Sistemas Distribuídos

Idea Broker

José Pedro Marques & Samuel Nunes
Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologias da Universidade de Coimbra
2011144548 | 2011158011
Outubro de 2013

Índice

Introdução

O objetivo deste projeto é implementar uma aplicação, do género de uma rede social, que permita aos utilizadores criar ideias e discuti-las sempre com respondendo com novas ideias a favor, contra ou de forma neutra à principal ou à sub-ideia à qual pretendem dar resposta. Está previsto também que seja possível comprar ações de uma ideia postas à venda pelo criador da mesma ou por outro utilizador que tenha comprado ações e pretenda revender.

Para além destas linhas gerais, o programa é também capaz de permitir a criação de tópicos, aos quais se deve associar uma nova ideia, a remoção de ideias, a listagem das transações efetuadas pelo utilizador. Permite ainda ao utilizador consultar os seus *packs* de ações e a partir daqui tornar ações disponíveis ou indisponíveis para venda ou alterar o preço de venda das ações do *pack* em causa. É ainda possível anexar um ficheiro à ideia criada.

Para tudo isto foi criado um sistema de registo/login para que o utilizador possa navegar no programa e realizar tudo o que acima foi referido.

Existe um servidor RMI que trata da informação, recebe pedidos e devolve resultados. A informação é guardada numa base de dados, pelo que esta parte do programa se vai encarregar da gestão da mesma.

Há também um servidor TCP que se encarrega de receber os pedidos do cliente e preparar o pedido para que o possa enviar para o servidor RMI. Este programa está preparado para que corra um servidor do mesmo género e que, caso o principal falhe, assumirá o seu papel sem que o utilizador seja incomodado com estas falhas.

Arquitetura Interna do Serviço

A arquitetura do projeto segue a arquitetura sugerida no enunciado e representada na Figura 1. Foram implementados dois tipos de servidor, um baseado no protocolo TCP e outro baseado no protocolo RMI. No servidor TCP implementou-se uma solução para possíveis falhas ocorridas no servidor, descrita mais adiante neste relatório.

Os servidores TCP e RMI comunicam entre sim através do protocolo RMI. Já a comunicação entre servidor TCP e cliente é feita pelo protocolo TCP. Existe ainda comunicação entre o servidor TCP principal e o secundário recorrendo ao protocolo UDP.

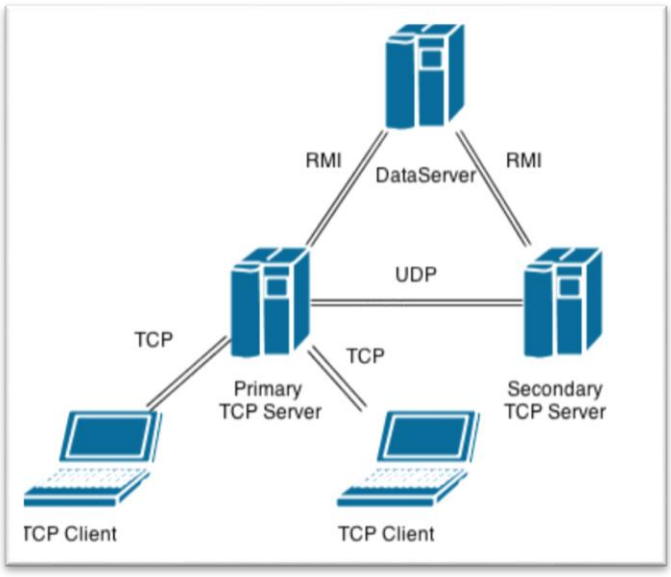


Figura 1 - Arquitetura do Sistema

No servidor RMI apenas é estabelecida a comunicação com a base de dados e colocado o servidor à escuta de ligações RMI.

