Desenvolvimento de Sistemas de Software

Ano letivo 2016/2017
Universidade do Minho

Carlos Campos
A74745

Pedro Fonseca
A74166

Paulo Guedes
A74411

Diogo Gomes
A73825

Introdução

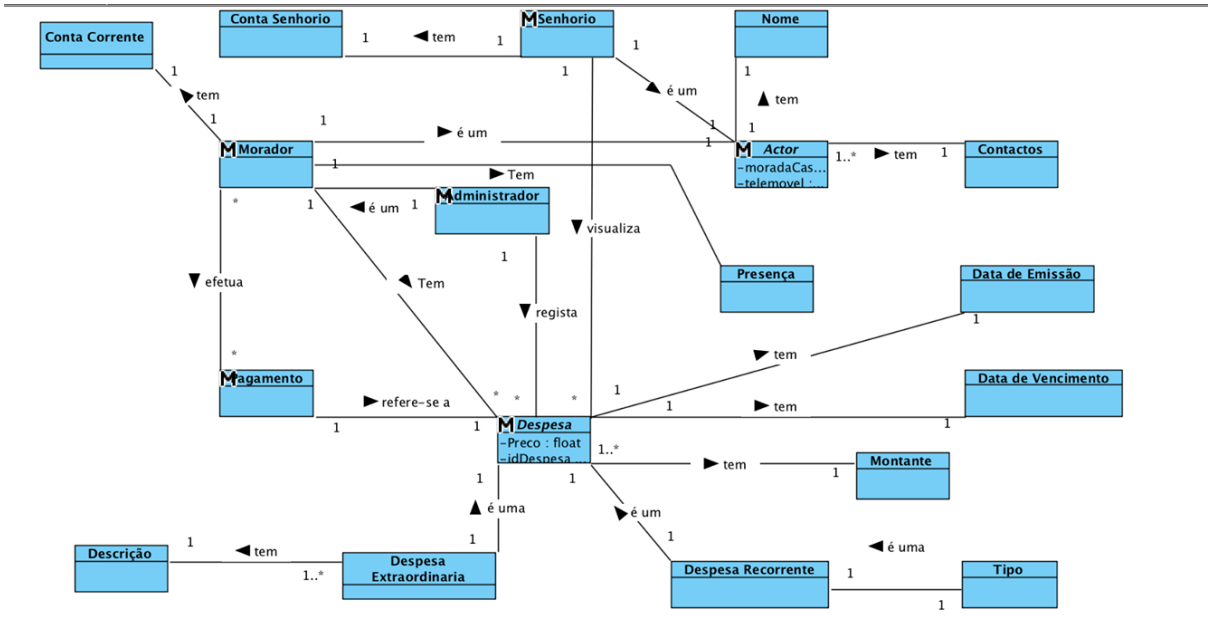
Este trabalho tem como objetivo o desenvolvimento de um sistema de suporte à partilha de despesas num apartamento.

Para tal, é necessário que o sistema desenvolvido seja suficientemente flexível para suportar quer diferentes formas de divisão das despesas, quer diferentes modalidades de pagamento.

Tendo em conta todos estes aspetos, iremos desenvolver uma proposta de Modelo de Domínio para satisfazer estes requisitos, assim como os Diagramas de Use Case para a sua utilização. Serão apresentados vários diagramas de sequencia para explicar o comportamento de cada subsistema e de cada operação logica da aplicação. Também estarão presentes diagramas de classes, de instalação e de pacotes para ajudar o estudo da estrutura do trabalho. E por fim os diagramas das maquinas de estado, serão ilustrativos para representar o funcionamento da interface.

O projeto desenvolvido pelo nosso grupo tem como objetivo, facilitar o gerenciamento das despesas incorrentes da utilização do apartamento, assim como das despesas fixas que são independentes dessa mesma utilização. Como tal, focamo-nos em fazer a distinção entre os vários tipos de despesa assim como a correta divisão das mesmas, e também na flexibilidade da forma de pagamento.

Modelo de Domínio



O nosso modelo de domínio pode ser partido em 2 partes, a parte dos Atores e das Despesas, o nosso grupo decidiu não adicionar a entidade Apartamento, pois este já é assumido, sendo que este modelo de domínio é referente a utilização do mesmo.

