Arquiteturas de Software (4º ano de MIEI)

ESS Trading Platform

Relatório de Desenvolvimento (Entrega Final)

Luís Alves (a80165)

Rafaela Rodrigues (a80516)

30 de Novembro de 2019

Resumo

Este relatório inicia-se com uma breve contextualização, seguindo-se a declaração dos objetivos deste trabalho e em que é que este consiste. De seguida são listados todos os requisitos do sistema, seguindo-se um esquema de navegação que é acompanhado por *mockups* da implementação da interface com o utilizador. É também neste capítulo que são explicitados os atributos de qualidade escolhidos e respetivos cenários. Depois, são apresentados os diversos diagramas que foram elaborados para documentar a arquitetura final do sistema, desde os de estrutura aos de comportamento para cada uma das funcionalidades descritas no diagrama de casos de uso. É incluído um modelo físico da base de dados no fim deste capítulo. A seguir, surgem os padrões utilizados no sistema. É discutido o padrão arquitetural adotado e de seguida são listados e discutidos os padrões de desenho utilizados. Por fim, são apresentadas as alterações feitas ao sistema devido à inclusão do novo requisito em cada uma das 3 camadas. É também explicitado um exemplo de funcionamento do novo requisito.

Conteúdo

1 Introdução **6**

1.1 Contextualização 6

1.2 Objetivos e Trabalho Proposto 6

2 Análise e Funcionalidades **7**

2.1 Requisitos 7

2.1.1 Utilizador não registado 7

2.1.2 Negociador 7

2.1.3 Administrador 8

2.1.4 Outros 8

2.2 Esquema de Navegação 9

2.3 Atributos de Qualidade 14

2.3.1 Modificabilidade 14

2.3.2 Interoperabilidade 14

2.4 Condicionantes 15

3 Modelação **16**

3.1 Modelo de Domínio 16

3.2 Diagrama de *Use Cases* 17

3.3 Diagramas Comportamentais 17

3.3.1 Registar Negociador 18

3.3.2 Estabelecer CFD 18

3.3.3 Encerrar CFD 19

3.3.4 Adicionar Saldo 20

3.3.5 Atualizar Ativo 21

3.4 Diagrama de *Packages* 21

3.5 Diagrama de Classe 22

3.6 Modelo Físico da Base de Dados 23

4 Padrões **25**

4.1 Padrão Arquitetural - Camadas . . . 25
4.2 Padrões de Desenho . . . 25
4.2.1 *Facade* . . . 25
4.2.2 *Observer* . . . 26
4.2.3 *Data Access Object* . . . 28
4.2.4 *Template* . . . 28
4.2.5 *Factory* . . . 29

5 Novo Requisito **31**
5.1 Alterações ao Sistema . . . 31
5.1.1 Camada de Apresentação . . . 31
5.1.2 Camada de Negócio . . . 31
5.1.3 Camada de Persistência . . . 32
5.2 Reutilização de Funcionalidades . . . 32
5.3 Exemplo de funcionamento . . . 32

6 Conclusão **34**

A Diagrama de Classe (Primeira Fase) **35**

Lista de Figuras

2.1 Diagrama de Esquema de Navegação 9
2.2 Menu inicial 9
2.3 Login 10
2.4 Registar negociador 10
2.5 Página Inicial 10
2.6 Estabelecer CFD 11
2.7 Estabelecer CFD (continuação) 11
2.8 Consultar CFD 12
2.9 Terminar CFD 12
2.10 Consultar ativo 13
2.11 Adicionar saldo 13

3.1 Modelo de Domínio 16
3.2 Diagrama de *Use Cases* 17
3.3 Diagrama de Sequência: Registar Negociador 18
3.4 Diagrama de Sequência: Estabelecer CFD 19
3.5 Diagrama de Sequência: Encerrar CFD 20
3.6 Diagrama de Sequência: Adicionar Saldo 20
3.7 Diagrama de Sequência: Atualizar Ativo 21
3.8 Diagrama de *Packages* 22
3.9 Diagrama de Classe 23
3.10 Esquema físico da Base de Dados 24

4.1 Padrão Estrutural - *Facade* 26
4.2 Padrão Comportamental - *Observable* 27
4.3 Padrão Comportamental - *Observer* 27
4.4 Padrão Estrutural - *DAO* 28
4.5 Padrão Comportamental - *Template* no CFD 29
4.6 Padrão Comportamental - *Template* no Ativo 29
4.7 Padrão de Criação - *Factory* 30

5.1 Ativos subscritos 33
5.2 Notificação de um ativo subscrito 33

A.1 Diagrama de Classe da Primeira Fase 35

Lista de Tabelas

2.1 Cenário para Modificabilidade 14
2.2 Cenário para Interoperabilidade 14

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito da entrega final do Primeiro Trabalho Prático da Unidade Curricular de Arquiteturas de Software, que se insere no 1º semestre do 4º ano do Mestrado Integrado em Engenharia Informática.

1.2 Objetivos e Trabalho Proposto

Para a entrega final do trabalho, foi proposto o desenvolvimento de uma solução para uma plataforma de *trading* de ativos via CFDs (*Contract for Differences*), sendo necessária a definição das funcionalidades a implementar e a sua explicitação com recurso a diagramas UML (*Unified Modeling Language*).

Para além disso, foi proposta a escolha de dois atributos de qualidade que devem influenciar as decisões tomadas na definição da arquitetura da solução.

Por fim, deverá ser implementada a arquitetura proposta, dando resposta aos requisitos enunciados e satisfazendo os atributos de qualidade.

Capítulo 2

Análise e Funcionalidades

Neste capítulo são apresentados os diversos requisitos levantados e organizados de acordo com os atores do sistema, é apresentada uma proposta de esquema de navegação na aplicação, são apresentados os atributos de qualidade escolhidos e qual a sua relação com o padrão arquitetural selecionao e, por fim, são apresentadas algumas condicionantes ao projeto.

2.1 Requisitos

2.1.1 Utilizador não registado

1. É um utilizador que ainda não possui uma conta no sistema
2. Podem (e devem!) registar-se e, assim, tornam-se em Negociadores.

2.1.2 Negociador

1. Um Negociador é um utilizador registado no sistema.
2. Um Negociador deverá poder obter informação acerca de todos os CFDs que contratualizou e ainda não foram encerrados.
3. Um Negociador deverá poder consultar o valor dos ativos disponíveis no sistema.
4. Cada ativo deverá apresentar o valor correspondente a uma unidade desse mesmo ativo.
5. Um Negociador poderá aceder à sua conta utilizando o seu NIF e *password*.
6. Um Negociador pode adicionar e remover fundos da sua conta (adicionando e subtraindo saldo).

Estabelecer CFD

1. Um Negociador deve poder estabelecer um novo CFD.
2. Um CFD (*Contract for Differences*) é constituído por uma quantidade de um ativo com um determinado valor.

3. Um Negociador apenas poderá estabelecer CFDs cujo investimento seu saldo permita.
4. Um Negociador poderá estabelecer um CFD *Long* ou *Short*.
5. Um CFD pode ser terminado automaticamente, com recurso a limites impostos aquando do momento de criação do CFD.
6. Um Negociador pode definir um *Stop Loss*, que termina automaticamente o CFD caso este desvalorize até a um valor definido pelo Negociador.
7. Um Negociador pode definir um *Take Profit*, que termina automaticamente o CFD caso este valorize até a um valor definido pelo Negociador.
8. Caso o CFD seja de *Long*, então o mesmo valoriza caso o ativo escolhido valorize, e desvaloriza caso o ativo escolhido desvalorize.
9. Caso o CFD seja de *Short*, então o mesmo desvaloriza caso o ativo escolhido valorize, e valoriza caso o ativo escolhido desvalorize.

Encerrar CFD

1. Um Negociador pode terminar um CFD quando pretender.
2. Um Negociador pode ver o seu CFD ser terminado automaticamente, caso tenha definido limites de *Stop Loss* ou *Take Profit* aquando do estabelecimento do CFD.
3. Quando um Negociador termina o CFD, o seu valor é transformado em saldo que é adicionado à sua conta.
4. O valor de um CFD *Long* é calculado multiplicando a quantidade de ativos investida pelo valor atual do ativo.
5. O valor de um CFD *Short* é calculado duplicando o investimento inicial e subtraindo o número de unidades de ativo compradas pelo valor atual do ativo.

2.1.3 Administrador

1. O Administrador deve poder popular o sistema com ativos provenientes de uma fonte externa

2.1.4 Outros

1. O sistema deve atualizar regularmente o valor dos seus ativos.
2. O sistema deve fechar automaticamente os CFDs que ultrapassem os seus limites, quando atualizar o valor dos ativos.

2.2 Esquema de Navegação

Para ser possível antecipar o fluxo de navegação do programa, foi criado um esquema de navegação que se encontra na figura 2.1. É possível verificar que será a partir da Página Inicial que um Negociador poderá usufruir das funcionalidades mais importantes do sistema.

