

**Desenvolvimento de Sistemas de Software**

**ConfiguraFácil**

Filipa Correia Parente (82145) José André Martins Pereira(a82880)

Rafaela Maria Soares da Silva(a79034) Ricardo André Gomes Petronilho(a81744)

5 janeiro, 2018

**Índice**

[Introdução 4](#_Toc534150879)

[Modelo de Domínio 5](#_Toc534150880)

[Diagrama de Use Case 7](#_Toc534150881)

[Especificação textual dos Use Cases 9](#_Toc534150882)

[Use Case Criar Configuração Individualizada 9](#_Toc534150883)

[Use Case Criar Configuração Ótima 10](#_Toc534150884)

[Use Case Iniciar Produção 11](#_Toc534150885)

[Use Case Atualizar Stock 12](#_Toc534150886)

[Use Case Autenticar no Sistema 12](#_Toc534150887)

[Protótipo da Interface Gráfica 13](#_Toc534150888)

[Login 13](#_Toc534150889)

[Menu de Administrador 13](#_Toc534150890)

[Adicionar novo utilizador 13](#_Toc534150891)

[Remover utilizador 14](#_Toc534150892)

[Menu de Funcionário da Fábrica 14](#_Toc534150893)

[Iniciar produções 14](#_Toc534150894)

[Adicionar stock 15](#_Toc534150895)

[Menu de Funcionário do Stand 15](#_Toc534150896)

[Criar Configuração 16](#_Toc534150897)

[Criar Configuração Ótima 16](#_Toc534150898)

[Diagrama de estados 17](#_Toc534150899)

[Estado não autenticado, efetuar login e confirmação 17](#_Toc534150900)

[Estado autenticado no sistema 18](#_Toc534150901)

[Diagrama de sequência de Sistema 19](#_Toc534150902)

[Use case Criar Configuração Individualizada 19](#_Toc534150903)

[Use case Iniciar Produção 22](#_Toc534150904)

[Use case Criar configuração Ótima 22](#_Toc534150905)

[Use Case Adicionar Stock 23](#_Toc534150906)

[Diagrama de sequência com subsistemas 25](#_Toc534150907)

[Diagrama de *package* 26](#_Toc534150908)

[Diagrama de classe com estruturas de dados e com ORM 27](#_Toc534150909)

[Implementação em Java 29](#_Toc534150910)

[Conclusões 31](#_Toc534150911)

**Resumo**

No âmbito da Unidade Curricular de Desenvolvimento de Sistemas de Software, foi proposto a realização de uma aplicação, desde a sua planificação, feita através de modelos UML, até à sua implementação em Linguagem Java.

Numa primeira fase, foi apresentada a análise de requisitos, correspondente à fase da planificação do projeto, na qual foi inicializada a modulação.

Já numa segunda fase, foi elaborada a finalização da modulação, criação de uma base de dados e a implementação em linguagem Java.

# Introdução

Este projeto consiste na elaboração de uma aplicação *desktop* capaz de possibilitar a encomenda de um carro novo num *stand*.

Esta aplicação será maioritariamente utilizada pelo funcionário do *stand* que, junto do cliente, irá configurar o carro e proceder à sua encomenda.

Durante o processo de configuração, o utilizador seguirá uma sequência de fases, nas quais o cliente dirá quais são os itens desejados.

No entanto, alguns desses itens poderão não ser compatíveis com a configuração atual e poderá ser necessário adicionar ou remover um ou vários componentes. A decisão da adição de um ou vários componentes necessários ou remoção de um ou vários componentes incompatíveis para a adição do pretendido é exclusivamente do cliente.

A secção da escolha de pacotes é constituída por pacotes pré-definidos. Não será possível optar apenas por alguns dos componentes incluídos do mesmo. A seleção do pacote requer a integração total dos componentes que o constituem. Caso exista alguma incompatibilidade da configuração atual com o pacote selecionado e o cliente não aceite a proposta de resolução desse problema, o pacote não poderá ser aplicado.

O cliente tem a possibilidade de optar entre dois tipos de configuração – a configuração individualizada e a configuração ótima.

A configuração individualizada procede a uma sequência de fases ordenada pela escolha do modelo, da motorização, de componentes de exterior, de componentes de interior e de pacotes. Em todas estas fases, caso haja incompatibilidade com a configuração atual e o cliente não aceite, a configuração mantém-se inalterada, isto é, mantém-se com os mesmos componentes que o mesmo escolheu anteriormente sem o componente incompatível. Se este processo ocorrer corretamente, é feita a verificação da existência de pacotes e, de seguida, a configuração é submetida para a fábrica, pronta para ser produzida.

Já a configuração ótima procede de forma diferente. Nesta, o cliente apenas informa ao funcionário do *stand* o modelo desejado e o montante máximo que está disposto a pagar pelo carro. Após transmitir essas informações, é gerada uma proposta. De referir que o valor proposto tem de ser superior ao mínimo estabelecido.

# Modelo de Domínio

Para a realização do modelo de domínio, numa primeira fase, identificaram-se os principais intervenientes do sistema, sendo estes o **cliente/comprador**, o **utilizador/funcionário do *stand***, o **utilizador/funcionário da fábrica e o administrador**. O **utilizador do *stand*** é o intermediário entre o **comprador** e o **utilizador da fábrica**.

Depois de identificados os atores e percecionadas as suas relações, foi necessário interpretar a interação que iria ser feita entre o cliente, o *stand* e a fábrica.

Pode-se constatar que é o cliente que escolhe a configuração que será enviada posteriormente para a fábrica, onde esta é realizada. Contudo, entre estas duas entidades existe um intermediário que faz o pedido, e o envia para a fábrica, o utilizador do *stand*.

Assim, os atores com funcionalidades no sistema serão o funcionário do *stand*, o funcionário da fábrica e o administrador. Especificamente:

* O administrador tem a incumbência de gerir os funcionários.
* O funcionário do stand é responsável pela geração de uma configuração.
* O funcionário da fábrica está encarregue de iniciar a produção de uma configuração e de atualizar o stock de um determinado componente.

Estes terão um identificador(*id*) e credenciais, de forma a permitir a autenticação na interface gráfica.

Na modulação, considerou-se que uma configuração é personalizada pelo modelo do carro, motorização, exterior e interior. Esta pode conter também pacotes. Como referido anteriormente, pode ser individualizada ou ótima. No caso de existência de pacote, o cliente usufruirá de um desconto.

A prestação deste serviço implica um pagamento de um determinado valor mediante o preço do modelo do carro e de cada componente integrado na configuração, bem como pela sua possível inclusão de pacote. Cada **componente** possui um ***stock****.*

De forma a ser possível validar a inserção de qualquer componente ou pacote à configuração em questão, considerou-se relevante que o componente possua componentes necessários e compatíveis consigo mesmo.

Após a configuração estar finalizada, esta é enviada para uma “fila de espera de configurações”.

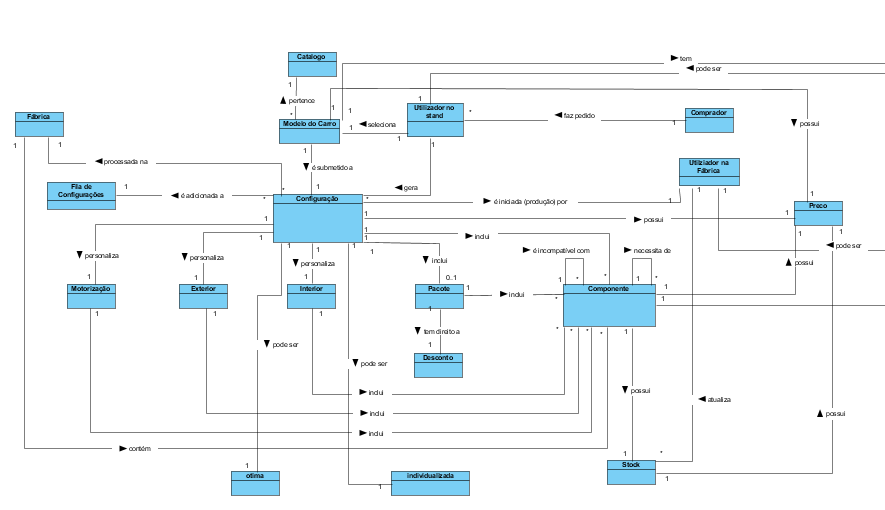


Figura 1 - Modelo de domínio

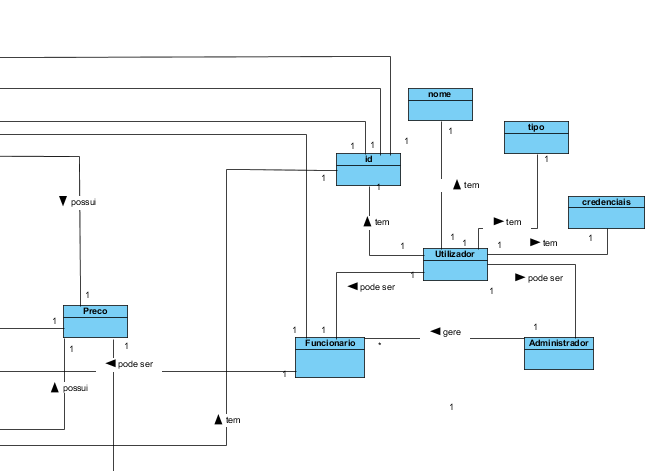


Figura 2 - Continuação do modelo de domínio

# Diagrama de Use Case

Após a conclusão do Modelo de Domínio, foram várias as considerações feitas durante o desenvolvimento do modelo Use Case.

