Instituto Superior de Engenharia de Lisboa

Mestrado em Engenharia Informática e Multimédia

Mestrado em Engenharia Informática e Computadores

Ano Letivo 2022/2023

Computação Distribuída

TP2 – Sistema Encomendas

Data: 28/12/2022 *Docente: Luís Assunção*

Grupo: 08

Alunos (Nome e número):

Alexandre Moreira 47463
Diogo Amorim 47248
Gonçalo Ferreira 50309

Índice

Introdução

A aplicação *Courier App*, associada ao servidor *Courier* eleito recebe a morada de recolha e a morada de destino de uma encomenda e começa o seu trabalho.
Inicialmente o Cliente realiza um pedido com as informações do ponto de recolha e do ponto de destino da encomenda. De seguida, devido ao grande número de pedidos existe um grupo *Spread* de aplicações *Worker* que processam os pedidos submetidos.
Os *Workers*, enviam em multicast as mensagens para um grupo *Spread* de *Couriers*, em que cada um representa no sistema, um *Courier App (estafeta)*.
Os servidores *Courier* do grupo da região, definem qual o *Courier* que vai transportar a encomenda. O *Courier* eleito envia a resposta ao Cliente com informação do ID da sua encomenda e da chave para selar a mesma através da componente *Publish/Subscribe*.
A aplicação *Courier App*, associada ao servidor *Courier* eleito recebe a morada de recolha e a morada de destino de uma encomenda e começa o seu trabalho.

Arquitetura Geral

Neste capítulo serão tratados aspetos gerais da arquitetura das várias componentes utilizadas neste sistema bem como a estrutura das mensagens entre essas componentes.

Arquitetura Publish/Subscribe

Este sistema utilizará uma componente do tipo *Publish/Subscribe* através de *RabbitMQ*. Esta arquitetura poder-se-á dividir em duas componentes: a componente que permite ao cliente fazer pedidos de transporte de encomendas com as informações necessárias e a componente que permite ao cliente receber as informações específicas da sua encomenda.

Cliente-Worker

Para o Cliente conseguir publicar os seus pedidos na componente *Publish/Subscribe* foi adotada a seguinte arquitetura:

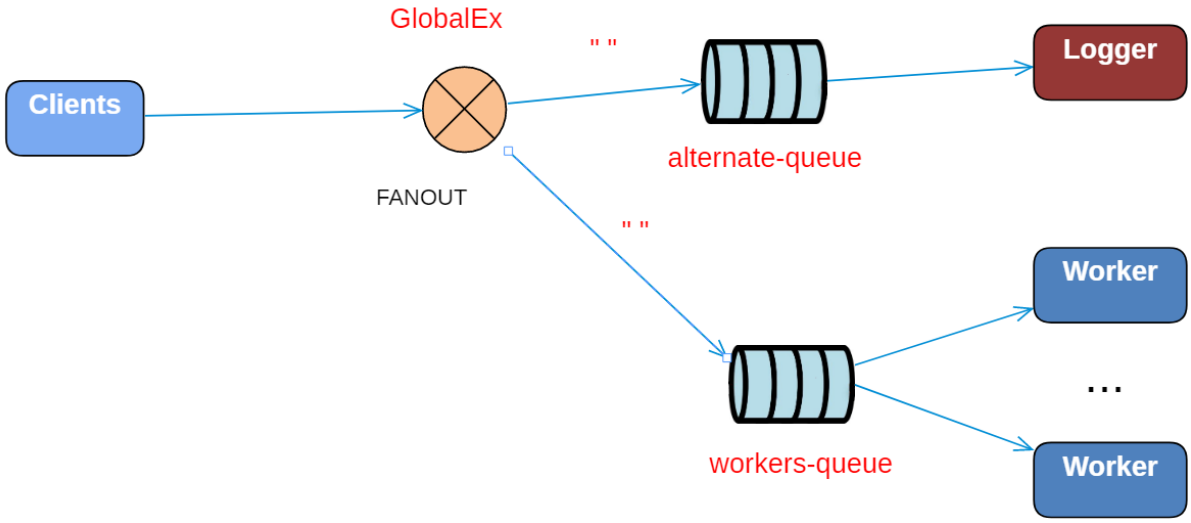

Figura 1 - Organização do pubsub (server-side)

Existirá um *exchange* global onde todos os clientes publicam os seus pedidos. Para possibilitar um maior controlo de todo o tráfego foi implementado um *Logger* que recebe todos os pedidos feitos pelos clientes. Devido ao número elevado de pedidos existem várias aplicações prontas a receber os pedidos dos clientes, pelo que foi implementado uma *queue* direcionada para as estas aplicações. Para possibilitar a paralelização das aplicações, foi implementado um padrão *work-queue* onde, é necessário que várias aplicações *worker* estejam associadas à mesma *queue*. Isto possibilita que, quando o cliente envia um pedido apenas uma destas aplicações o irá receber, facilitando assim eventuais problemas de escolha da aplicação que iria tratar o pedido.