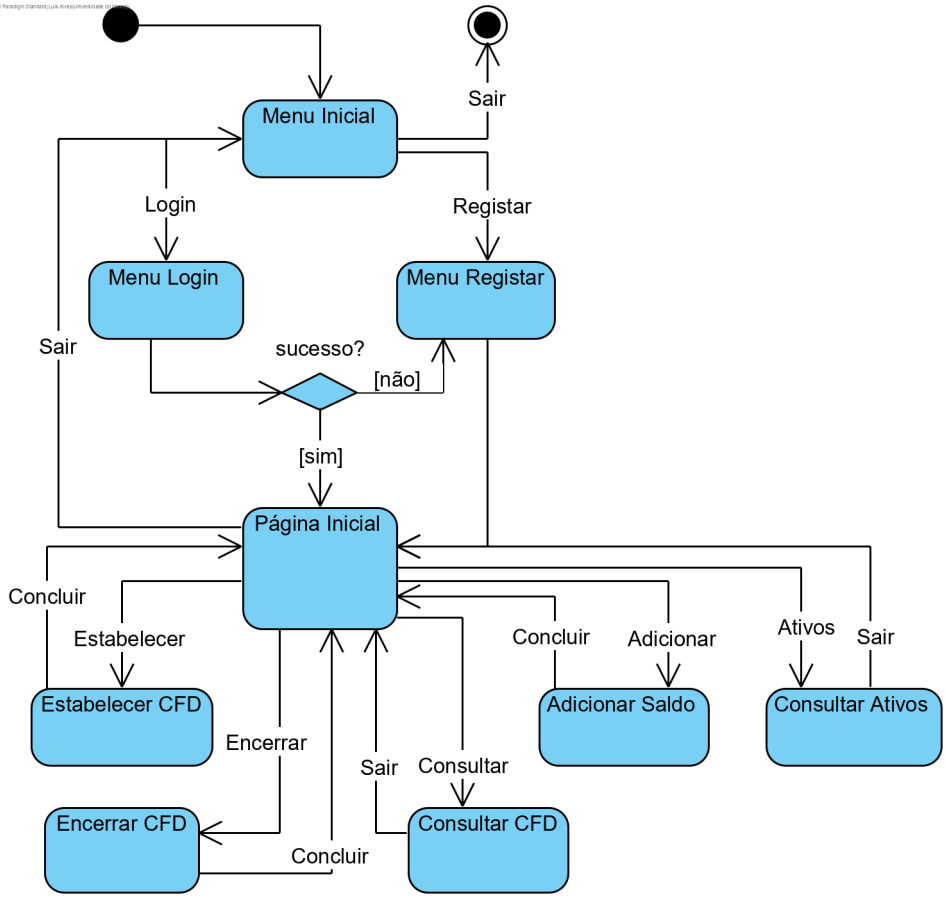

Figura 2.1: Diagrama de Esquema de Navegação

De seguida, apresentam-se os *Mockups* relativos a cada um dos Menus previstos acima.

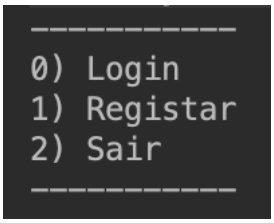

Figura 2.2: Menu inicial

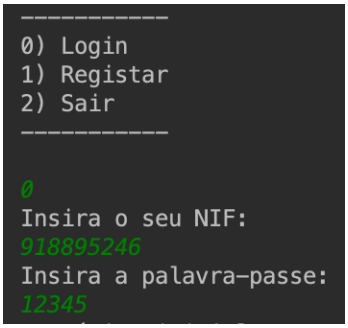

Figura 2.3: Login

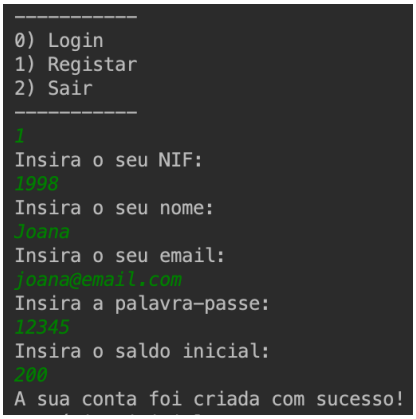

Figura 2.4: Registar negociador

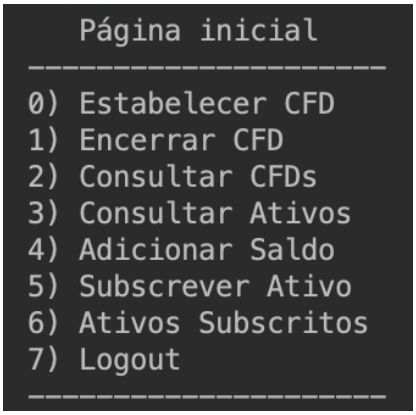

Figura 2.5: Página Inicial

```
   Página inicial
----------------------
0) Estabelecer CFD
1) Encerrar CFD
2) Consultar CFDs
3) Consultar Ativos
4) Adicionar Saldo
5) Subscrever Ativo
6) Ativos Subscritos
7) Logout
----------------------
0
---------------
0) Ações
1) Commodities
2) Índices
3) Moedas
---------------
2
Indique o ativo no qual deseja investir:
------------------------------------------------
0) Id: ind_NX6GzO
Nome: S&P 500
Valor por Unidade: 3140.98
Número de empresas: 0
1) Id: ind_2zvNzA
Nome: Dow Jones Industrial Average
Valor por Unidade: 28051.41
Número de empresas: 0
2) Id: ind_qzErBy
Nome: Dow Jones Composite Average
Valor por Unidade: 9233.75
Número de empresas: 0
3) Id: ind_bXZeGg
Nome: Dow Jones Transportation Average
Valor por Unidade: 10857.57
Número de empresas: 0
4) Id: ind_5X70Gz
Nome: Dow Jones Utility Average
Valor por Unidade: 851.72
Número de empresas: 0
5) Id: ind_gYWvd2
Nome: NASDAQ 100 Index
Valor por Unidade: 8403.68457
Número de empresas: 0
6) Id: ind_gPrpGL
Nome: NASDAQ Composite Index
Valor por Unidade: 8665.470703
Número de empresas: 0
```

Figura 2.6: Estabelecer CFD

```
------------------------------------------------
4
Escolha o tipo de CFD que deseja estabelecer:
--------
0) Long
1) Short
--------

1
Indique as unidades que deseja adquirir:
Possui 500000.0€ para investir
100
Deseja definir um Stop Loss?(s/n)
Vai investir 85172.0€
s
Indique o valor:
10
Deseja definir um Take Profit?(s/n)
Vai investir 85172.0€
n
CFD estabelecido com sucesso!
Id: 3
NIF: 918895246
Data: 2019-11-30T14:45:19.174772
Unidades de Ativo: 100.0
Valor por Unidade na Compra: 851.72
Take Profit: null
Stop Loss: 10.0
Id do Ativo: ind_5X70Gz
Aberto?: true
Valor Por Unidade no Final: 0.0
Short
```

Figura 2.7: Estabelecer CFD (continuação)

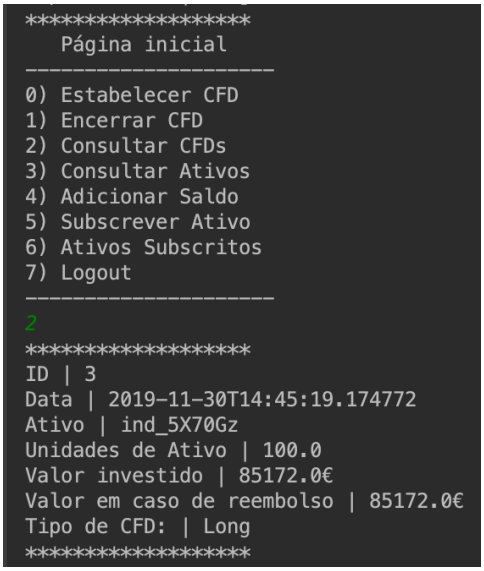

Figura 2.8: Consultar CFD

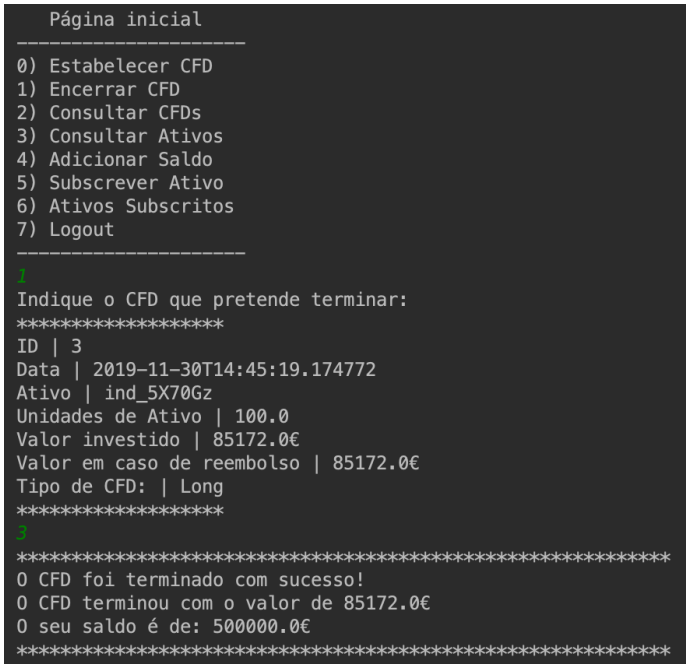

Figura 2.9: Terminar CFD

```
******************
   Página inicial
--------------------
0) Estabelecer CFD
1) Encerrar CFD
2) Consultar CFDs
3) Consultar Ativos
4) Adicionar Saldo
5) Subscrever Ativo
6) Ativos Subscritos
7) Logout
--------------------
3
Que tipo de Ativo deseja consultar?
--------------
0) Ações
1) Commodities
2) Índices
3) Moedas
--------------

2
Id: ind_NX6GzO
Nome: S&P 500
Valor por Unidade: 3140.98
Número de empresas: 0
Id: ind_2zvNzA
Nome: Dow Jones Industrial Average
Valor por Unidade: 28051.41
Número de empresas: 0
Id: ind_qzErBy
Nome: Dow Jones Composite Average
Valor por Unidade: 9233.75
Número de empresas: 0
Id: ind_bXZeGg
Nome: Dow Jones Transportation Average
Valor por Unidade: 10857.57
Número de empresas: 0
Id: ind_5X70Gz
Nome: Dow Jones Utility Average
Valor por Unidade: 851.72
Número de empresas: 0
Id: ind_gYWvd2
Nome: NASDAQ 100 Index
Valor por Unidade: 8403.68457
Número de empresas: 0
Id: ind_gPrpGL
Nome: NASDAQ Composite Index
Valor por Unidade: 8665.470703
Número de empresas: 0
```

Figura 2.10: Consultar ativo

```
   Página inicial
----------------------
0) Estabelecer CFD
1) Encerrar CFD
2) Consultar CFDs
3) Consultar Ativos
4) Adicionar Saldo
5) Subscrever Ativo
6) Ativos Subscritos
7) Logout
----------------------
4
Indique o valor:
100000
O seu saldo é de: 514828.0
```

Figura 2.11: Adicionar saldo

2.3 Atributos de Qualidade

Para este sistema, consideramos relevantes a **Modificabilidade** e a **Interoperabilidade**.

