Paradigmas de Sistemas Distribuídos

Trabalho Prático

Relatório de Desenvolvimento

Miguel Oliveira	Pedro Moura	César Silva
pg41088	pg41094	pg41842

Universidade do Minho,

20 de Janeiro de 2020

Conteúdo

Capítulo 1

Introdução

O seguinte relatório descreve o desenvolvimento do trabalho prático da UC de Paradigmas de Sistemas Distribuídos.

O trabalho prático tem como objetivo aplicar os conhecimentos adquiridos nas aulas, mais nomeadamente protocolos de comunicação entre diferentes (Java e Erlang)linguagens utilizando serviços de serialização como *Protocol Buffers*, programação com atores, *Message Oriented Middleware* e serviços REST.

Capítulo 2

Cliente

O cliente é a componente do sistema que interage com o utilizador, podendo desempenhar dois papéis: **Fabricante** e **Importador**, sendo que são disponibilizadas diferentes interfaces consoante estes papéis.

2.1 Fabricante

Um fabricante é um cliente que disponibiliza ofertas para a produção de artigos de uma determinada área, indicando o **nome do artigo**, **quantidades mínima e máxima**, **preço unitário** e **período de tempo** durante o qual um importador pode fazer ofertas.

Um fabricante é criado da seguinte forma:

```
/client manufacturer @area
```

Onde *manufacturer* é o campo que identifica que o cliente é um fabricante, e *@area* é o nome da área em que é especialista (ex.: Tecnologia, Alimentação, etc).

2.2 Importador

Um importador é um cliente que demonstra interesse em certos fabricantes, podendo efetuar encomendas a artigos dos mesmos, indicando a **quantidade** desejada e o **preço unitário** que está disposto a pagar. Para além disto, pode demonstrar interesse em novos fabricantes.

Um importador é criado da seguinte forma:

```
/client importer @area @m1 @m2 ...
```

Os primeiros 2 parâmetros são análogos ao que foi visto em cima. Os parâmetros *@m* são os nomes dos fabricantes ao qual deseja subscrever (sendo que dentro da aplicação pode subscrever a novos).

Tanto o importador como o fabricante devem ser sempre notificados acerca do resultado de uma negociação de uma encomenda.

2.3 Comunicação

O cliente comunica principalmente através de *Protocol Buffers* com o servidor *Frontend*. É através desta comunicação que é feita a autenticação no sistema, bem como o envio de ofertas e encomendas por parte dos fabricantes e importadores, respetivamente.

Além disto, os clientes conseguem comunicar entre si através de *sockets* do `ZeroMQ`, de forma a implementar um modelo *publish-subscribe*. Este modelo permite que sempre que um fabricante publique uma oferta, os importadores interessados nessa oferta sejam notificados, de modo a poderem realizar uma encomenda.

Capítulo 3

Front-End

O comportamento inicial do Front-End começa pela criação de um *ListeningSocket* para aceitar ligações e um actor especial que denominamos *Login Manager* que comunica via mensagens nativas *Erlang*, cuja função é validar credenciais recebidas.
Quando é recebida uma ligação, é alocado um ator para se encarregar desta. Este pode ser dividido em 2 grandes fases - autenticado e não autenticado. Tudo o que um utilizador não autenticado pode fazer é realizar tentativas de autenticação. Por sua vez quando um utilizador é autenticado, este abre uma ligação a um **Negociador** e a partir deste ponto, simplesmente encaminha pedidos vindos do cliente para este.
A comunicação usada entre o **Cliente** e o **Front-End** usa *gpb*, uma implementação em *Erlang* de *Protobuffers* - `https://github.com/tomas-abrahamsson/gpb` - o qual apresenta um guia extenso em como usar o módulo.

Capítulo 4

Negociador

Esta componente é crucial nas operações de iniciar uma produção de um artigo e na adição de uma ordem de compra por parte de um importador.