Courier-Cliente

Depois do *Worker* enviar o pedido a ser processado para o *Courier* este irá gerar um ID para a encomenda e a chave para o cliente a conseguir selar. Para enviar estas informações ao cliente o *Courier* tirará partido da componente *Publish/Subscribe* do sistema. Abaixo encontra-se a arquitetura utilizada para tal efeito:

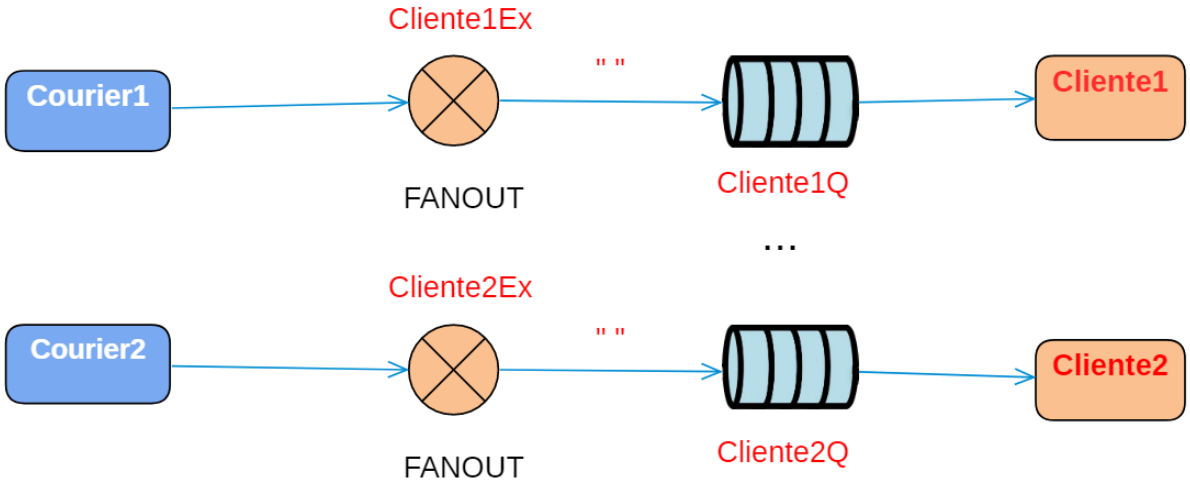

Figura 2 - Estrutura do Pubsub (client-side)

Dado que as respostas aos clientes são privadas, cada cliente terá de ter o seu próprio *exchange* e *queue*. Sendo que as únicas mensagens que irão passar por estes *exchanges* são as respostas aos clientes, não será necessário qualquer tipo de filtro para as mesmas, sendo a *routing key* vazia.

Arquitetura Spread

Na mensagem que o cliente envia o mesmo indica a região da sua encomenda. Baseado nessa região o *Worker* enviará em *multicast* para o grupo de *Spread* dessa região. Os *Couriers* decidirão qual dos integrantes desse grupo irá processar o pedido. De seguida o *Courier* encarregado do processamento do pedido gera o ID da encomenda e uma chave para o cliente poder selar a mesma. Por cada *Courier* existe um *Courier App* que representa o estafeta que se dirige ao local de recolha que o cliente indicou e transporta a encomenda até ao local de destino.