Deste modo, temos diversas entidades associadas à primeira parte, tais como:

- Actor: esta entidade representa as diversas pessoas que estão envolvidas neste apartamento;
  - ➢ Nome: esta entidade está associada ao Actor, pois este precisa de um nome, de modo a ser distinguido dos restantes atores;
  - ➢ Contactos: esta entidade está associada ao Actor, pois este precisa de um contacto, de modo a ser distinguido dos restantes atores;

- Senhorio: esta entidade representa o dono do apartamento, ou a pessoa encarregue deste, sendo este também um Actor;
  - ➢ Conta Senhorio: esta entidade está associada ao Senhorio, e indica que este é detentor de uma conta;

- Morador: esta entidade representa os que habitam no apartamento, sendo este também um Actor;

- ➢ Administrador: esta entidade está associada ao Morador, porque o administrador vai ser o morador que estará encarregue das despesas da casa, todos os outros lhe vão pagar, e ele é que irá pagar as despesas;
- ➢ Conta Corrente: está associada ao Morador, e indica que este é detentor de uma conta;
- ➢ Pagamento: está associada ao Morador, e representa que um pagamento foi efetuado, referente a uma despesa;
- ➢ Presença: está associada ao Morador, representa quem estava presente num determinado momento;

Nesta primeira parte podemos ver alguns dos caminhos que o nosso grupo decidiu seguir, tal como, haver um administrador no apartamento, pois esta situação verifica-se na realidade, e facilita aos moradores e ao senhorio. Para além deste caminho também optamos por dar um acesso especial ao senhorio, de modo a ele ter uma noção do que se está a passar no apartamento.

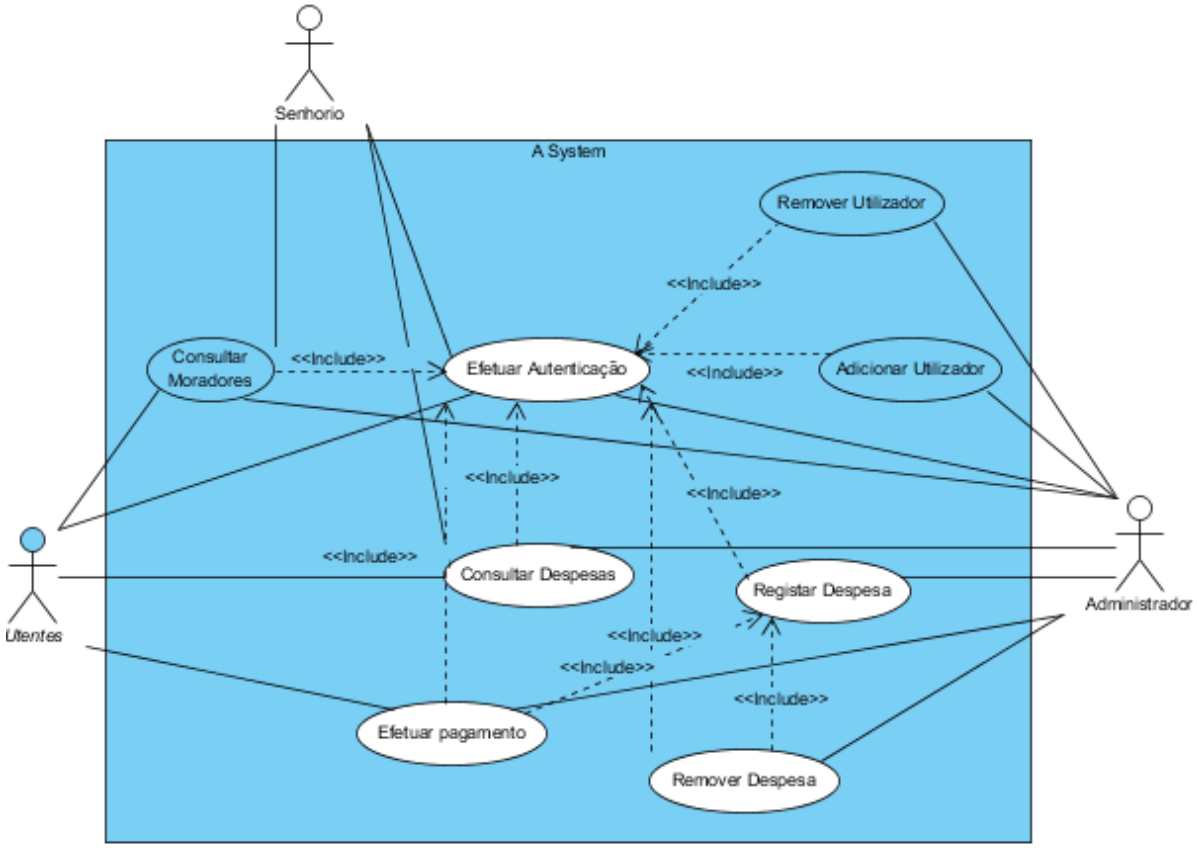
Na parte das despesas podemos verificar que, como referido anteriormente, apenas o administrador pode registar despesas, e o senhorio apenas pode consultar. Deste modo temos o seguinte:

- Despesa: está associada ao Morador, ao Administrador, ao Senhorio, sendo que apenas o administrador pode registar uma despesa;
  - ➢ Montante: está associada à Despesa, e indica o valor da despesa;
  - ➢ Data de Emissão: está associada à Despesa, e indica a data em que um determinado despesa foi emitida.
  - ➢ Data de Vencimento: está associada à Despesa, e indica a data em que um determinado despesa tem de ser entregue.

