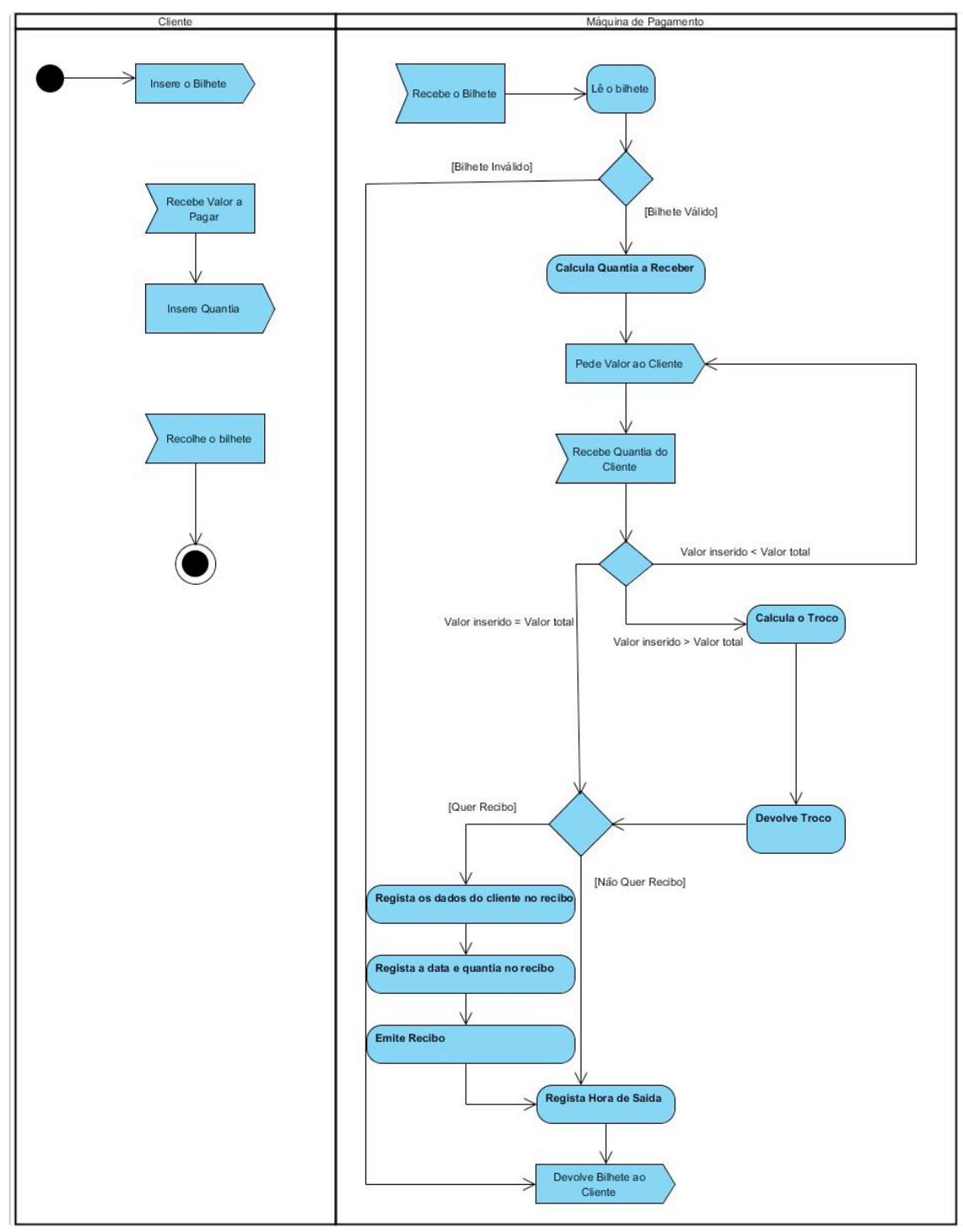
A acção é iniciada quando o cliente se aproxima do portal de entrada, este detecta o veículo, verifica se o parque está cheio. Tendo o parque alguns lugares livres é verificado, se o cliente está registado e é valido então é enviado o sinal de abertura à barreira de entrada. A barreira ao receber o sinal abre. O portal de entrada envia o sinal de fecho à barreira e quando esta o recebe fecha a barreira. Por último, envia os dados à base de dados onde são guardado. Se o parque estiver cheio, ou o cliente for inválido o portal informa o cliente do respectivo problema.

ESTACIONAR VEÍCULO



Quando o cliente estaciona o carro o sensor detecta a presença de um veículo altera a cor do sensor para vermelho e envia o sinal ao PARKUM-CS. O PARKUM-CS quando recebe o sinal actualiza o painel de informação e o painel global. O PARKUM-CS envia os dados do estacionamento à base de dados. Depois de receber o sinal a dados a base regista os dados.

PAGAR BILHETE

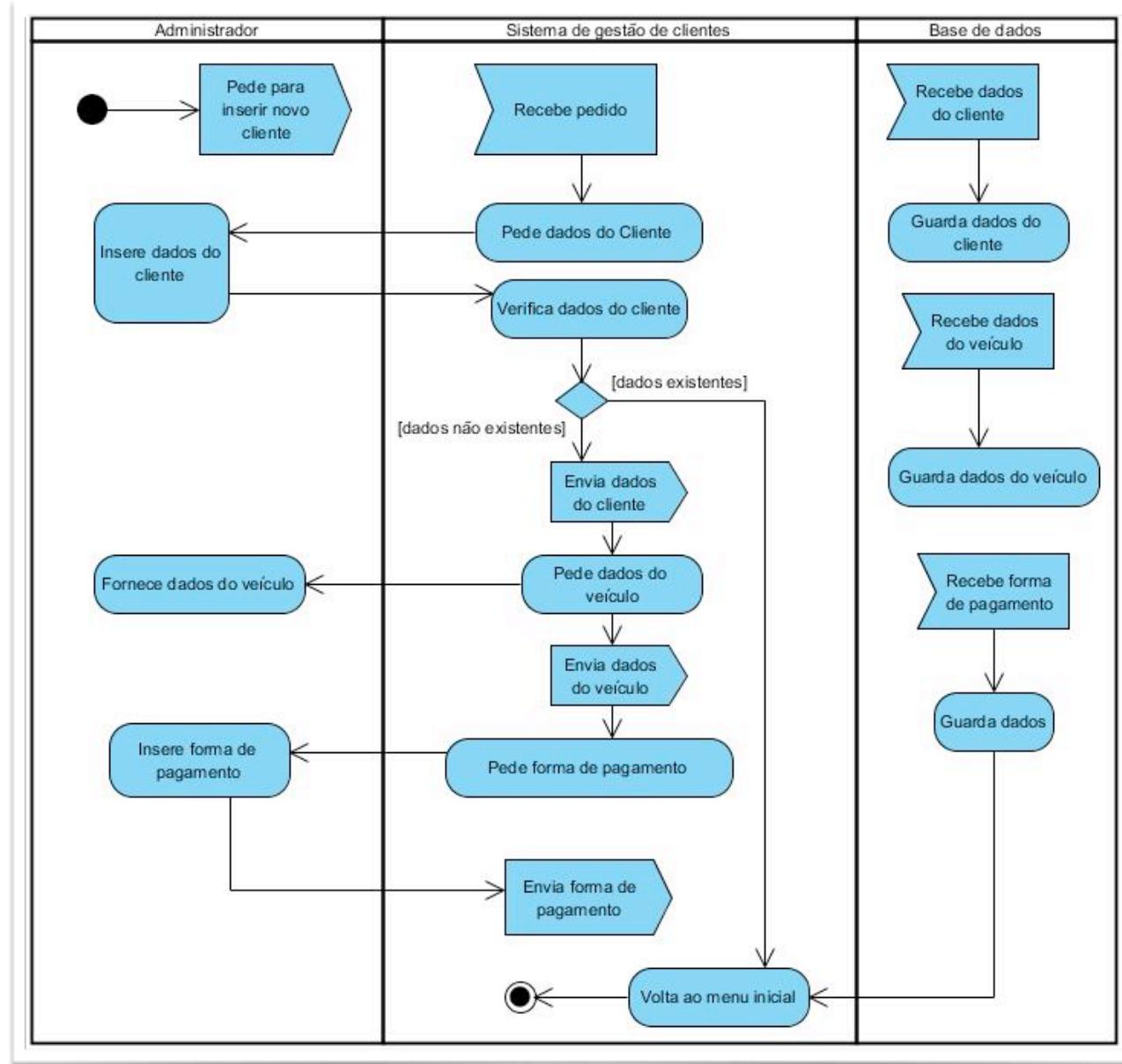


A actividade pagar bilhete começa quando o cliente insere o bilhete na máquina de pagamento. Depois de receber o bilhete, a máquina lê e verifica se é válido. Caso o bilhete seja inválido este é logo devolvido ao cliente. Se for válido, calcula a quantia a receber e pede a quantia ao cliente. O cliente recebe a informação do valor pedido e insere a quantia a pagar.

Enquanto a quantia não for suficiente, é pedido o valor em falta ao cliente. Se for superior a máquina de pagamento calcula o valor do troco. Devolve o troco e questiona o cliente se este quer recibo. Se, eventualmente, o cliente quiser recibo a máquina regista no bilhete os dados do cliente, a data e a quantia do pagamento e emite o recibo.

Termina registando a hora de saída e devolve ao cliente o bilhete.

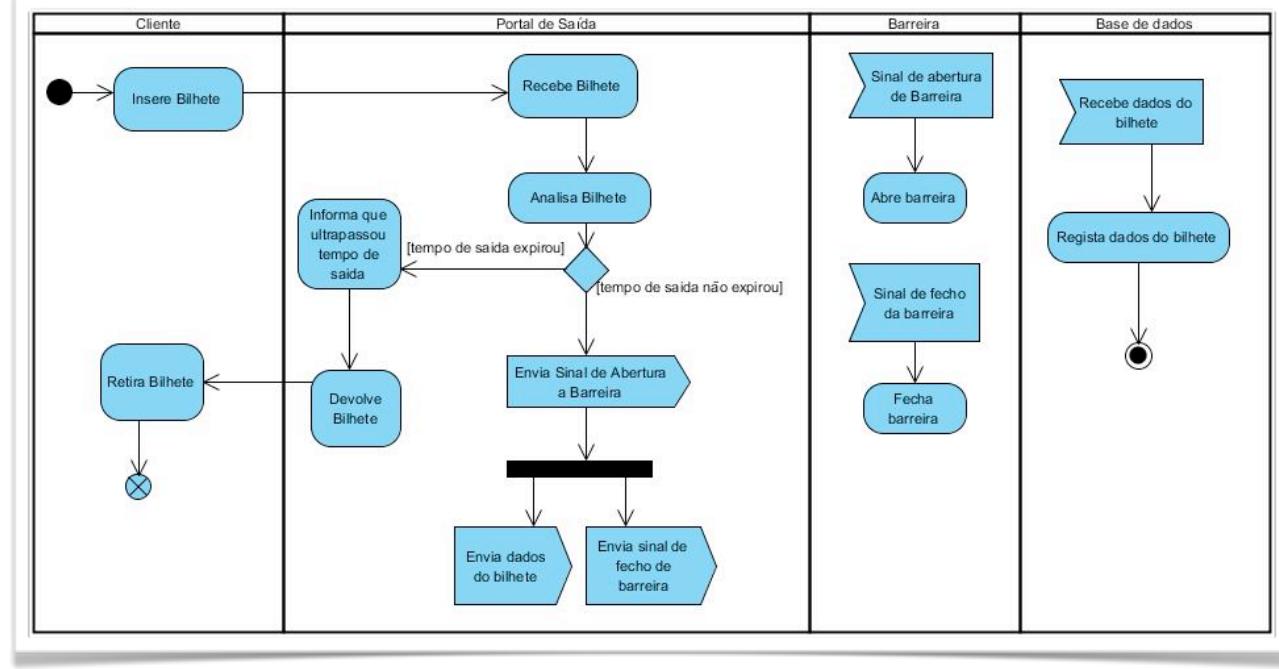
R E G I S T A R C L I E N T E



Neste caso, o administrador pede ao subsistema de gestão de clientes a opção de registo de cliente. O subsistema pede os dados do cliente. Depois de inseridos verifica se os dados do cliente já existem. Se, eventualmente, os dados do cliente já existirem a gestão de clientes can-

cela operação. Se não existirem os dados do cliente são enviados e registados na base de dados. Seguidamente, são pedidos os dados do veículo. Após o administrador introduzir esses dados a gestão de clientes envia à base de dados essa informação. Por último o sistema pede forma de pagamento que o cliente pretende. E após a informação ser fornecida por parte do funcionário os dados são novamente enviados à base de dados.

SAIR DO PARQUE (CLIENTE COM BILHETE)



Esta acção é desencadeada quando o cliente insere o bilhete no portal de saída. Por sua vez, o portal recebe o bilhete e analisa se ele é válido. Caso o bilhete não esteja válido o portal de saída informa o cliente e devolve o bilhete. Caso contrário, envia o sinal de abertura à barreira. Depois de receber o sinal a barreira abre. Seguidamente, a barreira de saída recebe o sinal para fechar.

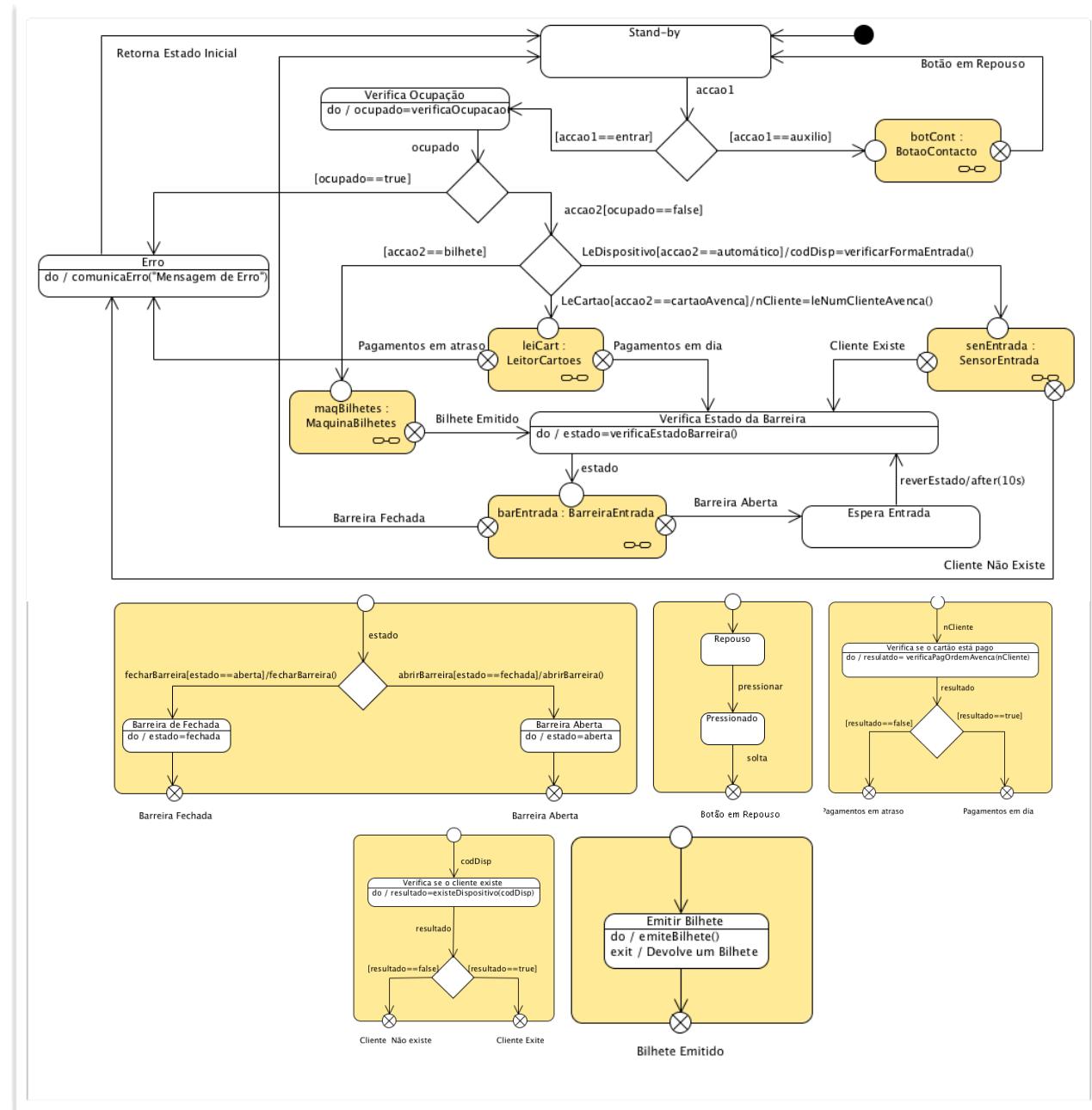
Por último o portal envia os dados do bilhete à base de dados, que os guarda.

DIAGRAMAS MÁQUINAS DE ESTADOS

Nos diagramas de estados um objecto possui um comportamento e um estado. É no diagrama de estados mostra os possíveis estados de um objecto e as transacções responsáveis pelas mudanças de estado.

Os diagramas de estados permitiram modelar o comportamento dos seguintes subtemas:

PORTAL DE ENTRADA

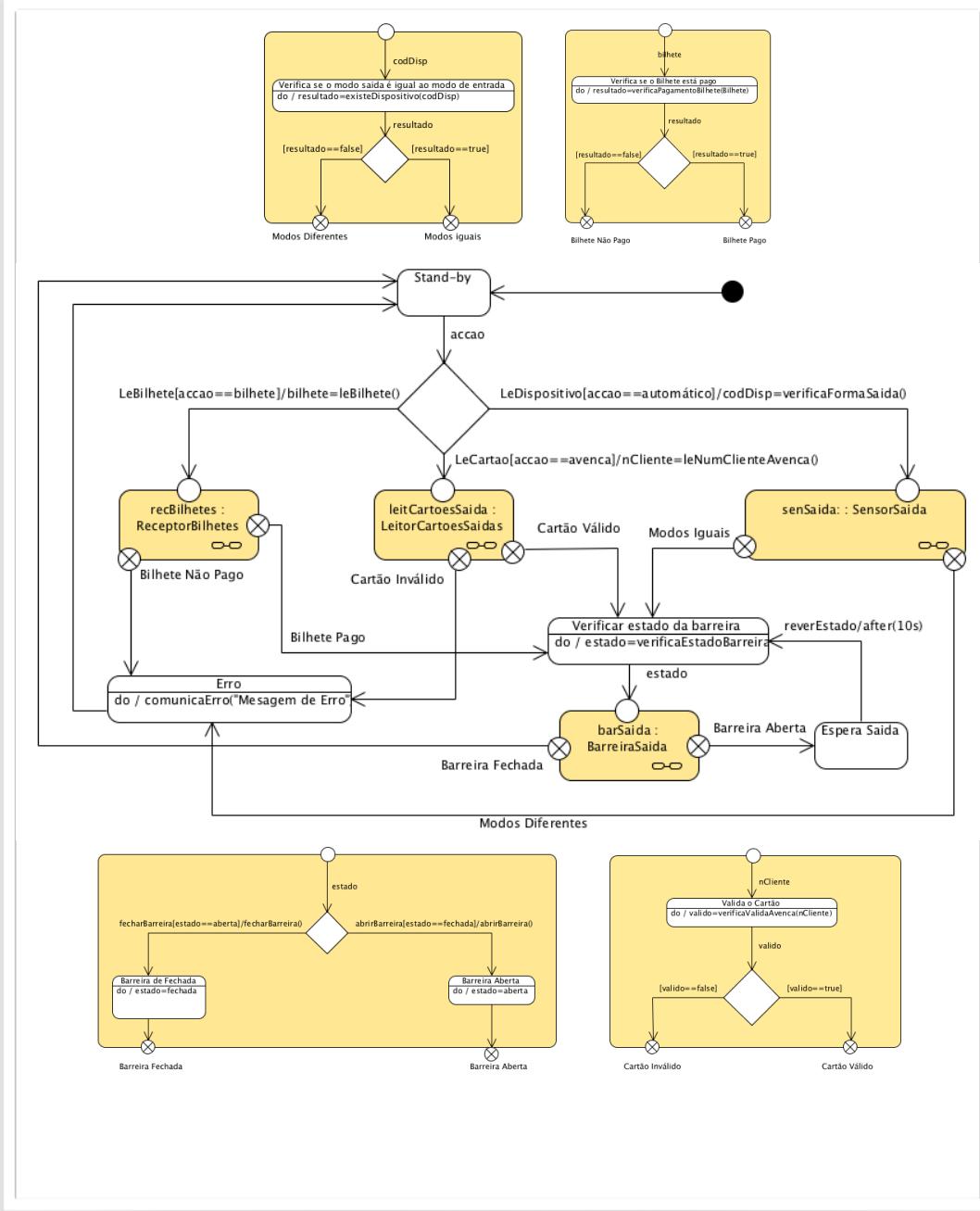


Este diagrama de estados mostra-nos os vários estados do portal de entrada, que se inicia em stand-by e aguarda uma acção. Esta acção pode ser desencadeada accionando o botão de contacto com o funcionário ou verificando a ocupação do parque para que um cliente possa entrar no parque.