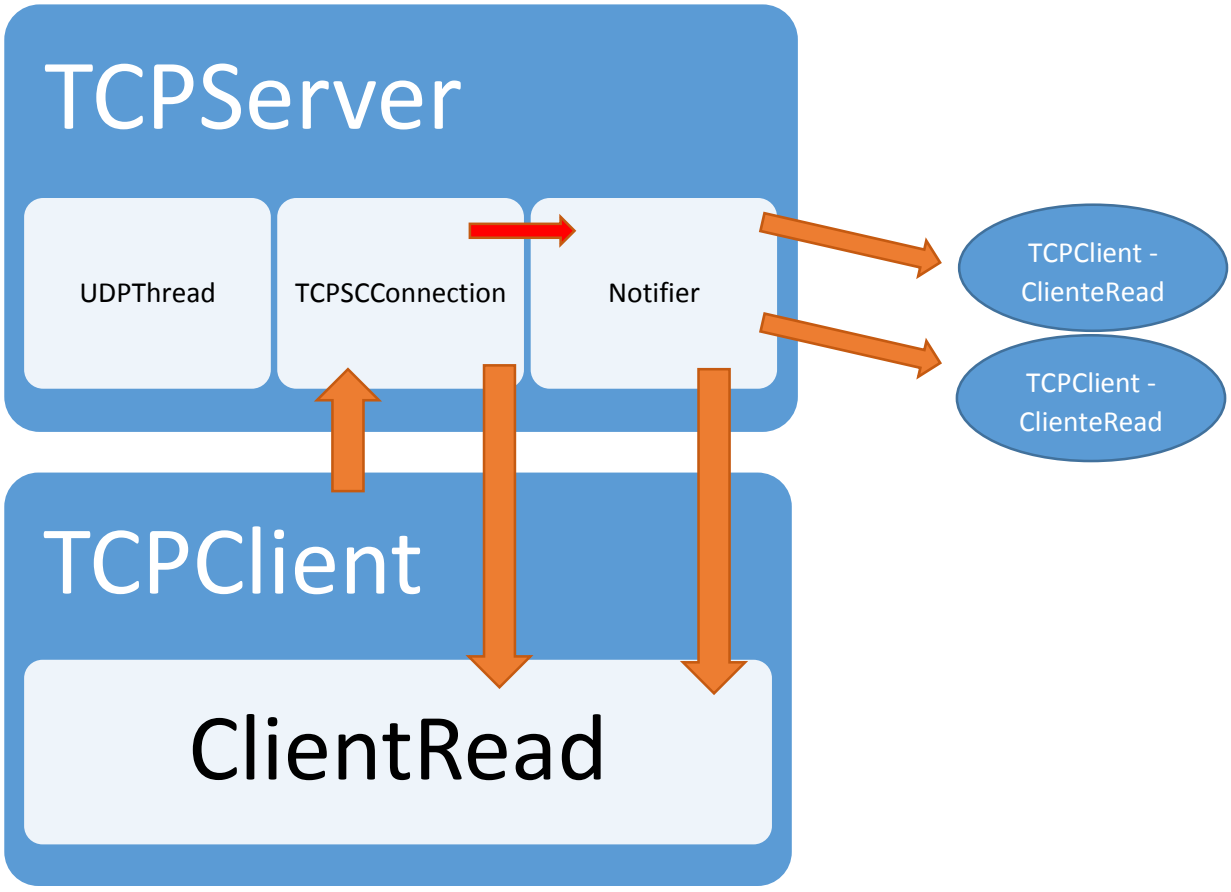
Ao iniciar o servidor TCP é aberto um *Stream* para comunicação UDP, uma vez que antes de qualquer outra coisa a aplicação verifica se já existe algum servidor com as mesmas funções do que o que pretendemos iniciar. Caso o programa obtenha alguma resposta ao pedido UDP significa que já existe um servidor primário em execução e, neste caso, o programa continua a enviar pedidos UDP até que deixe de receber resposta. Se deixar de receber resposta significa que o servidor primário deixou de funcionar e nesse momento vou assumir o papel concreto de servidor (1). Por outro lado, caso não seja obtida uma resposta ao pedido UDP, parte-se do princípio que não existe nenhum servidor ligado e avançamos para a inicialização do servidor (1).