Como referido anteriormente, verificou-se que deveriam existir três atores – o administrador, o utilizador da fábrica e o utilizador do *stand* (note-se que o comprador/cliente não interage diretamente com o sistema, logo não pode ser considerado um ator).

De seguida, associou-se a cada ator os use cases que se consideraram necessários. Todos os atores poderão autenticar-se no sistema.

O administrador será capaz de criar e remover funcionários. Já o utilizador do *stand* está habilitado a criar configurações, uma vez que é este que a executa, junto do cliente. No caso do utilizador da fábrica, este poderá iniciar a produção das configurações e atualizar o stock de um certo componente.

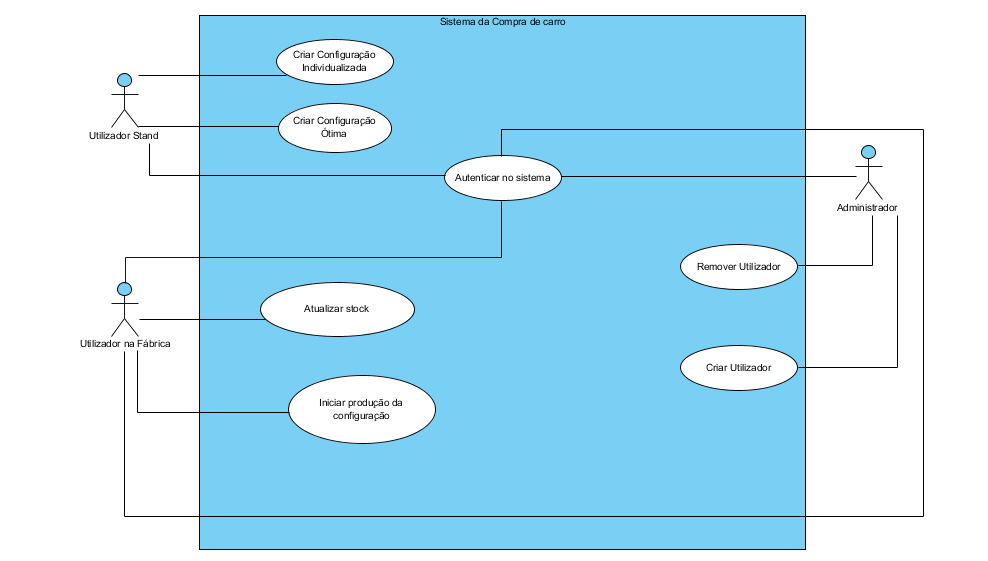


Figura 3 - Diagrama de Use Case implementado

Numa perspetiva de continuação do desenvolvimento desta aplicação no futuro, sugere-se o seguinte diagrama de use cases mais completo:

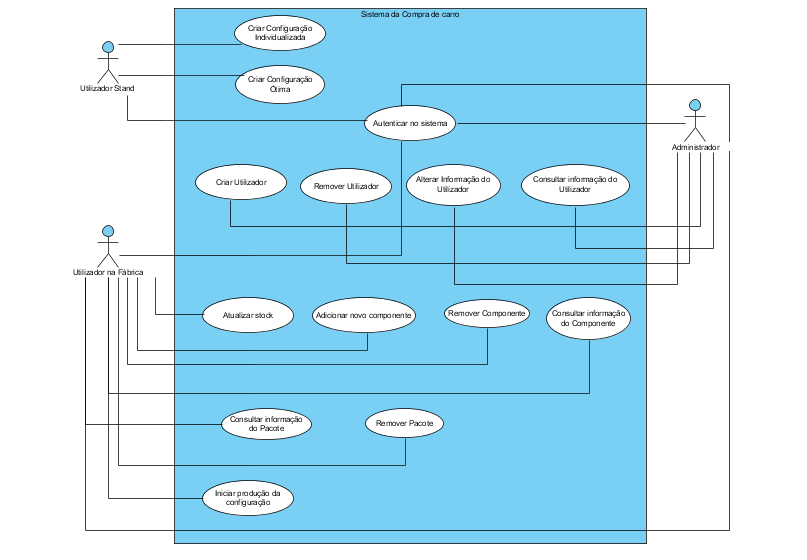
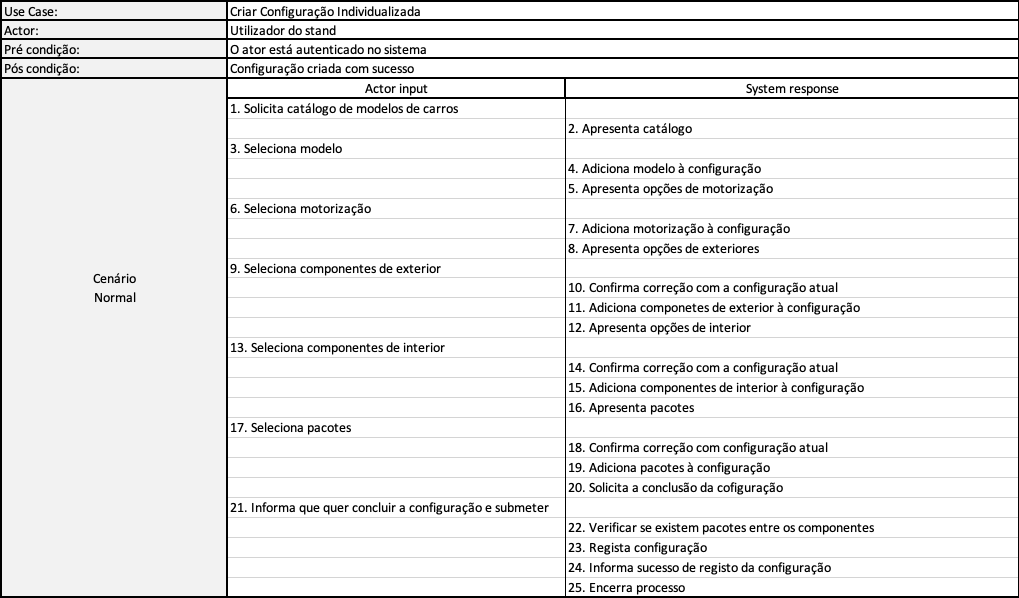
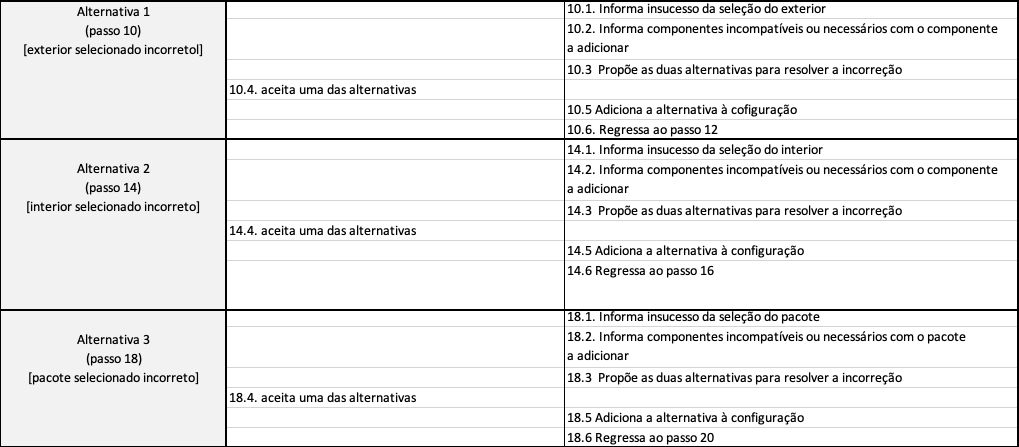