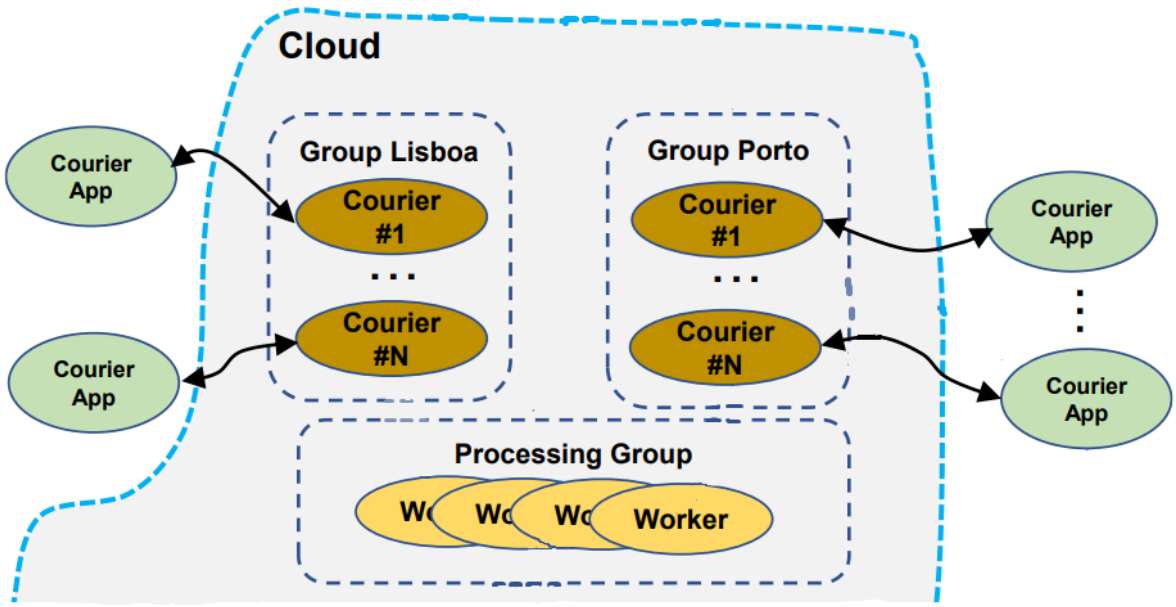

Figura 3 - Arquitetura do Spread

Na mensagem que o cliente envia o mesmo indica a região da sua encomenda. Baseado nessa região o *Worker* enviará em *multicast* para o grupo de *Spread* dessa região. Os Couriers decidirão qual dos integrantes desse grupo irá processar o pedido. De seguida o Courier encarregado do processamento do pedido gera um *boxID*, para o ID da encomenda e uma chave para o cliente poder selar a mesma. Por cada *Courier* existe um *Courier App* que representa o estafeta que se dirige ao local de recolha que o Cliente indicou e transporta a encomenda até ao local de destino.

Contratos e Estrutura das Mensagens

Para possibilitar que as várias componentes do sistema se interliguem tiveram de ser criados protocolos de comunicação entre as mesmas. A seguir serão descritas as várias mensagens entre as componentes do sistema.

Pedido do Cliente

Quando o cliente pretende enviar um pedido para transporte da sua encomenda este deve especificar as várias informações abaixo ilustradas.

```
message ClientePedido {
    string IDRequest;
    string moradaRecolha;
    string moradaDestino;
    string regiao;
    string nomeExchange;
}
```

Para ser possível, para o cliente, identificar os vários pedidos que o mesmo envia, este gera automaticamente um ID por cada pedido. De seguida deve inserir qual a morada de recolha da encomenda e de destino, bem como a região onde a mesma se encontra (e.g Lisboa, Porto…). Para o cliente receber as mensagens é criado um *exchange* na componente *Publish/Subscribe*, pelo que é necessário enviar o nome do mesmo no pedido para que, posteriormente, quem processa o pedido, poder publicar a resposta nesse *exchange*.

Pedido do Worker

```
message WorkerPedido {
    string IDRequest;
    string moradaRecolha;
    string moradaDestino;
    string nomeExchange;
}
```

Depois de um *Worker* receber o pedido de um cliente, extrairá a informação referente à região para poder fazer o *multicast* para o grupo de *Spread* indicado. As restantes informações irão ser reencaminhadas para o grupo indicado.

Intra Couriers

As mensagens troadas entre Couriers podem assumir vários tipos, dependendo da sua função.
Caso o tipo de mensagem recebido seja um pedido para processar encomenda então o formato será o apresentado no pedido do *Worker*.

Caso algum servidor se junte ao grupo então será necessário informar ao recém-chegado todos os pedidos pendentes existentes. O formato desta mensagem será uma lista de pedidos com o formato descrito no pedido do *Worker*.

Outro tipo de mensagem existente será a de uma eleição que assumirá o seguinte formato:

```
message EleicaoPedido {
    string IDRequest;
}
```

A mensagem de eleição apenas indica o ID do pedido que se pretende processar. O primeiro *Courier* que indicar que pretende processar um certo pedido é o que fica com essa tarefa. Dado que a ordem de mensagem é garantida pelo *Spread*, todos os integrantes de um grupo irão receber todas as mensagens pela mesma ordem, logo existirá consenso em qual dos *Couriers* enviou a primeira mensagem de eleição para um determinado pedido.

Resposta para o Cliente

```
message ClienteResposta {
    string IDEncomenda;
    string chave;
}
```

Ao processar uma encomenda, um *Courier* irá gerar o ID dessa encomenda e a chave que permitirá ao cliente selar essa encomenda.