O primeiro, por este sistema ser desenvolvido no âmbito da Unidade Curricular de Arquiteturas de Software. Assim, este deverá ser capaz de acolher alterações ao código, devido à inclusão de novos requisitos e à alteração de outros já tidos em conta.

O segundo, por este sistema implicar o acesso a uma plataforma externa para consultar o valor de ativos. Tendo em conta a dificuldade que a equipa teve na primeira fase em encontrar uma plataforma estável, consideramos que este atributo devia ter um destaque maior nesta fase final uma vez que a solução encontrada deve poder vir a ser alterada facilmente para uma melhor.

2.3.1 Modificabilidade

Tendo em conta este atributo de qualidade, foi definido um cenário que se apresenta na tabela 2.1.

Modificabilidade	
Origem	*Developer*
Estímulo	Deseja alterar a API que devolve o valor dos Ativos
Artefacto	Código da aplicação
Ambiente	Momento de desenho
Resposta	API alterada e testada
Medida da Resposta	2 classes alteradas: a que consulta a API, e a que a instancia

Tabela 2.1: Cenário para Modificabilidade

De forma a ser possível controlar a modificabilidade no cenário descrito, optamos por **encapsular** o objeto que vai buscar as informações a uma API externa com recurso a **interfaces** do Java. Assim, basta alterar a instanciação dessa interface para uma outra classe que implemente de uma outra forma os métodos necessários para consultar a API. Torna-se assim possível modificar as fontes externas de informação relativa ao estado do mercado financeiro sem isso se refletir em grandes alterações no código da aplicação (apenas uma classe e respetiva instanciação).

2.3.2 Interoperabilidade

Tendo em conta este atributo de qualidade, foi definido um cenário que se apresenta na tabela 2.2.

Interoperabilidade	
Origem	Trading Platform
Estímulo	Requisita o valor de um ativo
Artefacto	API que disponibiliza valores de ativos
Ambiente	Sistemas conhecidos antes de *runtime*
Resposta	O pedido é aceite e é devolvido o valor mais recente do ativo em causa
Medida da Resposta	Em 95% dos pedidos é devolvido um valor corretamente

Tabela 2.2: Cenário para Interoperabilidade

De forma a ser possível lidar com a interoperabilidade com a API externa, implementamos uma classe que adapta as chamadas usadas por essa API a métodos mais simplificados que são os que o nossos sistema utiliza. Usamos então a tática de **Tailor Interface**, uma vez que escondemos do núcleo do nosso sistema as funções e métodos disponibilizados pela API, que ficam restringidos atrás de uma interface *Mercado*. Para além disso, a classe que implementa a interface *Mercado* lida também com os pedidos que não são atendidos corretamente, mantendo então a transparência para com o resto do sistema.

2.4 Condicionantes

O sistema teve de ser desenvolvido até ao dia 30 de novembro de 2019, e teve de ser implementado em **Java**.

Capítulo 3

Modelação

3.1 Modelo de Domínio

De forma a ser possível compreender melhor o contexto no qual foi desenvolvido este sistema, elaborou-se o modelo de domínio representado na figura 3.1.

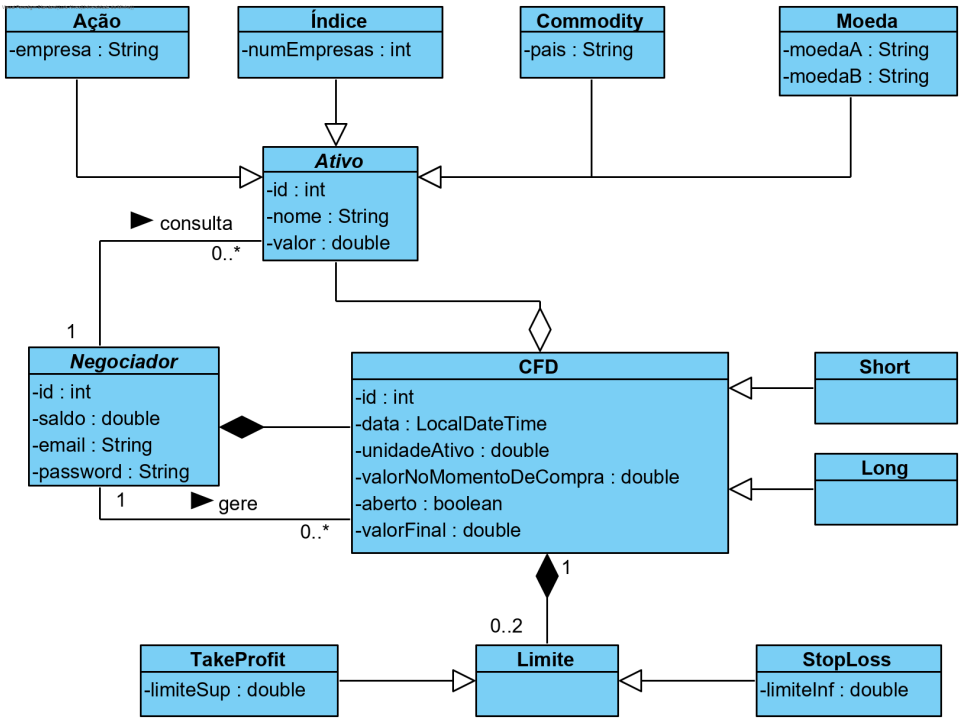

Figura 3.1: Modelo de Domínio

Neste modelo é possível verificar que os 3 principais atores no sistema são o CFD, o Negociador e o Ativo, sendo o CFD o elo de ligação entre estes dois últimos. Em relação à primeira fase, optou-se por tornar o diagrama mais conciso ao recorrer a agregadores e atributos, facilitando a leitura e compreensão do modelo.

3.2 Diagrama de *Use Cases*

Uma vez identificado o domínio, e tendo em conta os requisitos elicitados na secção 2.1, foi possível definir um diagrama de casos de uso, que se apresenta de seguida.

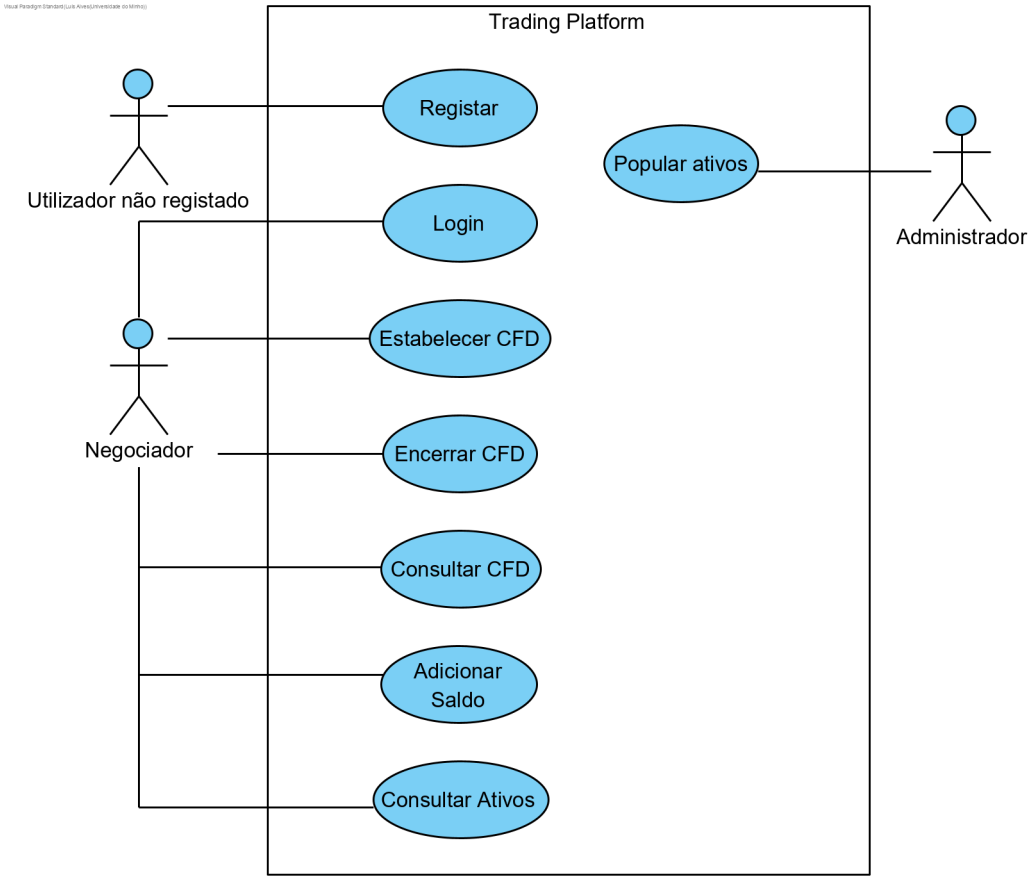

Figura 3.2: Diagrama de *Use Cases*

Neste diagrama é possível verificar a existência de 3 atores no sistema: o Utilizador não registado, o Negociador e o Administrador. Serão os casos de uso do Negociador e Utilizador não registado que serão devidamente especificados com recurso aos diagramas comportamentais expostos na secção 3.3. O Administrador, deverá ter uma forma de popular inicialmente o sistema com ativos para que os Negociadores possam estabelecer CFDs.

3.3 Diagramas Comportamentais

Nesta secção são apresentados diagramas de sequência que refletem o comportamento dos casos de uso de Registar Negociador, Estabelecer CFD, Encerrar CFD, Adicionar Saldo e Atualizar Ativo. Quanto ao Atualizar Ativo, ainda que não esteja no diagrama da figura 3.2, é uma funcionalidade necessária que é invocada regularmente pelo próprio sistema.