Se o cliente não tem registo, efectua o pedido a emissão um bilhete. Se o cliente possui cartão de avença, este deve ser lido e avaliar se as condições de entrada estão regularizadas. Finalmente, se o cliente possui um dispositivo de detecção automática, é dada permissão para entrar caso este seja reconhecido.

Todas estas acções geram a abertura da barreira no caso do sistema permitir a entrada do veículo. Depois da passagem do veículo, a barreira é fechada e o portal de entrada retorna ao estado de stand-by. No caso de insucesso, deve ser comunicado o erro, e voltar ao estado inicial.

PORTAL DE SAÍDA

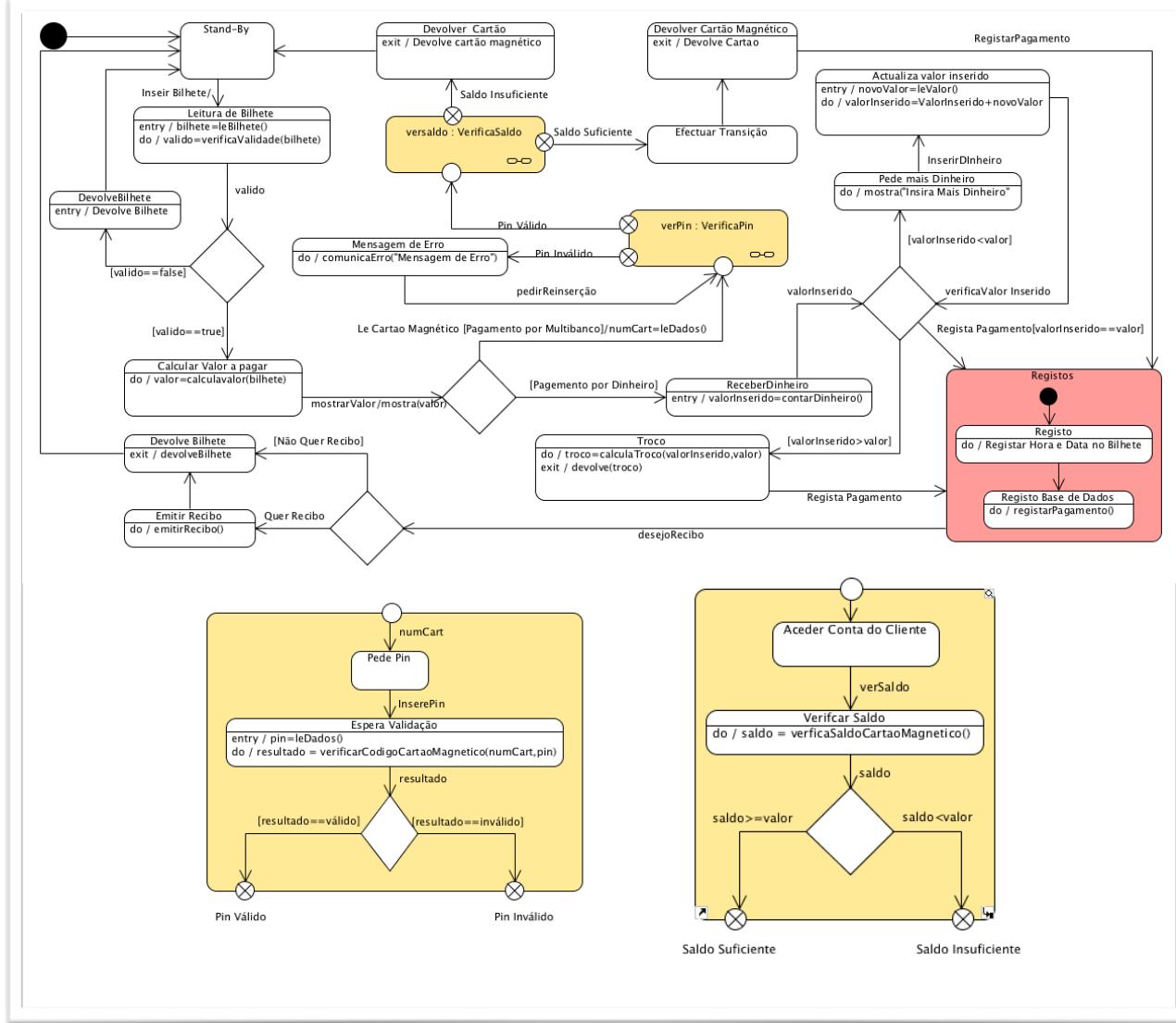


O estado inicial do portal de saída é de stand-by, podendo receber três acções: receber um bilhete, um cartão de avença ou uma saída automática. Estes, por sua vez, estão associados a dispositivos que são respectivamente receptor de bilhetes, leitor de cartões de saída e sensor de saída em que cada um possui um diagrama de estado associado.

Quando a saída é feita por devolução de bilhete, e o bilhete está pago, a barreira verifica o seu estado, abre e espera segundos pela saída do veículo. De seguida, volta a verificar o seu estado, fecha e o portal de saída volta ao estado inicial. O processo é semelhante para cartão de avença.

Se a saída é feita automaticamente, verifica-se apenas se o modo de entrada foi o mesmo, e são desencadeadas as acções de abrir e fechar a barreira. No final do processo, o estado do portal volta ao estado inicial.

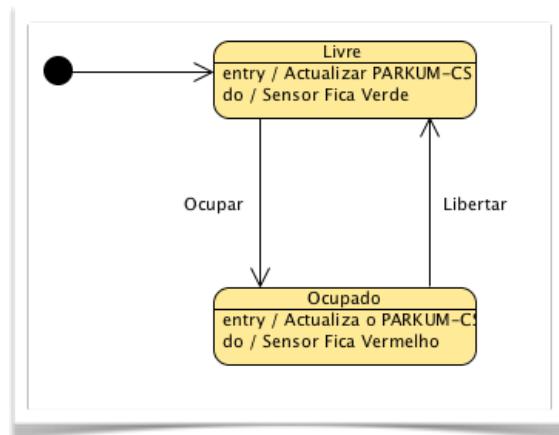
MAQUINAS DE PAGAMENTO



Inicialmente a máquina está em estado stand-by e ao ser inserido o bilhete lê e verifica se o bilhete é válido: Nesse caso, é calculado o valor a pagar, e, de seguida, mostra-o ao cliente e o cliente tem a opção de pagamento em dinheiro ou em multibanco. Se o cliente escolher a opção de pagamento em dinheiro, a máquina recebe o dinheiro e faz a contagem do mesmo. Quando o valor inserido é o correcto, regista a hora e data no bilhete, regista o pagamento na base de dados e devolve o bilhete ao cliente e volta para o estado de stand-by. Se o valor inserido for superior ao pedido calcula o troco e devolve. Caso o cliente escolha o pagamento em multibanco, é-lhe pedido o pin. Após a correcta inserção do pin é verificado o saldo do cartão, e de seguida é efectuado registo da transacção, o registo da hora e data no bilhete, registado o pagamento na base de dados e é devolvido o cartão magnético e por último o bilhete.

Na situação em que o pin inserido é errado, é exibido o código de erro e volta a pedir o pin. Se o saldo não for suficiente é devolvido o cartão e é de novo pedida a forma de pagamento que o cliente quer. Caso o bilhete não seja válido máquina devolve bilhete e volta ao estado de stand-by.

SENSORES DE LUGAR



Inicialmente o seu estado é livre e está com a cor verde. Depois da acção ocupar estacionamento, o estado passa a ocupado e cor altera-se para vermelho. Se o lugar voltar a ser desocupado volta ao estado livre.

DIAGRAMAS DE CLASSES

Os diagramas de classes fazem uma representação da estrutura e relações das classes que servem de modelo para os objectos do sistema. Cada classe contém os seus atributos e os respectivos métodos. As classes identificadas foram as seguintes:

P O R T A L E N T R A D A

Inclui todos os métodos para que o sistema permita a entrada de um cliente no parque, independentemente de ser registado ou não.

P O R T A L S A I D A

Inclui os métodos que permitem ao sistema autorizar a saída de um cliente do parque.

B A R R E I R A

Possui os métodos que permitem abrir e fechar a barreira.

M O N I T O R

Possui os métodos que permitem a comunicação entre o sistema e o cliente, através de mensagens de informação.

B I L H E T E

Possui apenas variáveis de instância, que correspondem às propriedades do bilhete.

G E S T A O C L I E N T E S

Inclui todos os métodos que permitem ao sistema efectuar a gestão dos clientes registados do parque.

F I C H A C L I E N T E

Possui apenas variáveis de instância, que correspondem aos dados do cliente.

P A G A M E N T O

É uma classe abstracta - inclui a classe PagamentoAutomático, que possui o método de processar o pagamento dos clientes que entram no parque de forma automática, isto é, sem necessidade de qualquer bilhete ou cartão; inclui também a classe PagamentoAvença, que possui o método de registo de pagamento do cartão de avença, para os clientes registados com este modo de entrada.

M A Q U I N A P A G A M E N T O

Inclui todos os métodos que, no seu conjunto, possibilitam a realização da tarefa central de pagamento de bilhete.

R E C I B O

Possui variáveis de instância que correspondem aos atributos usuais de um recibo, e um método expõe todas essas informações.

G E S T A O R E L A T O R I O S

Apresenta os métodos que permitem ao sistema gerar e apresentar todos os relatórios.

S E N S O R E S E S T A C I O N A M E N T O

Possui o métodos que actualiza cada sensor, no caso de se efectuar um estacionamento ou a desocupaçāo de um lugar, e ainda outra que comunica o seu estado para efeitos de actualização de todo do sistema .

P A R K U M - C S

Possui os métodos que permitem ao sistema efectuar a contagem do número de viaturas no parque, e ainda incrementar e decrementar o número de viaturas, consoante a informação recebida pelos sensores.

P A I N E L I N F O R M A C A O E P A I N E L G L O B A L

Incluem métodos que possibilitam a actualização dos painéis de informação de nível e do painel global.

B A S E D A D O S

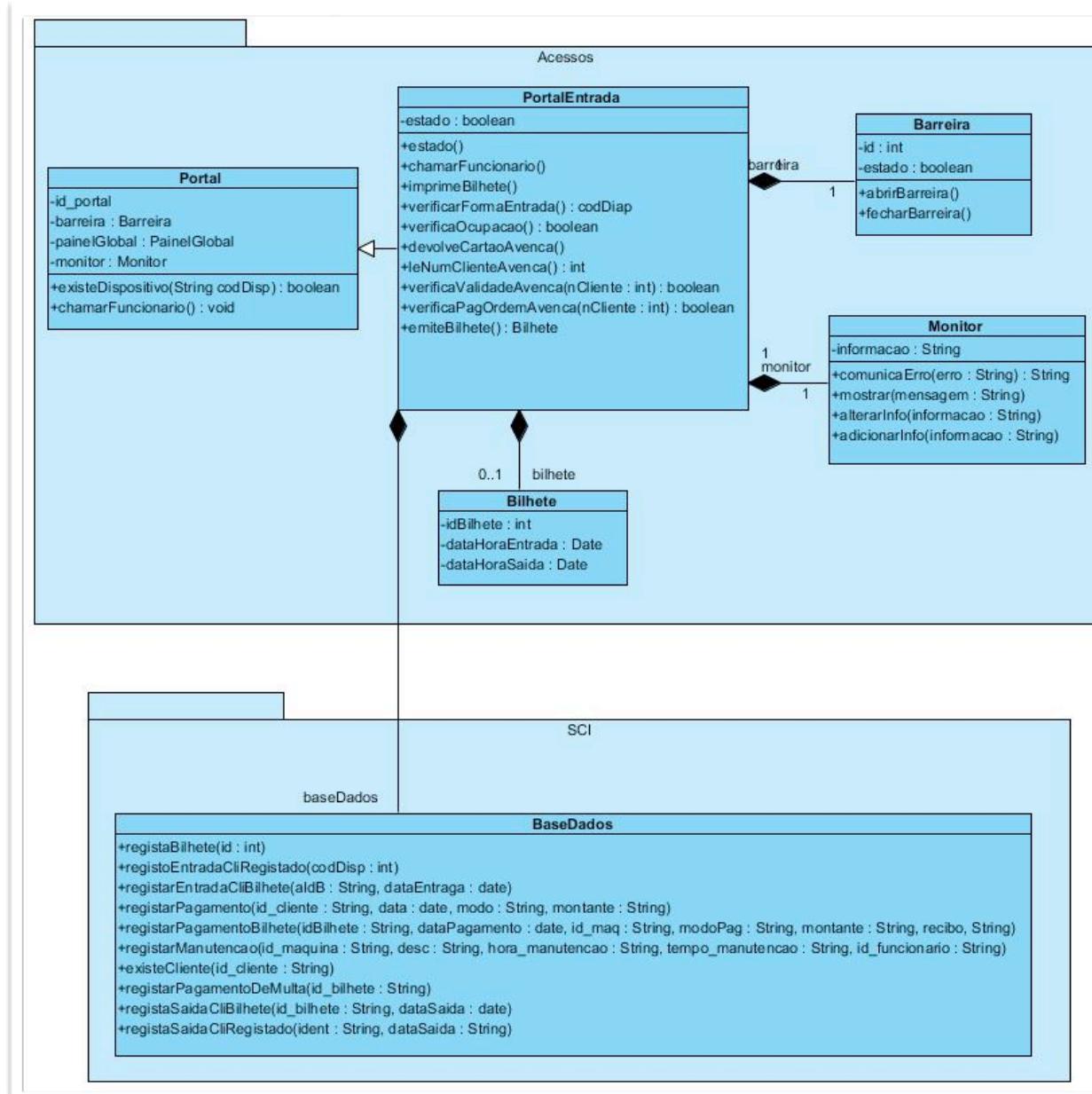
Inclui os métodos emergentes para que sejam efectuados todos registo necessários para o normal funcionamento de um parque de estacionamento.

AGRUPAMENTO EM PACKAGES

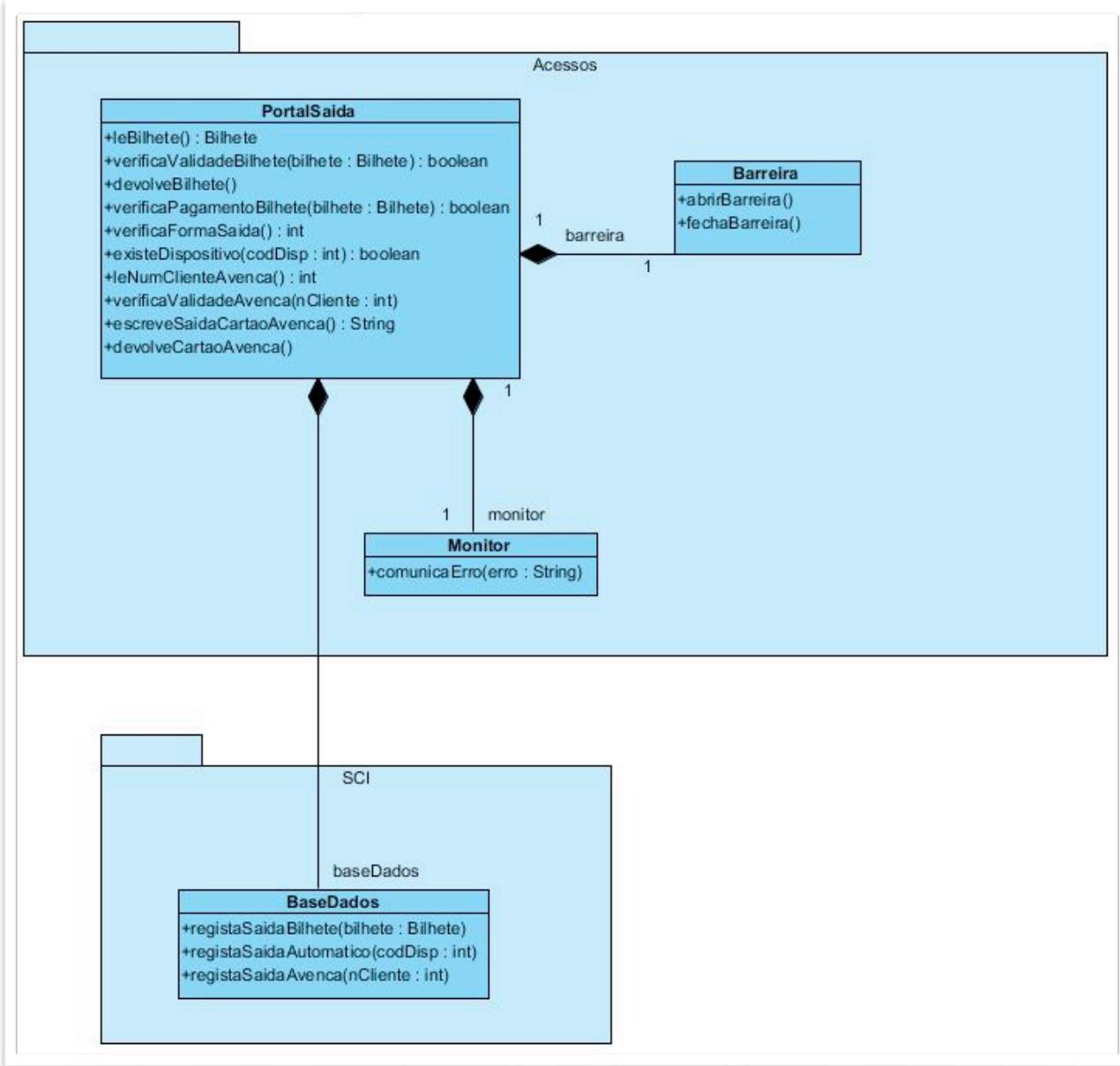
Depois de desenvolvidos todos os diagramas de classes, as diversas classes identificadas foram agrupadas em *packages*, possibilitando assim a representação estrutural para implementar as diversas funcionalidades do parque, designadamente:

A C E S S O S

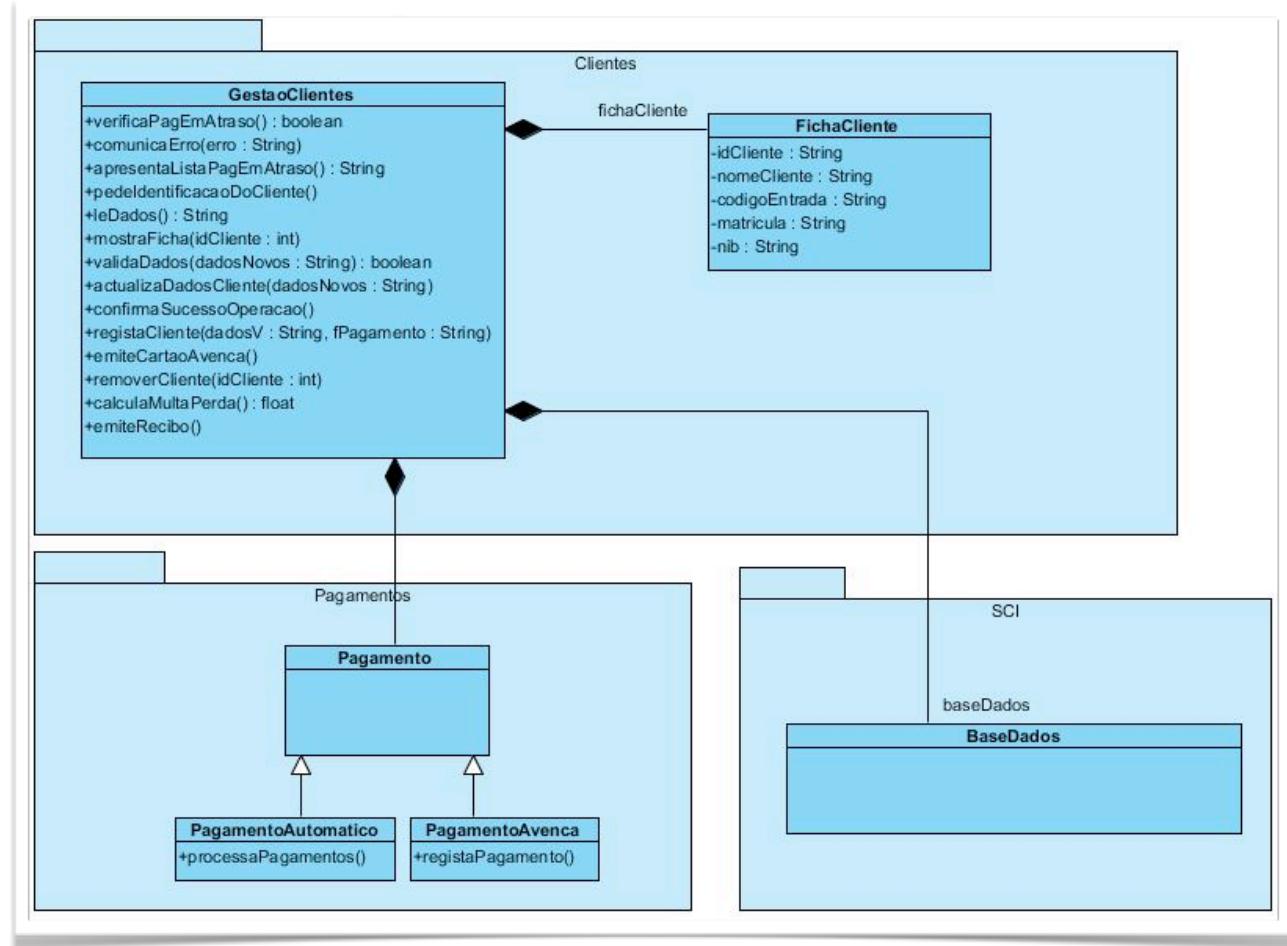
- Portal de Entrada



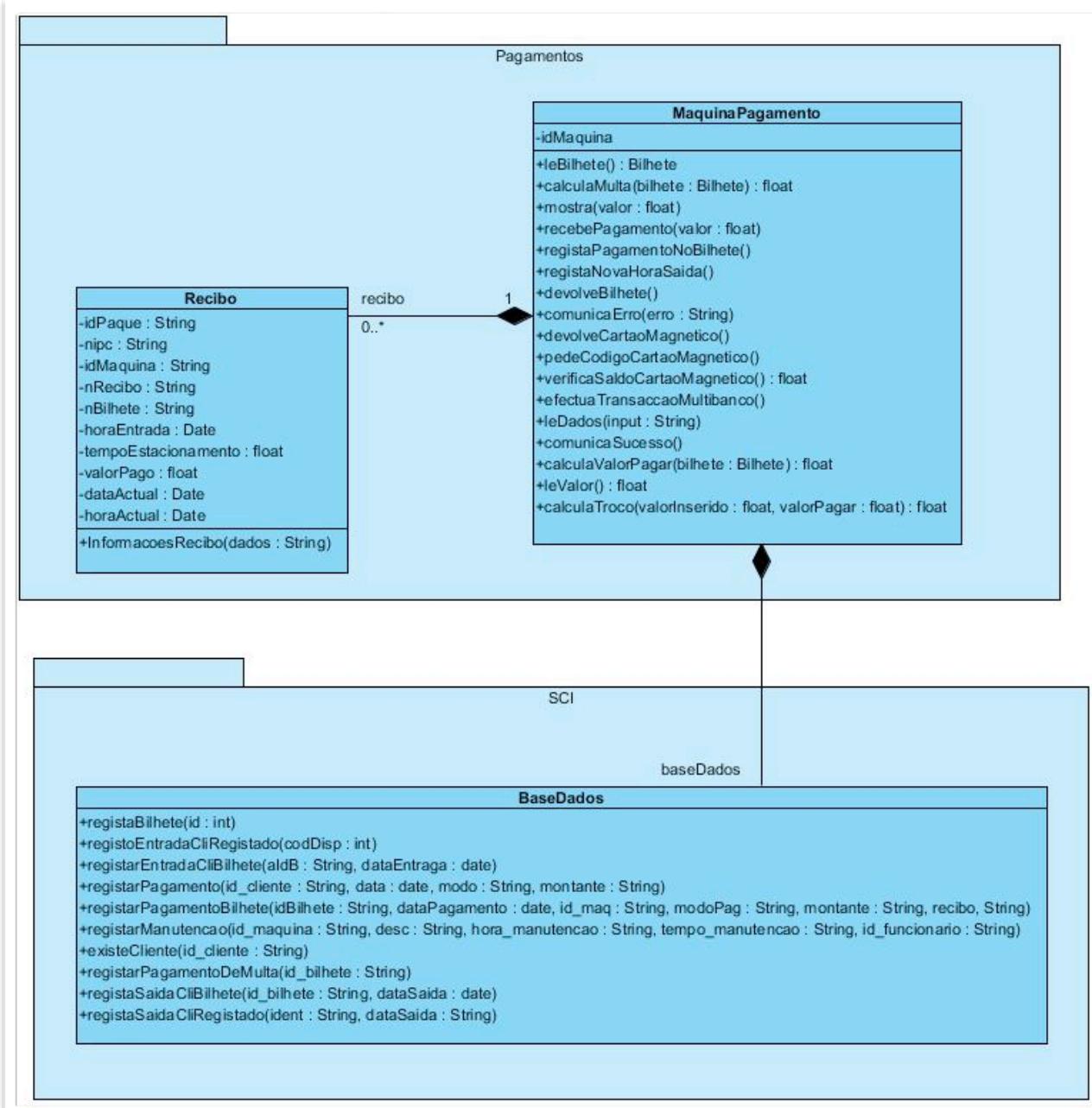
- Portal de Saída



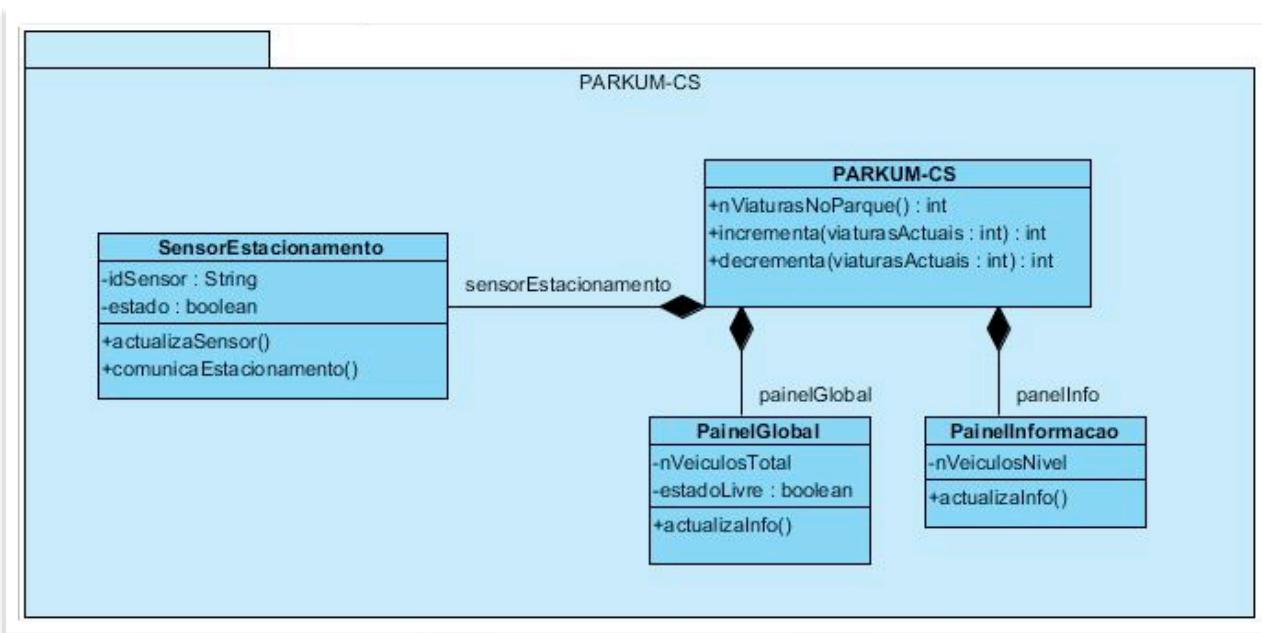
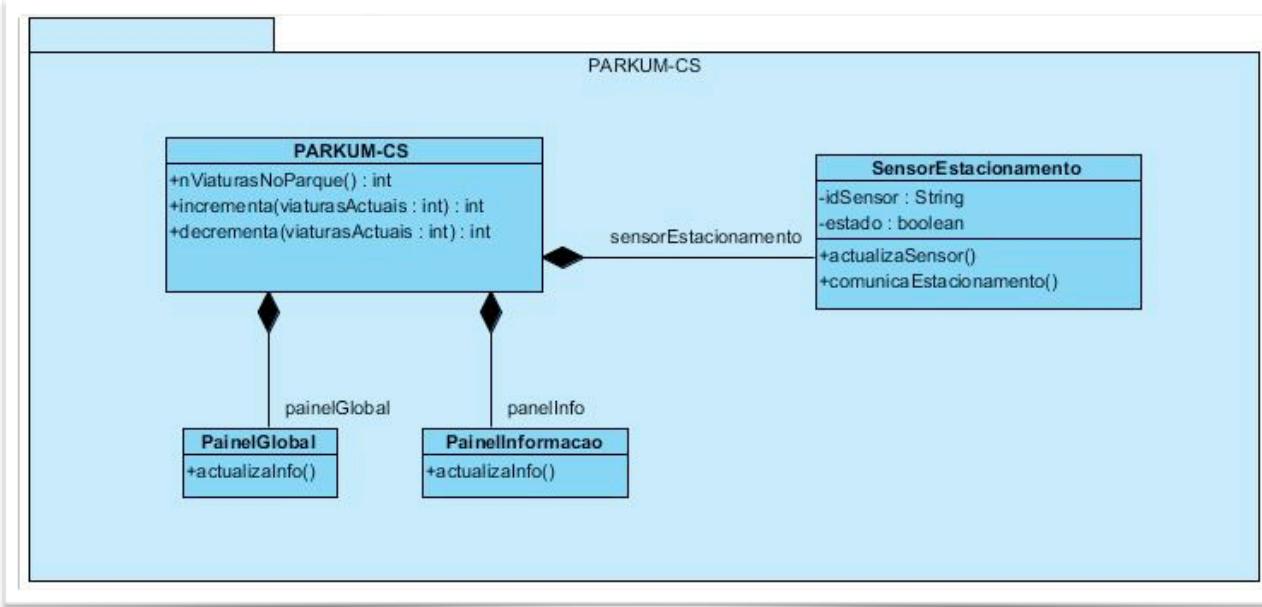
CLIENTES



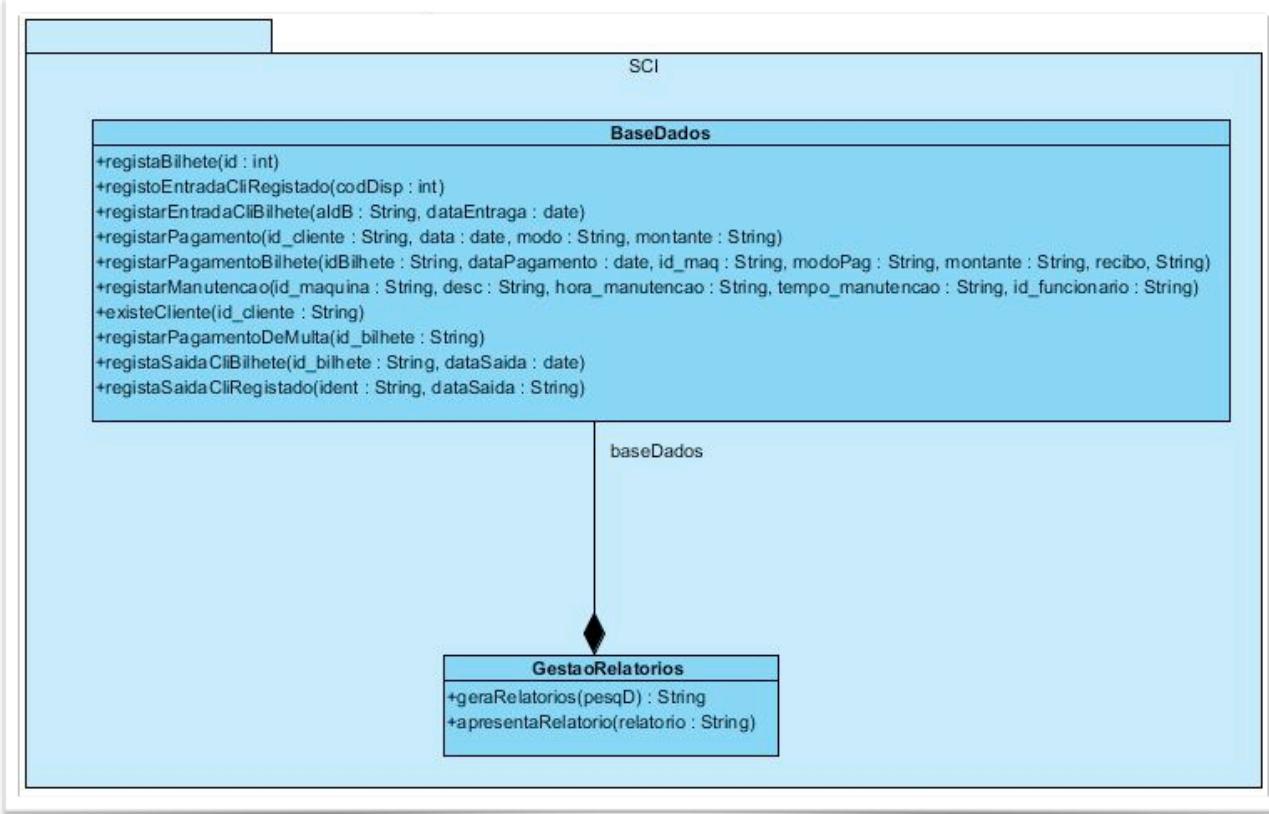
PAGAMENTOS



P A R K U M - C S



SCI: SISTEMA CENTRAL DE INFORMAÇÃO



IMPLEMENTAÇÃO

Para concluir os requisitos deste projecto, desenvolvemos uma aplicação em JAVA. Esta aplicação pretende simular toda a actividade interna do parque, tentando aproximar-se das funcionalidades de um parque de estacionamento real o mais possível.

BASE DE DADOS

Todo o estado inicial do parque foi concebido a partir de registos contidos na base de dados. O parque está configurado com 5 níveis, e com 50 lugares cada nível. Foram designados os funcionários do parque, que são 10 (dois por nível), e as máquinas de pagamento, existindo duas no primeiro nível e uma nos restantes níveis. Foram criadas de fichas de clientes registados no parque, registo de entradas e saídas, de estacionamentos e de veículos ainda estacionados, de pagamentos efectuados, e de manutenção de máquinas de pagamento.

Recorremos à utilização de JDBC para a criação e acesso aos dados da camada de persistência.

INTERFACE

A interface da aplicação foi desenvolvida em Java Swing, e pretende ser usada por dois tipos de utilizadores distintos: administrador e funcionários do parque. Esta interface foi idealizada para representar um sistema que seria instalado, de facto, num parque de estacionamento. Desta forma, é possível fazer a gestão de todas as funcionalidades do parque, tendo em conta essencialmente os requisitos enunciados. No mini-manual de utilização da aplicação está explicado o modo de funcionamento da interface.

TABELAS CRIADAS

Para a realização do trabalho foi necessário povoar a base de dados com informação relativa ao parque. As tabelas criadas foram bastantes, mas de seguida apresentamos aquelas com mais relevância para procura e registo de informações:

T A B E L A C L I E N T E S

ID_CLIENTE	NOME	ID_ENTRADA	MATRICULA	NIB
1	Ana Sampaio	CA	97-AI-91	253817994200723000000
2	Miguel Costa	CW	83-PC-50	187590182183401000000
3	Catarina Campos	VV	93-CJ-45	132209822204810000000
4	Hugo Frade	CW	10-GN-02	192702656083175000000
5	Andreia Silva	CA	17-NA-30	992237456423757000000
6	Pinto da Costa	RM	56-ET-99	586046365055740000000
7	Jose Saramago	VV	41-JS-12	411693727874434000000
8	Jose Socrates	VV	80-BN-02	678022475088760000000
9	Luis Figo	RM	09-HM-37	571527645034226000000
10	Cristiano Ronaldo	CW	07-CR-09	559808178481802000000
11	Naria Carlitos	RM	QX-79-84	793926278497171000000
12	Silvia Rosa	CA	30-81-AA	658016600973839000000
13	Anabela Josefa	RM	GU-41-84	792499983788587000000
14	Madalena Eustaquio	CA	79-BI-99	685438245361989000000
15	Cornelio Flavia	RM	CD-15-29	114034237529456000000
16	Emilio Olegario	CA	41-BE-81	692831611987127000000
17	Branca Aldo	CW	72-93-FQ	460754507366937000000
18	Renata Celestino	RM	PG-08-70	742508930731356000000
19	Lourenco Alfredo	VV	AA-11-22	758415585293017000000
20	Amilcar Catia	CW	AB-12-23	705410789633529000000
21	Casimiro Bernardo	VV	AC-13-24	413956006379929000000
22	Clara In?cio	RM	AD-14-25	117028261984476000000
23	Aloisio Sofia	CW	AE-15-26	205883728504026000000
24	Dulce Vera	CA	AF-16-27	658958014365297000000
25	Elisa Reinaldo	RM	AG-17-28	329512709505527000000
26	Bonifacio Victor	CW	AG-17-29	88158028543639900000
27	Mauricio Egídio	RM	AG-17-30	418055188821681000000
28	Bruna Henrique	VV	AG-17-31	786431604643029000000
29	Fausto Rebeca	RM	AG-17-32	451548609095133000000
30	Hipolito Eduardo	VV	AG-17-33	451210337971813000000

T A B E L A F U N C I O N Á R I O

ID_FUNCIONARIO	NOME_FUNCIONARIO	PALAVRA_CHAVE
Fo1	Manuel	fo1
Fo2	Fernando	fo2
Fo3	Antonio	fo3
Fo4	Jose	fo4
Fo5	Paulo	fo5
Fo6	Alcino	fo6
Fo7	Alberto	fo7
Fo8	Rui	fo8
Fo9	Olga	fo9
Fo10	Patricia	fo10

T A B E L A M A Q U I N A S

ID_MAQUINA	ID_PISO	LOCALIZACAO	MONTANTE
11	1	Ao lado da porta de saida	500
12	1		500
21	2		500
31	3		500
41	4	Ao lado do lugar 414	500
51	5		500

T A B E L A M O D O _ E N T R A D A

ID_ENTRADA	MODO_ENTRADA
CA	Cartao de Avenca
RM	Reconhecimento de Matricula
VV	Via Verde
CW	Chamada Wireless

T A B E L A M O D O S _ P A G A M E N T O

ID_MODO_PAGAMENTO	MODO_PAGAMENTO
0	Nao especificado
1	Dinheiro
2	Cheque
3	Multibanco

T A B E L A P I S O S

ID_PISO	N_LUGARES	ESTADO
1	50	0
2	50	0
3	50	0
4	50	0
5	50	0

CÓDIGO IMPLEMENTADO

CLASSE BARREIRA

```
public class Barreira {  
    private int id;  
    private boolean estado;  
    // true - aberta  
    // false - fechada  
  
    public Barreira(){ estado = false;}  
    public void abrirBarreira() {estado = true;}  
  
    public void fecharBarreira() {estado = false;}  
  
    @Override  
    public String toString(){  
        if (estado == true)  
            return "Barreira aberta";  
        else return "Barreira fechada";  
    }  
}
```

CLASSE BILHETE

```
public class Bilhete {  
    private int _idBilhete;  
    private String _dataHoraEntrada;  
    private String _dataHoraSaida;  
    private int _id_PortalEntrada;  
  
    public Bilhete (){  
        _idBilhete=0;  
        _dataHoraEntrada=null;  
        _dataHoraSaida=null;  
    }  
  
    public Bilhete(int id, String dataEntrada, String dataSaida){  
        _idBilhete=id;  
        _dataHoraEntrada= dataEntrada;  
        _dataHoraSaida= dataSaida;  
    }  
  
    public int getIdBilhete() {return _idBilhete;}  
  
    public String getHorasSaida() {return _dataHoraSaida;}  
  
    public String getHoraEntrada() {return _dataHoraEntrada;}  
  
    public int getPortalEntrada() {return _id_PortalEntrada;}  
  
    public void setPortalEntrada(int portal){_id_PortalEntrada=portal;}}
```

CLASSE MONITOR

```
public class Monitor {  
  
    private String _informacao;  
  
    public Monitor (){  
        _informacao = "";  
    }  
    public String comunicaErro(String aErro) {  
        return "";  
    }  
  
    public void mostrar(String info) { //getInformacao  
        System.out.println(info);  
    }  
  
    public void alterarInfo(String informacao){  
        _informacao = informacao;  
    }  
  
    public void adicionarInfo(String informacao){  
        _informacao = _informacao + "\n" + informacao;  
    }  
}
```

```

public class Portal {
    private int _id_portal;
    private boolean _estado; // false - fechado true-aberto
    public Barreira _barreira;
    public Monitor _monitor;
    public PainelGlobal _painelGlobal;

    public Portal (int id_portal){
        this._id_portal = id_portal;
        _barreira = new Barreira();
        _monitor = new Monitor();
        _painelGlobal = new PainelGlobal();
        _estado = false;
    }

    public int getID(){return _id_portal;}

    public Barreira getBarreira(){return _barreira;}

    public Monitor getMonitor(){return _monitor;}

    public PainelGlobal getPainelGlobal(){return _painelGlobal;}

    public void chamarFuncionario() {_monitor.mostrar("O funcionário foi contactado");}

    public boolean existeDispositivo(String codDisp) throws SQLException {

        ResultSet rSet = null;
        ArrayList<String> codigos = new ArrayList<String>();
        boolean encontrado = false;
        String sql = "SELECT * FROM CLIENTES WHERE ID_DISPOSITIVO = ''";
        rSet=Model.stmt.executeQuery(sql+codDisp+"'");
        while(rSet.next()){
            String code = rSet.getString(6);
            codigos.add(code);
        }
        for(String cod: codigos){
            if(cod.equals(codDisp)){
                encontrado = true;
            }
        }
        return encontrado;
    }
}