Contrato Courier-CourierApp

```
service CommunicationService {
    rpc connect(Connection) returns (stream Work);
    rpc disconnect(Void) returns (Void);
    rpc free(Void) returns (Void);
    rpc busy(Void) returns (Void);

}

message Connection {
    string region = 1;
}

message Work {
    string addressSrc = 1;
    string addressDest = 2;
}
```

Arquitetura Detalhada

Neste capítulo será abordado o funcionamento detalhado das várias componentes presentes no sistema.

Cliente-Worker (*Publish/Subscribe*)

Para um cliente enviar um pedido de transporte de encomenda terá de realizar várias ações, as quais estão descritas no diagrama abaixo.

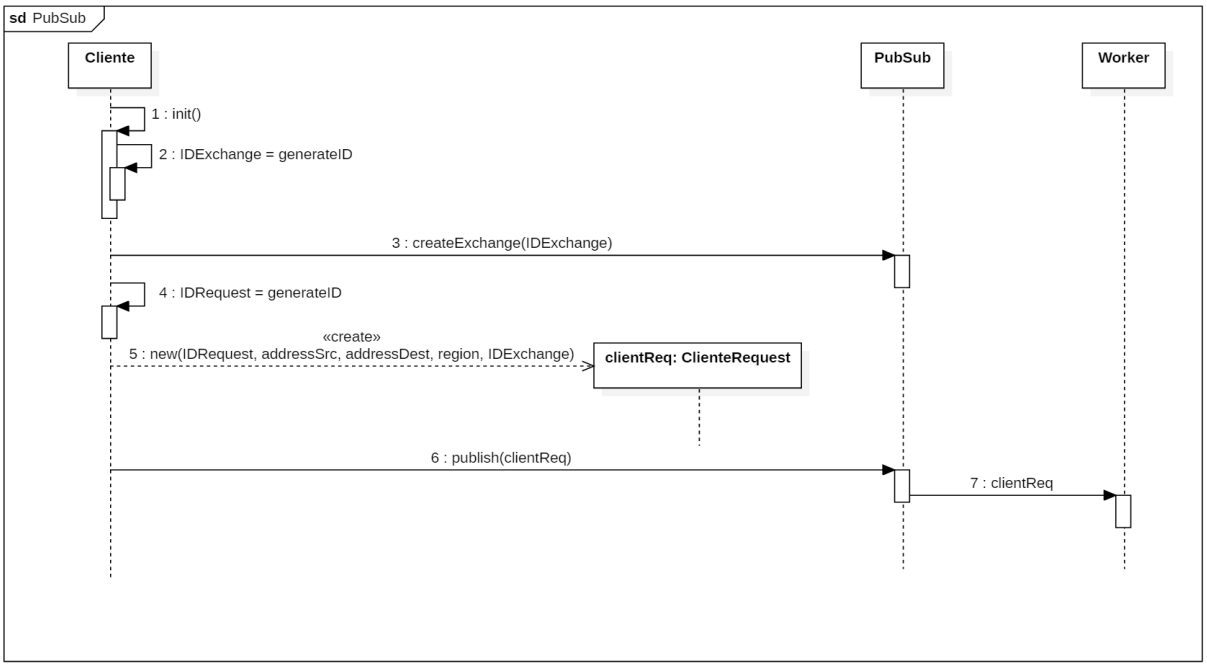

Figura 4 - Diagrama de entiração Cliente-Worker

Quando o cliente pretende enviar um pedido de transporte de encomenda, caso seja ainda não tenha feito nenhum pedido, terá de criar o seu *exchange* específico. De seguida cria um objeto *ClientRequest* onde imprimirá a morada de recolha, a morada de destino e a região. Em adição é necessária a geração de um ID para o pedido para o cliente poder distinguir as várias respostas aos seus pedidos. Também será necessário enviar o nome do *exchange* gerado para possibilitar que quem processe o pedido envie a resposta para o *exchange* adequado. De seguida o objeto criado será serializado num formato JSON e enviado para o *exchange* global onde existirão *Workers* prontos a consumir esta mensagem.

Worker-Courier (*Spread*)

Quando o *Worker* recebe uma mensagem proveniente da componente *Publish/Subscribe* o mesmo deverá realizar várias ações como é demonstrado no diagrama abaixo.