- Despesa Recorrente: está associada à Despesa, e representa as despesas que todos os meses é preciso pagar, tais como eletricidade e gás.
  - ➢ Tipo: está associada à Despesa Recorrente, e indica o tipo da despesa, ou seja, se representa uma despesa de eletricidade, ou de gás, ou outra despesa recorrente.

- Despesa Extraordinária: está associada à Despesa, e representa as despesas não recorrentes, como por exemplo pagar por uma televisão nova.
  - ➢ Descrição: está associada à Despesa Extraordinária, e é um breve comentário, sobre o porquê de esta despesa acontecer;

Diagramas de Use Case



Descrição dos Use Case

Consideramos que existem certos Use Case que são meritórios de serem enaltecidos no relatório pois têm grande importância no consequente desenvolvimento do software. São eles Efetuar Autenticação e Registar Despesa, sendo que o resto se encontra no anexo.

Efetuar Autenticação

<table>
<tr><td>Brief Description</td><td>Autenticação dos utilizadores</td></tr>
<tr><td>Preconditions</td><td>Estar registado</td></tr>
<tr><td>Post-conditions</td><td>Estar autenticado</td></tr>
<tr><td>Flow of Events</td><td>
<table>
<tr><th></th><th>Actor Input</th><th>System Response</th></tr>
<tr><td>1</td><td>Insere dados</td><td></td></tr>
<tr><td>2</td><td></td><td>Valida dados</td></tr>
</table>
</td></tr>
<tr><td>Alternativa 1 [Dados inválidos < 3 vezes] (Passo 2)</td><td>
<table>
<tr><th></th><th>Actor Input</th><th>System Response</th></tr>
<tr><td>1</td><td></td><td>Informa que dados estão inválidos</td></tr>
<tr><td>2</td><td>Insere de novo os dados</td><td></td></tr>
<tr><td>3</td><td></td><td>Regressa a 2</td></tr>
</table>
</td></tr>
<tr><td>Exceção 1 [Dados inválidos = 3 vezes] (Passo 2.3)</td><td>
<table>
<tr><th></th><th>Actor Input</th><th>System Response</th></tr>
<tr><td>1</td><td></td><td>Informa que dados estão inválidos</td></tr>
<tr><td>2</td><td></td><td>Bloqueia</td></tr>
</table>
</td></tr>
</table>

Registar Despesa

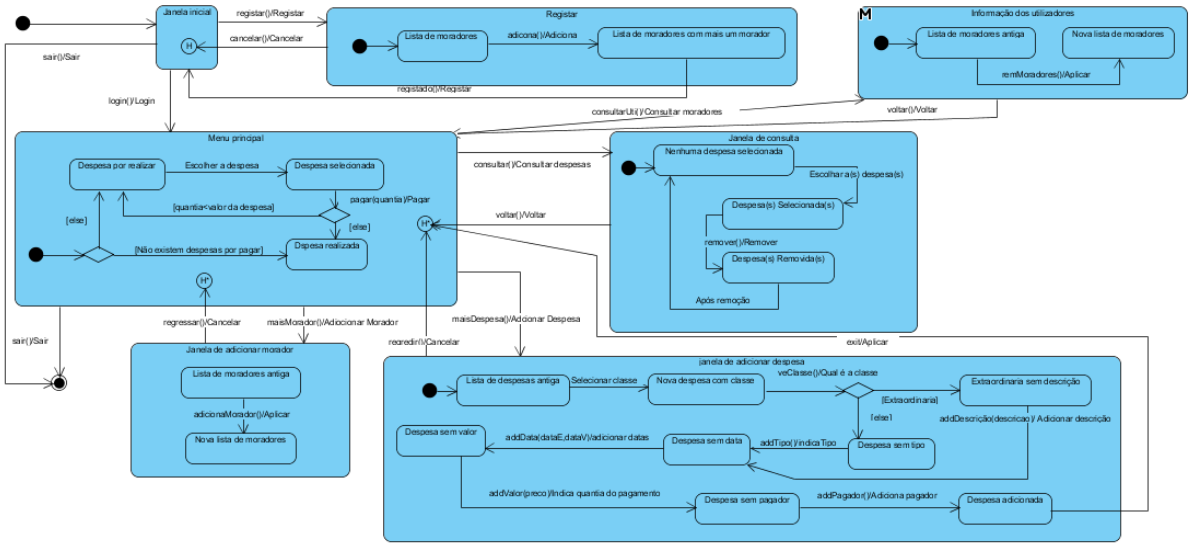
<table>
<tr><td>Super Use Case</td><td></td></tr>
<tr><td>Author</td><td>user</td></tr>
<tr><td>Date</td><td>28/out/2016 11:59:54</td></tr>
<tr><td>Brief Description</td><td>Registar despesas do apartamento</td></tr>
<tr><td>Preconditions</td><td>Estar autenticado</td></tr>
<tr><td>Post-conditions</td><td>Despesa registada</td></tr>
<tr><td>Flow of Events</td><td>
<table>
<tr><th></th><th>Actor Input</th><th>System Response</th></tr>
<tr><td>1</td><td>Inserir detalhes da despesa</td><td></td></tr>
<tr><td>2</td><td></td><td>Confirmar despesa</td></tr>
</table>
</td></tr>
<tr><td>Alternativa 1
[Se é despesa extraordinaria]
(Passo 2)</td><td>
<table>
<tr><th></th><th>Actor Input</th><th>System Response</th></tr>
<tr><td>1</td><td></td><td>Requer descrição</td></tr>
<tr><td>2</td><td>Insere descrição</td><td></td></tr>
<tr><td>3</td><td></td><td>Regressa a 2</td></tr>
</table>
</td></tr>
</table>

