Universidade do Minho

Mestrado Integrado de Engenharia Informática

Paradigmas de Sistemas Distribuídos

Sistema de Negociação

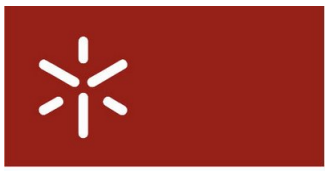
Autores:

A74817 Marcelo António Caridade Miranda

A75428 Bruno Miguel Sousa Cancelinha

A74576 José António Dantas Silva

5 de Setembro de 2018



Universidade do Minho

Conteúdo

1 | Introdução

Este relatório descreve a realização do trabalho prático realizado no âmbito de Sistemas Distribuídos. O objetivo deste trabalho é o desenvolvimento de um protótipo que fornece um serviço de intermediação de compra e venda de ações numa bolsa, aplicando os conceitos de serialização, programação baseada em atores, programação orientada a mensagens e arquiteturas REST, tal como aprendido no decorrer do semestre.

Os clientes do sistema deverão ser capazes de comprar e vendar uma quantidade arbitrária de ações de uma dada empresa. As negociações entre as várias partes interessadas ocorrem nas *exchanges*, sendo que cada empresa é negociada numa única *exchange*. Os clientes são capazes de obter a lista de empresas a ser negociadas através do *diretório de ações*, o qual contém as informações relativas às ações de cada empresa, como por exemplo o valor máximo atingido nesse dia ou o valor com que a *exchange* abriu. Opcionalmente, os clientes podem subscrever as empresas em que estão interessados, recebendo notificações das transações que ocorrem.

2 | Análise do problema

O cliente pode interagir com o sistema de três formas distintas. A primeira opção é comunicar com o diretório de ações, através do qual é possível obter a lista de empresas que são negociadas ou obter informações específicas de uma empresa. As informações de uma empresa obtíveis através do diretório são:

- *Exchange* em que esta é negociada (nome, endereço e porta);
- Preço máximo e mínimo que ocorreram no dia atual ou no dia anterior (caso tenham ocorrido transações);
- Preço de abertura e fecho da bolsa no dia atual e no dia anterior.

Outra opção é o cliente subscrever os eventos de uma empresa, recebendo notificações sempre que ocorre uma transação relacionada com aquela empresa. Por fim, o cliente pode ainda enviar pedidos de venda ou compra de ações.

Os vários pedidos do cliente são enviados para o servidor de *front-end* o qual, desde que o utilizador esteja autenticado, aceita pedidos de compra e venda. Estes pedidos devem indicar sempre a empresa, a quantidade de ações que se pretende comprar/vender, assim como o preço a que se pretende faze-lo. Dado que uma empresa é negociada apenas numa das várias *exchanges*,

o *front-end* necessita de reencaminhar o pedido para a *exchange* correta, a qual irá processar o pedido.

Na *exchange*, uma vez que surja um par de ordens compra/venda sobre a mesma empresa, nas quais o preço de venda é menor ou igual ao preço de compra, ocorre uma transação e as ações são vendidos pela média dos preços indicados. Caso um dos pedidos não seja completamente satisfeito, é gerado um novo pedido com a quantidade de ações que restou da transação.

Assim, caso surja uma ordem de venda de 100 unidades por 5€e uma ordem de compra de 50 unidades por 10€, ocorre uma transação de 50 unidades por 7,5€e é emitida uma nova ordem de compra com as restantes 50 unidades.

Sempre que há ocorrência de uma transação, é necessário que diretório de ações seja notificado para que os dados se mantenham atuais. Para além disso, é também essencial que seja enviada uma notificação a todos os clientes que demonstraram interesse naquela empresa.

3 | Desenho da Arquitetura

3.1 Visão geral

De todo o sistema, destacamos quatro elementos principais. O **cliente** é a interface do lado do utilizador; O **diretório de ações** que agrega informações de todas as empresas; As **exchanges** que representa uma bolsa de valores onde são negociadas várias empresas; o **front-end**, o coração do sistema, que se encarrega de processar os pedidos dos clientes. O comportamento de todos estes elementos são descritos no capítulo 4-Implementação.

Era necessário garantir que os clientes tinham a possibilidade de subscrever empresas, podendo ser notificados quando era completada uma transação com sucesso. Para isso, cada ***exchange*** pode publicar notificações que são recebidas por um ***proxy***, o qual as reencaminha para os vários clientes que a subscrevem.

Também era necessário que as ordens de um cliente, depois de processadas pelo ***front-end***, pudessem ser enviadas para as ***exchanges*** e receber a resposta. Era imperativo que todo este processo fosse assíncrono, para que o cliente não bloqueasse até que a sua ordem tenha sido correspondida e uma transferência tenha sido efetuada. O padrão ***push/pull*** é, portanto, o mais adequado. No entanto, é necessário ter atenção para que as mensagens sejam entregues às exchanges corretas. Assim, embora seja necessário ter um ***push*** para cada ***exchange***, apenas precisamos de um número reduzido de ***pulls*** para suportar todos os clientes. Para podermos ligar todos os ***push*** das ***exchanges*** ao ***pull*** do ***front-end*** foi necessário implementar um ***proxy***, tal como é visível pela figura 3.1.