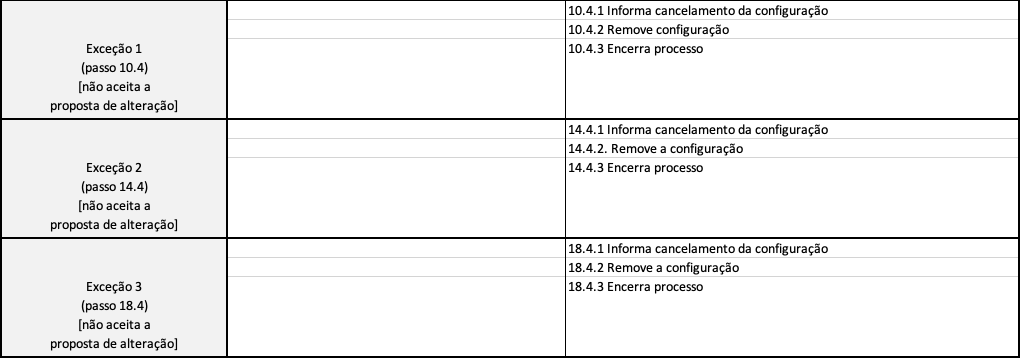
Figura 4 - Sugestão de diagrama de Use Case a implementar

# Especificação textual dos Use Cases

## Use Case Criar Configuração Individualizada







O use case especificado acima, tem como objetivo criar uma configuração individualizada(normal), percorrendo todas as categorias de componentes que podem constituir a mesma, garantindo que esta não contém incompatibilidades entre si e com todos os seus necessários.

Inicialmente o ator, utilizador do stand, solicita a apresentação do Catálogo, ou seja, os modelos disponíveis. As escolhas são sempre efetuadas pelo cliente, no entanto como quem interage com a aplicação é o utilizador do stand este seleciona o modelo, sendo que o sistema adiciona o mesmo à configuração.

De seguida, o sistema apresenta todos os motores compatíveis com este modelo, e o utilizador seleciona o motor pretendido, e este é adicionado à configuração por parte do sistema.

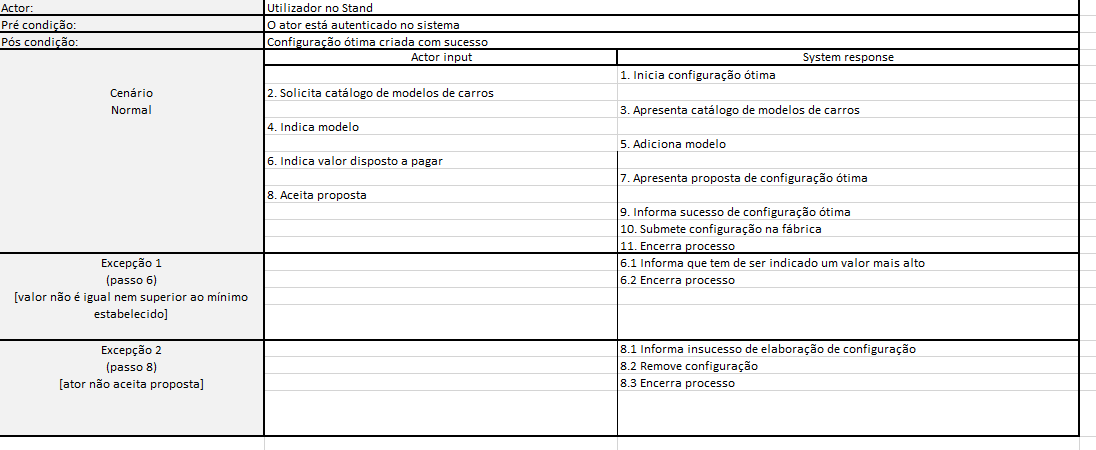
Após o passo anterior, são apresentados todos os componentes exteriores compatíveis com o modelo da configuração, o ator seleciona os exteriores que decidir, sendo que neste passo difere do anterior, pois é necessário verificar as incompatibilidades e os componentes necessários para cada componente selecionado. Para a resolução de incompatibilidades tem-se a Alternativa 1 que informa o insucesso da adição dos componentes e propõe duas alternativas para resolver. Por fim o ator escolhe uma das alternativas e estas são adicionadas à configuração.

Caso não haja incompatibilidades ou componentes necessários, o componente é adicionado normalmente à configuração.

O processo anteriormente descrito, é repetido do mesmo modo para a seleção de componentes do tipo interior, e pacotes.

Por fim, solicita-se o ator da conclusão da configuração, e após a confirmação da mesma, verifica-se a existência de pacotes por parte dos componentes adicionados, e em caso afirmativo é aplicado o desconto do mesmo, e submete-se à fábrica.

## Use Case Criar Configuração Ótima



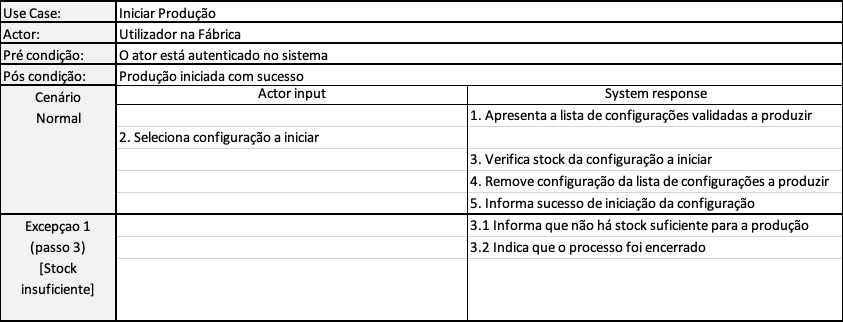
O Use Case Criar Configuração Ótima permite gerar uma proposta de configuração face a um modelo e a um preço apresentado.

Nesse intuito, primeiramente, solicita-se o catálogo de modelos de carros disponíveis.

Após a seleção do modelo, o sistema adiciona-o à configuração. No seguimento, o ator indica o valor disposto a gastar. Caso o valor seja superior ao mínimo estabelecido, o sistema apresenta uma proposta. Caso contrário, o sistema informa que tem de ser introduzido um valor superior.

Se o a ator aceitar a proposta, esta é submetida para a fábrica e está pronta para ser produzida. Caso contrário, ocorre insucesso e a configuração é removida e, consequentemente, o processo é encerrado.

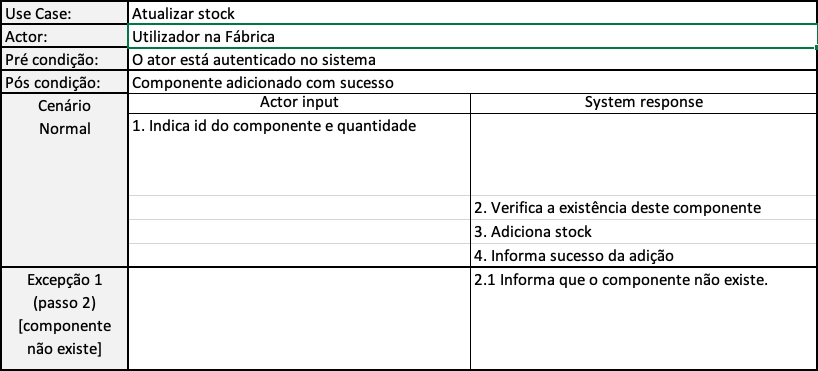
## Use Case Iniciar Produção

****

Inicialmente, o sistema apresenta ao ator as configurações que estão prontas a ser produzidas, ou seja, aquelas que foram submetidas para a fábrica, no entanto é importante referir que as mesmas que são apresentadas, podem não ter stock disponível.

O utilizador da fábrica seleciona uma das configurações, após este passo o sistema verifica se existe stock para todos os componentes desta configuração, caso não haja, é lançada a Exceção 1 [Stock insuficiente], que por sua vez informa o ator de que não existe stock e encerra o processo. Caso não ocorra exceção, o sistema remove esta configuração e informa o ator que o início de produção desta configuração foi realizado com sucesso.