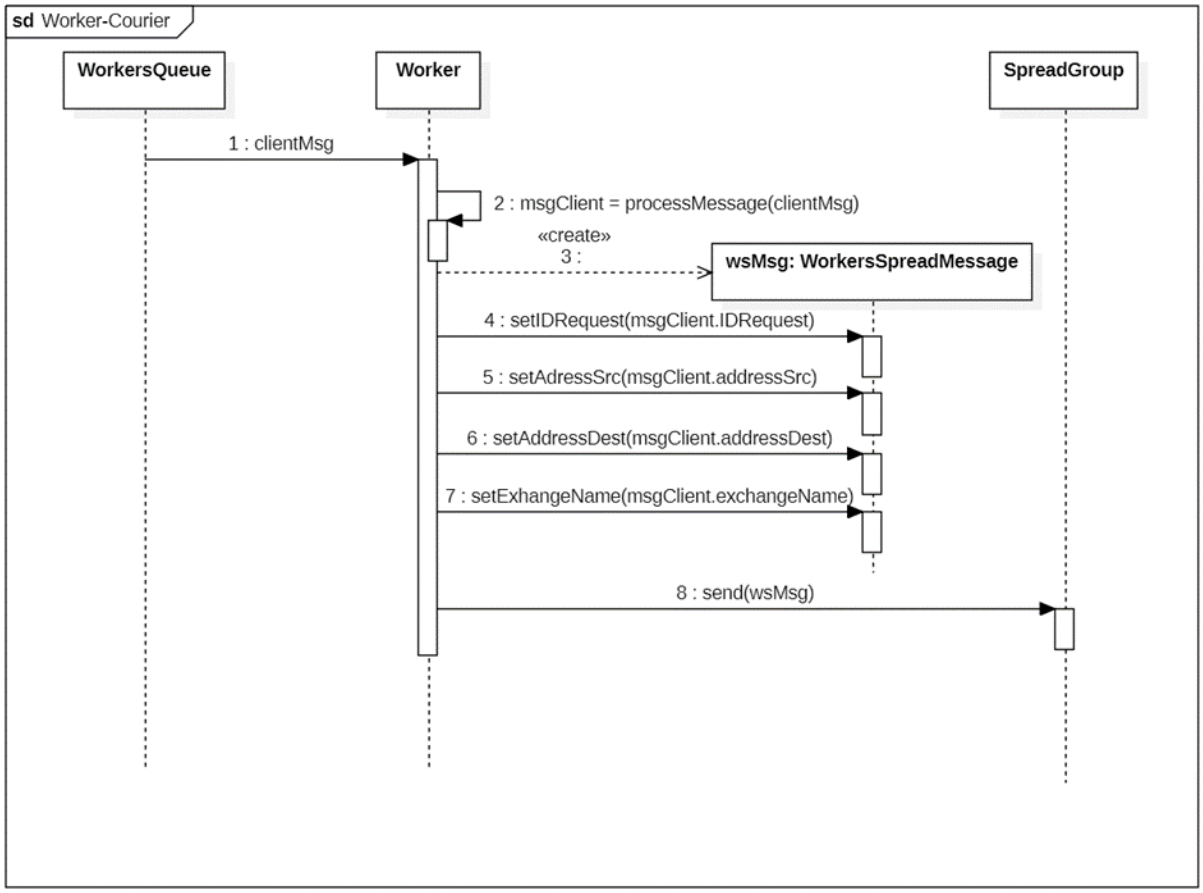

Figura 5 - Diagrama da interação Worker-Courier

O primeiro passo será converter a mensagem recebida numa classe que o mesmo consiga interpretar. De seguida a sua tarefa é extrair o campo correspondente à região da encomenda para saber para onde a deve enviar. Sabendo a região o mesmo deve imprimir os dados recebidos num novo objeto de uma classe que possa ser interpretada pelos *Couriers*. Tendo o objeto construído é feito o *multicast* para o grupo de *Spread* com o nome da região.

Intra Courier (*Spread*)

Eleição

Um courier apresenta 2 estados base: o estado de ocupado (quer seja por trabalho ou por indicação do utilizador da aplicação, representado pela *flag isBusy* a *true* no código) e o estado disponível (representado no codigo pela *flag isBusy* a false).
Este comportamento foi esquematizado no diagrama abaixo.