3.3.1 Registar Negociador

Esta funcionalidade é disponibilizada a qualquer utilizador da plataforma. Para que possa passar a ser um Negociador, este deverá inserir o seu NIF, o seu nome, o seu e-mail e uma palavra passe. Para além disso, também deverá ser possível definir um saldo inicial.

Este processo encontra-se ilustrado na figura 3.3.

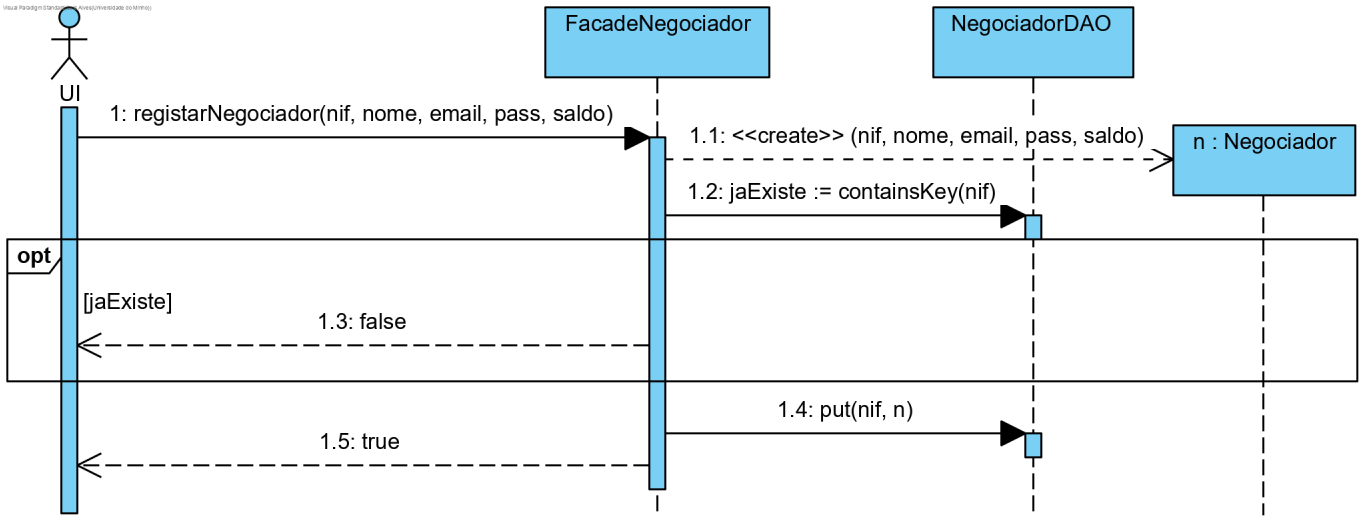

Figura 3.3: Diagrama de Sequência: Registar Negociador

3.3.2 Estabelecer CFD

Esta funcionalidade é disponibilizada a um Negociador após ter efetuado *login*. Assim, este deverá selecionar um ativo, definir uma quantidade que deseja adquirir e definir possíveis limites (*Take Profit* ou *Stop Loss*). Estando isto definido, o sistema irá verificar se o Negociador possui saldo suficiente para a operação em causa e tenta estabelecer um novo CFD com base no valor do Ativo no momento.

Este processo é ilustrado na figura 3.4.

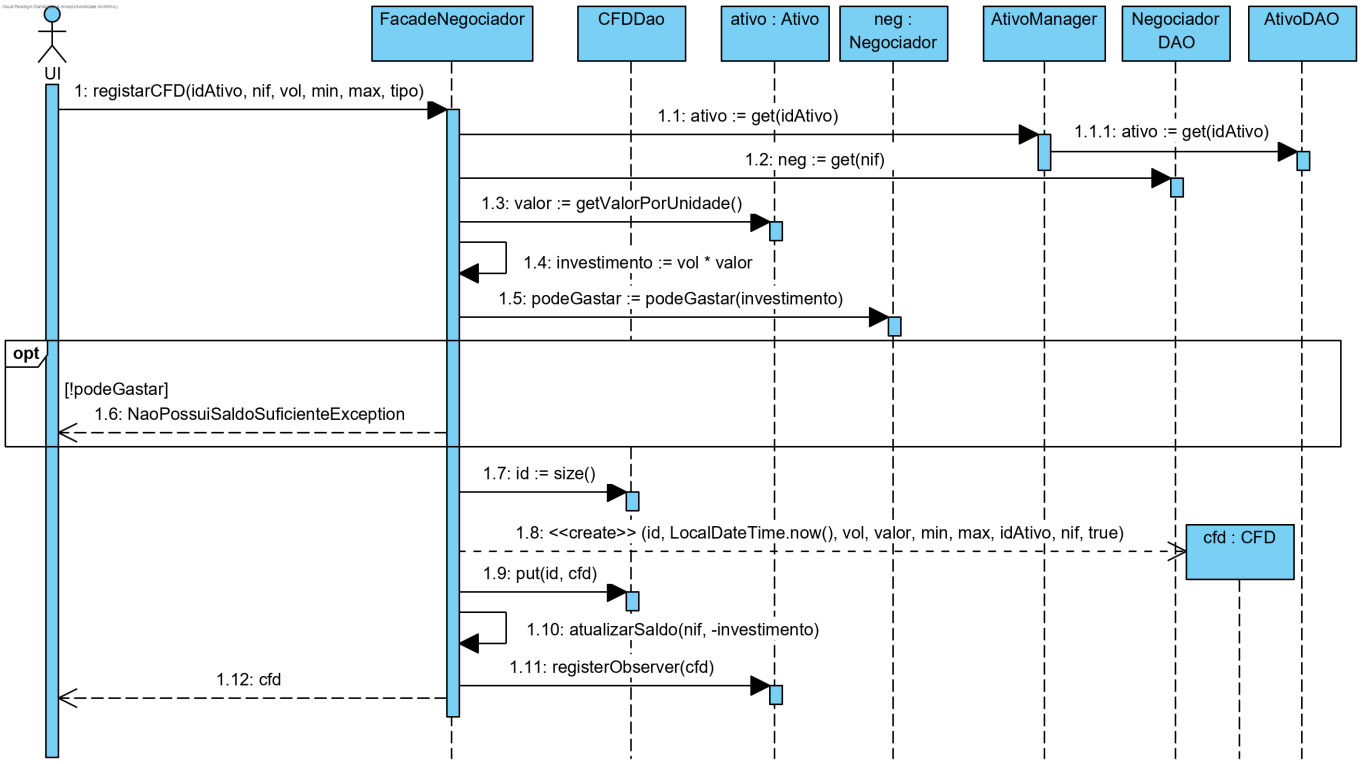

Figura 3.4: Diagrama de Sequência: Estabelecer CFD

3.3.3 Encerrar CFD

Esta funcionalidade é disponibilizada a um Negociador devidamente autenticado. Assim, este deverá selecionar um dos CFDs abertos que detém. De seguida, o sistema irá obter o valor de mercado do ativo relativo ao CFD em causa e calcular o valor obtido com a venda da quantidade de ativos associada a esse CFD. Tendo obtido esse valor, irá depositá-lo no saldo do Negociador.

Este processo é ilustrado na figura 3.5.

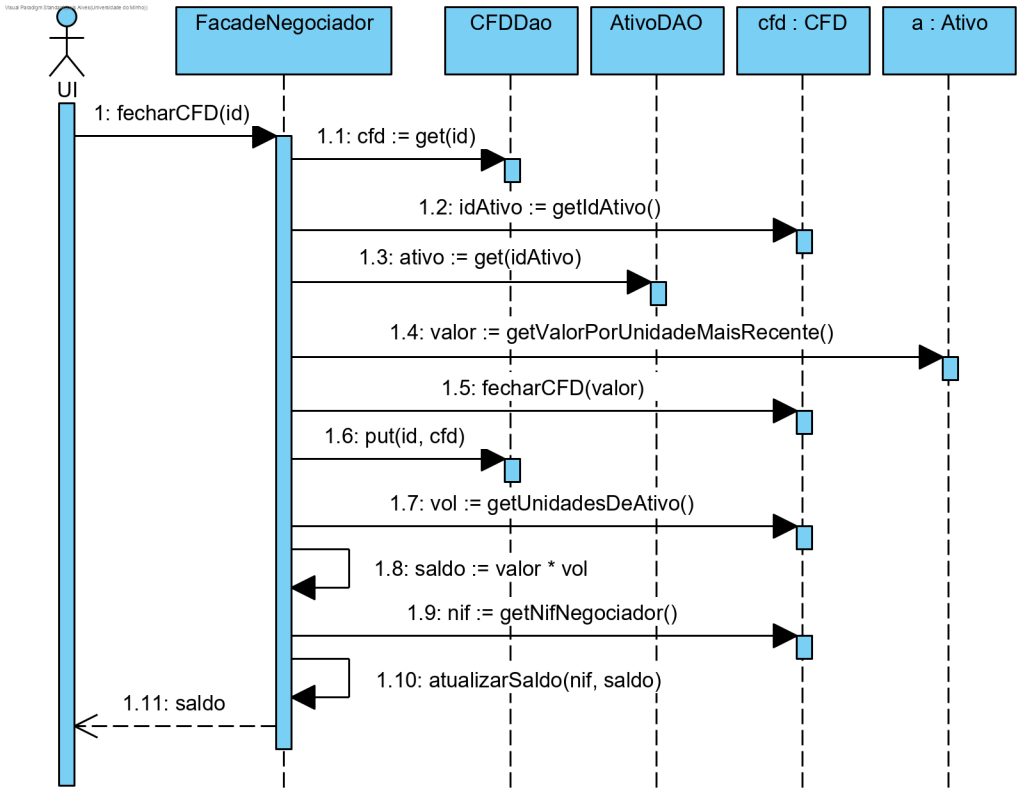

Figura 3.5: Diagrama de Sequência: Encerrar CFD

3.3.4 Adicionar Saldo

Esta funcionalidade é também ela disponibilizada a um Negociador devidamente autenticado. Para tal, este apenas necessita de indicar o valor a adicionar. Numa versão futura do sistema, este poderá recorrer a meios de pagamento reais, nomeadamente cartão de crédito, *PayPal*, ou até Multibanco.

