Sistema de negociação

Hugo Abreu | A76203 João Padrão | A76438
João Reis | A75372

Paradigmas de Sistemas Distribuídos
Universidade do Minho

23 de Janeiro de 2018

1 Introdução

O seguinte trabalho prático advém da necessidade de aplicação e consequente cimentação de conhecimentos adquiridos nas aulas relativas à unidade curricular de Paradigmas de Sistemas Distribuídos, visando temas como a correcta implementação de protocolos de comunicação inter-linguagens e compreenção das temáticas associadas. Assim sendo, o presente relatório reporta à fase final do projeto, estando, por conseguinte, visada a correta implementação de *Protocol Buffers*, tanto em Java com em Erlang. Nesta fase final é também esperado que os conhecimentos acerca de atores, implementação de abordagens com recurso a Message Oriented Middleware e serviços REST sejam corretamente aplicados.

Estratégias para a manipulação das tecnologias contempladas pela Exchange terão de ser adotadas. No final deste projeto, espera-se que tenham sido corretamente aplicados conceitos com significativa complexidade no âmbito da disciplina, de um modo organizado e estruturado, a fim de lançar as fundações adequadas à execução deste último segmento da componente prática de avaliação da unidade curricular.

2 Front-end Server

3 Exchange Server

De modo a processar ordens recebidas, foi necessário implementar um servidor que efetua transações para um para um dado mercado - *PSI20*, *NADAQ*, etc – o que implica que existirá várias instancias deste tipo de servidor a executar. Caberá, portanto, ao *front-server*, assim que recebe uma ordem, averiguar qual o mercado onde uma empresa se encontra e redirecionar para o servidor correto.

Para iniciar um servidor *Exchange*, é necessário indicar sobre parâmetro qual o mercado onde irá operar de modo a obter as informações necessárias via REST.

Dada cada empresa recebida, é criado um objeto *Company* que conterá queues de ordens de compra e venda, que não foram de imediato efetuadas uma transação. Está associado também um objeto *PriceInfo*, que contem os preços de abertura, fecho, máximo e mínimo. Sempre que uma transação é efetuada, este verifica com os valores contidos se existem alterações, e se conferir, é enviado ao servidor REST.

Este conjunto de objectos encontram-se agregados num mapa que será acedido por todas as conexões.

Entre essas informações obtidas, encontra-se também a porta onde irá escutar novas conexões do provenientes do *front-server*. É também iniciado um *Publisher* implementado em ZeroMQ, que publica informação sempre que uma transação é efetuada.

Estando, portanto, à escuta de novas conexões, estas originam um novo *Handler* que recebe um *protobuff order*. Se hora de receção estiver entre as 9:00 e 17:00 é replicado o *protobuff* recebido, indicando desta vez uma confirmação positiva e enviado para o socket, processando o pedido posteriormente. Caso contrário, envia-se apendas uma confirmação negativa.

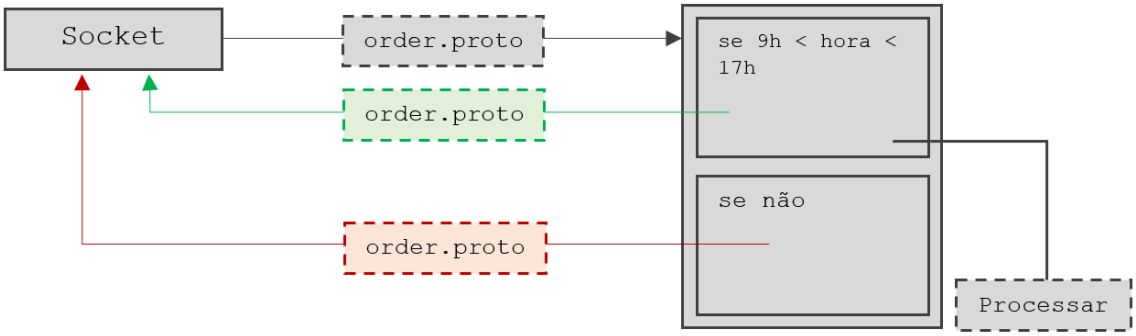

Figura 1: Aceitação de conexões

A *protobuff* order recebida é encaminhada para o objeto correspondente à empresa a que diz respeito. Consoante o tipo de order – compra ou venda – confere-se na contrária queue se existem outras que correspondam de modo a efetuar uma transação. No fim, se ainda existir quantidade por vender, é também colocada em queue.