Diagramas de Máquina de Estado

Os Diagramas de Estado permitem modelar o comportamento de um dado sistema de forma global. Modelam-se todos os estados possíveis que o sistema atravessa em resposta aos eventos que podem ocorrer.

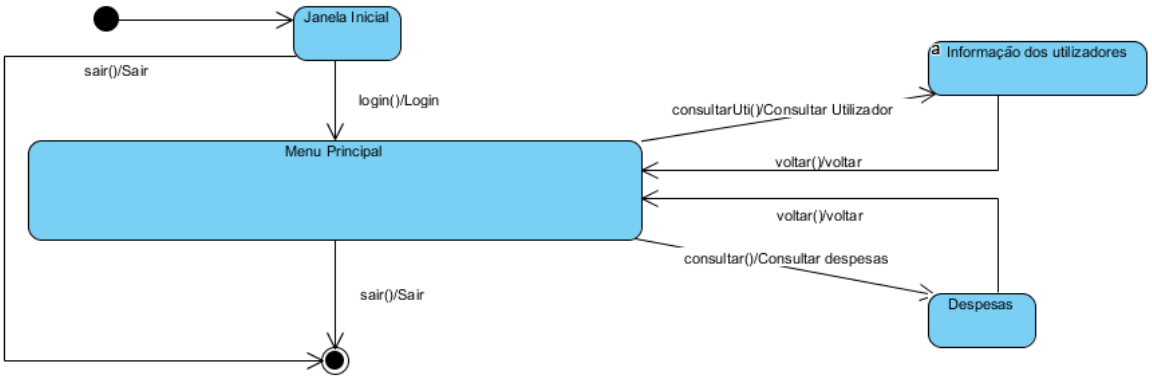
No nosso caso, criamos 3 Máquinas de Estado, para explicitar as 3 interfaces possíveis para os 3 tipos de utilizadores, Administrador, Senhorio e Morador.

Máquina de Estado – Administrador



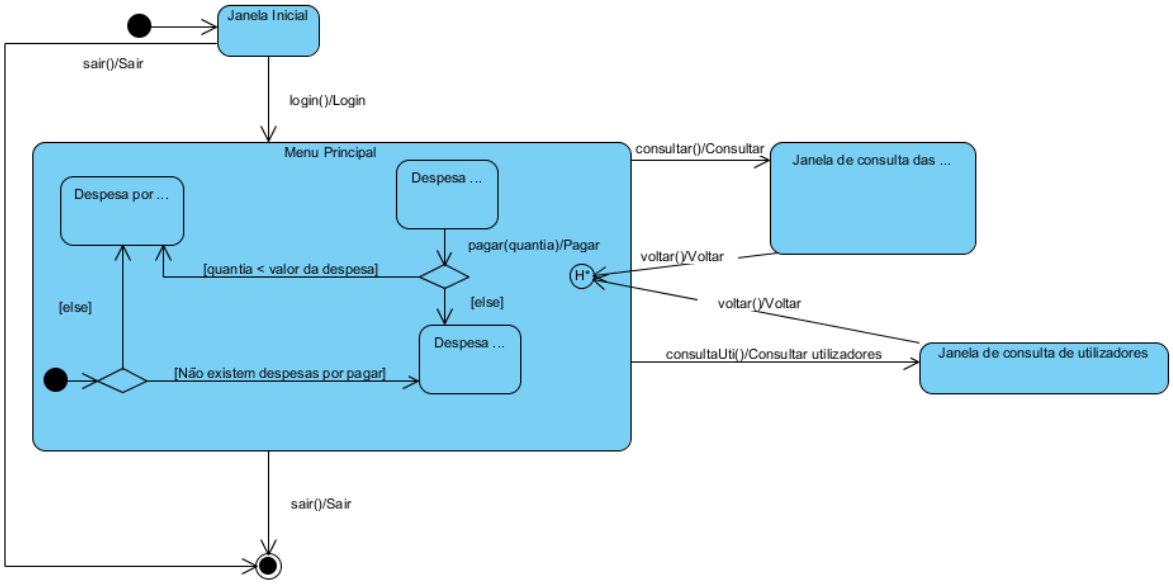
Aqui podemos observar todos os estados possíveis que o sistema de Administrador pode ter em resposta aos diferentes eventos que podem ocorrer. É uma máquina de estado relativamente complexa pois o administrador, de todos os atores, é o que maiores permissões têm, como por exemplo adicionar uma despesa assim como removê-la, e ainda de poder adicionar moradores.

