Paradigmas de Sistemas Distribuídos

Trabalho Prático

-

Mestrado em Engenharia Informática

Universidade do Minho

Relatório

Grupo	
PG41080	João Ribeiro Imperadeiro
PG41081	José Alberto Martins Boticas
PG41091	Nelson José Dias Teixeira

20 de Janeiro de 2020

Conteúdo

Capítulo 1

Introdução

Neste trabalho prático é requerido o desenvolvimento de um protótipo de uma plataforma de negociação entre fabricantes e importadores de produtos. Este protótipo é composto por cliente, servidor de *front-end*, negociador e catálogo de entidades e negociações em curso. Os clientes podem existir em elevado número, sendo que cada um deles desempenhará sempre ou o papel de fabricante ou de importador. Cada fabricante indicará a disponibilidade para produzir um determinado artigo, numa quantidade mínima e máxima, a um preço mínimo (unitário), bem como o período de tempo durante o qual os importadores poderão fazer ofertas de encomenda (período de negociação). Por sua vez, cada importador indica a quantidade e valor unitário a que está disposto a pagar por um determinado artigo de um fabricante. Os clientes autenticam-se no servidor de *front-end*, o qual encaminha as suas ordens para um (de entre vários) negociadores. O catálogo disponibilizará uma interface *RESTful*, que permitirá obter informação sobre os fabricantes, importadores, e negociações em curso. Como tal, por forma a implementar este protótipo, foi utilizada a linguagem de programação *Java* (cliente, negociador e catálogo), *Erlang* (servidor *front-end*), e, ainda, *Protocol Buffers*, *ZeroMQ* e *Dropwizard*.

Capítulo 2

O sistema

2.1 Funcionamento

O sistema é dividido em quatro partes principais: servidor *front-end*, cliente, negociador e catálogo.

O cliente é a entidade que representa importadores ou fabricantes. O mesmo apresenta uma interface rudimentar que permite ao utilizador fazer ofertas ou encomendas, dependendo se é um importador ou fabricante.

O servidor, por sua vez, é responsável por receber estas encomendas e fazê-las chegar ao negociador correspondente. Tem ainda um pequeno mecanismo de autenticação para os seus clientes. Foi ainda tomada a decisão de que cada produto seria atribuído a um negociador, sendo esse negociador o responsável por todas as negociações que existam com esse produto.

Um negociador é responsável por intermediar as negociações entre importadores e fabricantes, fazendo corresponder ofertas e encomendas que satisfaçam os requisitos de ambas. Quando uma negociação chega ao fim do seu tempo, envia as informações de fabricante, importadores e respetivas quantidades e preços para o servidor.

Por fim, o catálogo apresenta uma interface *RESTful* e ainda um sistema de *publish-subscribe*, sendo que os subscritores são importadores e são notificados quando um fabricante, que eles subscreveram, publica um oferta, sendo que esta chega pelo negociador responsável.

Por forma a facilitar a execução de todas as entidades envolvidas, é incluído, junto à implementação deste trabalho, um ficheiro ***Makefile*** com todas as intruções necessárias para garantir o arranque correto do sistema.

2.2 Implementação

2.2.1 Servidor

A implementação do servidor foi feita em *Erlang*. O servidor divide-se em diversos ficheiros, cada um dos quais corresponde a um tipo de ator, que se dividem em seis tipos:

- Servidor;
- `loginHandler`;
- Clientes;
- Importadorers;
- Fabricantes;
- `negotiatorsHandler`;
- Negociadores.

O servidor é apenas um ponto de ligação entre clientes e negociadores, sendo que decide ainda quais os negociadores que tratarão de determinadas ofertas/encomendas. Vejamos em mais detalhe o trabalho de cada tipo de atores e a forma como o servidor reage a cada situação.

2.2.1.1 Servidor central

Este é o primeiro ator criado, responsável pela criação de todos os outros atores e por aceitar novas conexões (clientes), criando um novo ator por cada uma. É ainda responsável pela conexão inicial de todos os negociadores, criando um ator por cada conexão. Assim, cada cliente e cada negociador terá um correspondente ator no servidor, com o qual comunicará diretamente, já que cada um destes atores será o responsável pelo *socket* correspondente.