Este processo, ainda que simples, encontra-se ilustrado na figura 3.6.

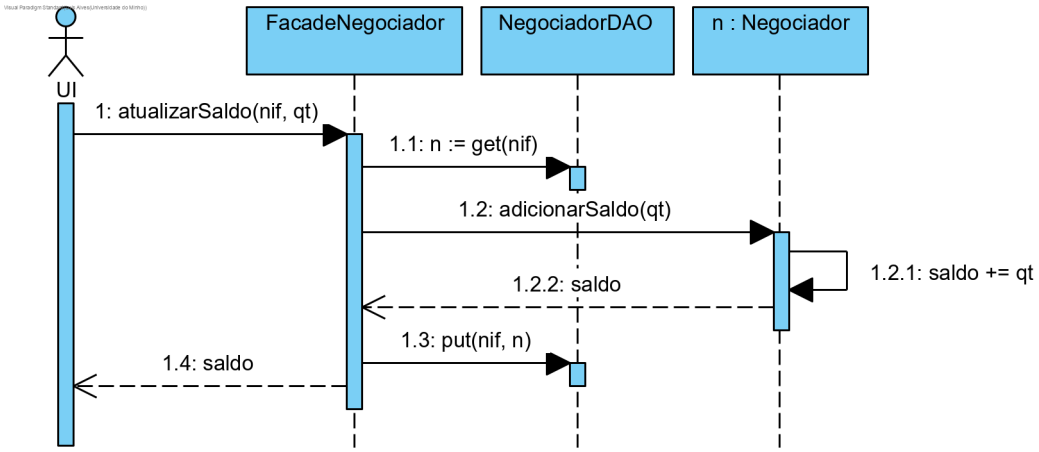

Figura 3.6: Diagrama de Sequência: Adicionar Saldo

3.3.5 Atualizar Ativo

Esta funcionalidade não é ativada pelo Negociador, mas sim pelo próprio sistema. Assim, a cada 30 segundos (ou outro valor configurável), é emitida uma nova ordem de atualização do valor dos ativos. Vai-se obter esse valor com recurso a uma API externa, e caso seja diferente do atual, atualizam-se os observadores do valor desse ativo.

Este processo é representado na figura 3.7.

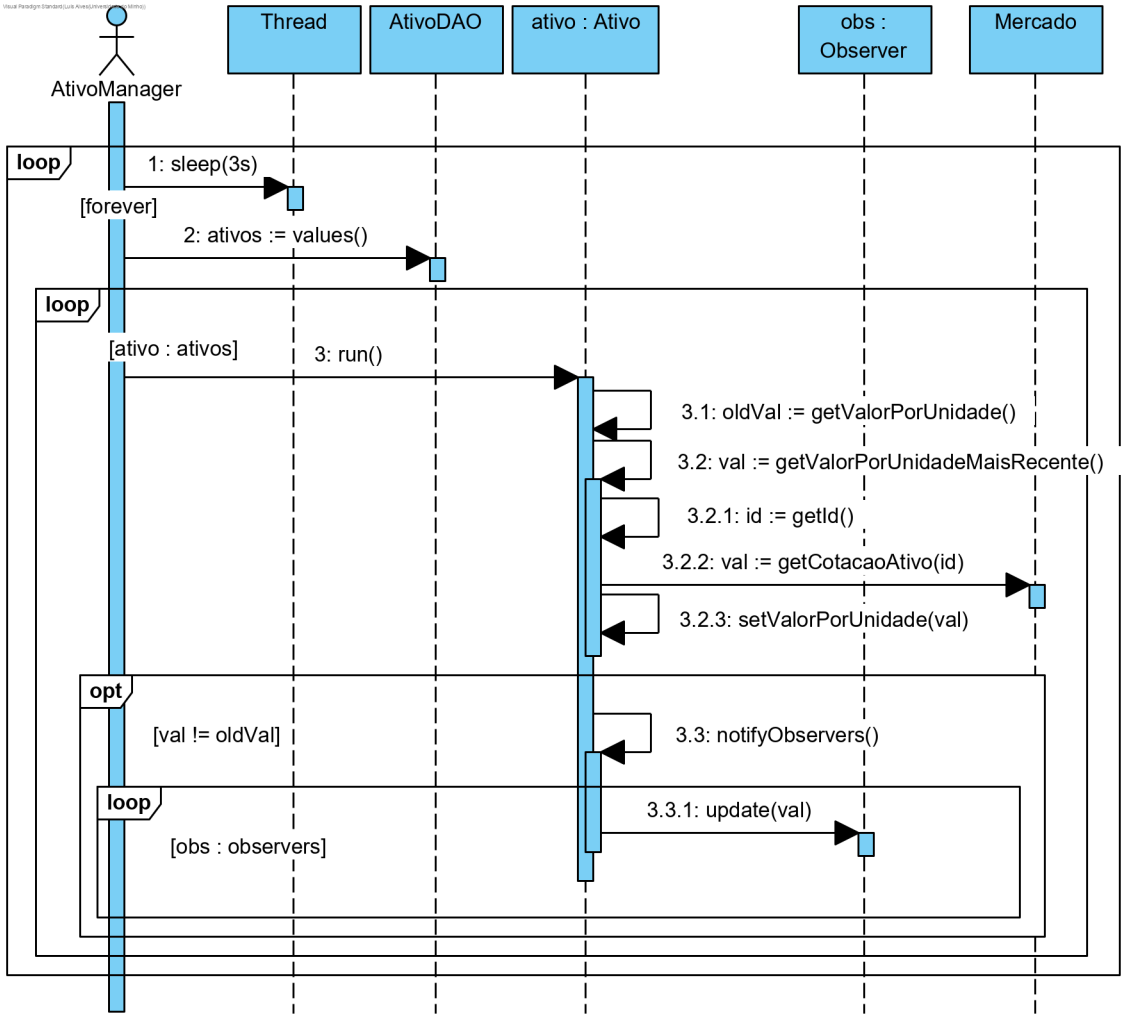

Figura 3.7: Diagrama de Sequência: Atualizar Ativo

3.4 Diagrama de *Packages*

Uma vez que optamos por uma **Arquitetura por Camadas** (ver capítulo 4), é possível identificar essas camadas e alguns dos seus principais componentes no diagrama da figura 3.8. Tendo em conta a complexidade do *Facade*, *Mercado* e diversidade de *Ativos*, estes estão contidos dentro de *packages* próprios no *package* de *Business*.

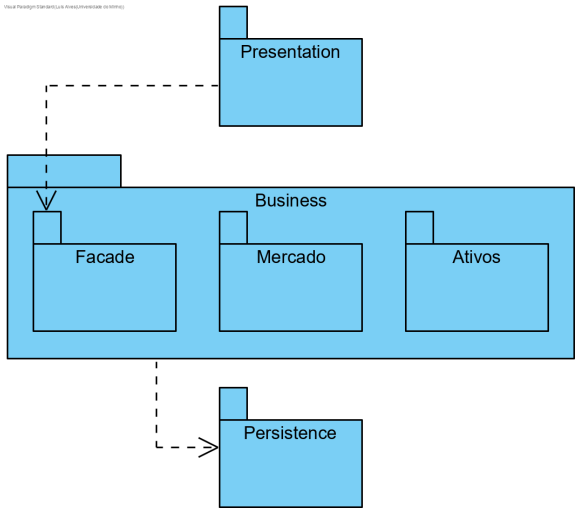

Figura 3.8: Diagrama de *Packages*

3.5 Diagrama de Classe

Tendo todos os diagramas anteriores, e optando por diversas decisões de desenho que são especificadas no capítulo 4, desenhou-se o diagrama de classe presente na figura 3.9.

Tendo em conta o diagrama desenvolvido na fase anterior (ver figura A.1), consideramos que a arquitetura final sofreu grandes alterações, tendo em conta a conveniência de utilização de padrões de desenho, bem como da necessidade de tornar as classes menos interdependentes e mais dependentes de interfaces ou classes abstratas, e não de implementações concretas.

As restantes considerações sobre a arquitetura do sistema, encontram-se tal como as decisões de desenho, no capítulo 4.