```

CLASSE PORTAENTRADA

```
public class PortalEntrada extends Portal {  
    private int nBilhete=1;  
    public PortalEntrada(int portalID){super(portalID); }  
    public int verificarFormaEntrada() {return 0;}  
  
    public boolean verificaOcupacao() throws SQLException {  
  
        ResultSet rSet = null;  
        ArrayList<String> estadosPisos = new ArrayList<String>();  
        boolean ocupado = true;  
        rSet=Model.stmt.executeQuery("SELECT * FROM PISOS");  
        while(rSet.next()){  
            String estado = rSet.getString(3);  
            estadosPisos.add(estado);  
        }  
        for(String estado: estadosPisos){  
            if(estado.equals("0")){  
                ocupado = false;  
            }  
        }  
        return ocupado;  
    }  
    public void devolveCartaoAvenca() {  
        throw new UnsupportedOperationException();}  
  
    public int leNumClienteAvenca() {  
        throw new UnsupportedOperationException();}  
    }  
    public boolean verificaValidadeAvenca(int aNCliente) {  
        throw new UnsupportedOperationException();}  
    }  
    public boolean verificaPagOrdemAvenca(int aNCliente) {  
        throw new UnsupportedOperationException();}  
    }  
    public Bilhete emiteBilhete() throws SQLException {  
        String dataEntrada = " " ;  
        String dataSaida = "a";  
        Bilhete b = new Bilhete(nBilhete,dataEntrada,dataSaida);  
        BaseDados.registarEntradaCliBilhete(""+nBilhete+"", dataEntra-  
da);  
        nBilhete++;  
        return b;  
    }  
}
```

CLASSE PORTALSAIDA

```
public class Portalsaida {  
  
    public BaseDados _baseDados;  
    public Barreira _barreiraSaida;  
    public Monitor _monitor;  
    public Bilhete leBilhete() {  
        throw new UnsupportedOperationException();  
    }  
    public boolean verificaValidadeBilhete(Bilhete aBilhete) throws SQLException {  
        ResultSet rSet = null;  
        int idBilhete = aBilhete.getIdBilhete();  
        String data = aBilhete.getHoraEntrada();  
        String dataHoraPagamento="", modoPagamento = "";  
        Boolean valido = false;  
        String sql = "SELECT * WHERE ID_BILHETE = '" + idBilhete + "'";  
        sql = sql + " AND DATA_HORA_ENTRADA = '" + data + "'";  
        rSet = Model.stmt.executeQuery(sql);  
        while(rSet.next()) {  
            dataHoraPagamento = rSet.getString(3);  
            modoPagamento = rSet.getString(5);  
        }  
        if (!dataHoraPagamento.equals("") && !modoPagamento.equals("")) {  
            valido = true;  
        }  
        return valido;  
    }  
    public void devolveBilhete() {  
        throw new UnsupportedOperationException();  
    }  
    public boolean verificaPagamentoBilhete(Bilhete aBilhete) {  
        throw new UnsupportedOperationException();  
    }  
    public int verificaFormaSaida() {  
        throw new UnsupportedOperationException();  
    }  
    public int leNumClienteAvenca() {  
        throw new UnsupportedOperationException();  
    }  
    public void verificaValidadeAvenca(int aCliente) {  
        throw new UnsupportedOperationException();  
    }  
    public String escreveSaidaCartaoAvenca() {  
        throw new UnsupportedOperationException();  
    }  
    public void devolveCartaoAvenca() {  
        throw new UnsupportedOperationException();  
    }  
}
```

CLASSE GESTAOCLIENTES

```
public class GestaoClientes {  
  
    public FichaCliente _fichaCliente;  
    public Pagamento _unnamed_Pagamento_;  
    public boolean verificaPagEmAtraso() throws SQLException {  
        boolean existe = false;  
        String resultado = sci.Query.pagamentoEmAtraso(_fichaCliente.getIdCliente());  
        if (resultado.equalsIgnoreCase("") == false ||  
            resultado.equalsIgnoreCase(null)==false) existe = true;  
        return existe;  
    }  
  
    public String comunicaErro(String aErro) {return aErro;}  
  
    public String apresentaListaPagEmAtraso() throws SQLException {  
        String resultado = sci.Query.pagamentoEmAtraso(_fichaCliente.getIdCliente());  
        return resultado;  
    }  
  
    public void pedeIdentificacaoDoCliente() {  
        throw new UnsupportedOperationException();  
    }  
  
    public String leDados() {throw new UnsupportedOperationException();}  
  
    public void mostraFicha(int aIdCliente) { _fichaCliente.toString();}  
  
    public boolean validaDados(String aDadosNovos) {  
        if (_fichaCliente.getIdCliente().equalsIgnoreCase(null)) return false ;  
        if (_fichaCliente.getCodigoEntrada().equalsIgnoreCase(null)) return false;  
        if (_fichaCliente.getMatricula().equalsIgnoreCase(null)) return false;  
        if (_fichaCliente.getNib().equalsIgnoreCase(null)) return false;  
        if (_fichaCliente.getNomeCliente().equalsIgnoreCase(null)) return false;  
        return true;  
    }  
  
    public void actualizaDadosCliente(String aDadosNovos) {}  
  
    public void confirmaSucessoOperacao() {throw new UnsupportedOperationException();}  
  
    public void registaCliente(String aDadosV, String aFPagamento) {  
        throw new UnsupportedOperationException();}  
  
    public void emiteCartaoAvenca() {throw new UnsupportedOperationException();}  
  
    public void removerCliente(int aIdCliente) {throw new UnsupportedOperationException();}  
  
    public float calculaMultaperda() {throw new UnsupportedOperationException();}  
  
    public void emiteRecibo() {}  
}
```

CLASSE FICHA CLIENTE

```
public class Fichacliente {  
  
    private String _idcliente;  
    private String _nomecliente;  
    private String _codigoEntrada;  
    private String _matricula;  
    private String _nib;  
  
    public Fichacliente() {  
        _idcliente      = null;  
        _nomecliente   = null;  
        _codigoEntrada = null;  
        _matricula     = null;  
        _nib           = null;  
    }  
    public Fichacliente(String idcliente, String nomecliente, String codigoEn-  
trada, String matricula, String nib){  
        _idcliente      = idcliente;  
        _nomecliente   = nomecliente;  
        _codigoEntrada = codigoEntrada;  
        _matricula     = matricula;  
        _nib           = nib;  
    }  
    public Fichacliente(Fichacliente fichacliente){  
        _idcliente      = fichacliente.getIdcliente();  
        _nomecliente   = fichacliente.getNomecliente();  
        _codigoEntrada = fichacliente.getCodigoEntrada();  
        _matricula     = fichacliente.getMatricula();  
        _nib           = fichacliente.getNib();  
    }  
  
    public String getIdcliente() {return this._idcliente;}  
    public String getNomecliente() {return this._nomecliente;}  
    public String getCodigoEntrada() {return this._codigoEntrada;}  
    public String getMatricula() {return this._matricula;}  
    public String getNib() {return this._nib;}  
    public void setIdcliente(String idcliente){ _idcliente = idcliente;}  
    public void setNomecliente(String nomecliente){_nomecliente = nomecliente;}  
    public void setCodigoEntrada(String codigoEntrada){_codigoEntrada = codigoEntrada;}  
    public void setMatricula(String matricula){_matricula = matricula;}  
    public void setNib(String nib){_nib = nib;}  
    @Override  
    public String toString(){  
        StringBuilder r = new StringBuilder();  
        r.append("Cliente Registrado\n");  
        r.append("ID: ").append(_idcliente).append("\n");  
        r.append("Nome: ").append(_nomecliente).append("\n");  
        r.append("Codigo Entrada: ").append(_codigoEntrada).append("\n");  
        r.append("Matricula: ").append(_matricula).append("\n");  
        r.append("Nib: ").append(getNib()).append("\n");  
  
        return r.toString();  
    }  
}
```

CLASSE MAQUINAPAGAMENTO

```
public class MaquinaPagamento {

    private int _idMaquina;
    private ArrayList<Recibo> _recibo = new ArrayList<Recibo>();
    public void leBilhete() {return;}
    public static float calculaMulta(Bilhete aBilhete) throws SQLException {
        String idBilhete = ""+aBilhete.getIdBilhete()+"";
        ResultSet rSet = null;
        String sql = "SELECT * FROM BILHETES WHERE ID_BILHETE = '"+idBilhete+"'";
        rSet= Model.stmt.executeQuery(sql);
        String data1="";
        GregorianCalendar dataActual = new GregorianCalendar();
        while(rSet.next()){
            data1=rSet.getString(3);
        }
        GregorianCalendar data = new GregorianCalendar(LerDatas.getAno(data1),
LerDatas.getMes(data1), LerDatas.getDia(data1), LerDatas.getHora(data1),
LerDatas.getMin(data1), LerDatas.getSec(data1));
        float resultado = TabelaPreco.calculaPreco(data, dataActual);
        return resultado;
    }
    public static void mostra(float avalor) {throw new UnsupportedOperationException();}
    public static void recebePagamento(float avalor) {throw new UnsupportedOperationException();}
    public static void registaNovaHoraSaida() {}
    public static void devolveBilhete() {return;}
    public static void comunicaErro(String aErro) { }
    public static void devolveCartaoMagnetico() {return;}
    public static void pedeCodigoCartaoMagnetico() {return;}
    public static float verificaSaldoCartaoMagnetico() {
        float saldo=0;
        return saldo;
    }
    public static void efectuaTransacciaoMultibanco() {return;}
    public static void leDados(String aInput) {}
    public static void comunicaSucesso() {}
    public static float calculaValorPagar(Bilhete aBilhete) {
        GregorianCalendar dataEntrada = new
        GregorianCalendar(LerDatas.getAno(aBilhete.getHoraEntrada()),
LerDatas.getMes(aBilhete.getHoraEntrada()), LerDatas.getDia(aBilhete.getHoraEntrada()),
LerDatas.getHora(aBilhete.getHoraEntrada()), LerDatas.getMin(aBilhete.getHoraEntrada()),
LerDatas.getSec(aBilhete.getHoraEntrada()));
        GregorianCalendar dataSaida = new
        GregorianCalendar(LerDatas.getAno(aBilhete.getHoraSaida()),
LerDatas.getMes(aBilhete.getHoraSaida()), LerDatas.getDia(aBilhete.getHorasaida()),
LerDatas.getHora(aBilhete.getHoraSaida()), LerDatas.getMin(aBilhete.getHorasaida()),
LerDatas.getSec(aBilhete.getHoraSaida())));
        float preco = TabelaPreco.calculaPreco(dataEntrada, dataSaida);
        return preco;
    }
    public static float leValor() {return 0;}
    public static float calculaTroco(float avalorInserido, float avalorPagar){
        float troco = avalorInserido-avvalorPagar;
        return troco;
    }
}
```

CLASSE PAGAMENTOAUTOMATICO

```
public class PagamentoAutomatico extends Pagamento {  
  
    public void processaPagamentos() {  
        return;  
    }  
}
```

CLASSE PAGAMENTOAVENCA

```
public class PagamentoAvenca extends Pagamento {  
  
    public static void registrarPagamento(String id_cliente, String data, String modo, String montante) throws SQLException, Exception {  
        sci.Query.adicionarPagamento(id_cliente, data, modo, montante);  
    }  
}
```

CLASSE RECIBO

```
public class Recibo {  
    private String _idPaque = "PARKUM";  
    private String _nipc = "987654321";  
    private String _idMaquina;  
    private String _nRecibo;  
    private String _nBilhete;  
    private String _horaEntrada;  
    private float _tempoEstacionamento;  
    private float _valorPago;  
    private String _dataActual;  
    private String _horaActual;  
    public MaquinaPagamento _unnamed_MaquinaPagamento_;  
    public void InformacoesRecibo(String aDados) {  
        System.out.println(_idPaque);  
        System.out.println("NIPC: "); System.out.println(_nipc);  
        System.out.println("Máquina automática:");  
        System.out.println(_idMaquina);  
        System.out.println("Recibo Nº: "); System.out.println(_nRecibo);  
        System.out.println("Bilhete Nº: "); System.out.println(_nBilhete);  
        System.out.println("Hora de Entrada: ");  
        System.out.println(_horaEntrada);  
        System.out.println("Tempo de estacionamento: ");  
        System.out.println(_tempoEstacionamento);  
        System.out.println("Valor pago:"); System.out.println(_valorPago);  
        System.out.println("Data: "); System.out.println(_dataActual);  
        System.out.println("Hora: "); System.out.println(_horaActual);  
        System.out.println("IVA incluído à taxa em vigor\n");  
        System.out.println("Recibo processado por computador\n");  
        System.out.println("Contribuinte Nº:\n");  
        System.out.println("Morada:\n");  
    }  
}
```

CLASSE PARKUM-CS

```
public class PARKUM_CS {  
    private PainelGlobal _paineGlobal;  
    private PainelInformacao _panelInfo;  
    private ArrayList<SensorEstacionamento> _sensorEstacionamento = new Ar-  
rayList<SensorEstacionamento>();  
    public int nViaturasNoParque(){  
        int nTotalViaturas=0;  
        for(SensorEstacionamento sensor: _sensorEstacionamento){  
            if(sensor.getEstaOcupado()==true){  
                nTotalViaturas++;}  
        return nTotalViaturas;  
    }  
    public int incrementa() {  
        int nViaturas = nViaturasNoParque();  
        return nViaturas++;  
    }  
    public int decrementa(int aviaturasActuais) {  
        int nViaturas = nViaturasNoParque();  
        return nViaturas--;  
    }  
}
```

CLASSE PAINELGLOBAL

```
public class PainelGlobal {  
  
    private int _nVeiculosTotal;  
    private boolean _estaLivre;  
    public PainelGlobal(){  
        _nVeiculosTotal=0;  
        _estaLivre = true;  
    }  
    public void actualizaInfo(int nVeiculos, boolean estaLivre) {  
        _nVeiculosTotal=nVeiculos;  
        _estaLivre=estaLivre;  
    }  
}
```

CLASSE PAINELINFORMACAO

```
public class PainelInformacao {  
  
    private int _nivel;  
    private int _nVeiculosNivel;  
  
    public PainelInformacao(){  
        _nivel = 1;  
        _nVeiculosNivel=0; }  
    public PainelInformacao(int nivel, int nveiculos){  
        _nivel=nivel;  
        _nVeiculosNivel=nVeiculos; }  
    public void actualizaInfo(int nveiculos) {  
        _nVeiculosNivel=nveiculos;}  
}
```

CLASSE SENSORESTACIONAMENTO

```
public class SensorEstacionamento {  
  
    private String _idSensor;  
    private boolean _estaOcupado;  
    public SensorEstacionamento(){  
        _idSensor = "0";  
        _estaOcupado = false;}  
    public SensorEstacionamento(String idSensor, boolean estaOcu){  
        _idSensor = idSensor;  
        _estaOcupado = estaOcu;}  
    public String getId(){return _idSensor;}  
    public boolean getEstaOcupado(){return _estaOcupado;}  
    public void setID(String id){_idSensor = id; }  
    public void setEstaOcupado(boolean estaOcupado){_estaOcupado=estaOcupado;}  
    public void actualizaSensor() {  
        if(_estaOcupado==true)  
            _estaOcupado = false;  
        else _estaOcupado= true;  
    }  
    public void comunicaEstacionamento(string data) throws SQLException {  
        String sql="";  
        if(_estaOcupado == false){  
            sql= sql + "INSERT INTO REGISTOS_LUGARES  
VALUES ('"+_idSensor+"',TO_DATE('"+data+"','yyyy-mm-dd hh24:mi:ss')),'';"  
        }  
        else {  
            sql = sql + "UPDATE REGISTOS_LUGARES SET DATA_HORA_LIVRE = TO_DA-  
TE('"+data+"','yyyy-mm-dd hh24:mi:ss')";  
        }  
        Model.stmt.executeQuery(sql);  
    }  
}
```

CLASSE FUNCIONARIO

```
public class Funcionario {  
  
    private int _idFuncionario;  
    private String _nomeFunc;  
    public void Funcionario(){  
        _idFuncionario=0;  
        _nomeFunc=null;  
    }  
    public void Funcionario(int idFuncionario, String nomeFuncionario){  
        _idFuncionario=idFuncionario;  
        _nomeFunc=nomeFuncionario;  
    }  
    public int getIdFuncionario(){return _idFuncionario;}  
    public String getNomeFuncionario(){return _nomeFunc;}  
    public void setIdFunc(int idFuncionario){_idFuncionario=idFuncionario;}  
    public void setNomeFunc(String nomeFuncionario){_nomeFunc=nomeFuncionario;}  
}
```

CLASSE LERDATAS

```
public class LerDatas {  
  
    public static int getAno(String data) {  
        String ano = data.substring(0, 4);  
        return Integer.parseInt(ano);}  
    public static int getMes(String data) {  
        String mes = data.substring(5, 7);  
        return Integer.parseInt(mes);}  
    public static int getDia(String data) {  
        String dia = data.substring(8, 10);  
        return Integer.parseInt(dia);}  
    public static int getHora(String data) {  
        String hora = data.substring(11, 13);  
        return Integer.parseInt(hora);}  
    public static int getMin(String data) {  
        String min = data.substring(14, 16);  
        return Integer.parseInt(min);}  
    public static int getSec(String data) {  
        String sec = data.substring(17, 19);  
        return Integer.parseInt(sec);}  
    public static float diferencaEntreDatas(GregorianCalendar data1, GregorianCalendar data2) {  
        long dt1 = data1.getTimeInMillis();  
        long dt2 = data2.getTimeInMillis();  
        float resultado = (float) (((dt2 - dt1) * 0.001)/60);  
        return resultado;}  
}
```

CLASSE LUGAR

```
public class Lugar {  
    private int _idLugar;  
    private boolean _estaLivre;  
    private SensorEstacionamento _sensor;  
    public void Lugar(){  
        _idLugar=0;  
        _estaLivre=true;  
        _sensor =new SensorEstacionamento(""+_idLugar+"",_estaLivre);}  
    public void Lugar(int idLugar, boolean estaLivre){  
        _idLugar=idLugar;  
        _estaLivre=estaLivre;  
        _sensor = new SensorEstacionamento(""+_idLugar+"",_estaLivre);}  
    public int getIdLugar(){return _idLugar; }  
    public boolean getEstaLivre(){return _estaLivre; }  
    public void setIdLugar(int idLugar){_idLugar=idLugar;}  
    public void setEstaLivre(boolean estaLivre){_estaLivre=estaLivre;}  
}
```

CLASSE NIVEL

```
public class Nivel {  
    private int _idNivel;  
    private int _nLugares;  
    private boolean _estaLivre;  
    private PainelInformacao _painelI;  
    public void Nivel() {  
        _idNivel = 1;  
        _nLugares = 0;  
        _estaLivre = true;  
        _painelI = new PainelInformacao(1,0);}  
    public void Nivel(int idNivel,int nLugares,boolean estaLivre){  
        _idNivel=idNivel;  
        _nLugares=nLugares;  
        _estaLivre=estaLivre;  
        _painelI = new PainelInformacao(idNivel, 0);}  
    public int getIdNivel(){return _idNivel;}  
    public int getNLugares(){return _nLugares;}  
    public boolean getEstaLivre(){return _estaLivre;}  
    public void setIdNivel(int idNivel){_idNivel=idNivel;}  
    public void setNLugares(int nLugares){_nLugares=nLugares;}  
    public void setEstaLivre(boolean estaLivre){_estaLivre=estaLivre;}  
}
```

CLASSE TABELAPRECO

```
public class TabelaPreco {  
  
    private static float precoBase;  
  
    public void tabelaPreco(){  
        precoBase= (float) 0.10; /* Preço Base Por cada 10 min*/  
    }  
  
    public float getPrecoBase(){  
        return precoBase;  
    }  
  
    public void setPrecoBase(float pB){  
        precoBase=pB;  
    }  
  
    public static float calculaPreco(GregorianCalendar data1, GregorianCalendar  
data2) {  
        float resultado = LerDatas.diferencaEntreDatas(data1,data2);  
        float preco;  
  
        preco =(resultado/10) * precoBase;  
  
        return preco;  
    }  
}
```

```

public class BaseDados {
    public static void registrarEntradaBilhete(String aIdB, String dataEntrada)
        throws SQLException {
        Model stmt.executeQuery("INSERT INTO BILHETES VALUES '" + aIdB + "','" + dataEntrada + "','" + "'");
    }
    public static void registroEntradaCliRegistado(String identificacao)
        throws SQLException, Exception {
        String sql = "SELECT * FROM CLIENTES WHERE ID_DISPOSITIVO = ''";
        ResultSet rSet = null;
        String idCliente = "";
        GregorianCalendar data = new GregorianCalendar();
        rSet = Model stmt.executeQuery(sql + identificacao + "'");
        while(rSet.next()){
            idCliente = rSet.getString(1);
        }
        String sql1 = "INSERT INTO REGISTOS_REGISTADOS VALUES ('" + idCliente + "','" + data.toString() + "','" + ")";
        Model stmt.executeQuery(sql1);
        Query.commit2();
    }
    public static void registrarPagamentoBilhete(String idBilhete, String dataPagamento, String id_maq, String modoPag, String montante, String recibo)
        throws SQLException{
        String sql = "UPDATE BILHETES SET DATA_HORA_PAGAMENTO = TO_DATE('" + dataPagamento + "','yyyy-mm-dd hh24:mi:ss') WHERE ID_BILHETE = '" + idBilhete + "'";
        Model stmt.executeQuery(sql);
        String sql1 = "INSERT INTO PAGAMENTOS_MAQUINAS
VALUES('" + id_maq + "','" + idBilhete + "','" + montante + "','" + modoPag + "',TO_DATE('" + dataPagamento + "','yyyy-mm-dd hh24:mi:ss'),'" + recibo + "')";
        Model stmt.executeQuery(sql1);
    }
    public static void registrarManutencao(String id_Mquina, String desc, String hora_manutencao, String tempo_Manutencao, String id_funcionario) throws SQLException, Exception{
        Query.adicionarManutencao(id_Mquina, desc, hora_manutencao, tempo_Manutencao, id_funcionario);
    }
    public static boolean existeCliente(String aIdCliente) throws SQLException {
        ResultSet rSet = null;
        String sql = "SELECT * FROM CLIENTES WHERE ID_CLIENTE = ''";
        rSet = Model stmt.executeQuery(sql + aIdCliente + "'");
        boolean existe = false;
        String idCliente;
        while(rSet.next()){
            idCliente = rSet.getString(1);
            if(idCliente.equals(aIdCliente)){
                existe = true;
            }
        }
        return existe;
    }
}