## Use Case Atualizar Stock



O use case acima é responsável pela atualização do stock de um componente, começando pela indicação por parte do ator, utilizador da fábrica, o id do componente e a respetiva quantidade a adicionar, de seguida o sistema verifica se o id inserido tem componente correspondente, isto é, se existe na base de dados. Após esta verificação, o stock é adicionado/atualizado e o sistema informa o ator do sucesso da adição. Caso o componente não exista na base de dados, é lançada a Exceção 1 a informar o ator de que o componente não existe.

## Use Case Autenticar no Sistema

# 

# Protótipo da Interface Gráfica

## Login

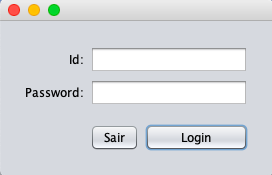


Figura 5 - Login

## Menu de Administrador

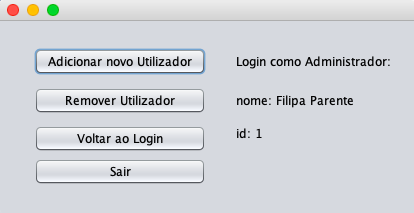


Figura 6 - Menu de Administrador

## Adicionar novo utilizador

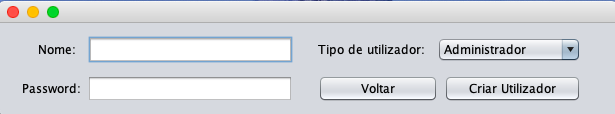


Figura 7 - Adicionar novo utilizador, funcionalidade do Menu de Administrador

## Remover utilizador

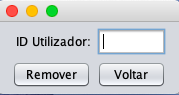


Figura 8 - Remover utilizador, funcionalidade do Menu de Administrador

## Menu de Funcionário da Fábrica

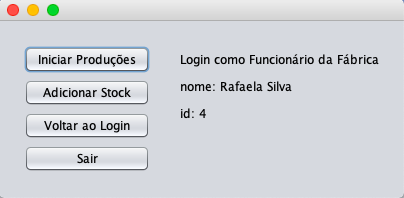


Figura 9 - Menu de Funcionário da Fábrica

## Iniciar produções

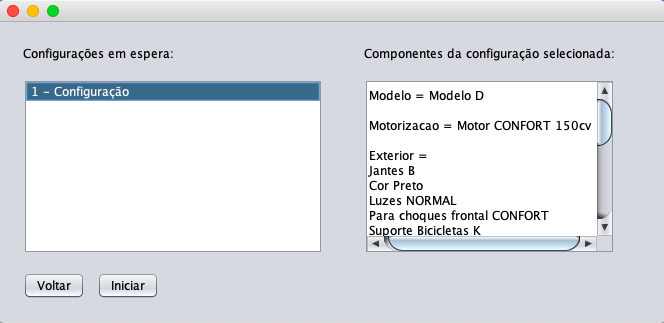


Figura 10 - Iniciar produções, funcionalidade do Menu de Funcionário da Fábrica

## Adicionar stock

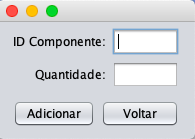


Figura 11 - Adicionar stock, funcionalidade do Menu de Funcionário da Fábrica

## Menu de Funcionário do Stand

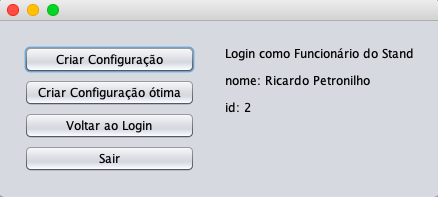


Figura 12 - Menu de Funcionário do Stand

## Criar Configuração

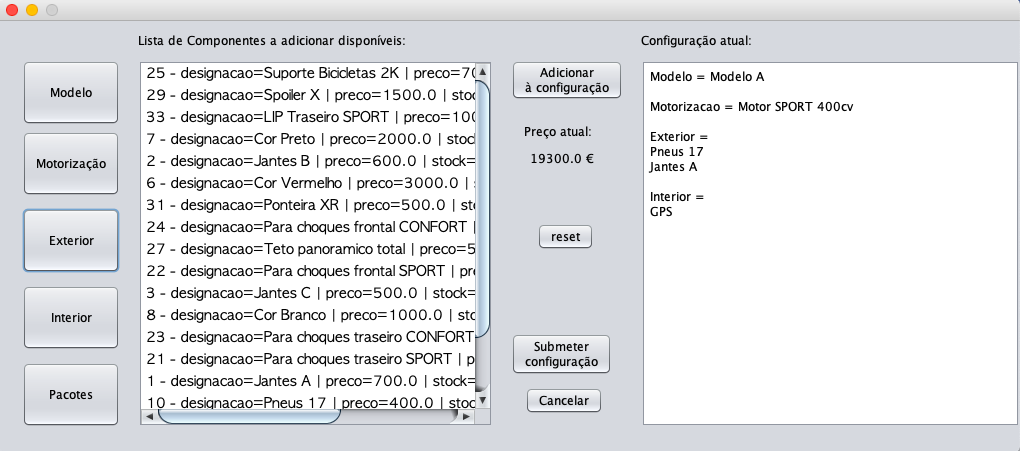


Figura 13 - Criar configuração, funcionalidade do Menu de Funcionário do Stand

## Criar Configuração Ótima

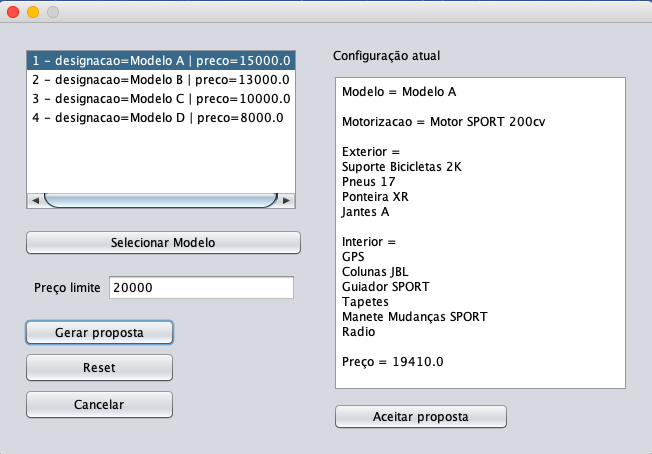


Figura 14 - Criar configuração ótima, funcionalidade do Menu de Funcionário do Stand

# Diagrama de estados

## Estado não autenticado, efetuar login e confirmação

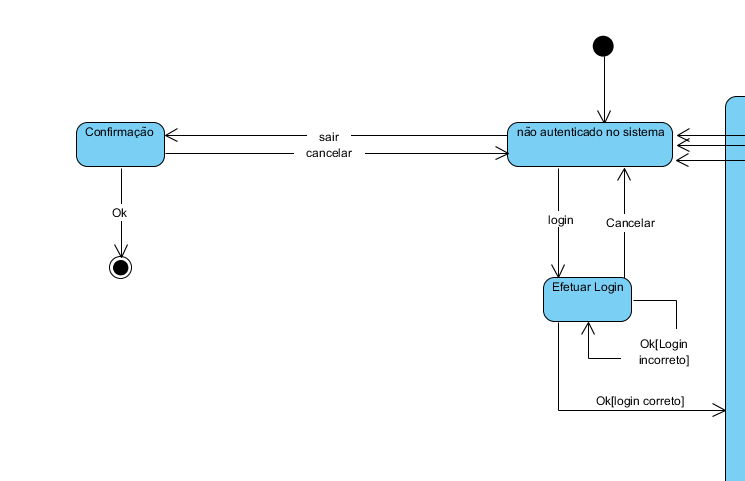


Figura 15 - Estado não autenticado, efetuar login e confirmação

O estado inicial de qualquer utilizador no sistema é de **não autenticado no sistema**, existindo duas opções de saída deste estado: sair para o estado de **Confirmação**, que questiona o utilizador se deseja sair ou não do sistema, e a segunda opção é sair para o estado **Efetuar Login**. No estado de Confirmação, o utilizador tem duas possíveis saídas: voltar ao estado **não autenticado no sistema** ou sair do sistema, ou seja, estado final.