Máquina de Estado – Senhorio



Aqui podemos observar todos os estados possíveis que o sistema de Senhorio pode ter em resposta aos diferentes eventos que podem ocorrer. Como este só pode observar e nada alterar no estado da aplicação, a sua Máquina de Estado limita-se a este pequeno diagrama com apenas um estado inicial e estado final.

Máquina de Estado – Morador



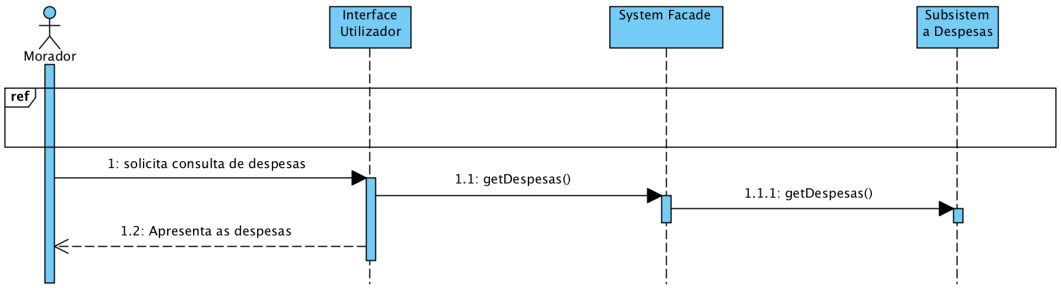
Aqui podemos observar todos os estados possíveis que o sistema de Morador pode ter em resposta aos diferentes eventos que podem ocorrer. O Morador comparativamente ao senhorio tem apenas o acrescento de ter de pagar as despesas, logo caso execute essa ação será criado um novo estado da aplicação como podemos observar no diagrama.

Diagramas de Sequência de Subsistemas

Os Diagramas de Sequência representam as interações, ordenadas de forma temporal, entre objetos através das mensagens que são trocadas entre eles.

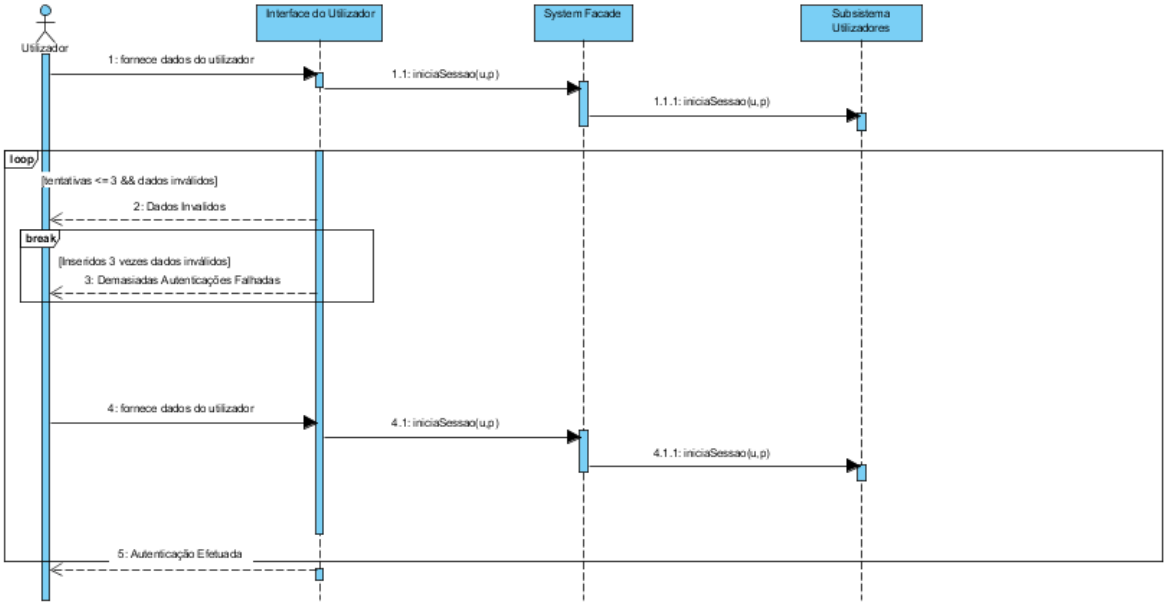
No nosso caso em específico, criamos os seguintes diagramas de sequência para satisfazer a especificação dos Use Case do problema. Utilizamos o mesmo método para fazer os diagramas de sequencia de sistemas, que se encontram em anexo.

Diagrama de Sequência de Subsistemas - Consulta de Despesas



Neste diagrama temos a especificação do Use Case Consulta de Despesas, onde o ator solicita a consulta das despesas e o sistema responde apresentando a lista das despesas.