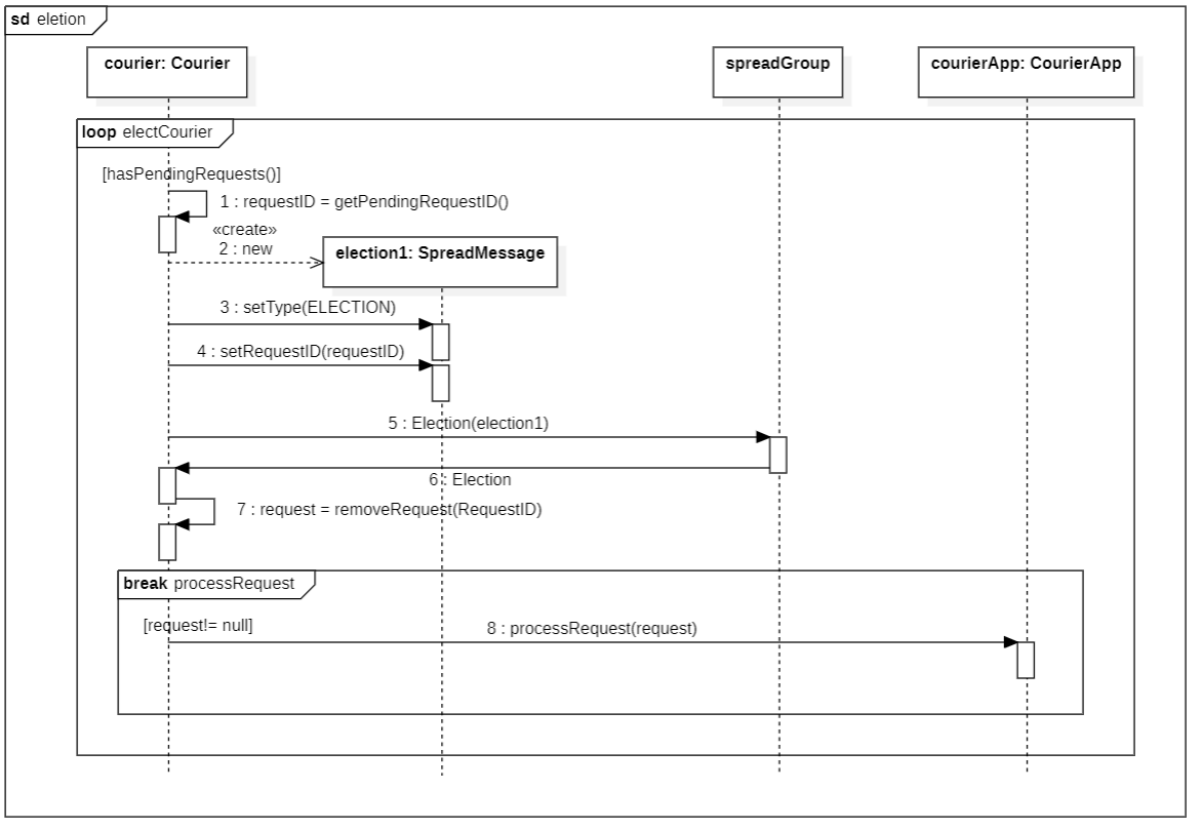

Figura 6 - Diagrama de sequencia da eleição

Uma eleição não necessita que todos os membros do grupo participem da mesma. Um membro participa de uma eleição se este não estiver ocupado (*flag isBusy* a false). Caso um membro participe da eleição, este envia uma mensagem de eleição com o ID do pedido que pretende processar(*requestID*). Ao receber as mensagens de eleição, assume-se que a primeira mensagem de eleição para um dado *request* é a mensagem de quem o vai processar, tendo em conta que no *Spread* se recebe a própria mensagem.
É usada a *flag canCandidate* para indicar se o próprio está apto para participar numa eleição. Considera-se que um elemento pode participar de uma eleição se este não está ocupado e já sabe o resultado da última eleição em que participou, para não correr o risco de ganhar 2 trabalhos ao mesmo tempo. Só consideramos que se sabe o resultado da eleição anterior quando recebe a sua própria mensagem. O processo de eleição é executado enquanto o *Courier* não estiver ocupado e existirem pedidos pendentes.
Quando se recebe uma mensagem de eleição remove-se o pedido da lista de pendentes (caso ainda esteja nos pedidos pendentes). Quando um *Courier* conclui que ganhou a eleição para um dado *request*, este remove o pedido em questão da lista de pendentes e processa o pedido.
O processo de eleição é ativo quando o *Courier* acaba um pedido e tem pedidos pendentes ou quando recebe um pedido e não está ocupado.

Novo Membro

Quando um novo membro se conecta a um grupo do *Spread*, é gerada uma mensagem de *memberShip* para informar a sua entrada. Em resposta a esta mensagem os restantes membros enviam as suas listas de pedidos pendentes para que este possa começar a

processar pedidos. (É usada uma *flag firstRun* no código para impedir que este processe pedidos antes de receber a lista de pedidos pendentes).
Após este receber a lista de pedidos pendentes, o mesmo atualiza a sua lista com essa informação e entra no modo de funcionamento normal.

Courier-Cliente (*Publish/Subscribe*)

Assim que o Courier ganha uma eleição o mesmo agora deve processar a encomenda onde terá de realizar as ações apresentadas no diagrama abaixo.