No **Efetuar Login**, existem três possibilidades de saída deste estado: a saída para o estado anterior, denominado não autenticado no sistema, sendo este acontecimento, quando o utilizador cancela o login, manter-se no mesmo estado, após a falha do login, e por fim, sair para o estado Autenticado no sistema, que ocorre quando o utilizador faz o login correto.

## Estado autenticado no sistema

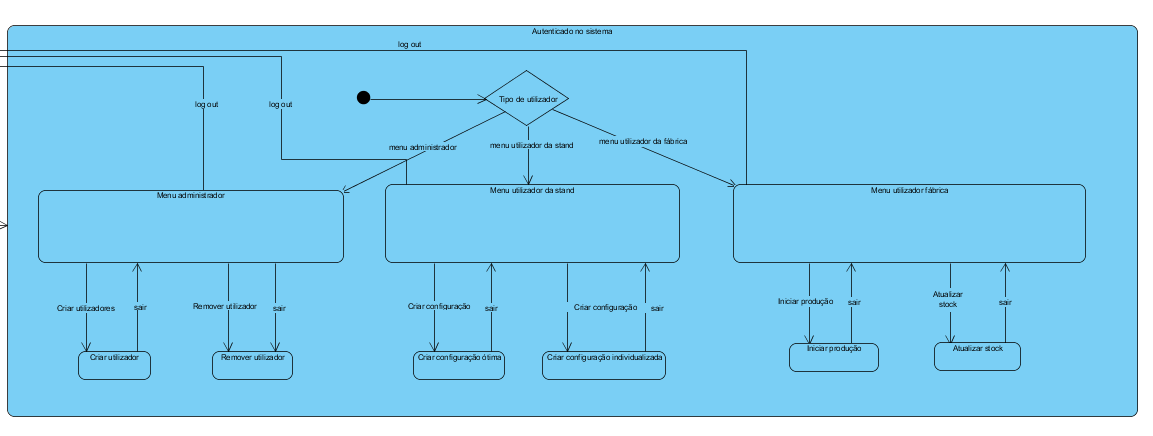
****

Figura 16 - Visão geral do estado Autenticado no sistema

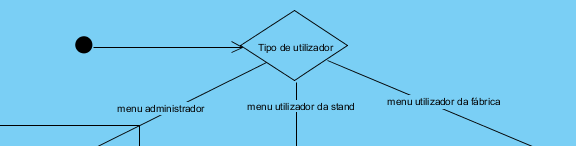


Figura 17 - Três tipos de menus principais diferentes

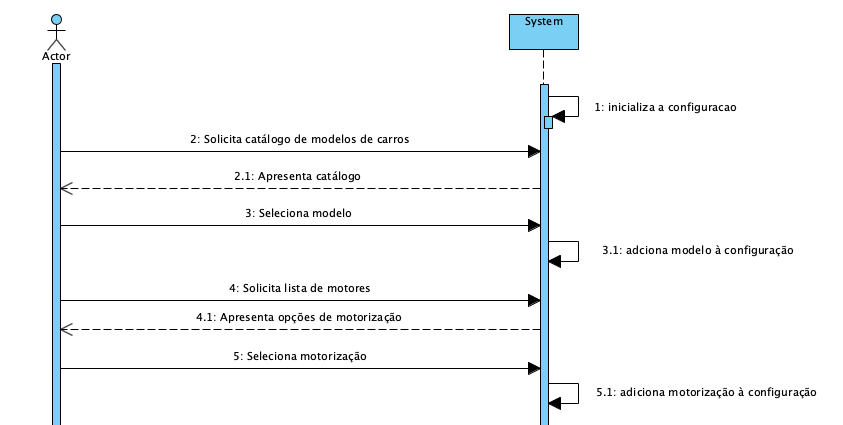
O estado **Autenticado no sistema** é composto por outros estados como se pode ver nas figuras acima, sendo assim um superestado. Deste modo, o seu estado inicial tem três possibilidades: **Menu administrador**, **Menu utilizador da stand** e **Menu utilizador da fábrica**, sendo esta escolha baseada no tipo de utilizador que está no sistema.

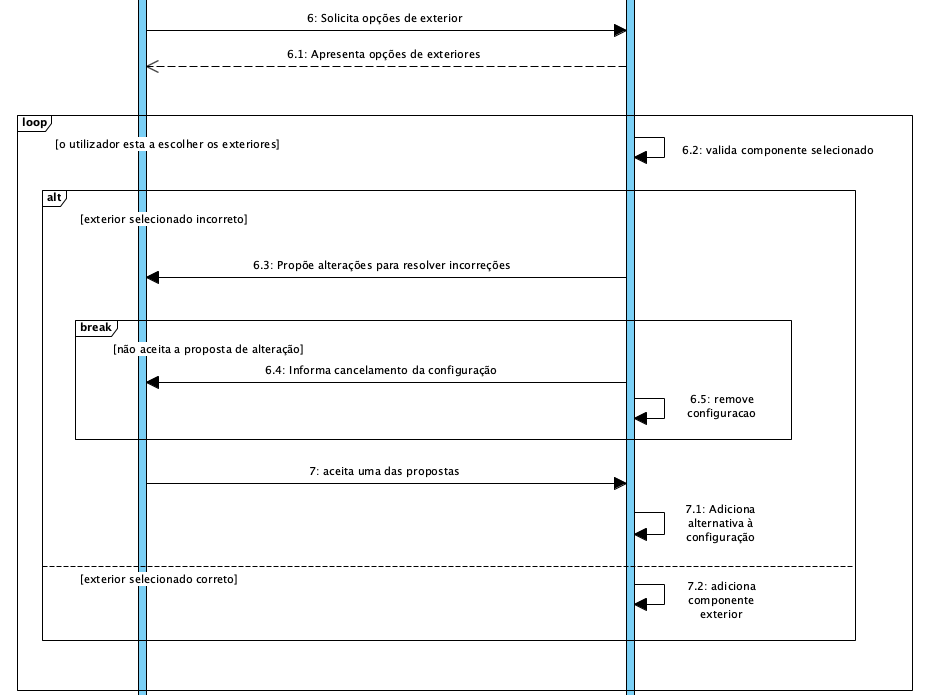
# Diagrama de sequência de Sistema

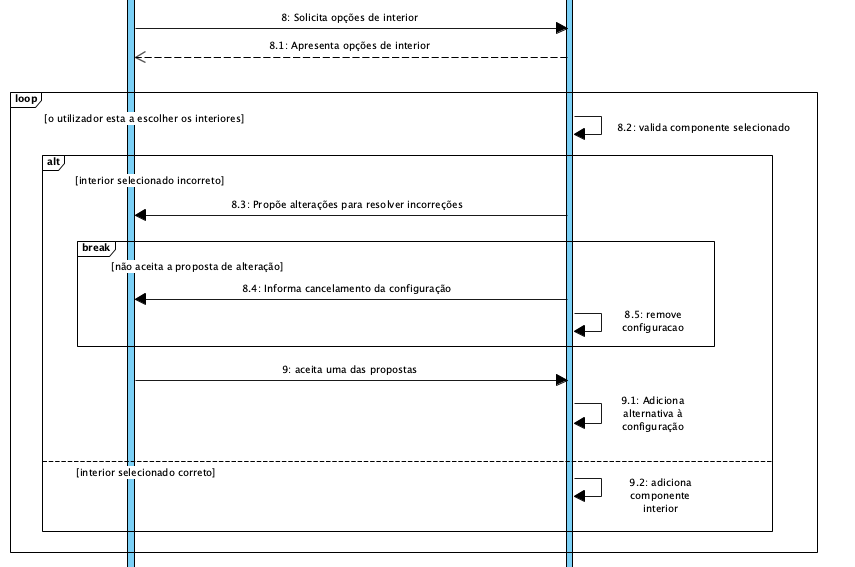
A criação do diagrama de sequência de sistema é totalmente mecânico caso o Use Case seja anteriormente, bem especificado. No entanto, tal não aconteceu em certas partes da especificação de cada Use Case.