```

```

public static void registrarPagamentoDeMulta(String idBilhete) throws SQLException {
    ResultSet rSet = null;
    String sql = "SELECT * FROM BILHETES WHERE ID_BILHETE = '" + idBilhete+"'";
    rSet=Model.stmt.executeQuery(sql);

    String idBi="", dataEntrada="", dataPagamento="", dataSaida="";
    while(rSet.next()){
        idBi=rSet.getString(1);
        dataEntrada = rSet.getString(2);
        dataPagamento= rSet.getString(3);
        dataSaida = rSet.getString(4);
    }

    int idInt= Integer.parseInt(idBi);
    Bilhete aux = new Bilhete(idInt, dataEntrada,dataSaida);
    GregorianCalendar dataActual = new GregorianCalendar();

    float montante = pagamentos.MaquinaPagamento.calculaMulta(aux);
    String sql1 = "INSERT INTO PAGAMENTOS_MULTAS
VALUES('"+aux.getIdBilhete()+"','"+montante+"',TO_DATE('"+dataActual+"','yyyy-mm-dd hh24:mi:ss'))";
    Model.stmt.executeQuery(sql1);
}

public static void registaSaidacliBilhete(String aIdB, String dataSaida)
throws SQLException{
    String sql = "UPDATE BILHETES SET DATA_HORA_SAIDA = '"+dataSaida+
WHERE ID_BILHETE ='"+aIdB+"'";
    Model.stmt.executeQuery(sql);

}

public static void registaSaidaCliRegistado(String ident, String dataSaida)
throws SQLException {
    String sql = "UPDATE REGISTOS_REGISTADOS SET DATA_HORA_SAIDA = '"+data-
Saida+"' WHERE ID_CLIENTE ='"+ident+"'";
    Model.stmt.executeQuery(sql);
}
}

```

CLASSE GESTAORELATORIOS

```
public class GestaoRelatorios {  
  
    public static String gerarRelatoriosDiarios(int relatorio, int dia, int  
mes, int ano, int niveis) throws SQLException {  
        String rel = null;  
  
        switch (relatorio) {  
            case 1:  
                rel = relatorioDiariototalEntradasSaidas(dia,mes,ano);  
                break;  
            case 2:  
                rel = relatorioDiarioOcupacaoPorNivel(niveis, dia,mes,ano);  
                break;  
            case 3:  
                rel = relatorioDiarioTemposEstacionamento(niveis, dia,mes,ano);  
                break;  
            case 5:  
                rel =  
relatorioDiarioDiferencaPagamentoBilhetesSaidaParque(dia,mes,ano);  
                break;  
            case 6:  
                rel = relatorioDiarioNumeroBilhetesEstraviados(dia, mes, ano);  
                break;  
            case 7:  
                rel =  
relatorioDiarioViaturasQueNaoEntraramNemSairam(dia,mes,ano);  
                break;  
        }  
  
        return rel;  
    }  
  
    public static void apresentaRelatorio(String aRelatorio) {  
        throw new UnsupportedOperationException();  
    }  
  
    public static String relatorioDiariototalEntradasSaidas(int dia, int  
mes, int ano) throws SQLException {  
  
        String rel = "";  
        int entradasBilhetes = 0;  
        int saidasBilhetes = 0;  
        int entradasRegistados = 0;  
        int saidasRegistados = 0;  
        int saidasTotal = 0;  
        int entradasTotal = 0;  
        ResultSet rSet = null;  
  
        String sqlEntradasRegistados = "SELECT * FROM registos_registados WHERE  
TO_CHAR(data_hora_entrada, 'dd') = "+dia;  
        sqlEntradasRegistados = sqlEntradasRegistados + " AND TO_CHAR(data_ho-  
ra_entrada, 'mm') = "+mes;  
        sqlEntradasRegistados = sqlEntradasRegistados + " AND TO_CHAR(data_ho-  
ra_entrada, 'yyyy') = "+ano;  
    }
```

```

System.out.println(sqlEntradasRegistados);
rSet = Model.stmt.executeQuery(sqlEntradasRegistados);
while(rSet.next()){
    entradasRegistados++;
}

String sqlEntradasBilhetes = "SELECT * FROM bilhetes WHERE TO_CHAR(da-
ta_hora_entrada, 'dd') = "+dia;
sqlEntradasBilhetes = sqlEntradasBilhetes + " AND TO_CHAR(data_hora_en-
trada, 'mm') = "+mes;
sqlEntradasBilhetes = sqlEntradasBilhetes + " AND TO_CHAR(data_hora_en-
trada, 'yyyy') = "+ano;
System.out.println(sqlEntradasBilhetes);
rSet = Model.stmt.executeQuery(sqlEntradasBilhetes);
while(rSet.next()){
    entradasBilhetes++;
}

String sqlSaidasRegistados = "SELECT * FROM registos_registados WHERE
TO_CHAR(data_hora_saida, 'dd') = "+dia;
sqlSaidasRegistados = sqlSaidasRegistados + " AND TO_CHAR(data_ho-
ra_saida, 'mm') = "+mes;
sqlSaidasRegistados = sqlSaidasRegistados + " AND TO_CHAR(data_ho-
ra_saida, 'yyyy') = "+ano;
System.out.println(sqlSaidasRegistados);
rSet = Model.stmt.executeQuery(sqlSaidasRegistados);
while(rSet.next()){
    saidasRegistados++;
}

String sqlSaidasBilhetes = "SELECT * FROM bilhetes WHERE TO_CHAR(da-
ta_hora_saida, 'dd') = "+dia;
sqlSaidasBilhetes = sqlSaidasBilhetes + " AND TO_CHAR(data_hora_saida,
'mm') = "+mes;
sqlSaidasBilhetes = sqlSaidasBilhetes + " AND TO_CHAR(data_hora_saida,
'yyyy') = "+ano;
System.out.println(sqlSaidasBilhetes);
rSet = Model.stmt.executeQuery(sqlSaidasBilhetes);
while(rSet.next()){
    saidasBilhetes++;
}

entradasTotal = entradasBilhetes+entradasRegistados;
saidasTotal = saidasBilhetes+saidasRegistados;

rel = rel + "RELATORIO DIARIO DO DIA "+dia+"-"+mes+"-"+ano+"\n";
rel = rel + "*****\n";
rel = rel + "Total de entradas de clientes registados: "+entradasRegis-
tados+"\n";
rel = rel + "Total de entradas de clientes com bilhetes: "+entradasBi-
lhetes+"\n";
rel = rel + "Total de saidas de clientes redistados: "+saidasRegista-
dos+"\n";
rel = rel + "Total de saidas de clientes com bilhetes: "+saidasBilhe-
tes+"\n";
rel = rel + "*****\n";
rel = rel + "Total de entradas: "+entradasTotal+"\n";

```

```

        rel = rel + "Total de saidas: "+saidasTotal+"\n";

        return rel;
    }

    public static String relatorioDiarioOcupacaoPorNivel(int niveis, int
dia, int mes, int ano) throws SQLException{
        String rel      = "";
        ResultSet rSet   = null;
        int totalNivel = 0;

        int i = 1;
        rel = rel + "RELATORIO DIARIO DO DIA "+dia+"-"+mes+"-"+ano+" POR NI-
VEL\n";
        rel = rel + "*****\n";

        while(i<=niveis){
            totalNivel = 0;
            String sql = "SELECT * FROM lugares, registros_lugares, pisos WHERE
lugares.id_piso = "+i;
            sql = sql + " AND TO_CHAR(registros_lugares.data_hora_ocupado, 'dd') "
= "+dia;
            sql = sql + " AND TO_CHAR(registros_lugares.data_hora_ocupado, 'mm') "
= "+mes;
            sql = sql + " AND TO_CHAR(registros_lugares.data_hora_ocupado,
'yyyy') = "+ano;
            sql = sql + " AND pisos.id_piso = lugares.id_piso";
            sql = sql + " AND registros_lugares.id_lugar = lugares.id_lugar";

            System.out.println(sql);
            rSet = Model.stmt.executeQuery(sql);
            while(rSet.next()){
                totalNivel++;
            }
            rel = rel + "Total de clientes que estacionaram no nivel "+i+":
"+totalNivel+"\n";
            i++;
        }
        rel = rel + "*****\n";
        System.out.println(rel);
        return rel;
    }

    private static String relatorioDiarioTemposEstacionamento(int niveis,
int dia, int mes, int ano) throws SQLException {
        String rel      = "";
        ResultSet rset   = null;
        float tempoMax      = 0;
        float tempoMin      = 0;
        float tempoMedio     = 0;
        float tempoTotal     = 0;
        int numeroEstacionamentos = 0;
        String data_inicio   = "";
        String data_fim       = "";
        GregorianCalendar dataInicio = new GregorianCalendar();
        GregorianCalendar dataFim  = new GregorianCalendar();

```

```

float diferenca = 0;

int i = 1;
rel = rel + "RELATORIO DIARIO DOS TEMPOS DE ESTACIONAMENTO DO DIA
"+dia+"-"+mes+"-"+ano+" POR NIVEL\n";
rel = rel +
"*****\n";
;

while(i<=niveis){
    tempoTotal = 0;
    tempoMax = 0;
    tempoMin = 10000;
    tempoMedio = 0;
    numeroEstacionamentos = 0;
    String sql = "SELECT to_char(data_hora_ocupado, 'yyyy-mm-dd
hh24:mi:ss'), to_char(data_hora_livre, 'yyyy-mm-dd hh24:mi:ss')";
    sql = sql + " FROM registos_lugares, lugares ";
    sql = sql + " WHERE to_char(data_hora_ocupado, 'dd') = "+dia;
    sql = sql + " AND to_char(data_hora_ocupado, 'mm') = "+mes;
    sql = sql + " AND to_char(data_hora_ocupado, 'yyyy') = "+ano;
    sql = sql + " AND to_char(data_hora_livre, 'dd') > 0 ";
    sql = sql + " AND registos_lugares.id_lugar = lugares.id_lugar ";
    sql = sql + " AND lugares.id_piso = "+i;
    System.out.println(sql);

    rSet = Model.stmt.executeQuery(sql);

    while(rSet.next()){
        data_inicio = rSet.getString(1);
        data_fim = rSet.getString(2);

        dataInicio.set(LerDatas.getAno(data_inicio),
LerDatas.getMes(data_inicio) , LerDatas.getDia(data_inicio),
LerDatas.getHora(data_inicio), LerDatas.getMin(data_inicio),
LerDatas.getSec(data_inicio));
        dataFim.set(LerDatas.getAno(data_fim),
LerDatas.getMes(data_fim) , LerDatas.getDia(data_fim),
LerDatas.getHora(data_fim), LerDatas.getMin(data_fim),
LerDatas.getSec(data_fim));

        diferenca = LerDatas.diferencaEntreDatas(dataInicio, dataFim);
        tempoTotal = tempoTotal + diferenca;
        numeroEstacionamentos++;
        if (diferenca>tempoMax) tempoMax = diferenca;
        if (diferenca<tempoMin && diferenca >0 ) tempoMin = diferenca;
        System.out.println("max "+tempoMax);
        System.out.println("min "+tempoMin);
        System.out.println("dif "+diferenca);

    }
    if(tempoMin==10000) tempoMin=0;
    if(numeroEstacionamentos>0) tempoMedio = tempoTotal/numeroEstaciona-
mentos;
    System.out.println("med "+tempoMedio);
    rel = rel + "Piso "+i+"\n";
    rel = rel + "Tempo medio de estacionamento: "+tempoMedio+"\n";
}

```

```

        rel = rel + "Tempo máximo de estacionamento: "+tempoMax+"\n";
        rel = rel + "Tempo minímo de estacionamento: "+tempoMin+"\n";
        rel = rel + "\n";
        i++;
    }
    return rel;
}

public static String relatorioDiarioNumeroBilhetesEstraviados(int dia, int
mes, int ano) throws SQLException{
/*
String sql = "SELECT * FROM BILHETES WHERE BILHETES.ID_BILHETE NOT IN
(";
sql = sql + " SELECT ID_BILHETE FROM BILHETES";
sql = sql + " WHERE TO_CHAR(DATA_HORA_SAIDA, 'dd') = '"+dia+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_SAIDA, 'mm') = '"+mes+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_SAIDA, 'yyyy') = '"+ano+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_ENTRADA, 'dd') = '"+dia+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_ENTRADA, 'mm') = '"+mes+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_ENTRADA, 'yyyy') = '"+ano+"'")";
*/
String sql = "";
sql = sql + "SELECT ID_BILHETE FROM BILHETES WHERE TO_CHAR(DATA_HO-
RA_ENTRADA, 'dd') = '"+dia+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_ENTRADA, 'mm') = '"+mes+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_ENTRADA, 'yyyy') = '"+ano+"'";
sql = sql + " AND BILHETES.ID_BILHETE NOT IN (";
sql = sql + " SELECT ID_BILHETE FROM BILHETES";
sql = sql + " WHERE TO_CHAR(DATA_HORA_SAIDA, 'dd') = '"+dia+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_SAIDA, 'mm') = '"+mes+"'";
sql = sql + " AND TO_CHAR(DATA_HORA_SAIDA, 'yyyy') = '"+ano+"'")";

String rel = "";
ResultSet rSet = null;

rSet = Model.stmt.executeQuery(sql);

rel = rel + "NUMERO BILHETES ESTRAVIADOS\n";
rel = rel + "*****\n";

int nBilhetes = 0;

while(rSet.next()){
    nBilhetes++;
}

rel = rel + "Foram estraviados: " + nBilhetes + "\n";
rel = rel + "*****";
return rel;
}

```

```

private static String
relatorioDiarioDiferencaPagamentoBilhetesSaidaParque(int dia, int mes, int ano)
throws SQLException {
    String rel = "";
    int numeroBilhetes = 0;
    float diferenca = 0;
    float tempoMax = 0;
    float tempoMin = 10000;
    float tempoMedio = 0;
    float tempoTotal = 0;
    String data_pagamento = "";
    String data_saida = "";
    GregorianCalendar dataPagamento = new GregorianCalendar();
    GregorianCalendar dataSaida = new GregorianCalendar();
    ResultSet rSet = null;

    String sql = "SELECT to_char(data_hora_pagamento, 'yyyy-mm-dd hh24:mi:ss'), to_char(data_hora_saida, 'yyyy-mm-dd hh24:mi:ss') FROM bilhetes";
    rSet = Model.stmt.executeQuery(sql);

    rel = rel + "RELATORIO DIARIO DOS TEMPOS ENTRE O PAGAMENTO E SADIDA DOS BILHETES\n";
    rel = rel + "DIA "+dia+"-"+mes+"-"+ano+" POR NIVEL\n";
    rel = rel +
"*****\n*****\n";
    while(rSet.next()){
        data_pagamento = rSet.getString(1);
        data_saida = rSet.getString(2);

        dataPagamento.set(LerDatas.getAno(data_pagamento),
LerDatas.getMes(data_pagamento) , LerDatas.getDia(data_pagamento),
LerDatas.getHora(data_pagamento), LerDatas.getMin(data_pagamento),
LerDatas.getSec(data_pagamento));
        dataSaida.set(LerDatas.getAno(data_saida),
LerDatas.getMes(data_saida) , LerDatas.getDia(data_saida),
LerDatas.getHora(data_saida), LerDatas.getMin(data_saida),
LerDatas.getSec(data_saida));

        diferenca = LerDatas.diferencaEntreDatas(dataPagamento, dataSaida);
        tempoTotal = tempoTotal + diferenca;
        numeroBilhetes++;

        if (diferenca>tempoMax) tempoMax = diferenca;
        if (diferenca<tempoMin && diferenca >0 ) tempoMin = diferenca;
        System.out.println("max "+tempoMax);
        System.out.println("min "+tempoMin);
        System.out.println("dif "+diferenca);
    }

    if(tempoMin==10000) tempoMin=0;
    if(numeroBilhetes>0) tempoMedio = tempoTotal/numeroBilhetes;
    System.out.println("med "+tempoMedio);

    rel = rel + "Tempo medio: "+tempoMedio+"\n";
}

```

```

        rel = rel + "Tempo máximo: "+tempoMax+"\n";
        rel = rel + "Tempo mínimo: "+tempoMin+"\n";
        rel = rel + "Número de bilhetes: "+numeroBilhetes+"\n";

        rel = rel +
"*****\n*****\n";
        return rel;
    }

    private static String relatorioDiarioViaturasQueNaoEntraramNemSairam(int dia, int mes, int ano) throws SQLException {
        String rel = "";
        int totalClientes = 0;

        ResultSet rSet = null;

        String sql = "SELECT * FROM clientes WHERE clientes.id_cliente NOT IN
(";
        sql = sql + " SELECT registos_registrados.id_cliente FROM registos_re-
gistrados";
        sql = sql + " WHERE to_char(registos_registrados.data_hora_entrada,
'dd') = "+dia;
        sql = sql + " AND to_char(registos_registrados.data_hora_entrada, 'mm')
= "+mes;
        sql = sql + " AND to_char(registos_registrados.data_hora_entrada,
'yyyy') = "+ano;
        sql = sql + " GROUP BY registos_registrados.id_cliente)";

        rel = rel + "RELATORIO DIARIO DOS CLIENTES QUE NAO ESTIVERAM NO PAR-
QUE\n";
        rel = rel + "DIA "+dia+"-"+mes+"-"+ano+" POR NIVEL\n";
        rel = rel +
"*****\n*****\n";
        rSet = Model.stmt.executeQuery(sql);
        while(rSet.next()){
            rel = rel + rSet.getString(1) + " " + rSet.getString(2) +"\n";
            totalClientes++;
        }

        rel = rel +
"*****\n*****\n";
        rel = rel + "TOTAL DE CLIENTES: "+totalClientes;
        return rel;
    }
}

```

```

public static String gerarRelatoriosMaqPagamento(int relatorio, String idMaq) throws SQLException {
    String rel = null;

    switch (relatorio) {
        case 1:
            rel = relatorioMaqPagamentoNumeroPagamentoTipo(idMaq);
            break;
        case 2:
            rel =
relatorioMaqPagamentoTotalRecebidoPorCadaPagamento(idMaq);
            break;
        case 3:
            rel = relatorioMaqPagamentoNumeroPercentagemRecibos(idMaq);
            break;
        case 4:
            rel =
relatorioMaqPagamentoNumAvariasOperaçõesManutencao(idMaq);
            break;
    }

    return rel;
}

private static String relatorioMaqPagamentoNumeroPagamentoTipo(String idMaq) throws SQLException {
    String rel = "";
    String sql = "SELECT * FROM pagamentos_maquinas WHERE id_maquina = ?";
    int numDinheiro = 0;
    int numMultibanco = 0;
    int totalPagamentos = 0;
    float montanteTotal = 0;
    float montanteDinehiro = 0;
    float montanteMultibanco = 0;

    rel = rel + "NUMERO PAGAMENTOS POR TIPO DA MÁQUINA "+idMaq+"\n";
    rel = rel + "*****\n";

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + idMaq + "");

    while(rSet.next()){
        if (rSet.getString(4).equalsIgnoreCase("1")){
            numDinheiro++;
            montanteDinehiro = montanteDinehiro +
Float.parseFloat(rSet.getString(3));
        }

        if (rSet.getString(4).equalsIgnoreCase("3")){
            numMultibanco++;
            montanteMultibanco = montanteMultibanco +
Float.parseFloat(rset.getString(3));
        }
    }
}