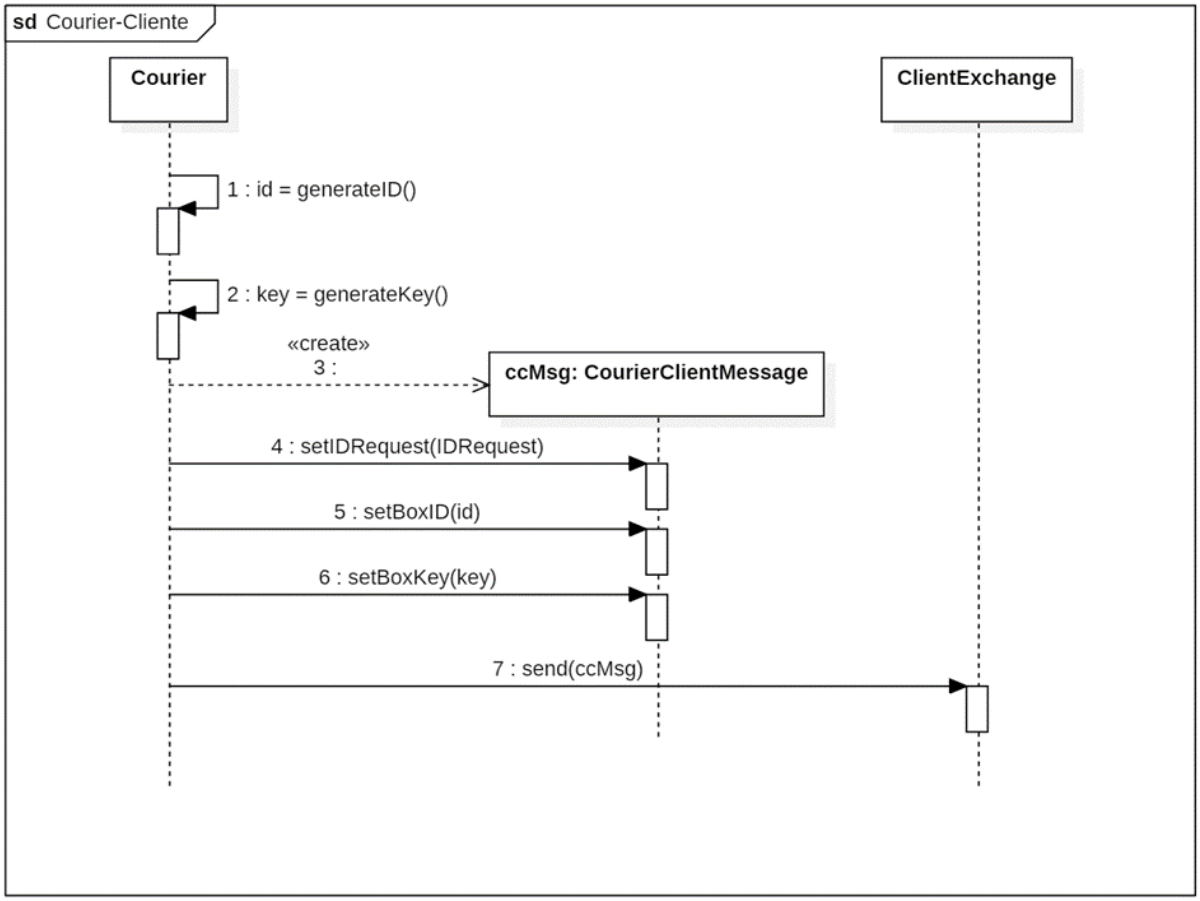

Figura 7 - Diagrama de interação Courier-Cliente

Primeiramente o deverá ser gerado um ID para a encomenda e uma chave para o cliente a poder selar. No pedido entregue pelo *Worker* também vêm duas informações adicionais: *IDRequest* e *exchangeName*. O *IDRequest* é o ID anteriormente gerado do pedido e o *exchangeName* é nome do *exchange* do cliente onde deverá ser publicada a resposta com o ID da encomenda, com o ID do pedido e com a chave.

Courier-Courier App (*gRPC*)

Quando um *Courier* ganha uma eleição também é necessário notificar o seu respetivo estafeta do trabalho a ser feito como é demonstrado no diagrama abaixo.