2.2.1.2 `loginHandler`

Este ator é registado e criado pelo anterior. É o responsável pela autenticação e registo dos clientes, guardando todas as informações nesse sentido. De realçar que só é contactado diretamente por outros atores do servidor, nomeadamente os criados a cada nova conexão de clientes, que veremos de seguida.

2.2.1.3 Clientes

Ator criado a cada nova conexão. Espera a receção de uma comunicação *TCP* com a autenticação do cliente e, usando o `loginHandler`, confirma a sua identificação. Segue-se a criação de novo ator, importador ou fabricante, que fica responsável pela comunicação com esse cliente. O ator atual é substituído pelo novo.

2.2.1.4 Importadores e Produtores

Ator criado pelo anterior, sempre que um cliente, do tipo importador ou fabricante, se autentica com sucesso. Fica então responsável pela comunicação com o cliente, recebendo as suas encomendas/ofertas e fazendo com que as mesmas cheguem ao respetivo negociador, através do `negotiatorsHandler`.

2.2.1.5 `negotiatorsHandler`

O `negotiatorsHandler` é um ator, registado pelo servidor central, aquando da conexão de todos os negociadores, que é contactado pelos atores dos importadores ou fabricantes que queiram fazer encomendas ou ofertas. Ao recebê-las, consoante o produto, passa-as para o respetivo negociador. Cada novo produto é atribuído a um negociador, em estilo *round-robin*.

2.2.1.6 Negociadores

Existirá um ator deste tipo para cada negociador (quantidade pré-definida). Um negociador é responsável por um certo número de produtos, tratando de todas as ofertas e encomendas desse produto. Quando um cliente faz uma oferta ou encomenda, a mesma acaba por chegar ao ator (negociador) responsável pelo produto, fazendo-a depois chegar ao negociador propriamente dito (exterior ao servidor).

2.2.2 Cliente

Tal como pedido nos requisitos do trabalho, a implementação do cliente foi feita em *Java*. Para além de todas as classes nativas, foram ainda usadas funcionalidades das bibliotecas *ZeroMQ*, uma implementação em *Java* da ferramenta *ZeroMQ* para envio/receção de mensagens assíncronas, e *Protocol Buffers*, um formato de serialização de dados desenvolvido pela *Google*. Posto isto, passaremos ao detalhe do funcionamento do cliente.

O cliente, aquando do início da sua utilização por parte de um utilizador, começa por pedir os dados de autenticação e entra em contacto com o servidor para verificar a sua validade. Em caso afirmativo, é criada uma *thread* adicional que recebe informações do servidor e o utilizador é enviado para um de dois menus: o de fornecedor ou de importador. Se o utilizador for um importador, é lançada ainda outra *thread*, responsável por receber updates do catálogo quanto a produtores que o utilizador subscreveu.

De seguida, após toda esta configuração inicial, é mostrado um dos seguintes menus com as operações indicadas:

1. **Importador**:
    - Oferta de encomenda;

- Subscrever notificações;
- Cancelar notificações.
- Atualizações.

2. **Fornecedor**:
    - Oferta de produção;
    - Atualizações.

Começando pelas operações do importador, temos que este pode encomendar um produto, tendo para isso de indicar o nome do produtor/produto, o número de unidades e o preço que está disposto a pagar por unidade. Pode ainda subscrever ou cancelar a subscrição de atualizações sobre um determinado produtor, bastando indicar o nome desse produtor.

Passando ao fornecedor, este pode apenas colocar uma oferta de produção, indicando o nome do produto, quantidade mínima/máxima, preço mínimo por unidade e período de negociação (em segundos).

Quanto às mensagens do servidor e do catálogo, estas são apresentadas sempre que há informações para mostrar, tendo em conta que o utilizador não está a realizar nenhuma operação ou quando este clica na opção "Atualizações".

Em ambos os casos, é dada a opção ao utilizador para sair da interface.