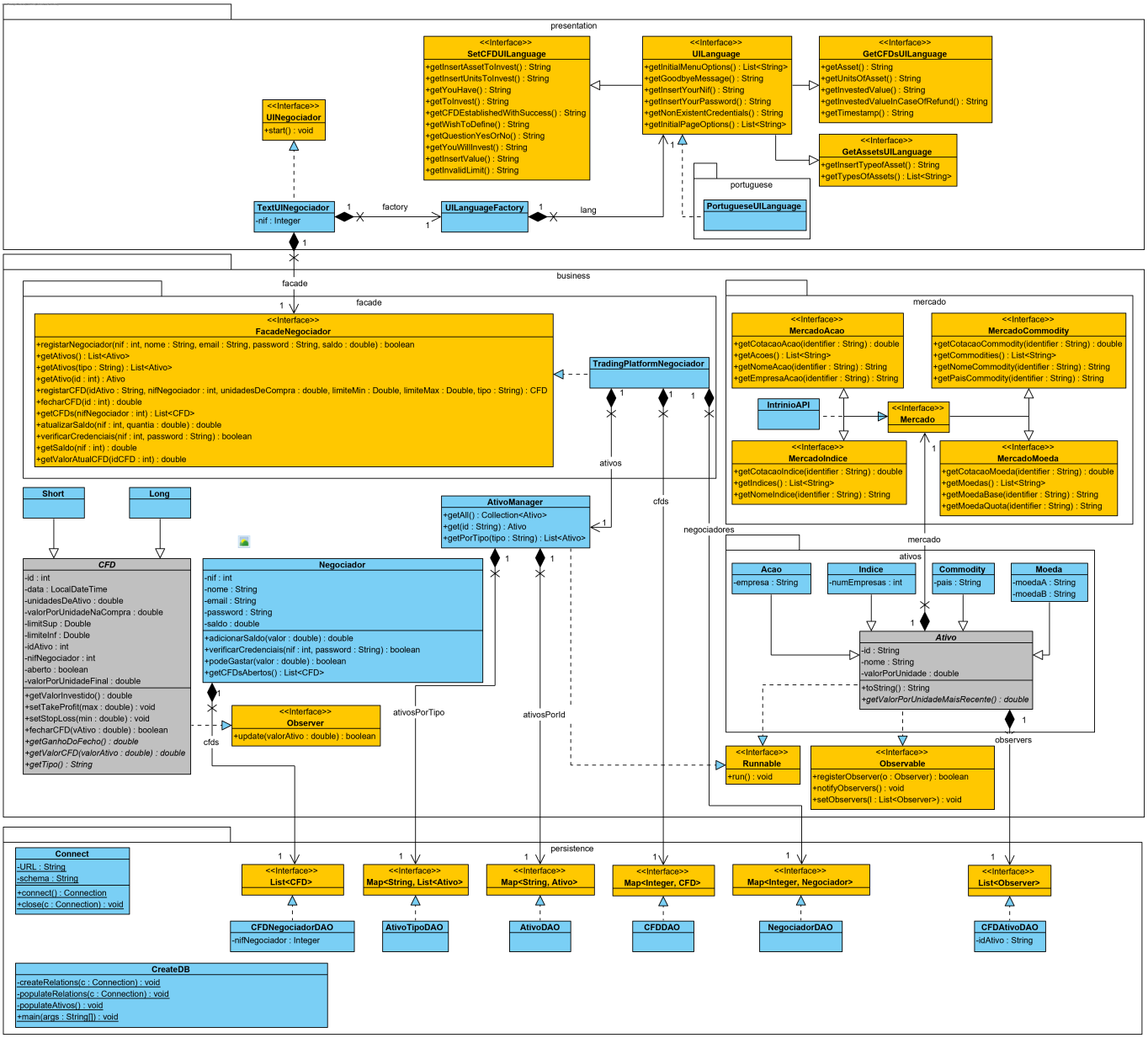

Figura 3.9: Diagrama de Classe

3.6 Modelo Físico da Base de Dados

Para a Base de Dados, desenvolveu-se um esquema físico que se encontra representado na figura 3.10. Com estas relações, é possível manter um histórico de todas as alterações que forem feitas ao sistema, sem que se tenha de manter tudo em memória.

É possível verificar que para adicionar um novo tipo de ativo, basta adicionar duas novas tabelas: uma correspondente ao que o ativo acrescenta em termos de atributos, e uma outra para relacionar com a tabela principal de ativo, que contém o nome e o respetivo valor por unidade (bem como um identificador).

Para implementação deste modelo, utilizamos o PostgreSQL.

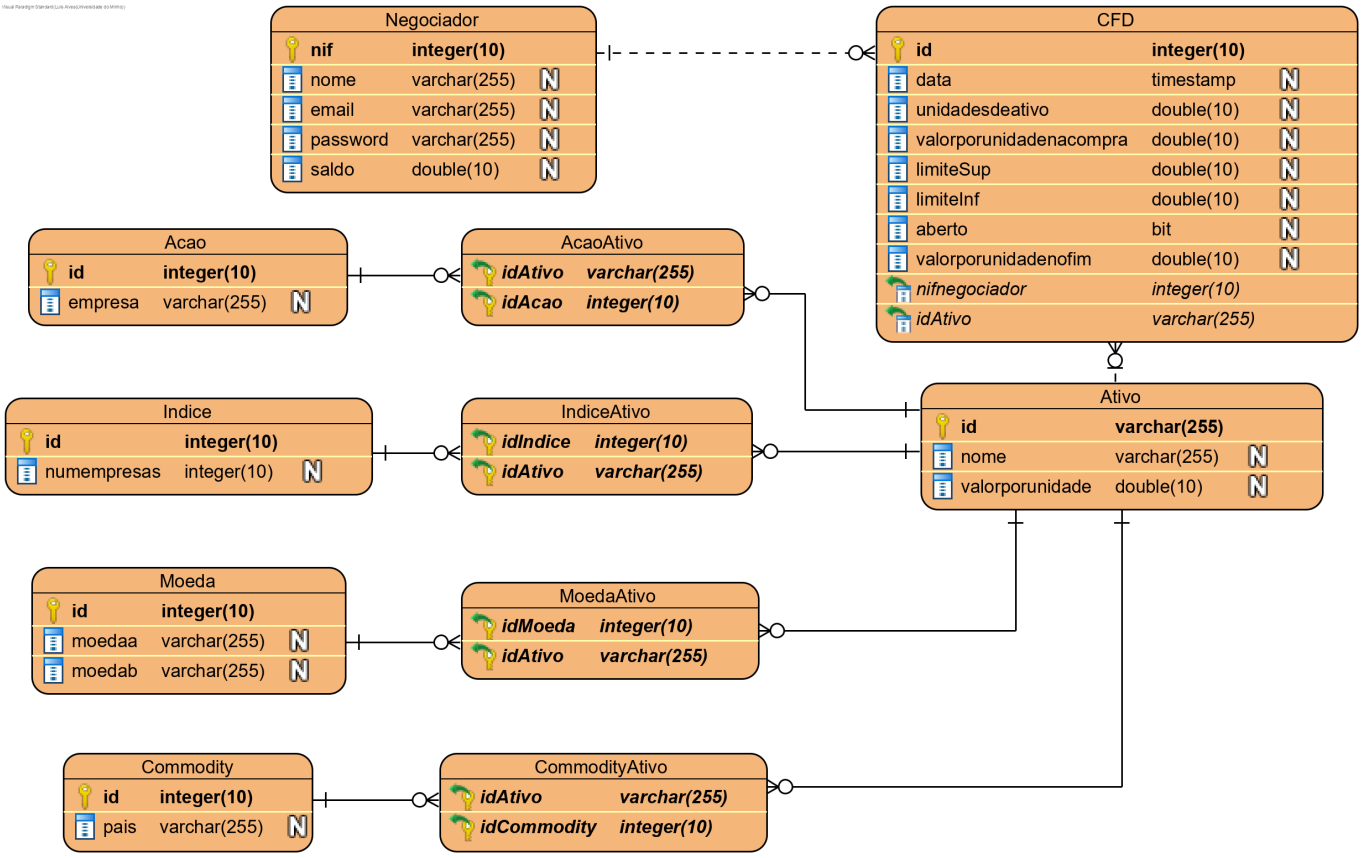

Figura 3.10: Esquema físico da Base de Dados

Capítulo 4

Padrões

Para o desenho final da arquitetura do sistema, adotamos um padrão arquitetural por camadas e recorremos a vários padrões de desenho para resolver pequenos problemas. São os que se apresentam nas secções seguintes.

4.1 Padrão Arquitetural - Camadas

Como padrão arquitetural, optamos pelo padrão em **Camadas**, uma vez que este valoriza a modificabilidade e a reusabilidade.

A **Modificabilidade** é um atributo de qualidade para o qual foi desenvolvido um cenário (ver tabela 2.1). Assim sendo, e tendo em conta as possíveis alterações ao sistema para que este passe a suportar uma API distinta para consulta dos valores dos ativos, ou uma UI sem ser pelo terminal, o grupo considerou que a possibilidade de trocar camadas numa fase futura seria benéfica para o desenvolvimento do sistema a longo prazo.

A **Reusabilidade** também é um atributo de qualidade relevante, uma vez que permitirá utilizar grande parte do código (por exemplo, da camada *Business* e *Persistence*) num sistema em que se recorra a uma interface gráfica *web*, por exemplo.

4.2 Padrões de Desenho

Tendo em conta o sistema a desenvolver, o grupo optou por incluir 5 padrões de desenho na solução final. A maioria deles foi aplicada por efetiva necessidade, sendo que apenas um deles (*Factory*) foi incluído para tentar introduzi-lo no sistema (provocando, na visão do grupo, um pouco de *over-design*). São os que se apresentam de seguida.

4.2.1 *Facade*

Para interligar a camada de apresentação à camada de negócio, optamos por utilizar o padrão **estrutural** *Facade*. Assim, simplificamos a complexidade do sistema ao cliente (neste caso, a UI) ao proporcionarmos uma única interface de acesso ao sistema.

É apenas através dos métodos disponibilizados nesta interface que o utilizador poderá efetuar alterações ao estado da aplicação.

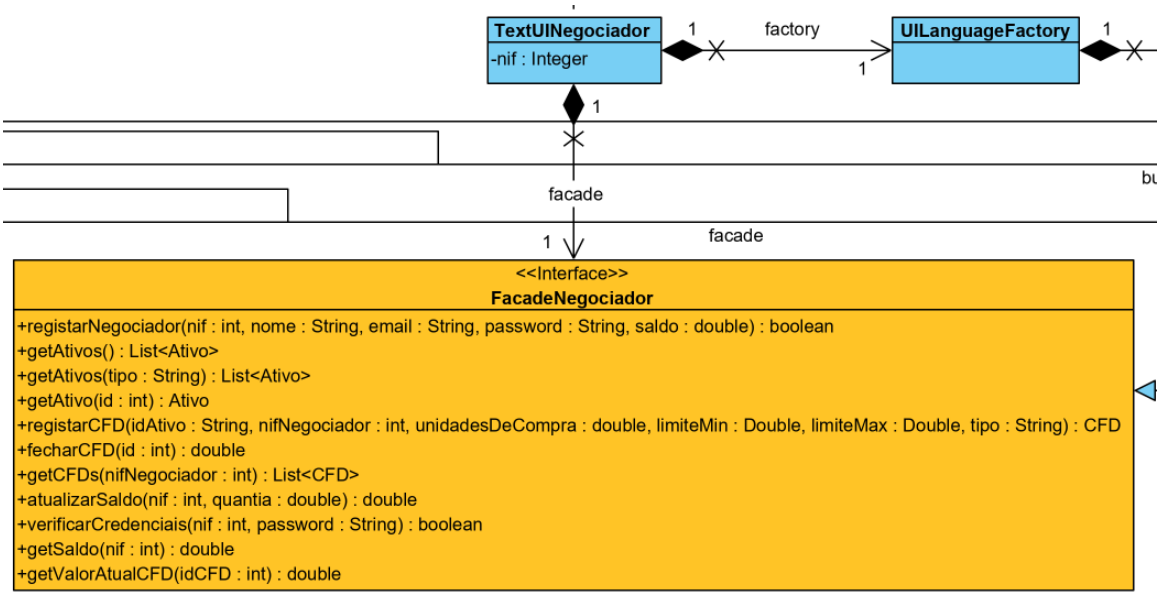

Figura 4.1: Padrão Estrutural - *Facade*

4.2.2 *Observer*

De forma a ser possível o Ativo propagar qualquer alteração ao seu valor unitário, optou-se por aplicar o padrão **comportamental** *Observer*. Assim, cada Ativo (um Observável) possui uma lista de Observadores, que monitorizam o estado do Ativo. Sempre que o valor por unidade for atualizado, então o Ativo notifica os seus Observadores através do método `notifyObservers()`, que por sua vez utiliza o método `update(double valorAtivo)` sobre cada um dos Observadores.