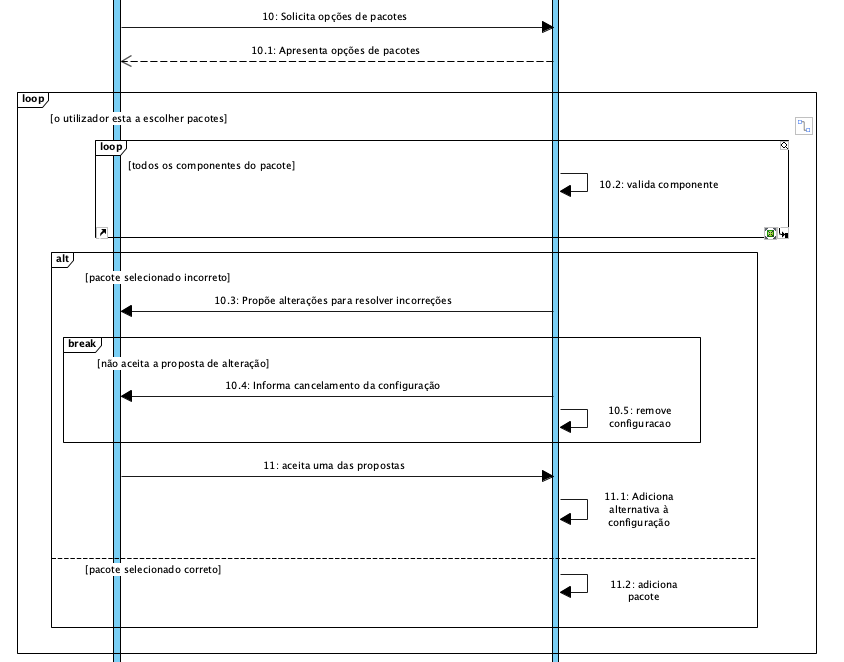
Assim, durante a conceptualização do diagrama de sistema reparamos que certos procedimentos não faziam sentido, mesmo cronologicamente, concluímos que o erro foi na especificação do Use Case.

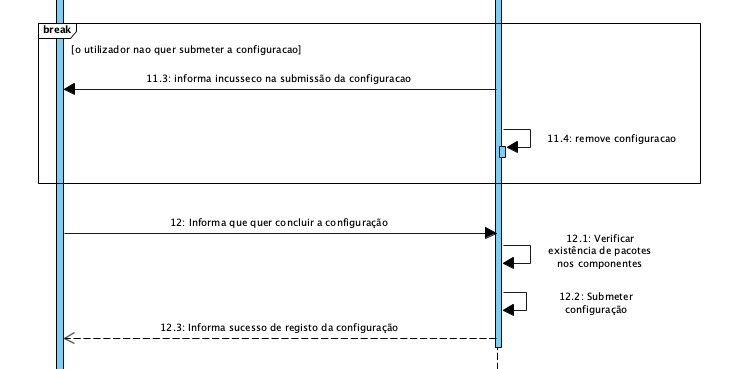
## Use case Criar Configuração Individualizada







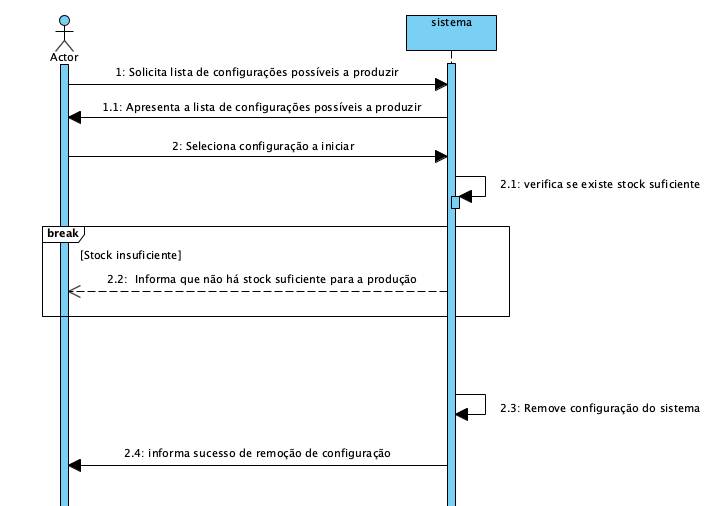




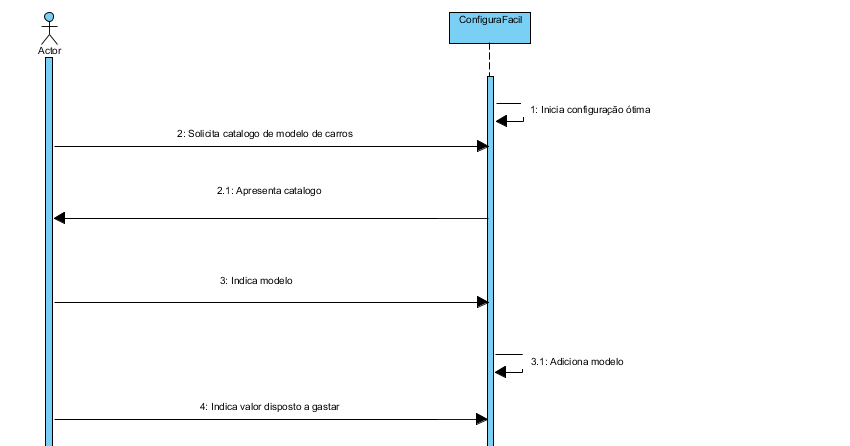
Note-se que a cronologia apresentada neste diagrama corresponde a uma situação ideal da seleção de cada categoria, isto é, primeiramente o utilizador seleciona o modelo, de seguida motor, exteriores, interiores e por último dos pacotes.

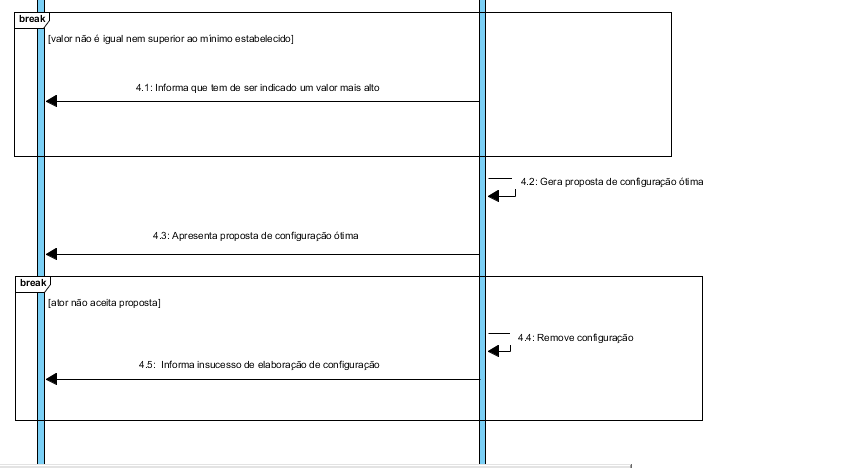
No entanto, o programa permite por exemplo, o utilizador estar na fase de seleção dos pacotes, e voltar a selecionar os componentes de exterior. A seleção do modelo e o motor são as únicas fases irreversíveis, desta forma caso o utilizador pretenda voltar a selecionar os mesmos necessitam de reiniciar a configuração.

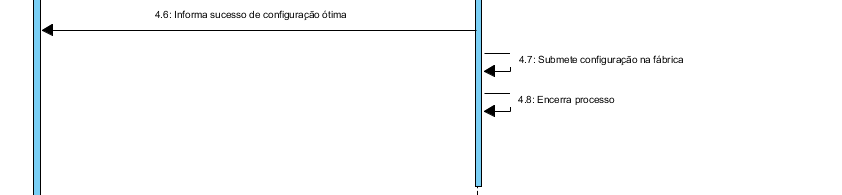
## Use case Iniciar Produção



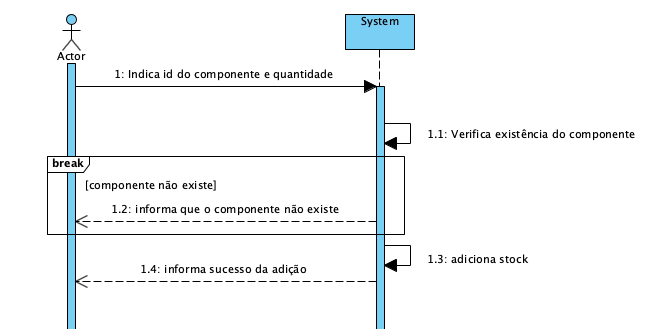
## Use case Criar configuração Ótima





****

## Use Case Adicionar Stock



# Diagramas de sequência com Facade, subsistemas e de implementação

Devido à dimensão dos mesmos, apresentamos os respetivos diagramas de Facade, subsistemas e implementação externo a este relatório nos respetivos ficheiros.vpp.