```

```

        totalPagamentos++;
        montanteTotal = montanteTotal +
Float.parseFloat(rset.getString(3));
    }

    rel = rel + "TOTAL PAGAMENTOS EM DINHEIRO: "+numDinheiro+"\n";
    rel = rel + "TOTAL PAGAMENTOS POR MULTIBACO: "+numMultibanco+"\n";
    rel = rel + "TOTAL PAGAMENTOS: "+totalPagamentos+"\n";
    rel = rel + "*****\n";
    /*
     rel = rel + "MONTANTE TOTAL RECEBIDO EM DINHEIRO: "+montanteDinehi-
ro+"€\n";
     rel = rel + "MONTANTE TOTAL RECEBIDO POR MULTIBANCO: "+montanteMulti-
banco+"€\n";
     rel = rel + "MONTANTE TOTAL RECEBIDO: "+montanteTotal+"€\n";
     rel = rel + "*****\n";
*/
}

return rel;
}

private static String
relatorioMaqPagamentoTotalRecebidoPorCadaPagamento(String idMaq) throws SQLException{
    String rel          = "";
    String sql          = "SELECT * FROM pagamentos_maquinas WHERE
id_maquina = ?";
    float montanteTotal = 0;
    float montanteDinehiro = 0;
    float montanteMultibanco= 0;
    float percentagemDinhe = 0;
    float percentagemMulti = 0;

    rel = rel + "NUMERO PAGAMENTOS POR TIPO DA MÁQUINA "+idMaq+"\n";
    rel = rel + "*****\n";

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + idMaq + "'");

    while(rSet.next()){
        if (rSet.getString(4).equalsIgnoreCase("1")) {
            montanteDinehiro = montanteDinehiro +
Float.parseFloat(rset.getString(3));
        }

        if (rSet.getString(4).equalsIgnoreCase("3")) {
            montanteMultibanco = montanteMultibanco +
Float.parseFloat(rset.getString(3));
        }

        montanteTotal = montanteTotal +
Float.parseFloat(rset.getString(3));
    }
    if (montanteTotal>0) percentagemDinhe = (montanteDinehiro/montanteTo-
tal)*100;
    if (montanteTotal>0) percentagemMulti = (montanteMultibanco/montanteTo-
tal)*100;
}

```

```

        rel = rel + "MONTANTE TOTAL RECEBIDO EM DINHEIRO: "+montanteDinehiro+"€\n";
        rel = rel + "MONTANTE TOTAL RECEBIDO POR MULTIBANCO: "+montanteMulti-
banco+"€ ("+percentagemMulti+"%)\n";
        rel = rel + "MONTANTE TOTAL RECEBIDO: "+montanteTotal+"€\n";
        rel = rel + "*****\n";

        return rel;
    }

    public static String relatorioMaqPagamentoNumeroPercentagemRecibos
(String idMaq) throws SQLException{
    String rel            = "";
    String sql             = "SELECT * FROM pagamentos_maquinas WHERE
id_maquina = ?";
    int numRecibos         = 0;
    int totalPagamentos   = 0;
    float percentagemRecibo = 0;

    rel = rel + "NUMERO/PERCENTAGEM RECIBOS EMITOS DA MÁQUINA "+idMaq+"\n";
    rel = rel +
"*****\n";

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + idMaq + "'");

    while(rSet.next()){
        if (rSet.getString(6).equalsIgnoreCase("1")) numRecibos++;
        totalPagamentos++;
    }

    if (totalPagamentos > 0) percentagemRecibo = ((float) numRecibos/to-
talPagamentos)*100;

    System.out.println("totalPagamento :" +totalPagamentos);
    System.out.println("numRecibos: " +numRecibos);
    System.out.println("%: " +percentagemRecibo);

    rel = rel + "TOTAL DE RECIBOS: "+numRecibos+"\n";
    rel = rel + "PERCENTAGEM RECIBOS: "+percentagemRecibo+" %\n";
    rel = rel + "TOTAL DE PAGAMENTOS: "+totalPagamentos+"\n";
    rel = rel +
"*****\n";

    return rel;
}

```

```

public static String
relatorioMaqPagamentoNumAvariasOperaçõesManutencao(String idMaq) throws SQLEx-
ception {
    String rel          = "";
    String sql          = "SELECT * FROM registos_manutencao WHERE
id_maquina = '"+";
    int numManutencoes = 0;

    rel = rel + "NUMERO DE AVARIAS E OPERACOES DE MANUTENCAO DA MAQUINA
"+idMaq+"\n";
    rel = rel +
"*****\n";

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + idMaq + "'");

    while (rSet.next()){
        numManutencoes++;
    }

    rel = rel + "TOTAL: "+numManutencoes+"\n";
    rel = rel +
"*****\n";

    return rel;
}

public static String gerarRelatoriosCliente(int relatorio, String idClien-
te) throws SQLException {
    String rel = null;

    switch (relatorio) {
        case 1:
            rel = relatorioNumeroEntradas(idCliente);
            break;
        case 2:
            rel = relatorioTemposMediosEstacionamento(idCliente);
            break;
        case 3:
            //      rel = relatorioValorMensalPagar(idCliente);
            break;
    }

    return rel;
}

```

CLASSE MODEL

```
public class Model extends Observable {  
  
    public static Statement stmt;  
  
    public static void connect() throws ClassNotFoundException, SQLException {  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        String url = "jdbc:oracle:thin:@localhost:1521:xe";  
        Connection conn = DriverManager.getConnection(url, "GRUPO1", "grupo1");  
        conn.setAutoCommit(false);  
        stmt = conn.createStatement();  
    }  
  
    public static void disconnect() throws SQLException {  
        stmt.close();  
    }  
}
```

CLASSE QUERYS

```
public class Query extends Model {  
  
    /*****  
     * Fazer commit da Base de Dados  
     *****/  
    public static void commit2() throws Exception {  
        Model(stmt.executeQuery("commit"));  
    }  
  
    /**  
     * Lista todos os clientes  
     * @return ResultSet  
     * @throws SQLException  
     */  
    public static ResultSet queryClientes () throws SQLException{  
        return Model(stmt.executeQuery("SELECT * FROM clientes"));  
    }  
  
    public static ResultSet queryPisos () throws SQLException{  
        return Model(stmt.executeQuery("SELECT * FROM pisos"));  
    }  
  
    public static ResultSet queryModosEntrada() throws SQLException{  
        return Model(stmt.executeQuery("SELECT * FROM modos_entrada"));  
    }  
  
    public static ResultSet queryModosPagamento() throws SQLException{  
        return Model(stmt.executeQuery("SELECT * FROM modos_pagamentos"));  
    }  
}
```

```

public static ResultSet procuraClientePorID(String sel) throws SQLException{
    String sql = "SELECT * FROM clientes WHERE id_cliente ='";
    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + sel + "'");
    return rSet;
}

public static ResultSet filtraLista(String sel) throws SQLException{
    String sql = "SELECT * FROM clientes WHERE id_entrada=''";
    ResultSet rSet = null;
    rSet=Model.stmt.executeQuery(sql+sel+"'");
    return rSet;
}

public static ResultSet procuraClientePorNome(String sel) throws SQLException{
    String sql = "SELECT * FROM clientes WHERE nome ='";
    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + sel + "'");
    return rSet;
}
public static String procuraModoEntradaPorID (String sel) throws SQLException{
    String sql = "SELECT * FROM modos_entrada WHERE id_entrada ='";
    String r = "";
    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + sel + "'");
    while(rSet.next()){
        r = rSet.getString(2);
    }
    return r;
}

public static String procuraModoPagamentoPorNome(String sel) throws SQLException{
    String sql = "SELECT * FROM modos_pagamentos WHERE MODO_PAGAMENTO ='";
    String r = "";
    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + sel + "'");
    while(rSet.next()){
        r= rSet.getString(1);
    }
    return r;
}

```

```

    public static String procuraModoEntradaPorNome (String sel) throws
SQLException {
    String sql = "SELECT * FROM modos_entrada WHERE modo_entrada ='";
    String r   = "";

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql + sel + "'");

    while(rSet.next()){
        r = rSet.getString(1);
    }
    return r;
}

public static int totalPisos() throws SQLException{
    String sql = "SELECT * FROM pisos";
    int numPisos = 0;

    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql);
    while(rSet.next()){
        numPisos++;
    }
    System.out.println("Total de Pisos: "+numPisos);
    return numPisos;
}

public static void adicionarCliente (String id_cliente, String nome_cien-
te, String modo_pagamento, String matricula, String nib) throws SQLException,
Exception{

    String sql = "INSERT INTO clientes VALUES
('"+id_cliente+"','"+nome_cliente+"','"+modo_pagamento+"','"+matricula+"','"+ni-
b+"')";

    System.out.println(sql);

    Model.stmt.executeQuery(sql);
    commit2();
}

public static void adicionarManutencao (String id_Maquina, String desc,s-
tring hora_manutencao, String tempo_Manutencao, String id_funcionario) throws
SQLException, Exception{

    String sql = "INSERT INTO REGISTOS_MANUTENCAO VALUES ('"+id_Maqui-
na+"','"+desc+"', to_date('"+hora_manutencao+"', 'yyyy-mm-dd
hh24:mi:ss'), '"+tempo_Manutencao+"','"+id_funcionario+"')";

    System.out.println(sql);
    Model.stmt.executeQuery(sql);
    commit2();
}

public static void adicionarPagamento (String id_cliente, String data,
String modo, String montante) throws SQLException, Exception{

```

```

        String sql = "INSERT INTO PAGAMENTOS VALUES ('"+id_cliente+"', to_da-
te('"+data+"', 'yyyy-mm-dd hh24:mi:ss'), '"+ procuraModoPagamentoPorNome(modo)
+"', '"+montante+"')";
        System.out.println(sql);
        Model.stmt.executeQuery(sql);
        commit2();
    }

    public static void alterarNomeCliente(String idCliente, String novoNome)
throws SQLException, Exception{
    String sql = "UPDATE clientes SET nome = '"+novoNome+"' WHERE id_cliente =
"+idCliente+"";
    Model.stmt.executeQuery(sql);
    commit2();
}

    public static void alterarModo(String idCliente, String novoModo) throws
SQLException, Exception{
    String sql = "UPDATE clientes SET id_entrada = '"+procuraModoEntrada-
PorNome(novoModo)+"' WHERE id_cliente = '"+idCliente+"'";
    Model.stmt.executeQuery(sql);
    commit2();
}

    public static void alterarMatricula(String idCliente, String novaMatricula)
throws SQLException, Exception{
    String sql = "UPDATE clientes SET matricula = '"+novaMatricula+"' WHERE
id_cliente = '"+idCliente+"'";
    Model.stmt.executeQuery(sql);
    commit2();
}

    public static void alterarNIB(String idCliente, String novoNIB) throws
SQLException, Exception{
    String sql = "UPDATE clientes SET nib = '"+novoNIB+"' WHERE id_cliente =
"+idCliente+"";
    Model.stmt.executeQuery(sql);
    commit2();
}

    public static ResultSet listarPagamentosCliente (String idCliente) throws
SQLException{
    String sql = "SELECT * FROM pagamentos WHERE pagamentos.id_cliente =
"";";
    ResultSet rSet = null;
    rSet = Model.stmt.executeQuery(sql +idCliente + "''");
    return rSet;
}

    public static GregorianCalendar listarUltimoPagamento (String idCliente)
throws SQLException{
    String sql = "SELECT * FROM pagamentos WHERE pagamentos.id_cliente =
"";";
    GregorianCalendar dataUltimoPagamento = new GregorianCalendar();
    GregorianCalendar dataTemp = new GregorianCalendar();
    String datas = "";
    float diferenca = 0;
}

```

```

ResultSet rSet = null;

rSet = Model.stmt.executeQuery(sql +idCliente + "");

while(rSet.next()){
    datas = rSet.getString(2);
    dataTemp.set(LerDatas.getAno(datas), LerDatas.getMes(datas),
LerDatas.getDia(datas), LerDatas.getHora(datas), LerDatas.getMin(datas),
LerDatas.getSec(datas));
    diferenca = LerDatas.diferencaEntreDatas(dataUltimoPagamento, data-
Temp);
    if (diferenca > 0) dataUltimoPagamento = dataTemp;
}
return dataUltimoPagamento;
}

public static GregorianCalendar listarUltimaEntradaParque (String idClien-
te) throws SQLException {
    String sql = "SELECT * FROM pagamentos WHERE pagamentos.id_cliente =
"";
    GregorianCalendar dataUltimoEntrada      = new GregorianCalendar();
    GregorianCalendar dataTemp                = new GregorianCalendar();
    String datas                      = "";
    float diferenca                  = 0;
    ResultSet rSet = null;

    rSet = Model.stmt.executeQuery(sql +idCliente + "");

    while(rSet.next()){
        datas = rSet.getString(2);
        dataTemp.set(LerDatas.getAno(datas), LerDatas.getMes(datas),
LerDatas.getDia(datas), LerDatas.getHora(datas), LerDatas.getMin(datas),
LerDatas.getSec(datas));
        diferenca = LerDatas.diferencaEntreDatas(dataUltimoEntrada, data-
Temp);
        if (diferenca > 0) dataUltimoEntrada = dataTemp;
    }

    return dataUltimoEntrada;
}

public static String pagamentoEmAtraso (String idcliente) throws SQLExcep-
tion{
    GregorianCalendar dataUltimoPagamento = new GregorianCalendar();
    GregorianCalendar dataUltimaEntrada   = new GregorianCalendar();
    GregorianCalendar dataPagar          = new GregorianCalendar();
    String resultado                     = "";
    float diferenca                      = 0;

    dataUltimaEntrada   = listarUltimaEntradaParque(idCliente);
    dataUltimoPagamento = listarUltimoPagamento(idCliente);

    LerDatas.diferencaEntreDatas(dataUltimoPagamento, dataUltimaEntrada);
    if (diferenca > 0) {
        dataPagar = dataUltimaEntrada;
        resultado = dataUltimaEntrada.getTime().toString();
    }
}

```

```
        return resultado;
    }

    public static void registrarEntradaBilhete (String idBilhete){
        String sql = "";
    }
}
```

DETECÇÃO DE PROBLEMAS

Durante a realização do projecto, foram detectadas alguns dificuldades que não conseguimos superar. É caso da implementação da Simulação da dinâmica do parque, que ficou por concretizar. Também não foi concluída a obtenção do relatório dos tempos entre a entrada e o efectivo estacionamento e a obtenção do valor a pagar por cada cliente registado no parque.

Outra dificuldade encontrada foi o facto de as data nem sempre se apresentassem no mesmo formato, o que provocou alguns conflitos. Em particular, detectamos problemas no cálculo do número de bilhetes extraviados e na selecção dos clientes com pagamentos em atraso, e portanto, esses métodos não funcionam correctamente.

Como trabalhos futuros, gostaríamos de corrigir os erros detectados após concluirmos o projecto, e terminar as tarefas que ficaram por concluir, como especial atenção para a implementação da simulação do sistema, que nos pareceu uma tarefa muito interessante.

CONCLUSÃO

A realização deste projecto revelou-se fundamental para consolidar as competências adquiridas nesta unidade curricular. Deste modo, desenvolvemos a capacidade de análise e compreensão dos requisitos que a aplicação exigia, o que permitiu a criação de diagramas UML mais consistentes e visualmente perceptíveis.

A concretização do projecto deu-nos, ainda, a oportunidade de juntar conceitos de base de dados ao paradigma da programação orientada aos objectos, o que faz com que a aplicação se torne num sistema mais próximo de um parque de estacionamento real.

Para além disso, este trabalho proporcionou-nos adquirir competências de análise de software mais rigorosas e detalhadas, que nos ajudaram a ter uma maior percepção do problema em causa.

Relativamente ao balanço geral do projecto verificamos que os objectivos primordiais foram desenvolvidos com sucesso, e por isso este trabalho relevou-se bastante produtivo e interessante.

MINI-MANUAL DE UTILIZAÇÃO DA APLICAÇÃO

PARKUM

Simulação de controlo e Gestão de um Parque de Estacionamento

Ana Isabel dos Anjos Sampaio nº 54740

Miguel Pinto da Costa nº 54746

Hugo Emanuel da Costa Frade nº 54750

Vanessa Catarina de Seabra Campos nº 54801

Andreia Patrícia Matias da Silva nº 56837

2 de Janeiro de 2011

Este mini-manual de utilização pretende dar a conhecer a aplicação desenvolvida, de forma a facilitar a sua utilização.

INTERFACE PARKUM

Esta interface corresponde ao sistema que idealmente seria instalado no parque de estacionamento. Foi concebida para ser utilizada pelo administrador e pelos funcionários do parque.

Esta interface está dividida em cinco funcionalidades como podemos ver nas páginas que se seguem.

GESTÃO DE CLIENTES

É nesta secção que o administrador do parque opera sobre tudo o que envolve a gestão de clientes registados.

Deste modo, é possível visualizar os clientes registados, tanto por número de cliente do parque como por nome, como se pode observar as figuras seguintes.

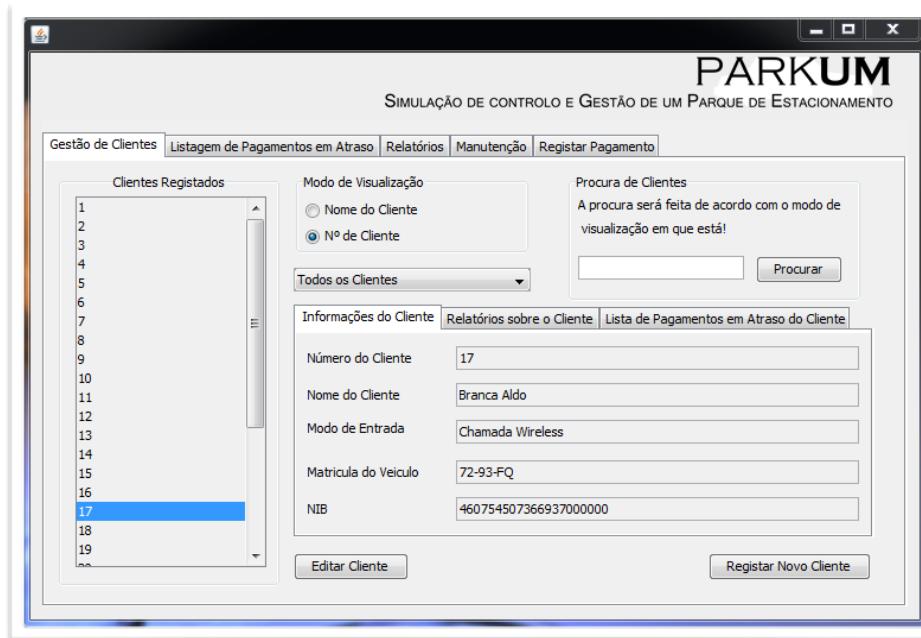


Fig 1: LISTAGEM DE CLIENTE POR NÚMERO DE CLIENTE DO PARQUE

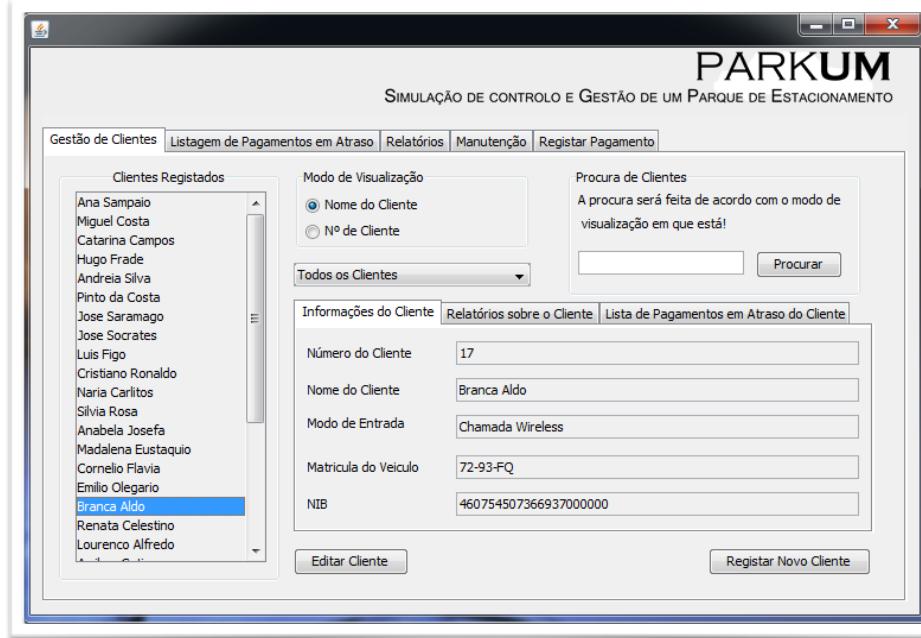


Fig 2: LISTAGEM DE CLIENTE POR NOME DO CLIENTE

Selecionando o cliente pretendido, é possível ter acesso à sua informação. Esta informação inclui, para além dos seus dados pessoais, os relatórios mensais sobre a utilização do parque e, no caso de possuir, a lista de pagamentos em atraso.