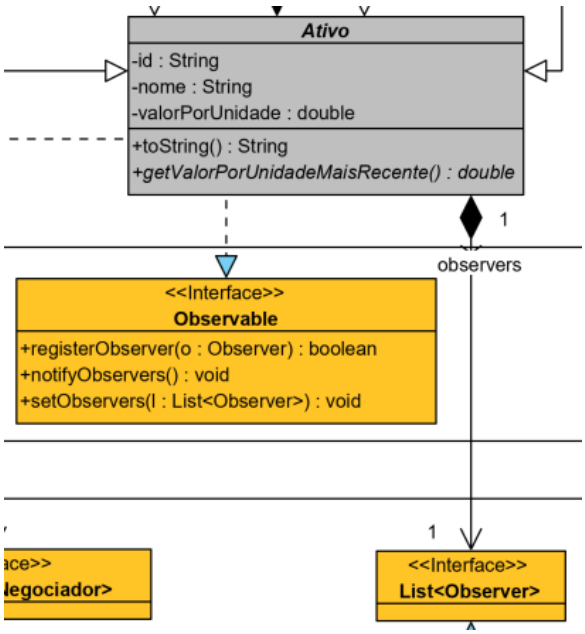

Figura 4.2: Padrão Comportamental - *Observable*

Por sua vez, o CFD implementa a interface Observador, uma vez que sempre que o Ativo recebe uma atualização de preço, o CFD deve verificar se deve encerrar automaticamente ou não (consoante os seus limites de ganho/perda).

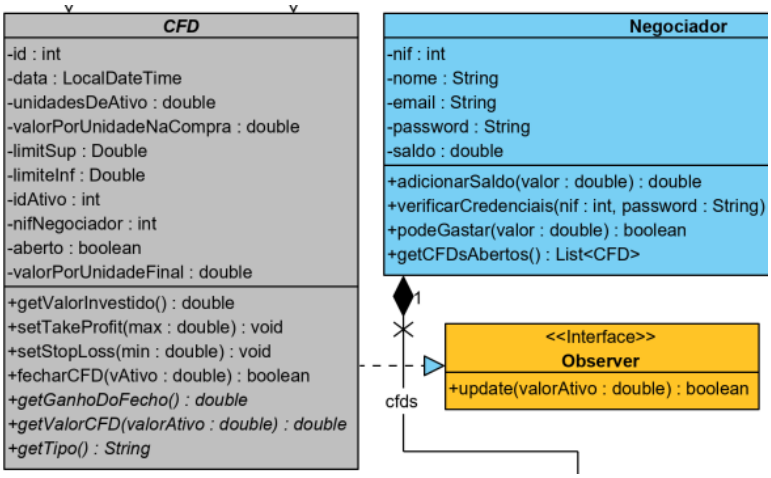

Figura 4.3: Padrão Comportamental - *Observer*

Assim, com este padrão, o Ativo desconhece a implementação do CFD, e o CFD desconhece a implementação do Ativo, apesar de ambos estarem relacionados através do padrão Observador.

No entanto, o grupo teve alguma dificuldade na implementação deste padrão, não na camada de negócio, mas sim na de persistência. Este problema é discutido na secção 4.2.3.

4.2.3 *Data Access Object*

De forma a ser possível persistir a informação sobre o estado do sistema em memória, optou-se por aplicar o padrão **estrutural** *DAO - Data Access Object*. Assim, mantendo a interface de Listas e Mapas do Java, foi possível desenvolver a lógica de negócio independentemente da decisão (que surgiu posteriormente) de como implementar a persistência dos dados.

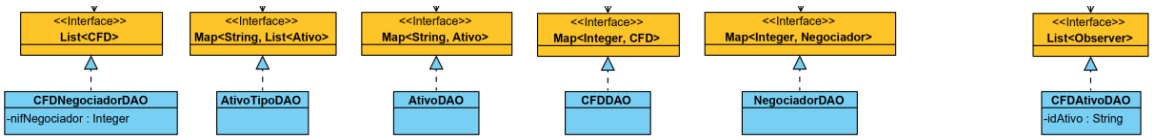

Figura 4.4: Padrão Estrutural - *DAO*

No entanto, a utilização deste padrão levantou vários problemas, nomeadamente na conciliação com o padrão *Observer*. Quando é feita a atualização do estado dos observadores, é percorrida a lista onde eles estão armazenados e é utilizado o método `update`. No entanto, para persistir a atualização do estado, é reposta a lista com os novos estados. Por parte da lógica de negócio este processo é transparente. No entanto, no DAO que implementa a Lista de Observadores, é necessário verificar a instanciação do objeto que é passado para adicionar à lista, de forma a poder ser armazenado corretamente na base de dados.

Para além disso, surgiu alguma repetição de código, que poderia ter sido mitigada pela equipa de desenvolvimento caso tivesse feito uma classe DAO agregadora de vários métodos mais concisos de acesso à base de dados, que seriam depois utilizados pelos métodos de cada um dos DAOs implementados. Ainda assim, foi positiva a utilização deste padrão para tornar transparente o uso de um mecanismo de persistência por parte da camada de negócio.

4.2.4 *Template*

Tendo duas classes abstratas na nossa arquitetura, de forma a minimizar a quantidade de código repetido, conseguimos aplicar também por duas vezes o padrão *Template*.

É possível aplicá-lo no caso dos CFDs, no método `update`. Uma vez que é necessário obter o valor do CFD para um dado valor de ativo, o método `update` recorre ao método declarado como abstrato `getValorCFD(double valorAtivo)`. Cada uma das classes que extende o comportamento do CFD (no nosso caso *Short* e *Long*) terá uma valorização diferente do CFD de acordo com as suas características. Assim, em vez de reimplementar o método `update` nas duas subclasses, delegou-se uma parte da implementação para as subclasses e definiu-se o algoritmo geral na classe abstrata (CFD).

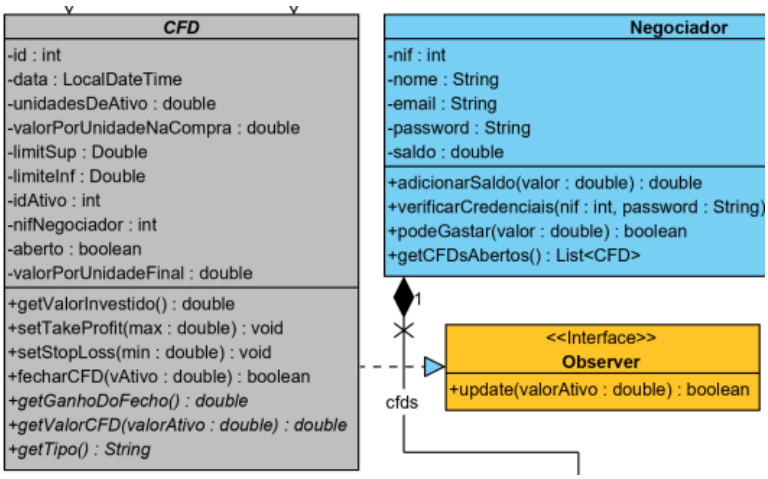

Figura 4.5: Padrão Comportamental - *Template* no CFD

Quanto ao Ativo, também foi utilizado o padrão *Template* para delegar uma parte da implementação do método `run`. Foi definido o método abstrato `getValorPorUnidadeMaisRecente()`, que é utilizado para ir buscar o valor mais atualizado do Ativo a uma API externa. Como esta implementação pode variar (e, na implementação concreta varia mesmo) de acordo com o tipo de Ativo, foi delegada a sua implementação para as subclasses que extendem a classe Ativo.

Assim, o algoritmo principal de atualização de valor de um Ativo fica independente das implementações feitas nas subclasses, sendo desnecessária a repetição de código para implementar o método `run`.

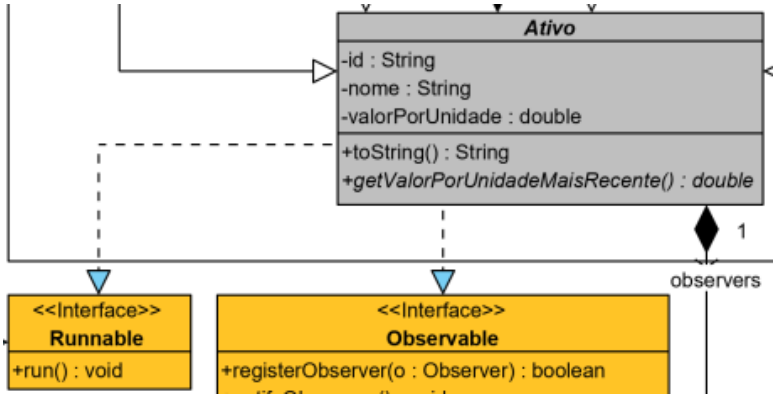

Figura 4.6: Padrão Comportamental - *Template* no Ativo

4.2.5 *Factory*

Por fim, na implementação da interface com o Utilizador, tentamos aplicar o padrão *Factory*. Decidimos tentar aplicar este padrão para facilitar a inclusão de novos idiomas na interface com o Utilizador. Assim, no momento de criação da UINegociador, é passado o argumento que indica o idioma pretendido. Dependendo desse idioma inicial, são devolvidas instanciações de pedaços de idioma consoante a disponibilidade desses idiomas. No caso da nossa aplicação, apenas foi implementado o idioma Português, mas torna-se fácil a adoção de qualquer outro idioma, desde que este implemente a interface *UILanguage*.