Sempre que uma transação é efetuada, são enviados a ambos intervenientes (via *front-server*, que estará à escuta na porta 3001) uma confirmação usando novamente uma protobuff order com a quantidade transacionada, assim como o preço a que foi feita. É também enviado ao publisher uma mensagem para anunciar a transação entre os dois utilizadores, bem como a quantidade e o preço.

Houve a necessidade de criar um *trigger* para que, aquando do fecho da exchange, o preço da última transação fosse atualizado no directório, desta forma, às 17:00 de cada dia, este timer é executado de maneira a enviar a última atualização diária no que toca ao preço de cada empresa transacionada.

4 Directory Server

Para projetar o servidor de directório, ou seja, o servidor que contém todos os meta-dados relativos ao sistema de negociação, portanto, esta entidade do sistema

mantém dados como, que empreas existem, em que exchange são transacionadas, que exchanges existem e que empresas têm e, os diferentes preços diários das empresas em questão. A utilização de um servidor REST para este tipo de acesso a dados, é o mais correto, visto que não existe a necessidade de manter estado entre pedidos e, desta forma, faz com que um só servidor consiga suportar muitos utilizadores em simultâneo.

O servidor construído, tal como requirido pelo docente no enunciado, é um servidor REST, stateless, programado em Java com recuros a framework Dropwizard. Para ser possível garantir o acesso a todos os recursos que o serviço oferece, foi necessário definir *end-points* para obter e/ou atualizar esses mesmos recursos. Os *end-points* definidos para o servidor REST foram os seguintes:

- GET
  - **/companies**
    Este *end-point* devolve uma lista com todas as empresas existentes no sistema. A lista devolvida é uma lista de objetos e, portanto, contém todos os atributos intrínsecos a cada empresa na enumeração.
  - **/company/{id}**
    Este recurso, quando fornecido um correcto id de empresa, a respetiva empresa, caso esta se encontre efetivamente no sistema. Neste recuso é devolvida a empresa e todos os atributos a esta relacionados.
  - **/company/{id}/today**
    O *end-point* descrito neste ponto, devolve a informação de preços do dia corrente, sendo que, esta informação também é devolvida no recurso anterior, desta forma, caso apenas seja necessário obter os preços, é possível reduzir a quantidade de informação que circula na rede.
  - **/company/{id}/yesterday**
    Da mesma forma que o recurso mencionado anteriormente retorna os preços do dia atual, este recurso retorna os preços do dia anterior.
  - **/exchanges**
    Este *end-point* devolve uma lista com todas as exchanges existentes no sistema. A lista devolvida é, tal como nas empresas, uma lista de objetos e, portanto, contém todos os atributos intrínsecos a cada exchange na enumeração.
  - **/exchange/{id}**
    Quando o recurso descrito é acedido, ao fornecer um id de exchange válido, este recurso devolve a dita exchange com todos os astributos a ela relacionados, como por exemplo, o IP+Porta.
  - **/exchange/{id}/companies**
    Este *end-point* devolve uma lista com todas as empresas da exchange que é fornecida como argumento (id). A lista devolvida é uma lista de objetos e, portanto, contém todos os atributos intrínsecos a cada empresa.
- PUT

Os recursos definidos como PUT, deverão ser idempotentes, ou seja, por muitas vezes que o recurso seja executado, o resultado é o mesmo e, portanto, é a operação ideal para, neste caso, o recurso listado abaixo.

- **/company/{id}/today**
  Este recurso tem como objetivo atualizar os preços atuais de uma dada empresa e, portanto é uma operação idempotente visto que, o resultado é sempre o mesmo não importando a quantidade de vezes que é executado.

Todos os recursos abordados acima foram corretamente implementados e a sua executção foi efetivamente testada, o grupo teve o cuidado de utilizar as primitivas da framework Jackson para permitir a correcta serialização dos dados.

Foi também adicionada a funcionalidade ao REST server de às 23:59:59 de cada dia, executar a alteração diária, ou seja, às horas definidas, é accionado um *trigger* que vai a todas as empresas trocar os preços actuais para o dia anterior, garantindo assim a atualização das informações no directório.