Diagrama de Sequência de Subsistemas - Efetuar Autenticação



No diagrama de sequência da especificação do Use Case Efetuar Autenticação temos os seguintes passos. Inicialmente o ator introduz os seus dados, caso os dados estejam incorretos o sistema informa o ator que os dados estão incorretos. Após 3 tentativas falhadas o sistema bloqueia e informa o ator que falhou demasiadas vezes a autenticação. Caso tudo esteja correto o ator fica autenticado.

Diagrama de Sequência de Subsistemas - Registar Despesa

Seguindo a lógica de termos a distinção entre despesa corrente e despesa extraordinária, aquando da inserção de uma nova despesa o sistema determina o seu tipo, de forma a que, caso a mesma seja extraordinária, o sistema solicite ao utilizador a correspondente descrição para posteriormente ser inserida nas “Despesas”.

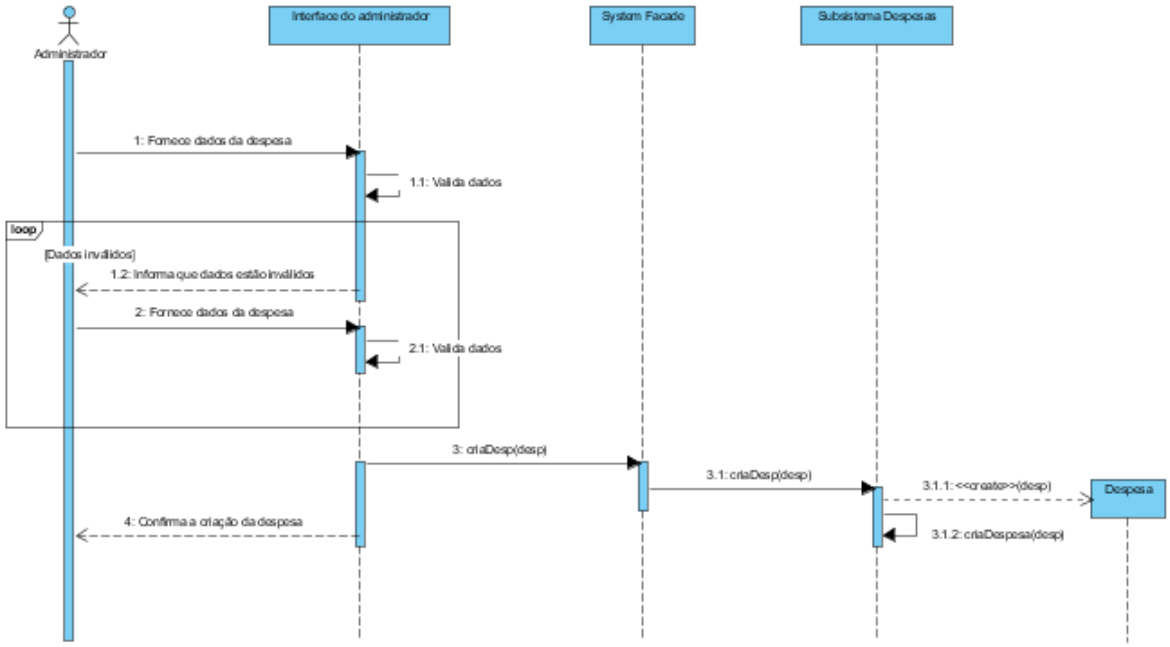
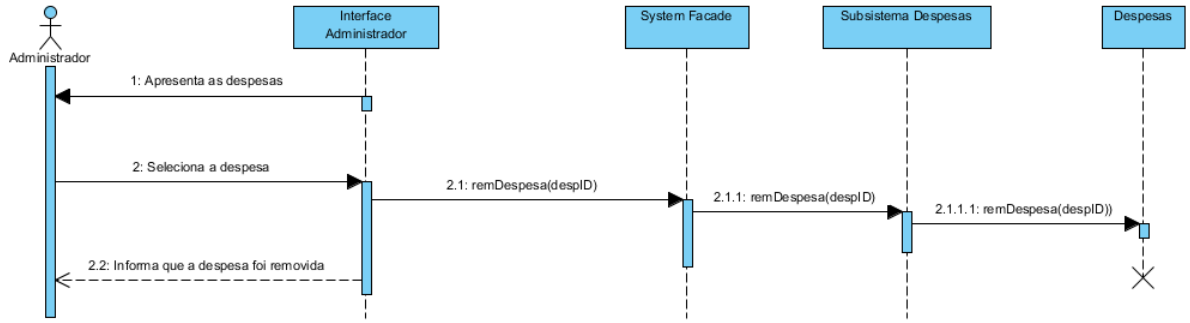


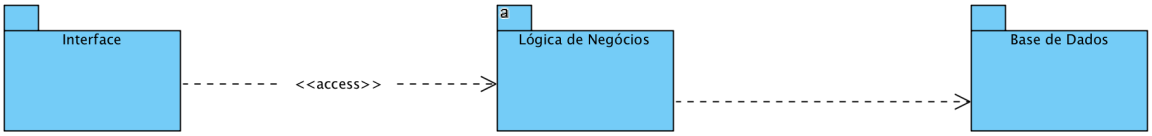
Diagrama de Sequência de Subsistemas - Remover Despesa