Adicionalmente, esta componente está responsável por enviar pedidos ao Cátalogo para reportar os registos de Fabricantes e quaisquer alterações relativas às Negociações em curso(como a adição de um pedido de encomenda de um Importador).

4.1 Diferenciação de Negociadores

Uma vez a necessidade de disponibilizar inúmeros artigos e portanto um elevado número de negociações em curso, optamos por separar as Negociações por Área, sendo cada Negociador responsável pelas negociações de uma certa Área.

Deste modo, cada Negociador corre numa porta específica conhecida pelo `Front-End` que irá redirecionar os pedidos, consoante a Área do Fabricante, ou do Importador.

4.2 Notificações

Para poder disponibilizar a funcionalidade de **Notificações** o Negociador interage como um `Broker` na tecnologia `ZeroMQ`, num padrão **Publisher-Subscriber** cuja implementação está disponível na classe `NotificationBroker` que executa como um `Thread` independente neste processo.

4.3 Término de Negociações

Para poder implementar o término de uma negociação necessitamos de correr um temporizador que verificasse o estado de cada negociação em curso e comparasse o tempo atual com o tempo final de negociação.

No fim do período de negociações de uma negociação, é verificado se a quantidade mínima de produtos imposta pelo Fabricante foi satisfeita, caso contrário, todas as partes interessadas(respetivo Fabricante, e todos os Importadores que registaram um pedido de encomenda) recebem uma mensagem a indicar que a oferta de produção foi cancelada.

Se existir pelo menos uma ordem de encomenda cuja quantidade seja maior que a quantidade mínima necessária para produção, procuramos a oferta de encomenda que contém o maior valor(`quantidade * preco_unitario`).

Capítulo 5

Catálogo

O Cátalogo é responsável por guardar a informação dos fabricantes e das negociações em curso e de disponibilizar uma interface `REST`(usando a *framework* `Dropwizard`) para obter as informações relativas aos Fabricantes e às Negociações.

Além de disponibilizar as operações de `GET` disponibilizamos também operações de `POST`, `PUT` e `DELETE` que só deverão ser usadas pelo **Negociador** para atualizar a informação do Catálogo.

Criamos 2 recursos, `FabricanteResource` e `NegociacaoResource` que estão responsáveis por disponiblizar as operações necessárias. Para o correto funcionamento destes Recursos, necessitamos de implementar várias **Representações**, em formato `JSON`, que serão usadas para a troca de informação das operações. Implementamos então as seguintes representações:

- `Fabricante` - Contém o nome do Fabricante, a lista dos seus `Artigos` disponíveis atualmente.
- `Artigo` - Contém a informação que o Fabricante disponibiliza quando a oferta de produção de um artigo.

  Ou seja, o nome do Artigo, a quantidade mínima a ser produzida, a quantidade máxima, o preço mínimo unitário e o período tempo permitido para negociações.
- `Negociação` - Contém o nome do fabricante que disponibilizou o Artigo, o Artigo que irá ser produzido e a Lista de `OrdemCompra` que representa as ofertas de encomendas feitas por importadores relativo a este artigo.
- `OrdemCompra` - Contém o nome do importador que realizou a oferta de encomenda, a quantidade do produto que o importador deseja comprar, e o preço unitário que está disposto a pagar.

Uma vez que as operações aos Recursos podem ser concorrentes, necessitamos de aplicar algum controlo de concorrência nas operações relativas às estruturas de dados, o que concretizamos utilizando a *keyword* do Java `synchronized` para assegurar atomicidade nas operações.

Capítulo 6

Conclusão

Achamos que a solução proposta, responde aos vários problemas propostos no enunciado, quer a nível de uso de conceitos usados - *ZeroMQ*, *REST*, *Erlang* - como também ao nível das funcionalidades pedidas.

Como trabalho futuro, poderiamos considerar a adesão de negociadores em *runtime*, tópicos de subscrição mais extensos ou até uma interface gráfica para o cliente, de maneira a consolidar ainda mais o implementado.