# Diagrama de *package*

De forma a obter gestão significativa das classes que irão surgir, decidiu-se criar um diagrama de *package* para agrupar as classes (em *packages*).

Assim, após a dedução dos subsistemas do nosso projeto através do diagrama de sequência, conseguiu-se identificar as dependências entre os *packages* considerados necessários – *presentation*, *data* e *business*. O package business por sua vez inclui dois packages, o comercial e o recursoshumanos.

Na perspetiva da evolução do projeto e criação de código sustentável foram criadas cinco classes Facade, uma por cada package. A existência de uma Facade por package é essencial, uma vez que garante abstração de implementação para quem usa a API do mesmo. Por exemplo, os métodos da classe Facade do package business ou mesmo as classes dentro do package podem ser alteradas, sem causar danos ao código externo de quem a usa, devido à existência de uma API constante no Facade.

Uma vez que a implementação do programa em formato MVC implica a existência de um Singleton que contenha a lógica do programa e de um package presentation que controle a UI, os packages com mais impacto no programa são o business e o presentation, por sua vez comunicam através dos respetivos Facade, SG e JLogin.

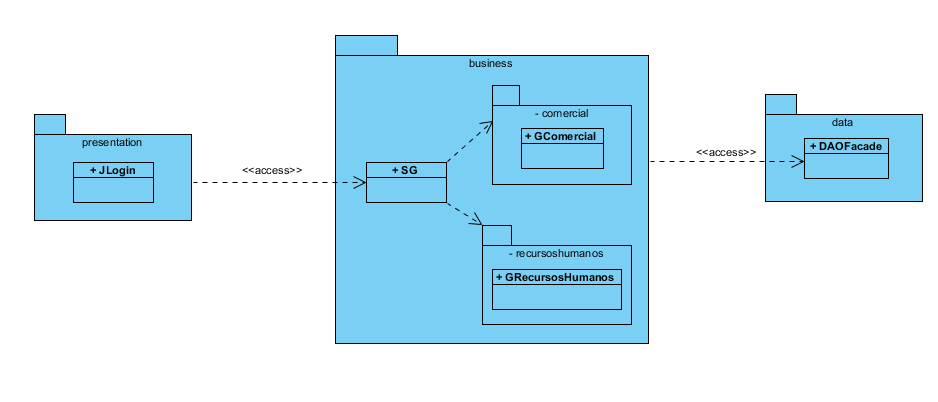


Figura 18 - Diagrama de *package*

# 

# Diagrama de classe com estruturas de dados e com ORM

Na figura seguinte apresenta-se o diagrama de classes com as estruturas de dados em memória:

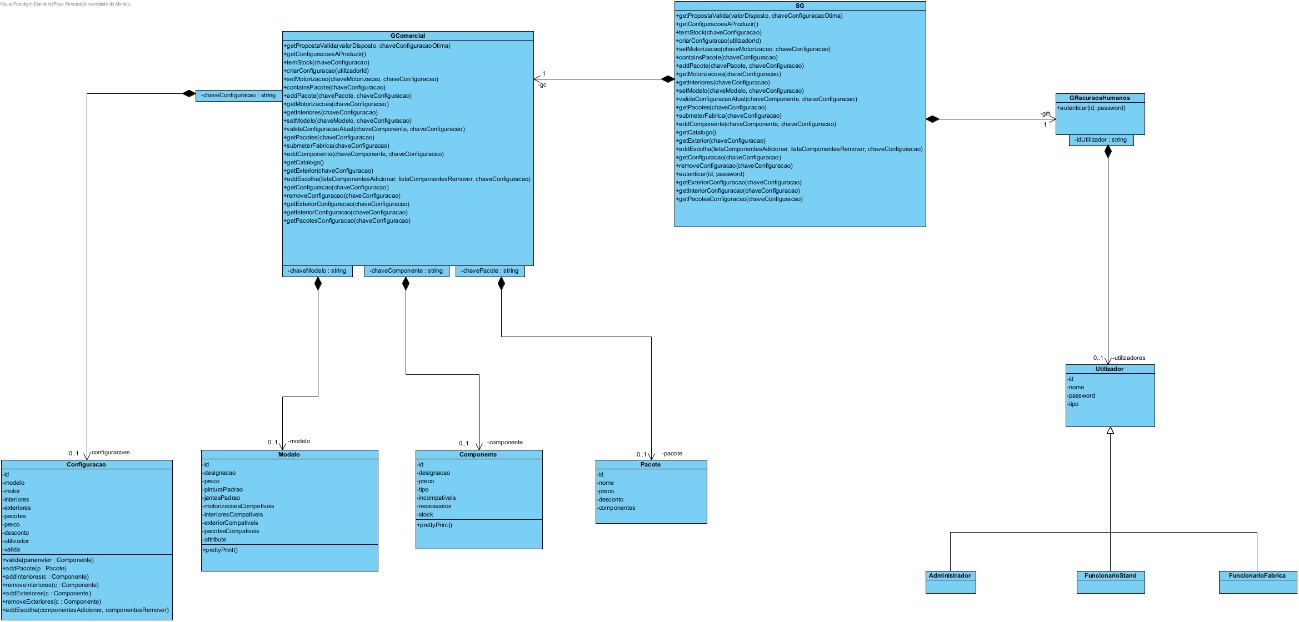


Figura 19 - Diagrama de *classes com estrutura de dados em memória*

A transição para o diagrama ORM baseou-se na transformação de todas as estruturas, existentes na classe Facade para DAO’s (Data Access Objects), distintas para cada uma, como se pode observar na imagem seguinte.

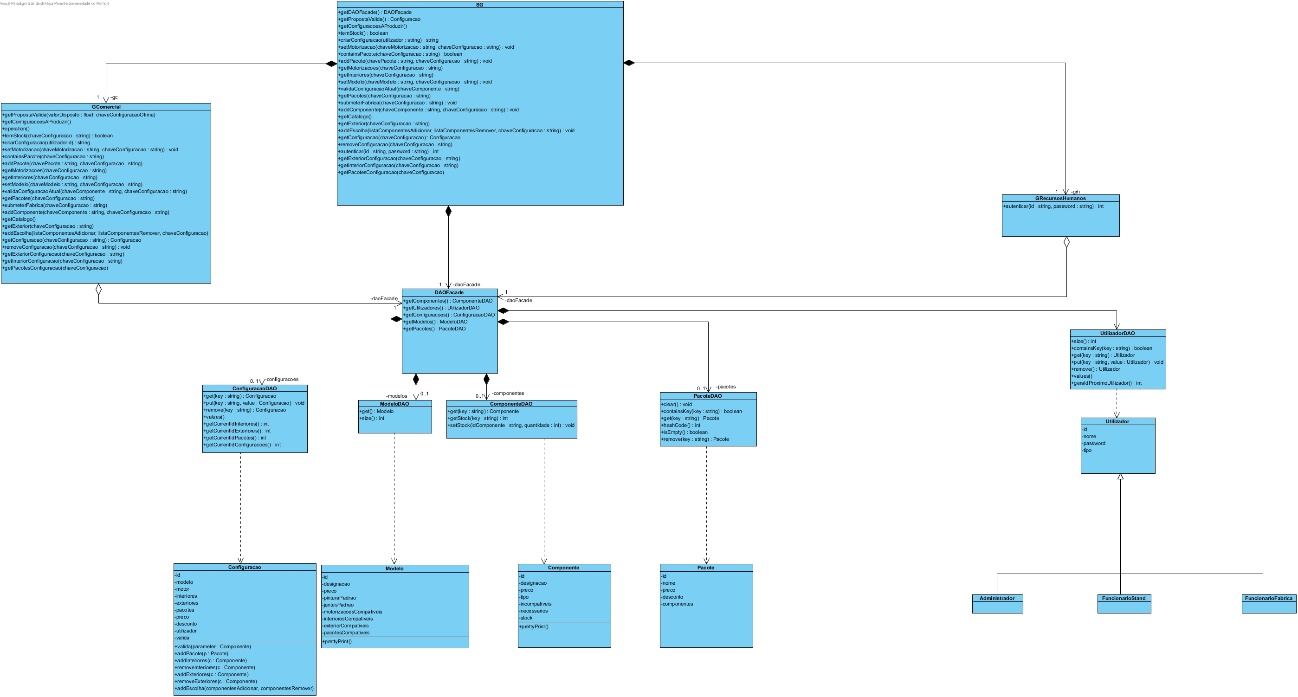


Figura 20 - Diagrama de *classes com ORM(“Object Relational Mapping”)*

# Implementação em Java

Depois da modulação estar consolidada e validada, decidiu-se proceder à implementação em linguagem Java.