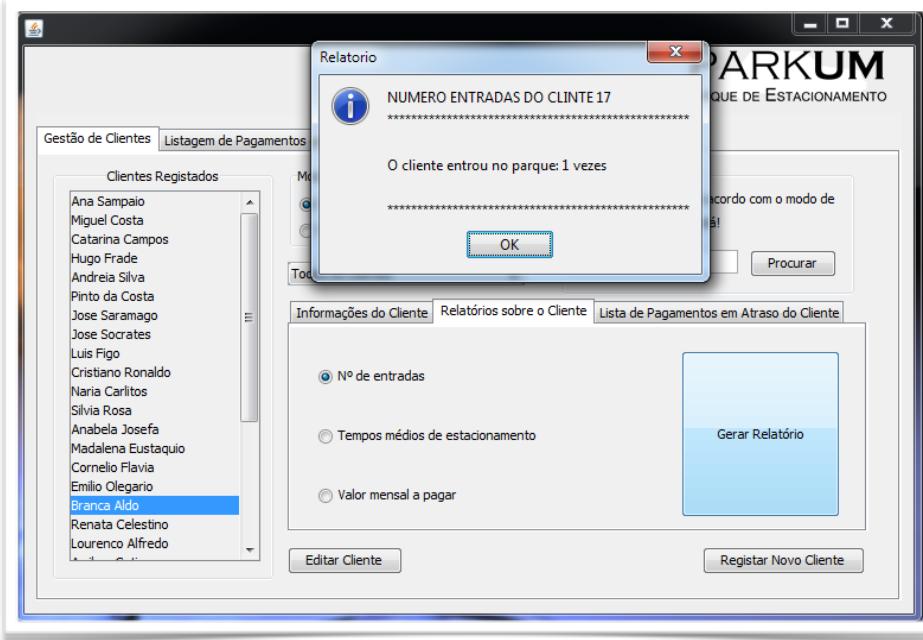


Fig 3: OBTENÇÃO DO RELATÓRIO DO NÚMERO DE ENTRADAS DE UM CLIENTE

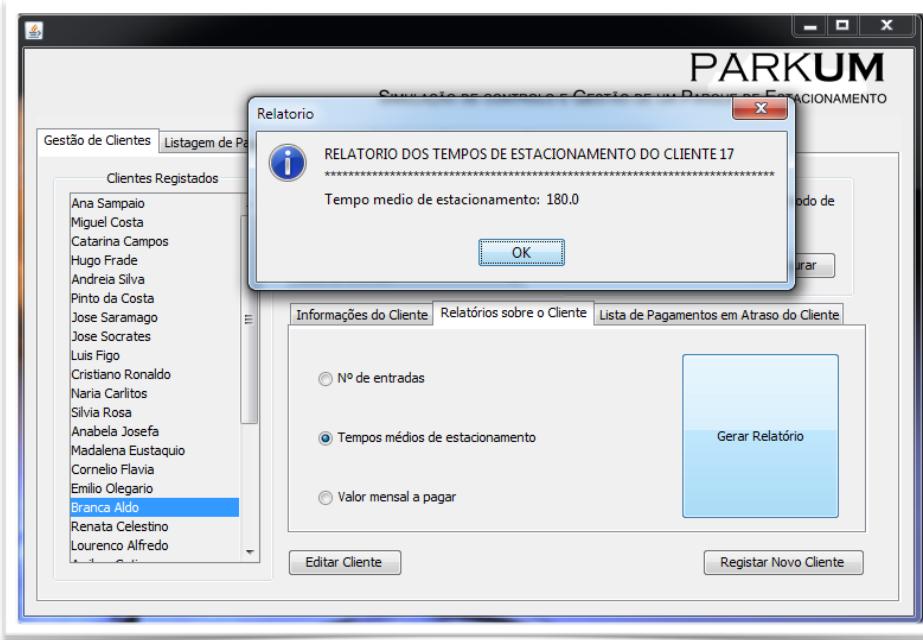


Fig 4: OBTENÇÃO DO RELATÓRIO COM OS TEMPOS MÉDIOS DE ESTACIONAMENTO

É também possível editar os dados de cada cliente, pesquisar a informação um cliente registado, por número de cliente ou por nome, e ainda filtrar os clientes conforme o tipo de pagamento.

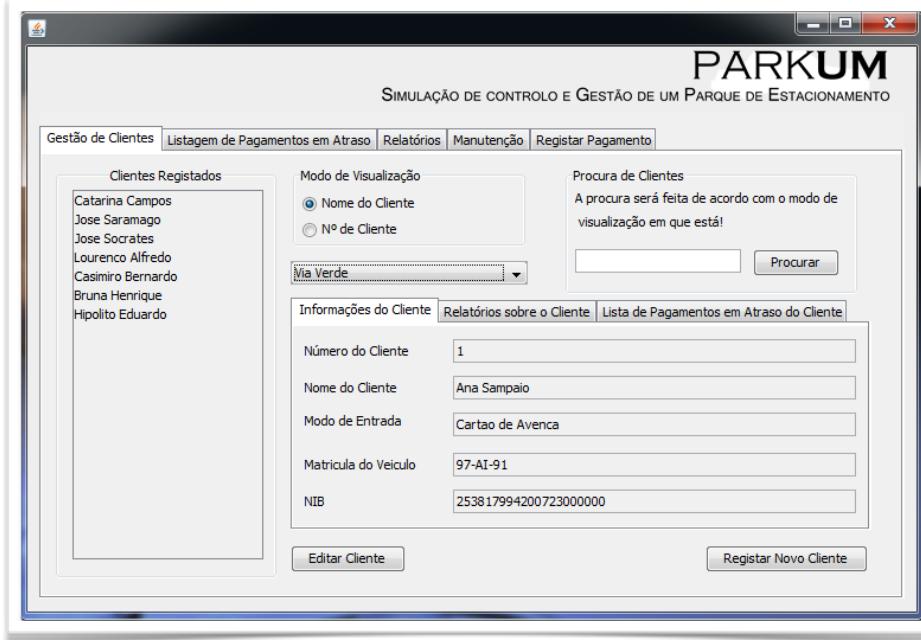


Fig 5: LISTAGEM DE CLIENTES CONFORME MODO DE ENTRADA VIA VERDE

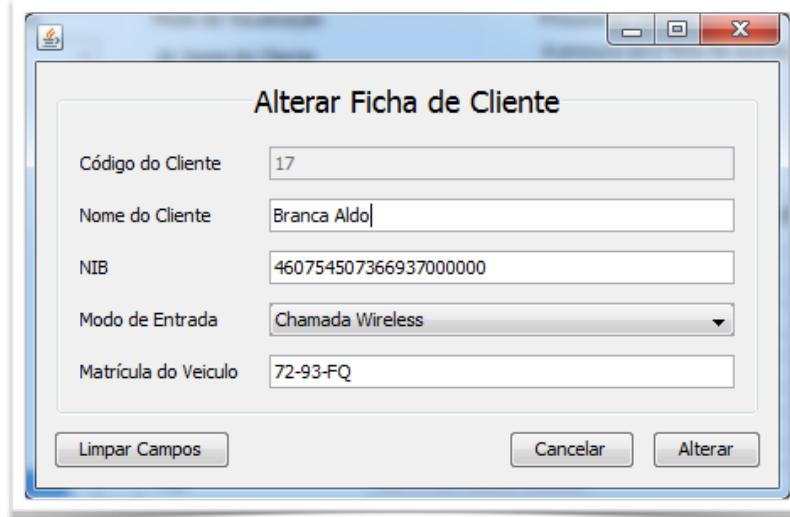


Fig 6: ALTERAÇÃO DA FICHA DE UM CLIENTE

Esta aplicação permite ainda registar um novo cliente no parque. Para isso, é necessário preencher a ficha do cliente com os seus dados pessoais, a matrícula do veículo e determinar o modo de entrada no parque que o novo cliente pretende.

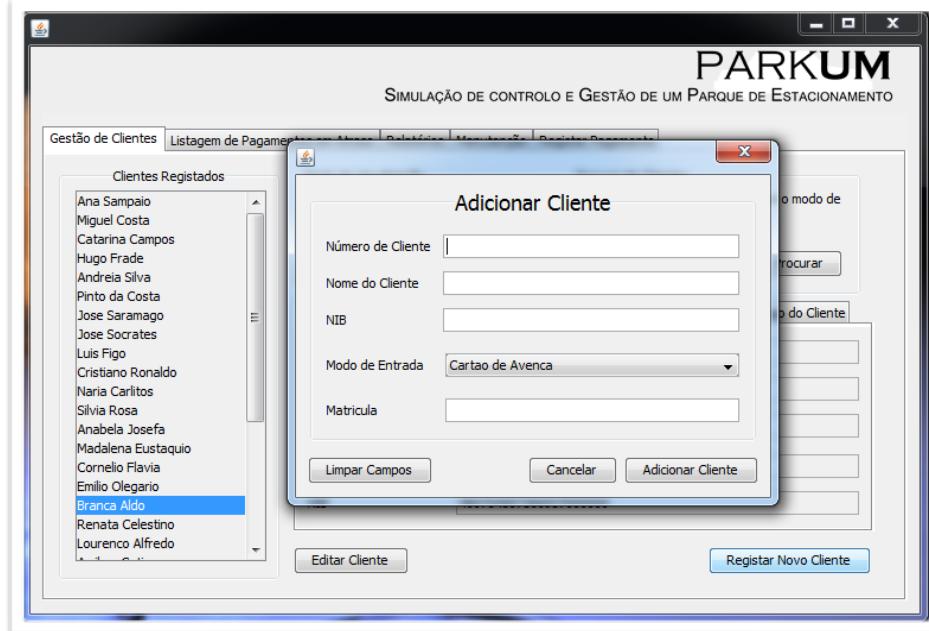


Fig 7: ADICIONAR UM NOVO CLIENTE NO SISTEMA

L I S T A G E M D E P A G A M E N T O S E M A T R A S O

É possível ter acesso à listagem de cliente que possuem pagamentos em atraso, através desta secção.

RELATÓRIOS

O sistema PARKUM permite a obtenção de relatórios diversos, sobre toda a informação relacionada com o funcionamento do parque:

- Relatórios diários, com informações gerais do parque

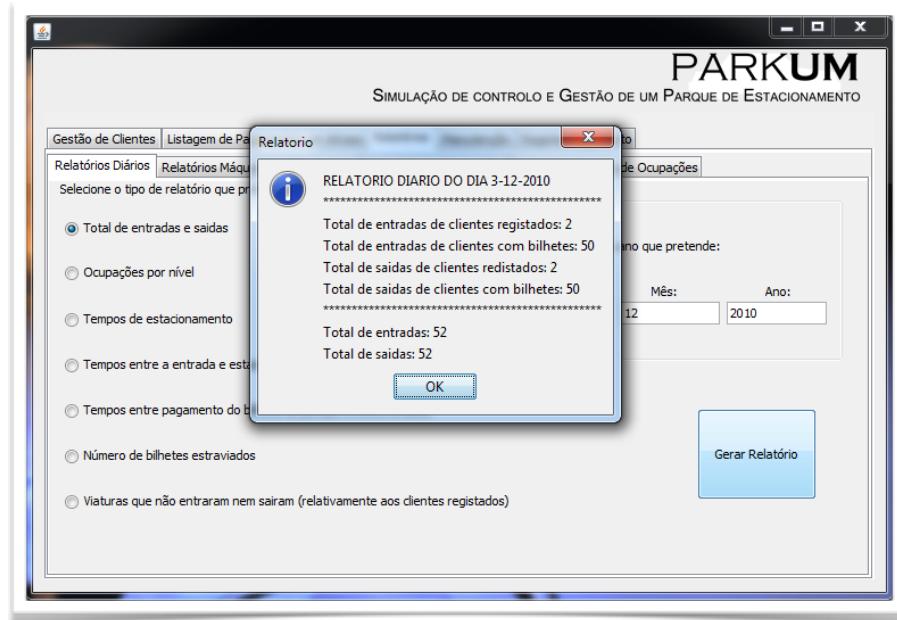


Fig 8: OBTENÇÃO DO RELATÓRIO DIÁRIO DO TOTAL DE ENTRADAS E SAÍDAS NO PARQUE, REFERENTE AO DIA 3-12-2010

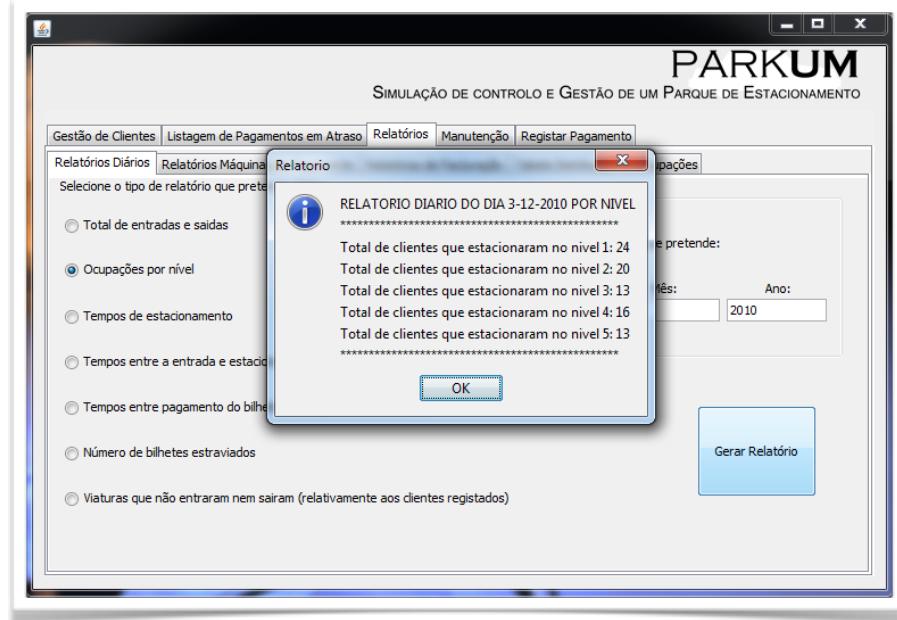


Fig 9: OBTENÇÃO DO RELATÓRIO DIÁRIO DAS OCUPAÇÕES NO PARQUE, EM TODOS OS NÍVEIS, REFERENTE AO DIA 3-12-2010

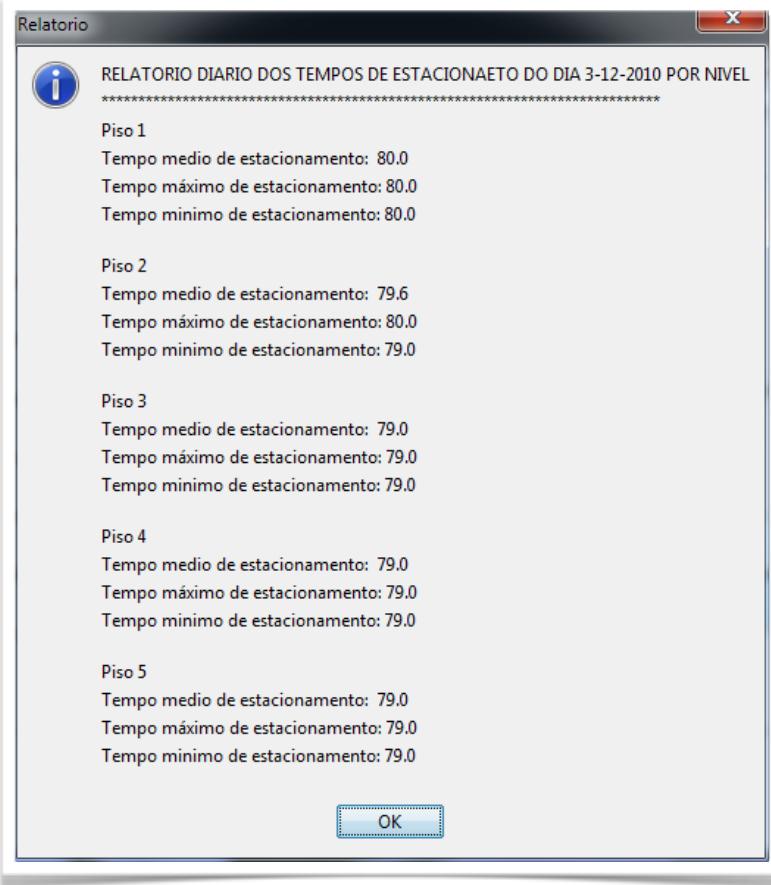


Fig 10: RELATÓRIO DIÁRIO DOS TEMPOS DE ESTACIONAMENTO, REFERENTE AO DIA 3-12-2010.

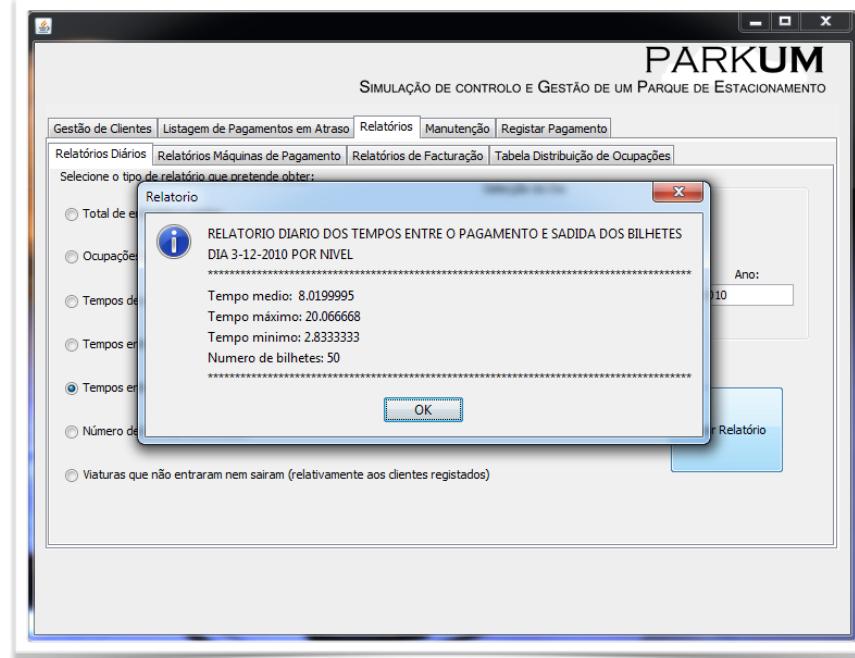


Fig 11: OBTENÇÃO DE RELATÓRIOS DIÁRIOS DOS TEMPOS DE ENTRE PAGAMENTOS E SAÍDAS, REFERENTES AO DIA 3-12-2010.

- Relatórios sobre o funcionamento das máquinas de pagamento

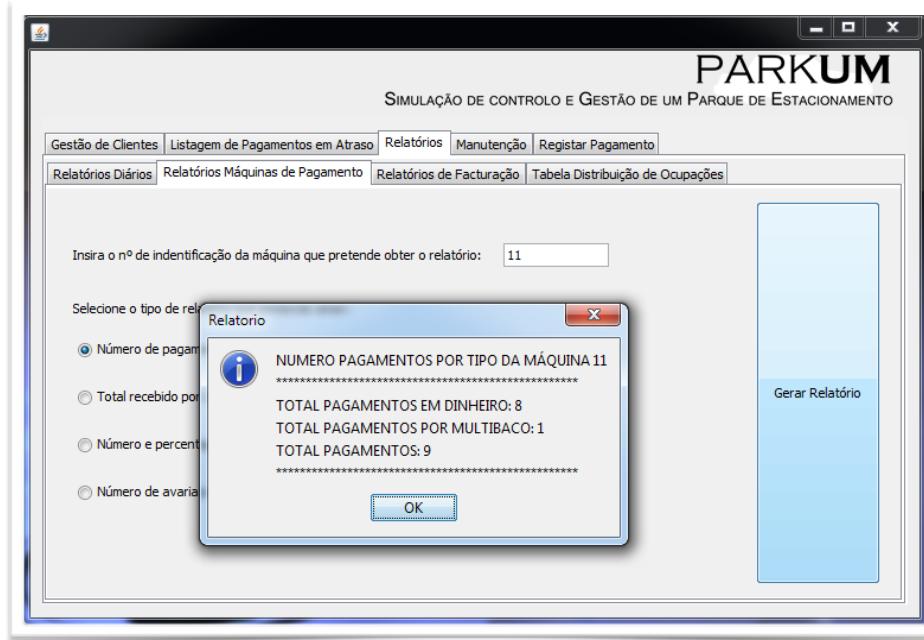


Fig 12: OBTEÇÃO DO RELATÓRIO REFERENTE AO NÚMERO DE PAGAMENTOS, POR TIPO, DA MÁQUINA 11.