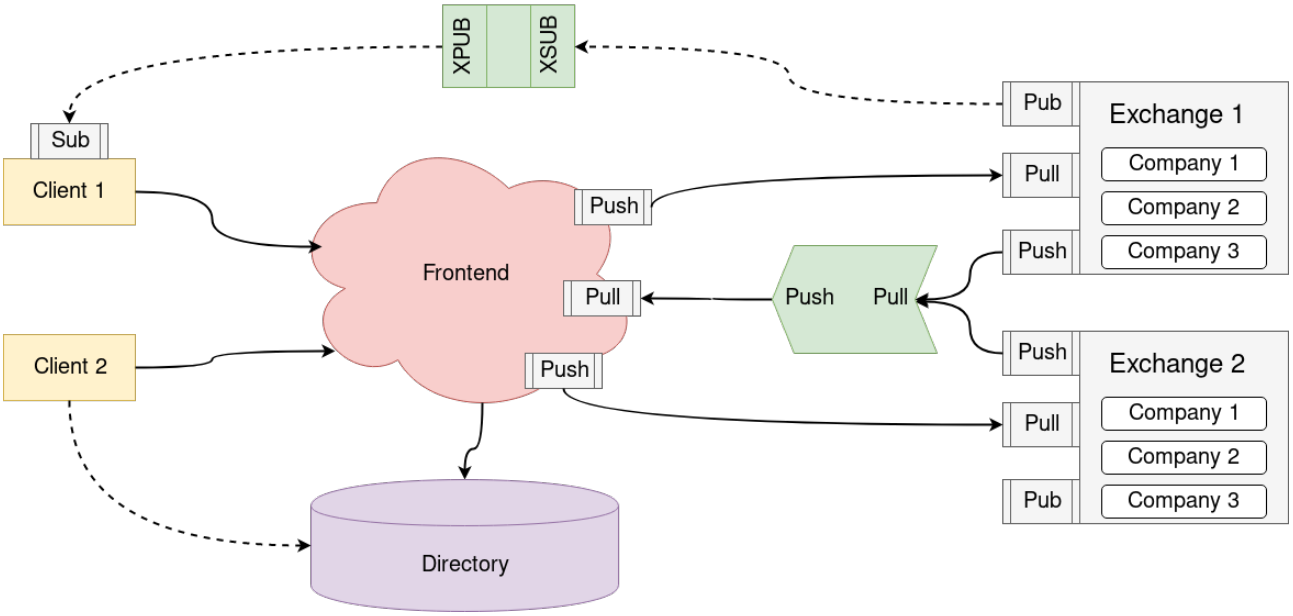


Figura 3.1: Visão geral da arquitetura do projeto

3.2 Cliente

O cliente, desenvolvido em *java*, funciona como uma interface de interação com o sistema, enviando pedidos ao *front-end* e recebendo deste a confirmação das transações nas quais está envolvido de forma assíncrona. Para além desta **mailbox** das suas transações pessoais, existe também a possibilidade de subscrever e cancelar a subscrição de informação relativamente às transações de uma dada empresa, efetuando a subscrição das mesmas, podendo a qualquer momento verificar na sua caixa de notificações as transações ocorridas.

Após o registo e login todas estas funcionalidades descritas anteriormente estão disponíveis, assim como a possibilidade de efetuar pedidos de compra e venda de ações de uma dada empresa, assim como observar estatísticas relativamente às empresas disponíveis e as *exchanges* em que são negociadas.

3.3 Front-end

O ***front-end*** é, provavelmente, das componentes mais interessantes do trabalho. Foi implementado usado programação por atores, sendo a única parte do trabalho desenvolvida em ***Erlang***. Possui uma rede rica de comunicações entre os vários atores que a compõem.

Vamos agora examinar um comportamento normal de um cliente a registar uma nova ordem. Inicialmente, um cliente conecta-se ao ***login manager***, quando é aceite no sistema, esse ator "transforma-se"num ***user session*** que serve como seu recetor de pedidos. Quando o ***user***

session recebe o pedido para uma nova ordem, pergunta ao ***exchange manager*** pelo ***exchange producer***, responsável pela ***exchange*** que contém a empresa especificada na ordem. O ***exchange manager*** guarda uma ***cache*** mas, se não encontrar o ator responsável, pergunta ao diretório, que age como um servidor de ***naming***, pela informação da ***exchange***, podendo ou não criar um novo ***exchange producer***. Este ***exchange producer*** trata de enviar para a ***exchange*** correta.