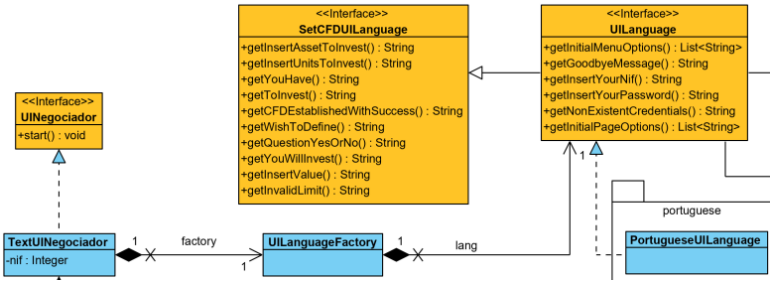

Figura 4.7: Padrão de Criação - *Factory*

Capítulo 5

Novo Requisito

A 48 horas do prazo de entrega final foi proposto um novo requisito a cumprir pela aplicação. O requisito é o seguinte:

O Sistema deve permitir aos *traders* seguirem a cotação de um conjunto de ativos, sendo notificados quando um desses ativos sofre uma variação significativa no seu valor.

Assim sendo, foram efetuadas algumas alterações no código fonte, mas a arquitetura geral do sistema manteve-se praticamente intacta, sendo apenas adicionada uma relação, uma interface e uma classe que implementa essa interface. As alterações feitas são discutidas de seguida.

5.1 Alterações ao Sistema

De forma a acomodarmos este novo requisito, tivemos que adaptar as 3 camadas arquiteturais.

5.1.1 Camada de Apresentação

Relativamente à camada de apresentação, foi necessário criar duas novas entradas na Página Inicial, para que o Negociador pudesse subscrever Ativos e pudesse consultar quais os Ativos que mantinha subscritos. Isso implicou apenas a adição de novos métodos, e à alteração do método `getInitialPageOptions()`.

5.1.2 Camada de Negócio

Quanto à camada de negócio, houve mais algumas alterações. De forma a ligar a funcionalidade disponibilizada na camada de apresentação ao estado da aplicação, foram adicionados dois novos métodos ao *Facade*, para que um Negociador pudesse subscrever um ativo (dado um NIF e um ID do ativo) e para que pudesse consultar os ativos subscritos (dado o NIF).

Para além disso, foi necessário que o Negociador passasse a implementar a interface *Observer*, uma vez que este deveria verificar o seu estado sempre que o Ativo que subscreveu atualizou o seu valor por unidade. Foi necessário então implementar o método `update` no Negociador, para que este verificasse se a alteração ao valor do Ativo era significativa ao ponto de despoletar um aviso e removesse a subscrição desse Ativo.

De forma a poder consultar quais os Ativos que um Negociador tinha subscrito, foi necessário adicionar um Map ao Negociador, que dado um ID de Ativo devolve a instância do Ativo correspondente. A imple-

mentação desse Map implicou a definição de um novo DAO: o NegociadorAtivoDAO, que persiste na base de dados a informação relativa aos ativos subscritos por um Negociador (mais sobre isto na secção 5.1.3).

Para que o Negociador saiba qual o Ativo que foi atualizado, foi infelizmente necessário atualizar o método `update`, que ao invés de ter como parâmetros apenas o valor do ativo, teve que incluir um parâmetro ID do Ativo.

5.1.3 Camada de Persistência

Relativamente à camada de persistência, foi necessário criar um novo DAO e atualizar um já existente.

Dado o conflito que já havíamos discutido na secção 4.2.3, esta foi a camada que requereu maior trabalho.

Foi necessário definir uma nova tabela (NegociadorAtivo) que apenas tem uma chave primária composta pelas colunas ID do Ativo e NIF do Negociador. Foi também necessário adaptar o CFDAtivoDAO (que implementa a interface `List<Observer>`), para que pudesse lidar com o facto de um *Observer* agora poder ser um Negociador. Como neste DAO "quebramos"a transparência que existe na camada de negócio, foi necessário também aqui verificar a instanciação de cada um dos Observadores para armazenar e atualizar corretamente os valores na base de dados.

Para além desse DAO, foi criado um NegociadorAtivoDAO, que permite ao Negociador adicionar e consultar os ativos que subscreveu.

5.2 Reutilização de Funcionalidades

A reutilização mais evidente foi a do método `run` no Ativo, que se manteve praticamente inalterada (apenas foi necessário passar o ID do Ativo ao método `update`). Para além disso, também grande parte do código do CFDAtivoDAO foi mantido, sendo que o resto do sistema se manteve inalterado apesar da incorporação do novo requisito.

As restantes alterações que foram feitas, limitaram-se a adicionar novas funcionalidades, e não a reescrever partes do sistema que já haviam sido implementadas corretamente.

5.3 Exemplo de funcionamento

O processo de subscrição de um ativo é similar ao processo de estabelecer um CFD: começa-se por selecionar o tipo de ativo que pretende seguir e depois o ativo em si.

Relativamente a consultar os ativos subscritos, estes aparecerão ao negociador como é apresentado na figura 5.1.

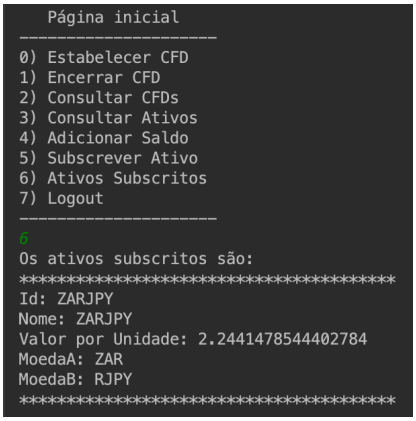

Figura 5.1: Ativos subscritos

Quando houver uma variação considerável, é apresentado ao negociador uma notificação. A figura 5.2 é um exemplo de uma notificação gerada.

Figura 5.2: Notificação de um ativo subscrito

Capítulo 6

Conclusão

Com este trabalho foi notória a evolução desde a arquitetura idealizada para a primeira fase com esta arquitetura final. Graças ao uso de padrões de desenho, tornou-se o código mais independente de implementações concretas, e mais modular, por estar também orientado a interfaces.

Este tipo de decisões refletiu-se positivamente na inclusão do requisito extra, divulgado com apenas 48h de antecedência relativamente à data final de entrega. O *core* da aplicação manteve-se intacto, e foi apenas necessário adicionar novas estruturas para armazenar a informação.

No entanto, também foi notória a dificuldade acrescida com a incorporação de tantos padrões. Ao fazê-lo, complica-se um pouco a estrutura do código, e isso consome mais tempo para a fase de conceção do desenho, adiando a fase de implementação.

Ainda assim, a equipa está satisfeita com o resultado final, uma vez que aplica os conhecimentos adquiridos na Unidade Curricular e satisfaz os requisitos propostos no enunciado.

Apêndice A

Diagrama de Classe (Primeira Fase)

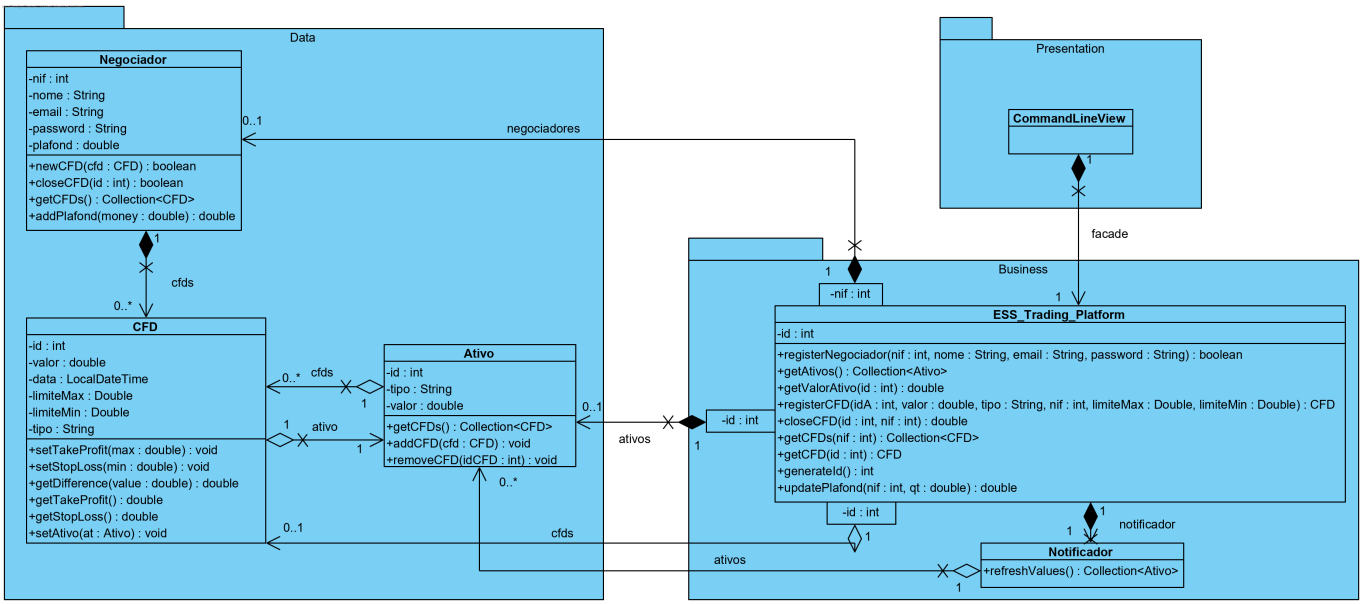

Figura A.1: Diagrama de Classe da Primeira Fase