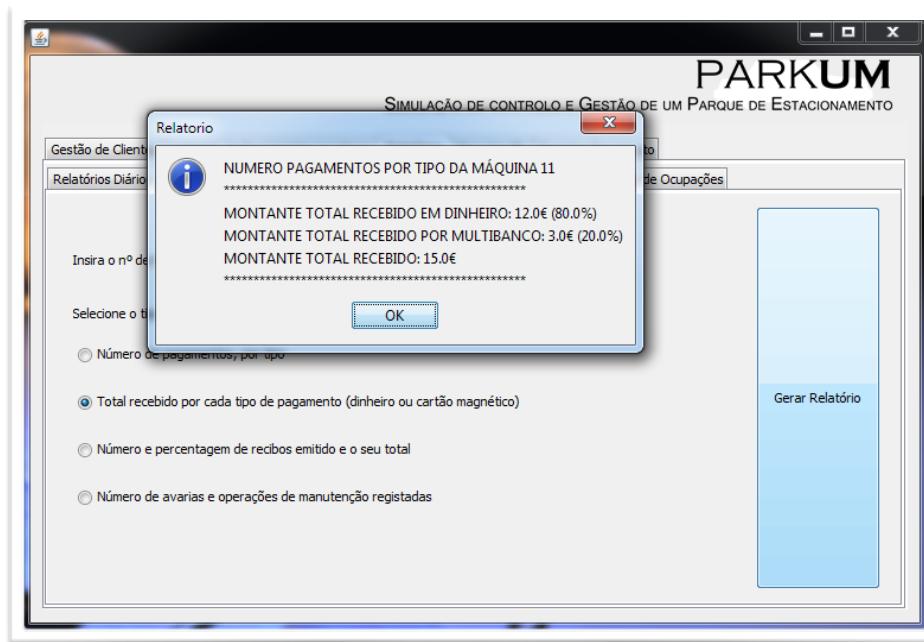


Fig 13: OBTEÇÃO DO RELATÓRIO REFERENTE AO MONTANTE ADQUIRIDO, POR FORMA DE PAGAMENTO

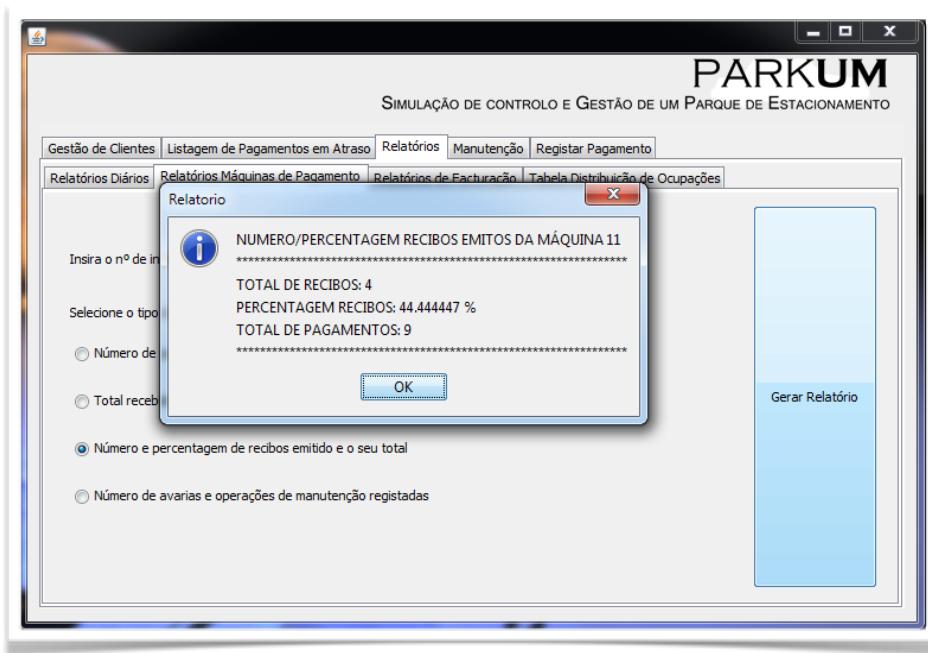


Fig 14: OBTENÇÃO DO RELATÓRIO REFERENTE AO NÚMERO E PERCENTAGEM DE RECIBOS EMITIDOS DA MÁQUINA II.

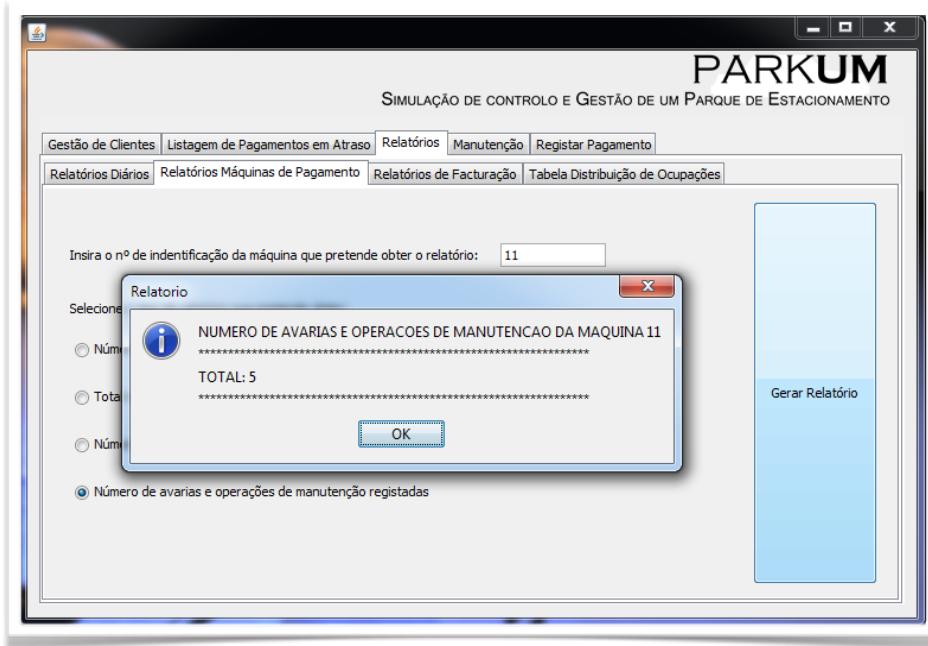


Fig 15: OBTENÇÃO DO RELATÓRIO REFERENTE AO NÚMERO DE AVARIAS E OPERAÇÕES DE MANUTENÇÃO DA MÁQUINA II

- Relatórios de facturação diários

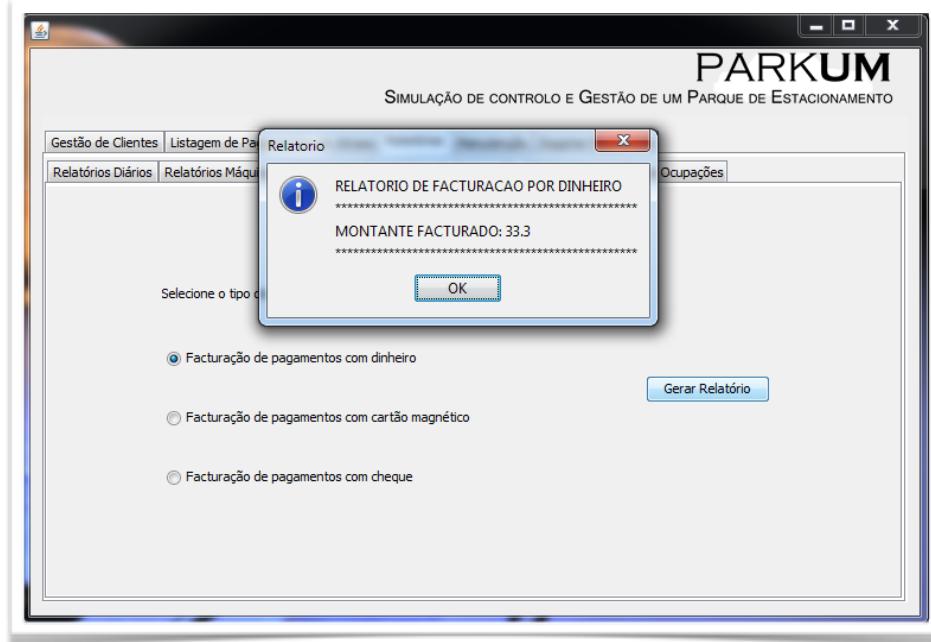


Fig 16: OBTENÇÃO DO RELATÓRIO REFERENTE À FACTURAÇÃO DE PAGAMENTOS COM DINHEIRO

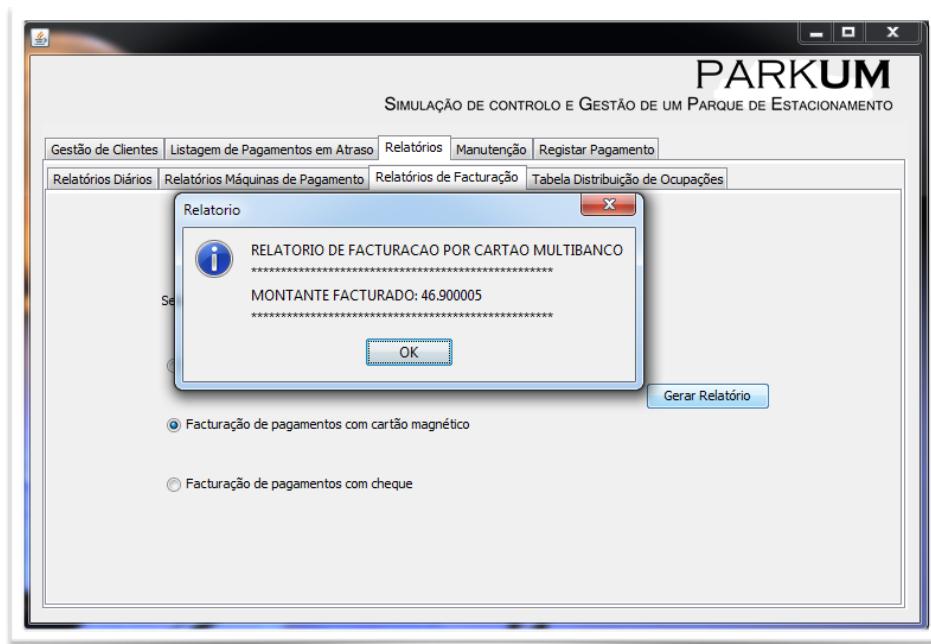


Fig 17: OBTENÇÃO DO RELATÓRIO REFERENTE À FACTURAÇÃO POR CARTÃO MAGNÉTICO

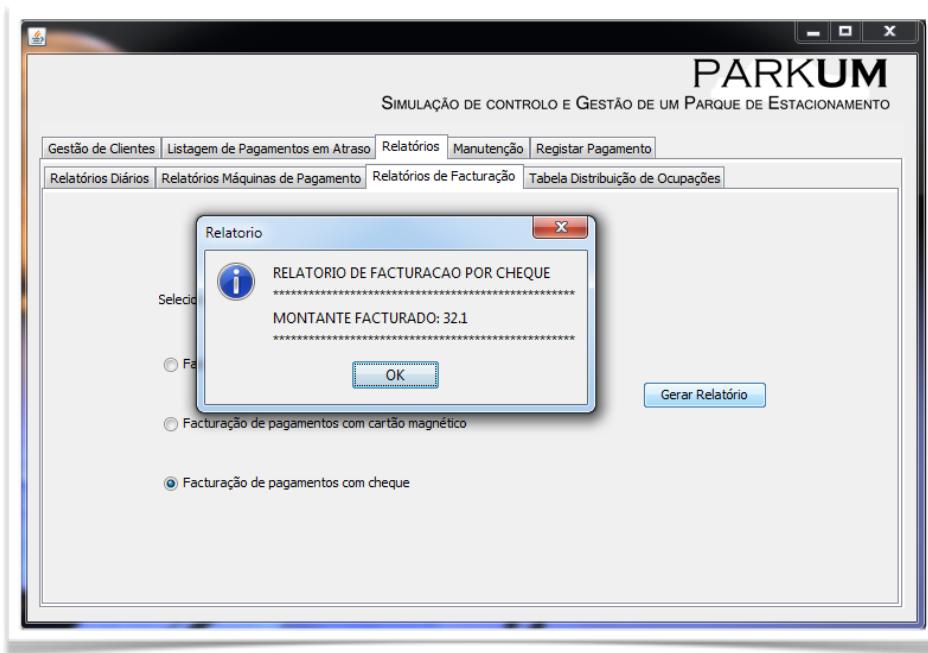


Fig 18: OBTENÇÃO DO RELATÓRIO REFERENTE À FACTURAÇÃO POR CHEQUE

- Tabelas de distribuição de ocupações de lugares

Tabela de Distribuição de Lugar																																			
Escolha o nível que pretende obter o relatório:	2																																		
Escolha a hora do dia que pretende obter o relatório	12 h																																		
	Gerar Tabela																																		
<table border="1"> <tr><td>LUGAR: 208</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 209</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 210</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 211</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 212</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 213</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 214</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 215</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 216</td><td>ESTADO: OCUPADO</td></tr> <tr><td>LUGAR: 217</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 218</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 219</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 220</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 221</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 222</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 223</td><td>ESTADO: LIVRE</td></tr> <tr><td>LUGAR: 224</td><td>ESTADO: LIVRE</td></tr> </table>		LUGAR: 208	ESTADO: OCUPADO	LUGAR: 209	ESTADO: OCUPADO	LUGAR: 210	ESTADO: OCUPADO	LUGAR: 211	ESTADO: OCUPADO	LUGAR: 212	ESTADO: OCUPADO	LUGAR: 213	ESTADO: OCUPADO	LUGAR: 214	ESTADO: OCUPADO	LUGAR: 215	ESTADO: OCUPADO	LUGAR: 216	ESTADO: OCUPADO	LUGAR: 217	ESTADO: LIVRE	LUGAR: 218	ESTADO: LIVRE	LUGAR: 219	ESTADO: LIVRE	LUGAR: 220	ESTADO: LIVRE	LUGAR: 221	ESTADO: LIVRE	LUGAR: 222	ESTADO: LIVRE	LUGAR: 223	ESTADO: LIVRE	LUGAR: 224	ESTADO: LIVRE
LUGAR: 208	ESTADO: OCUPADO																																		
LUGAR: 209	ESTADO: OCUPADO																																		
LUGAR: 210	ESTADO: OCUPADO																																		
LUGAR: 211	ESTADO: OCUPADO																																		
LUGAR: 212	ESTADO: OCUPADO																																		
LUGAR: 213	ESTADO: OCUPADO																																		
LUGAR: 214	ESTADO: OCUPADO																																		
LUGAR: 215	ESTADO: OCUPADO																																		
LUGAR: 216	ESTADO: OCUPADO																																		
LUGAR: 217	ESTADO: LIVRE																																		
LUGAR: 218	ESTADO: LIVRE																																		
LUGAR: 219	ESTADO: LIVRE																																		
LUGAR: 220	ESTADO: LIVRE																																		
LUGAR: 221	ESTADO: LIVRE																																		
LUGAR: 222	ESTADO: LIVRE																																		
LUGAR: 223	ESTADO: LIVRE																																		
LUGAR: 224	ESTADO: LIVRE																																		

Fig 19: OBTENÇÃO DAS TABELAS DE DISTRIBUIÇÃO DE OCUPAÇÕES, REFERENTES AO NÍVEL 2, ÀS 12H.

M A N U T E N Ç Ã O

Todas as máquinas de pagamento recebem operações de manutenção efectuadas pelos funcionários do parque, que devem ser registadas nesta secção. Devem também ser designados os dados da máquina, as operações efectuadas e o tempo gasto.

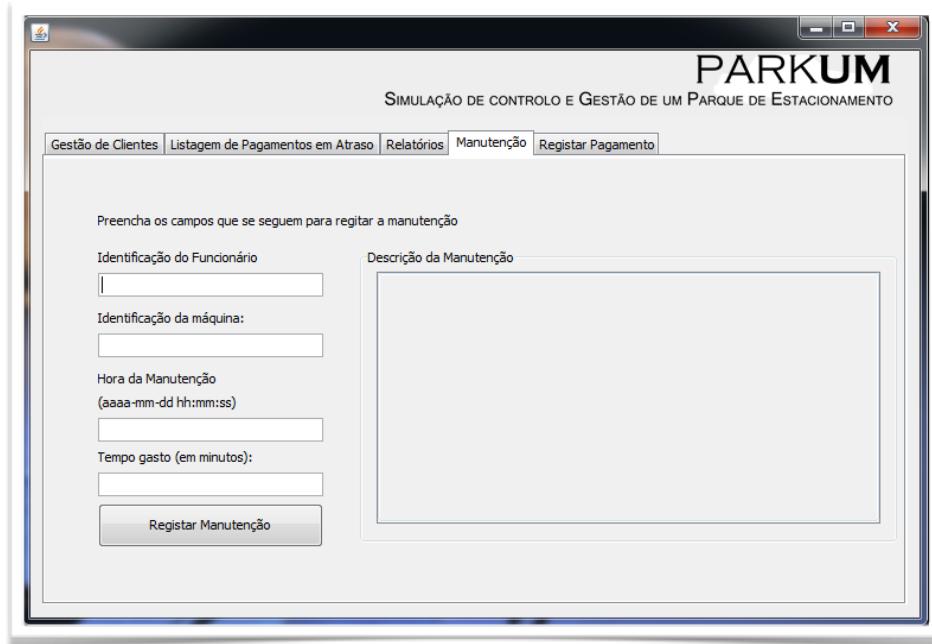


Fig 20 - REGISTAR MANUTENÇÃO DA MÁQUINA DE PAGAMENTO

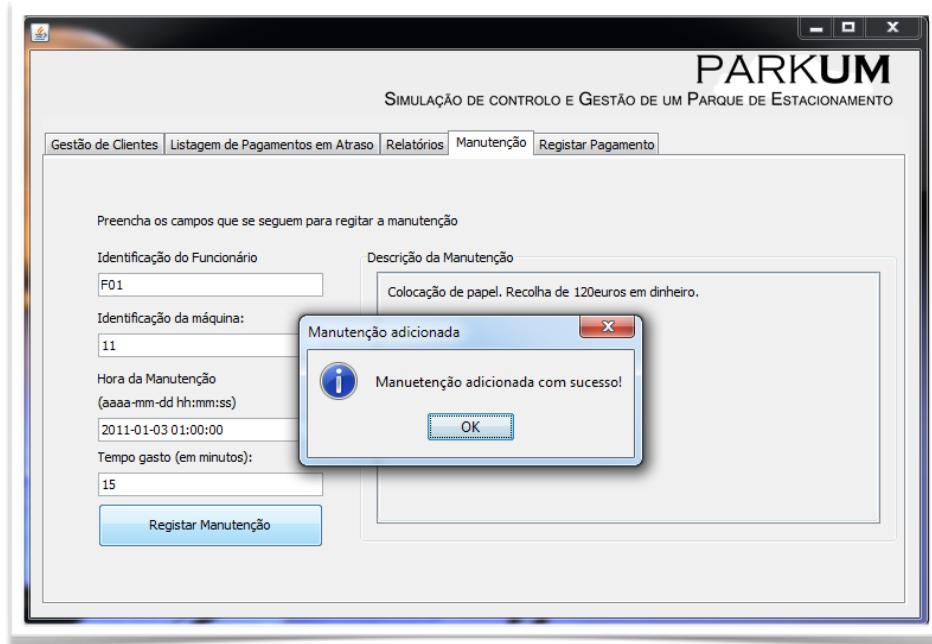


Fig 21: REGISTO DA MANUTENÇÃO DA MÁQUINA II COM SUCESSO

REGISTAR PAGAMENTOS

Quando um cliente efectua um dado pagamento ao funcionário, este pode registá-lo aqui. Esta secção destina-se essencialmente ao registo de pagamentos automáticos, por exemplo, cartão de avença, e de multas por perda do bilhete. Para isso, o funcionário deve preencher os campos pedidos e escolher o modo de pagamento. Clicando em “Registar Pagamento”, o pagamento é automaticamente registado na base de dados.

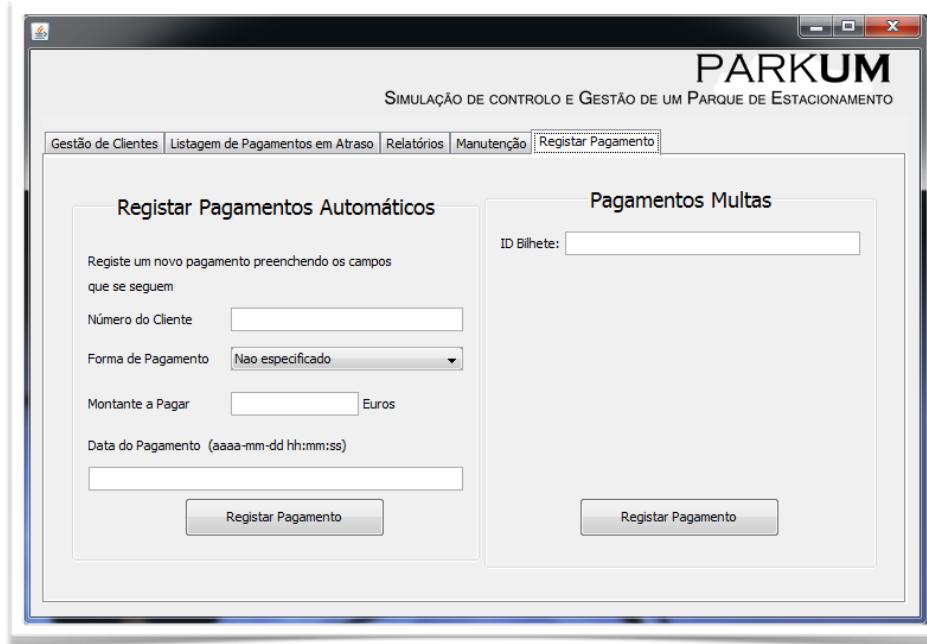


Fig 22: REGISTO DE PAGAMENTOS PARA CLIENTES QUE ENTRAM NO PARQUE DE FORMA AUTOMÁTICA E REGISTO DE PAGAMENTO DE MULTAS.