5 Client

Para permitir o acesso às funcionalidades completas da Exchange e todos os recursos do eco-sistema associado, foi construído um Client que visa a utilização por utilizadores ditos normais.

No que toca ao registo e login, este é efetuado no front-end server com recurso a *Protocol Buffers* (account.proto), esta classe contém o username, a password e uma flag que indica se é referente a um registo ou um login. O menu referente ao utilizador não logado encontra-se abaixo.

```
[1] Login
[2] Registo
[0] Sair
```

No cliente foram aplicadas todas as funcionalidades requiridas pelo docente no enunciado disponibilizado, as opções de um utilizador com o login efetuado são as seguintes:

```
[1] Ver empresas.
[2] Ver exchanges.
[3] Ver uma empresa específica.
[4] Verificar preços da empresa.
[5] Obter real time updates.
[6] Cancelar real time updates.
[7] Abrir posição.
```

As operações descritas são operações que, além de requiridas no enunciado do trabalho prático, são necessárias para a correta funcionalidade de toda a aplicação. A opção **[1]**, contacta o serviço de directório (REST) de forma a obter a lista de todas as empresas que se encontram disponíveis para transacionar. Assim o utilizador tem acesso à lista de empresas que podem ser transacionadas e respetivos ids para efetuar os pedidos. De salientar que, os pedidos que acedem ao servidor REST mantêm uma

cache para que sempre que é necessário obter, por exemplo, uma empresa, não exista a necessidade imediata de efetuar uma comunicação, existindo apenas a necessidade de aceder a cache local.

Ao selecionar a opção **[2]**, é possível verificar todas as exchanges existentes, respetivas cidades e endereços. No caso da escolha **[3]**, é possível fornecer o id da empresa para obter mais informações mais específicas acerca desta. É possível, tal como requirido no enunciado, obter os preços do dia atual e anterior, de forma simples e segmentada com a selecção da opção **[4]**.

A selecção **[5]** e **[6]**, são respetivamente a efetuar subscrições e anular estas. Estas opções utilizam a tecnologia de Message Oriented Middleware, o ZeroMQ, para subescrever aos diferentes servidores de exchange que contêm a empresa a subscrevem, desta forma obtendo as atualizações de transações em tempo real.

Por fim, o último ponto abre posições, quer de compra, quer de venda nas diferentes Exchanges, este método utiliza *Protocol Buffers* (orders.proto) para comunicar com o front-end server e com a exchange de forma a garantir que os seus pedidos são efetivamente processados. Para a correta execução deste pedido são utilizadas duas classes de *Protocol Buffers*, uma para efetuar o pedido e uma para a resposta. Quando é efetuado o pedido, este é preenchido com a quantidade, preço pretendido, o utilizador que pertende efetuar a transação, a empresa à qual o pedido se refere e por fim o tipo de pedido (compra/venda).

Quando existe efetivamente uma transação, além das notificações que são entregues pelo MOM, os intervinientes da compra/venda são também notificados da quantidade, preço e todas as informações intrínsecas à transação, portanto é também utilizado o mesmo *Protocol Buffer* acima mencionado mas com objetivos e compreensões diferentes.

6 Conclusão

Com o término da fase final do projeto, é admissível afirmar que todos os objetivos requeridos pelo docente no enunciado cedido foram alcançados.

Conhecimentos relativos à implementação de Message Oriented Middleware, serviços REST, Protocol Buffers e atores em Erlang foram incrementados e por consequência cimentados. Em relação à linguagem Erlang, o grupo teve a necessidade de utilizar um gerenciador de pacotes semelhante ao Maven, o *rebar3* que, permitiu aprofundar o conhecimento relativo à linguagem e respetiva utilização de diferentes pacotes desenvolvidos nesta.

O maior foco incidiu sobre a correta implementação dos diferentes protocolos inter-linguagem e, em particular na implementação do front-end server que é parte fundamental na comunicação de sistemas.

A qualidade do trabalho apresentado agrada ao grupo, tendo havido tempo suficiente para o estudo das soluções encontradas ao problema proposto, sem precipitações e com espaço de maneio suficiente para que todos os membros pudessem inteirar-se não somente do segmento que lhes foi atribuído, mas também das restantes duas terças-partes que não exigiram tanta da sua atenção.