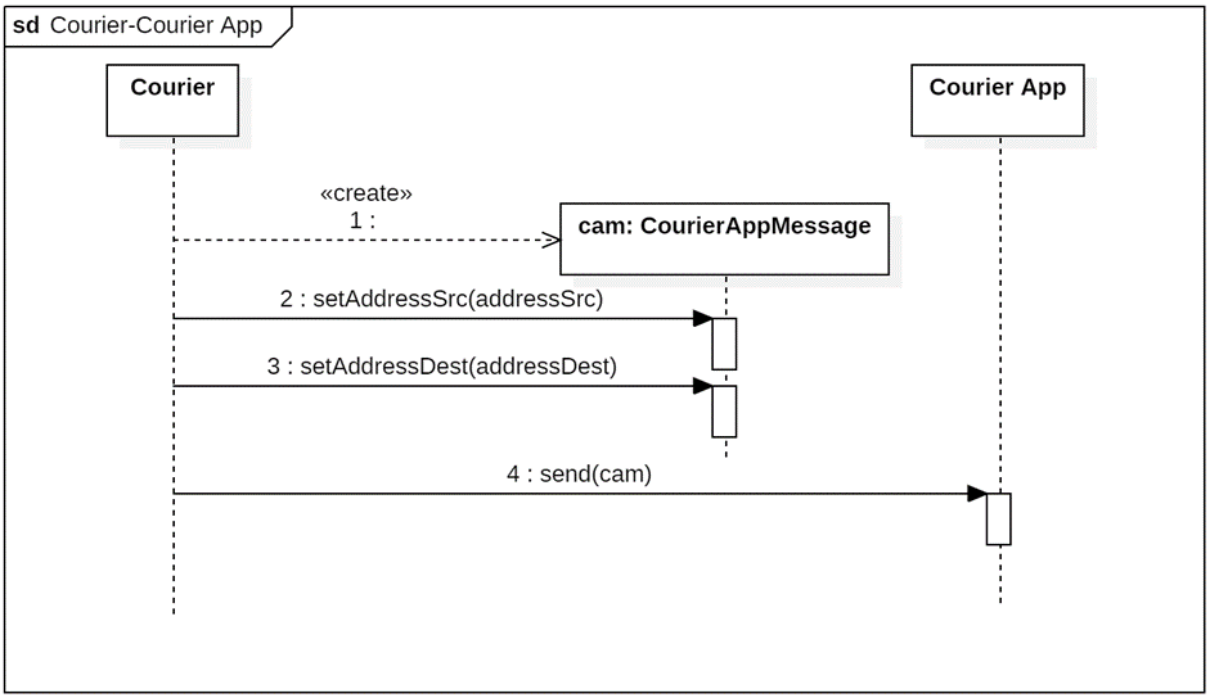

Figura 8 - Diagrama de interação Courier-Courier App

No pedido que o *Worker* envia existe um atributo que indica a morada de recolha e a morada de destino. O *Courier* extrai os valores os valores das moradas e imprime-os num novo objeto que possa ser interpretado pelo *Courier App* através de *gRPC*. Depois se criar o objeto o mesmo é enviado para o *Courier App* podendo assim este realizar a tarefa que lhe foi delegada. Quando é enviada esta mensagem o *Courier* fica em estado ocupado e só poderá sair deste estado caso o *Courier App* lhe indique que já terminou a sua tarefa.

Conclusões

Neste trabalho tivemos contacto com variadas tecnologias de âmbitos diferentes.
Para suportar os pedidos dos clientes foi utilizado o *RabbitMQ* como tecnologia de suporto a um modelo *Publish/Subscribe*. Neste modelo o cliente publicava pedidos num *exchange* global onde estariam aplicações em modo *work-queue* para tratar o mesmo. Estas aplicações *Worker*, por sua vez, reencaminhavam estes pedidos para um determinado grupo *Spread* onde estariam vários servidores conectados. Assim que os servidores deste grupo recebiam um pedido, os servidores disponíveis começavam uma eleição para averiguar qual deles é que processaria o pedido e enviava a resposta ao cliente. O servidor que ganhasse a eleição processava o pedido e gerava um ID da encomenda e uma chave para que o cliente possa selar a sua encomenda de forma segura. Depois de geradas estas informações é enviada ao cliente a resposta para o *exchange* que o mesmo indicou no seu pedido. Para se realizar o transporte da encomenda do cliente terá de existir também um estafeta. O *Courier* comunica com o estafeta através de *gRPC* onde os dois estabeleceram um contrato de comunicação *protobuf*, onde o mesmo lhe envia a morada de recolha da encomenda e a morada de destino.

No contrato entre o *Courier* e a *Courier App* foi implementado uma *stream* de servidor. Isto implica que o Courier App teria de estar permanentemente conectado ao servidor e pronto para receber trabalho a qualquer hora desde que estivesse disponível, o que pode não ser fácil num caso real. Este tipo de implementação é sensível a problemas de conexão entre o *Courier* e o *Courier App* o que pode ser comum num contexto onde o estafeta passe por zonas de menor acessibilidade ou por e simplesmente a rede seja instável. Uma outra hipótese seria a de implementar uma chamada unária onde seria o cliente que ativamente comunicava com o seu servidor para requisitar tarefas. Neste modelo caso houvesse pedidos pendentes, o *Courier* iniciava uma eleição para o mesmo e caso ganhasse, faria todo o processo já descrito.