No intuito de facilitar a implementação da interface, recorremos ao *NetBeans*. Neste, tentamos dar primazia à aplicação da modulação realizada.

Inicialmente, começou-se por criar as Classes Componente, Configuração, Modelo, Pacote, Utilizador, Administrador, FuncionarioStand, FuncionarioFabrica, sendo que para cada classe implementamos todas as convenções aprendidas em POO, isto é, todos os métodos gets , sets, clone, equals, toString() e respetivos construtores.

As variáveis mais importantes a referir na classe Componente e que já foram referidas no diagrama de classes é a collection que guarda os ids dos componentes incompatíveis com o componente e os respetivos necessários.

No caso do Modelo, também são guardados os ids dos componentes de cada categoria e pacotes, compatíveis com o modelo.

Na Configuracao foi usada a variável de instância valida que indica se a Configuração já foi submetida para fábrica.

Após este processo, implementamos a interface Map para cada classe referida acima, ComponenteDAO, ConfiguracaDAO, ModeloDAO, PacoteDAO, e no caso do Utilizador como é superclasse de Administrador, FuncionarioStand, FuncionarioFabrica, criou-se a classe UtilizadorDAO.

Neste processo foram tomadas decisões face aos diagramas definidos anteriormente, pois não era necessário implementar todos os métodos da interface Map, sendo que os diagramas facilitaram essa escolha, pois conseguiu-se verificar os métodos necessários.

Os dados são guardados num servidor de base de dados, e cada classe DAO está conectada ao mesmo, permitindo guardar e buscar dados.

De seguida, criou-se a classe DAOFacade, que é usada para comunicar com o exterior do pacote data. Do mesmo modo, criou-se a classe SG que é a classe Facade do pacote business, e que implementa todos os métodos necessários para a lógica da aplicação, sendo que nos baseamos no diagrama de packages anteriormente referido.

Depois do processo anterior estar concluído e funcional, começou-se por definir os métodos necessários para os use cases referenciados anteriormente, seguindo os diagramas de sequência de implementação de cada um. Por fim, criou-se o pacote presentation, onde a classe JLogin é o Facade. Neste pacote, implementou-se os Menus para cada tipo de utilizador e janelas para cada use case. É importante realçar que se tentou respeitar ao máximo as convenções apresentadas no início do semestre, onde na classe Facade (JLogin), tem-se o Singleton que é a classe Facade do business, e este é enviado como argumento a cada classe da user interface, permitindo assim que haja um controlo do acesso aos métodos do business.

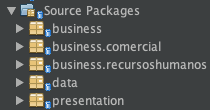


Figura 21 – Estrutura do projeto em *NetBeans*

Para o acesso a uma base de dados, utilizamos o *MySQL Workbench*, no qual criamos um modelo de base de dados adequado ao sistema.

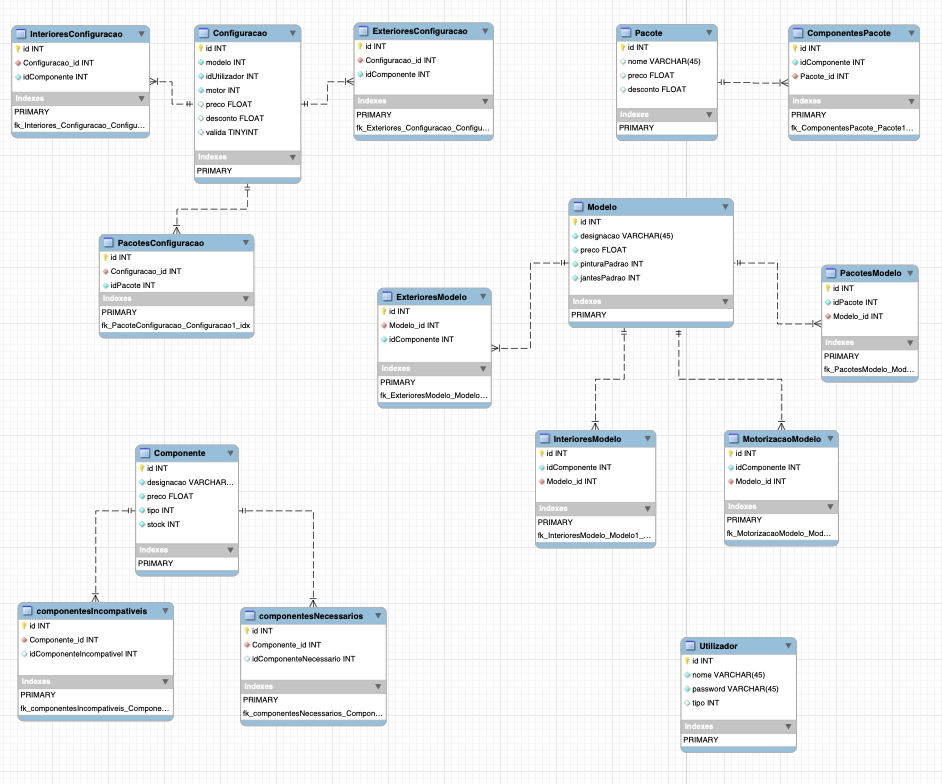


Figura 22 - Modelo lógico

# Conclusões

Ao longo das duas fases, surgiram alguns entraves no desenvolvimento do Modelo de Domínio e do Modelo de *Use Case*, uma vez que estes necessitaram de diversas revisões até à validação final.

Relativamente à elaboração do Modelo de Domínio, na primeira fase, começamos por elaborar um modelo simples, o que nos desafiou a alcançar um cariz mais complexo, quando nos apercebemos de que faltava considerar certas entidades, como a fila de espera de configuração e algumas das secções de configuração(exterior, interior e motorização), por exemplo. Para além disso, levantaram-se algumas questões acerca da multiplicidade da relação entre algumas entidades. Estas foram esclarecidas entre o grupo e com o auxílio do docente.

Na segunda fase, apercebemo-nos que, de certa forma, complicamos, quando admitimos que o funcionário da fábrica criasse pacotes e que existisse um fornecedor a fornecer o stock, por exemplo. Assim, tivemos de proceder a algumas alterações no modelo de domínio, no modelo de *Use Case* e no diagrama de estado.

No que diz respeito à conceção do Modelo de Use Case, na primeira fase, inicialmente atribuímos ao utilizador no *stand* a denominação de utilizador, o que causou bastante ambiguidade no desenvolvimento da especificação dos *Use Cases*. Além disso, nas especificações de *Use Cases* com mais de um ator, no desenvolvimento da interação com o sistema não nos referíamos a apenas ator, o que nos fez alterar alguns dos *Use Cases*.

Na segunda fase, apercebemo-nos também de que não iríamos ter tempo suficiente para elaborar todos os que tínhamos inicialmente ponderado. Devido a esse facto, apresentamos no relatório uma sugestão do que gostaríamos de ter implementado, visto que só implementamos os que consideramos imprescindíveis e necessários para corresponder aos requisitos das funcionalidades do sistema.

Quanto à criação do protótipo da interface, na primeira fase, tentamos dar primazia à coerência com o modelo de uses cases definido, o que foi algo desafiador também, pois inconscientemente pensávamos em adicionar funcionalidades que não estavam especificadas anteriormente.

O diagrama de estado suscitou algumas dúvidas que tentamos esclarecer, recorrendo ao material disponibilizado da unidade curricular.

No desenvolvimento do diagrama de sequência de implementação do Use Case Criar Configuração Ótima, especificamente no método getPropostaValida, ocorreram diversas modificações de conceptualização. Quando tentamos implementar o método, verificamos que não iríamos ser coerentes ao que inicialmente estava proposto. Por esse motivo, optamos por utilizar uma self message.

Na implementação em linguagem Java, fomos forçados a alterar o que tínhamos proposto inicialmente no protótipo da interface, visto que não implementamos todos os Use Cases propostos na primeira fase.