2.2.3 Negociador

No que toca ao desenvolvimento do negociador, este foi implementado em *Java* com recurso à linguagem neutra *Protocol Buffers* para efetuar a serialização de dados envolvidos nos processos de comunicação.

Por forma a satisfazer os requisitos impostos, foram declaradas duas variáveis relativas às negociações em curso no sistema e, ainda, aos importadores que não possuem de momento nenhum conjunto de produtores associado. De forma a instanciar o negociador em causa, foi declarado um *socket* que irá comunicar com o servidor, processando vários pedidos em simultâneo.

O negociador presente neste protótipo, ao receber pedidos do servidor, consegue processar se o mesmo corresponde a um pedido proveniente de um produtor ou de um importador. Caso este seja de um produtor, é aberto um período limitado de negociação para determinar quais os importadores que serão escolhidos para proceder à produção de um determinado produto. O critério adotado nestas situações é escolher qual o importador que possui o maior valor total de encomenda para o artigo em questão. Por outro lado, se o pedido for de um importador, é verificado se o mesmo está associado à produção de um determinado tipo de artigo. Em caso afirmativo, é atualizada, com essa informação, a variável relativa às negociações em curso. Caso contrário, o importador é colocado na variável que diz respeito aos importadores que não possuem de momento nenhum conjunto de produtores associado.

As mensagens que circulam na comunicação entre negociador e servidor encontram-se serializadas (através da linguagem *Protocol Buffers*) por forma a garantir o mesmo grau de interpretabilidade dos dados em ambas as entidades.

2.3 Catálogo

Relativamente ao catálogo presente neste sistema, este é um componente onde se guarda informação útil acerca dos fabricantes e negociações em curso. Esta entidade do protótipo pode ser vista como um ficheiro de *logs*, que é atualizado e acedido sempre que for necessário. Isto possibilita guardar um histórico do estado momentâneo do funcionamento da aplicação.

Para tal, foram declaradas 4 variáveis na classe deste componente por forma a armazenar toda a informação pertinente:

- **negociations**: variável com os dados das negociações vigentes;
- **importers**: variável correspondente ao conjunto dos importadores;
- **producers**: variável correspondente ao conjunto dos produtores;
- **subscriptions**: variável com os dados das subscrições registadas.

Sobre este componente podem ser exercidos 6 tipos distintos de operações:

1. **POSTNegotiation**: inserção da informação relativa à iniciação de um novo periodo de negociação;
2. **DELETENegotiation**: remoção da informação relativa à negociação em vigor;
3. **GETProducerInfo**: obtenção da informação relativa ao produtor;
4. **GETEntities**: obtenção dos dados de todas as entidades (importador e produtor);
5. **Subscribe**: operação relativa à subscrição de atualizações de um produto de um produtor;
6. **Unsubscribe**: operação que permite cancelar uma subscrição de novas atualizações de um determinado produto de um pordutor.

Com estas operações é possível reconhecer o estado atual do funcionamento do sistema implementado, acedendo e alterando o mesmo consoante o desenrolar da execução do protótipo.

Capítulo 3

Conclusão

Após a realização deste projeto prático, foi possível identificar e especificar as entidades principais que intervêem neste protótipo, e, ainda, as suas respetivas caraterísticas. Para cada uma destas, impôs-se um desafio de implementação na medida em que as linguagens de programação adotadas são heterógeneas entre si. Para além disso, foi possível constatar a dificuldade de integrar diferentes linguagens com diferentes paradigmas, bem como os desafios da construção de um sistema distribuído deste tipo. Os elementos que compõem este grupo depararam-se com algumas adversidades na utilização da linguagem neutra *Protocol Buffers* para a implementação do servidor.

Capítulo 4

Webgrafia

- ***Protocol Buffers***:
  https://developers.google.com/protocol-buffers
- ***Protocol Buffers - Java***:
  https://developers.google.com/protocol-buffers/docs/reference/java-generated
- **Documentação - *Java***:
  https://docs.oracle.com/en/java/javase/11/docs/api/index.html