De forma a especificar o Use Case Remover Despesa, no diagrama de sequência temos a seguinte disposição:

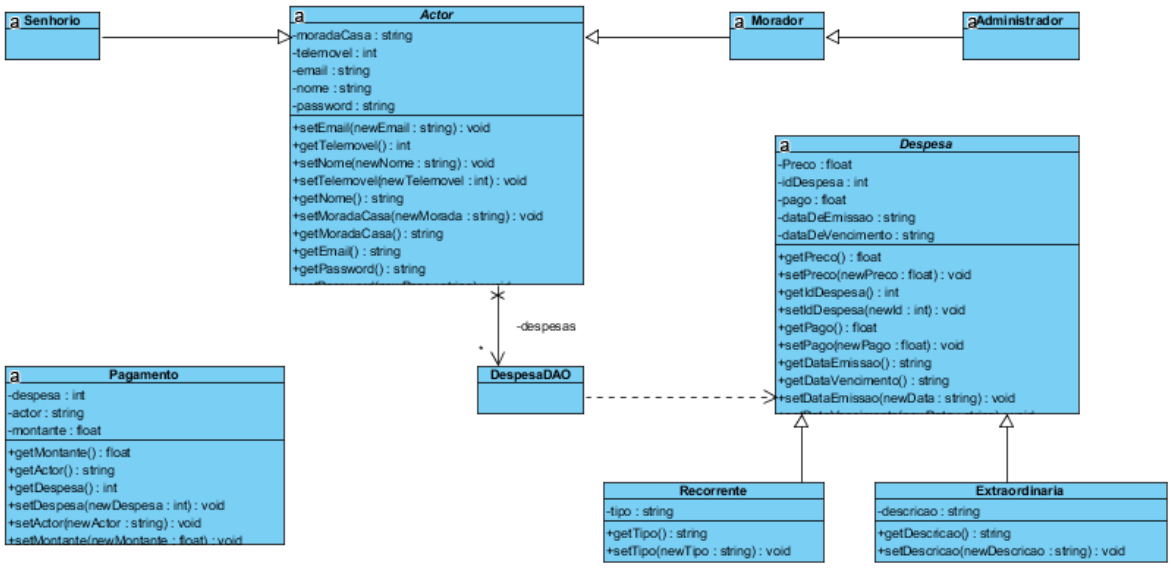
O utilizador seleciona a despesa que deseja remover e o sistema faz a sua consequente remoção do objeto “Despesas” emitindo uma reposta de volta para o utilizador a confirmar a remoção.

Diagrama de Pacotes Geral



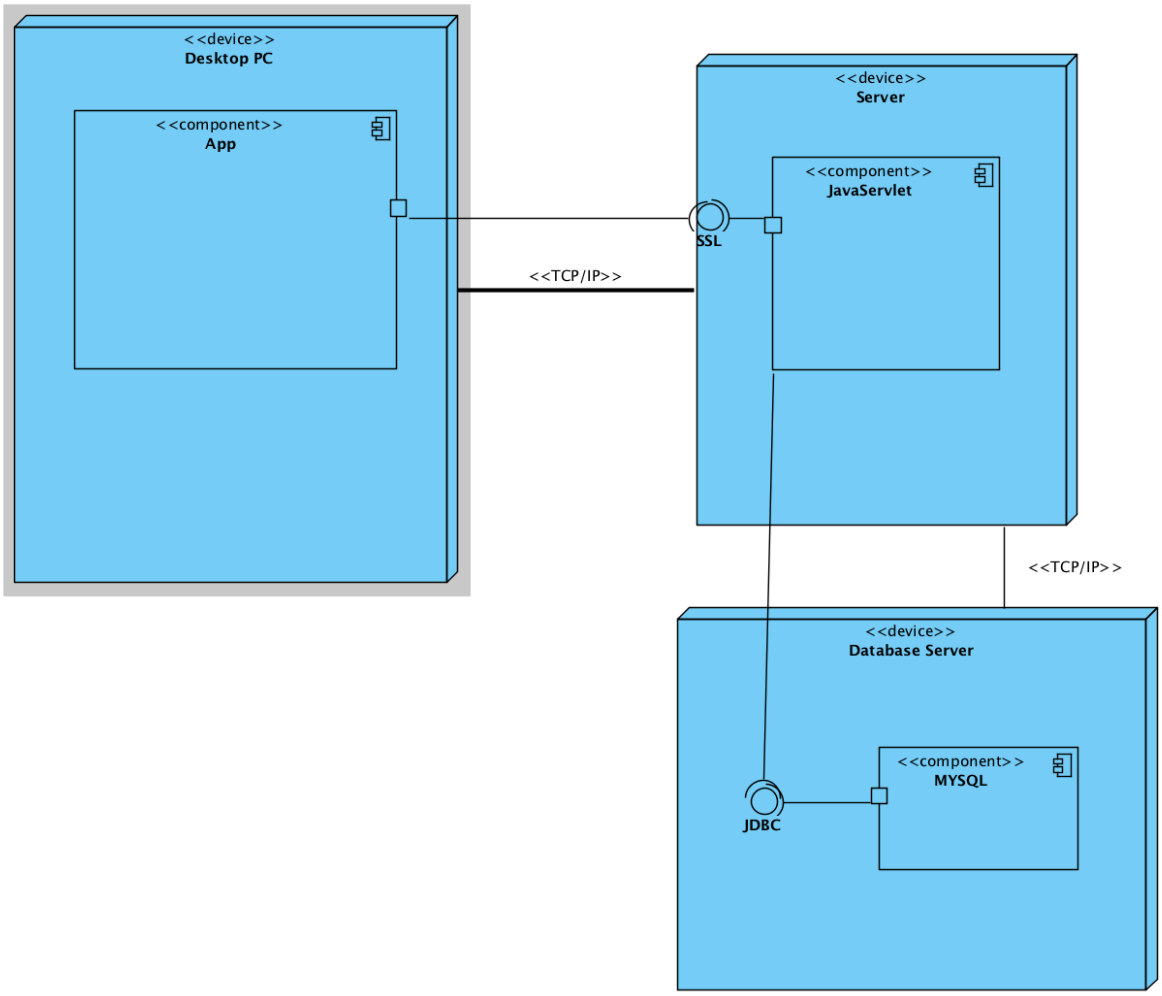
Neste diagrama de pacotes podemos ver que a interface permite o acesso à lógica de negócios, e a lógica de negócios é dependente da base de dados.
Os restantes diagramas de pacotes, específicos aos respetivos diagramas de sequencia, estão em anexo.