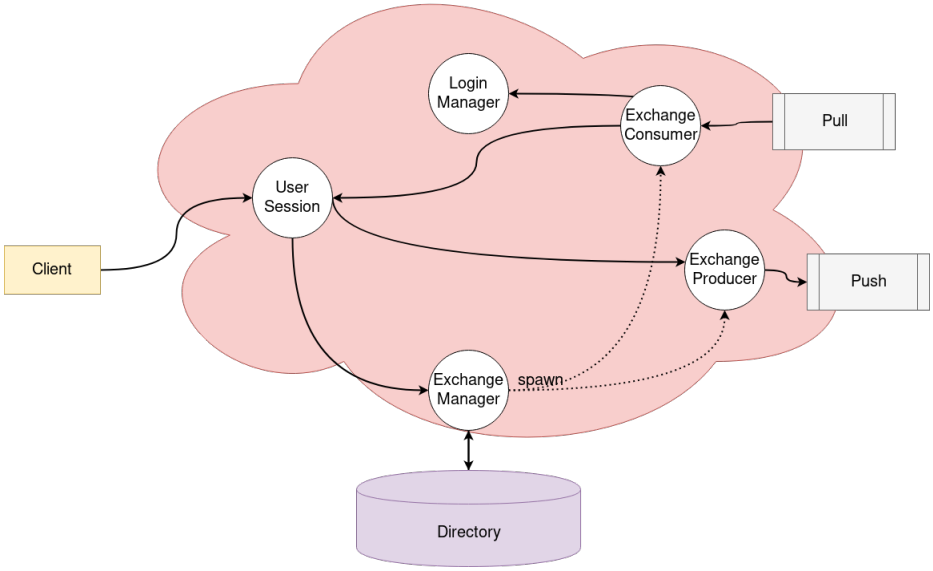


Figura 3.2: Comportamento dos atores no front-end

Quando uma ***exchange*** finaliza uma transação, informa o cliente através do frontend. Como já foi descrito acima, várias ***exchanges*** podem comunicar com o mesmo ***pull*** (ou para um conjunto reduzido de ***pulls***). O ator responsável para ler estas mensagens é o ***exchange consumer***, no entanto, este ator não tem informação necessária para poder enviar a notificação ao ***user session*** correspondente ao utilizador. É obrigado a perguntar ao ***login manager*** por esta informação, e procede a enviar-lhe a mensagem.

3.4 Exchange

A **exchange** é a entidade responsável por receber todas as ordens de compra e venda efetuadas pelos clientes, relativas às empresas que nela são negociadas, gerando transações quando é efetuado uma correspondência entre duas ou mais ordens. A **exchange** foi desenvolvida em java com recurso ao middleware de messaging **ZeroMQ**, permitindo uma fácil ligação entre componentes e ao mesmo tempo a escrita de código sequencial, sendo garantido pelo mesmo a gestão da concorrência. Quando uma transação é gerada, são enviadas ao *front-end* as mensagens que permitem que os envolvidos na mesma sejam notificados, o diretório é atualizado com essa mesma transação com recurso a um pedido http **PUT**, e é publicado através do

socket **PUB** uma notificação para que todos os subscritores de uma dada empresa recebam a notificação acerca da transação.

3.5 Diretório de Ações

O diretório de ações é um servidor que disponibiliza uma interface RESTful, através da qual é possível obter informações sobre as empresas. Esta interface disponibiliza apenas o recurso *Company* que, por sua vez, disponibiliza três operações distintas:

- **GET /companies** devolve um *array* com os nomes das várias empresas cujas ações podem ser negociadas;
- **GET /company/{name}** devolve um objeto *JSON* que descreve a empresa *name*, fornecendo os dados da *exchange* em que esta é negociada, assim como informações sobre o estado das suas ações nos últimos dois dias. Caso não seja encontrada uma empresa com o nome indicado é devolvida uma resposta com código *404 Not Found*;
- **PUT /company/{name}** atualiza os dados relativos às ações daquela empresa de acordo com a nova transação. Dado que os dados recebidos são apenas usados para estatística e posteriormente descartados, optamos pelo método *PUT* pelas suas características de idempotência. Caso o pedido seja recebido fora do horário de funcionamento das *exchanges*, é devolvida uma resposta com código *403 Forbidden*.

3.6 Serialização

Para que os dados sejam transmitidos sem problemas de heterogeneidade entre sistemas, foi utilizada serialização com recurso a **protocol buffers** na troca de mensagens entre o **cliente** e o **front-end** assim como na ligação entre o **front-end** e a **exchange**. Toda a comunicação com o diretório de ações é realizada através de JSON.

4 | Conclusão

Com este projeto verificamos que construir um sistema distribuído nem sempre é uma tarefa assustadora quando usadas as tecnologias mais apropriadas. Uma vez planeada a arquitetura do sistema, construir os vários componentes revelou-se uma tarefa relativamente simples. Graças ao uso de protocolos de serialização foi possível obter uma comunicação eficaz entre os vários componentes. Assim, o facto de possuirmos vários componentes trouxe apenas simplicidade ao projeto, permitindo que cada componente fosse construído utilizando a tecnologia mais apropriada.

Apesar da falta de tempo para realizar o trabalho concluímos que atingimos os resultados pretendidos, mesmo que alguns dos requisitos descritos do enunciado não tenham sido realizados. No geral, o grupo encontra-se satisfeito por ter sido capaz de desenvolver um sistema distribuído evitando as complexidades do controlo de concorrência aprendidas nas unidades curriculares anteriores.