Diagrama de Classe



No diagrama de classes, representamos as classes da parte logica da nossa aplicação, contendo os seus atributos e respetivos get's e set's

Diagrama de Instalação



Os diagramas de instalação permitem especificar a arquitetura física do sistema assim como especificar a distribuição dos componentes e ainda facilitar a identificação de estrangulamentos de desempenho. No nosso sistema temos a app a correr num dispositivo que neste caso será um PC, que comunica tem uma porta de comunicação com o servidor Java através da interface SSL. O PC e o servidor comunicam por TCP/IP. E de seguida para aceder aos dados da base de Dados temos o servidor da Base de Dados onde se encontram os dados por assim dizer. Através da interface é feita esta comunicação pelas “ports” do servidor Java e a base

de dados em MYSQL. O dispositivo servidor e o servidor da base de dados comunicam por TCP/IP também.

Anexo:

Em anexo o nosso grupo colocou o código utilizado e a base de dados também utilizada. Para complementar também foi colocado em anexo outros diagramas, sendo que estes utilizaram a mesma realizados pelo mesmo método que os diagramas do mesmo tipo aqui demonstrados.

Conclusão

Como foi referido anteriormente, optamos por criar um administrador no apartamento, pois esta situação verifica-se na realidade, e facilita a gestão dos gastos por parte dos moradores, mas também por parte do senhorio. Para além deste caminho optamos também por dar um acesso especial ao senhorio, de modo a que ele possa observar como está a situação das despesas no apartamento, bem como aceder a informação dos moradores, como por exemplo os seus contactos, permitindo assim ter um maior controlo e capacidade de ação caso seja necessário.

Achamos necessário fazer uma partição de uma despesa, em varias, cada uma com o seu respetivo pagador. Estivemos em consideração de colocarmos uma espécie de índice secundário nas despesas que fizessem parte da mesma conta, ou seja, ao administrador chegava uma conta da luz, ao adicionar na base de dados, em vez de adicionar uma despesa, ele criaria n despesas, consoante o numero de moradores envolvidos, como essas despesas teriam o mesmo tipo, data de emissão e data de vencimento, deixamos essa ideia de lado. Como tal criamos um índice na tabela Despesa, da base de dados, em relação a essas datas.

Decidimos também, que seriamos nós a fazer a inscrição do senhorio na base de dados, depois de lhe pedirmos as sus informações, visto que só existe um apartamento para gerir e assim evita-se a adição de novos senhorios por parte dos moradores.

Como grupo consideramos estes, uns dos pontos fortes do nosso trabalho pois dá uma maior flexibilidade ao sistema, visto como tal foi pedido no enunciado, a nossa aplicação garantes várias modalidades de pagamento.

No entanto tem determinadas limitações pois está muito dependente do administrador do apartamento, pois sem este, muitas das ações não poderão ser executadas, como por exemplo registar ou remover uma despesa.

Em suma, achamos que criamos um modelo equilibrado, com pontos fortes, mas também com limitações, mas que é mais do que capaz de fazer a gestão das despesas num apartamento.