(1) A inicialização do servidor passa por criar uma *Thread* de execução paralela destinada exclusivamente a responder a pedidos UDP (Solução Fail-Over). Após isto, é feita a ligação ao servidor RMI e só depois se inicia o *Socket* TCP e se põe o programa à escuta, ou seja, à espera de novas conexões por parte de clientes.

Quando se inicia um cliente é estabelecida conexão com o servidor TCP primário. O servidor TCP cria uma *Thread* para responder a todos os pedidos do novo cliente. No cliente é criada um *Thred* para receber todas as respostas aos pedidos efetuados ao servidor bem como notificações e todas as informações que o utilizador pedir para visualizar.

Modelo de Dados da aplicação

Modelo Multi-*Thread*:



TCPServer – cria uma *Thread* UDP, estabelece a ligação com o servidor RMI e fica à escuta de ligações de novos clientes.

UDPThread - esta *Thread* encarrega-se de responder a pedidos UDP para o caso de se ligar um novo servidor.

TCPSCConnection – é iniciada uma nova *Thread* deste tipo por cada novo cliente ligado para gerir a comunicação com este. Estabelece um *Socket* com o novo cliente e recebe e envia informação pelos *Streams* desse *Socket.*

Notifier – Quando se liga um novo cliente é colocado um objecto *Client* na *Thread Nofifier* que contém o seu *User Name* e o *OutputStream* correspondente ao *Socket* desse cliente para que seja possível enviar-lhe notificações.

TCPClient – estabelece a comunicação com o servidor e encarrega-se de enviar os pedidos do utilizador. Fica à espera de resposta a pedidos bloqueando até que a *Thread* ClientRead a notifique de que já recebeu resposta e pode avançar.

ClientRead – Esta *thread* trata de receber e mostrar ao utilizador as informações que recebe do servidor sejam elas respostas a pedidos do cliente ou notificações.

A troca de pedidos/respostas é feita com recurso a um tipo específico de objeto (Data) que leva a ordem que o servidor deve interpretar e toda a informação que o servidor precisa para tratar daquele pedido. A resposta funciona da mesma forma. A *Thread* de leitura no cliente interpreta o tipo de resposta que deve interpretar consoante o código que recebe.

Pedidos Cliente – Servidor (TCP)			**Resposta Servidor - Cliente**	
-2	- Pedido de logout		-2	Mensagem para desligar
-1	- Mudar user	|nome |		
0	- Registo de novo utilizador	|nome |password|	1/-1	Sucesso/Insucesso
1	- Login de utilizador	|nome | password|	1/-1	Sucesso/Insucesso
2	- Carteira		-3	Mensagem com as informações da carteira
3	- Histórico de Transações		-3	Mensagem com a informação das transacções
4	- Criar Novo Tópico	|Nome do Tópico|	1/-1	Sucesso/Insucesso – o tópico já existe
5	- Criar Nova Ideia	|ID(s) do(s) tópico(s) com que se relaciona | ID da ideia com que se relaciona | Nome da Ideia | Descrição da ideia | Número de ações em que se divide a ideia | Preço por ação| ações que devem ser colocadas à venda | Relação com a ideia principal	1	Sucesso

6	- Eliminar Ideia	|ID da ideia|	1/-1	Sucesso/Insucesso – Não pode eliminar esta ideia
7	- Consultar Tópicos		-3	Mensagem com os tópicos
8	- Consultar Ideias	|ID do tópico|	-3	Mensagem com as ideias
9	- Consultar Sub-Ideias	|ID da ideia|	-3	Mensagem com as sub-ideias
10	- Consultar informações de uma Ideia	|ID da ideia|	-3	Mensagem com as informações de uma ideia
11	- Mudar preço de venda de ações	|ID do pack | novo preço|	1	Sucesso
12	- Comprar ações	|ID do pack | ID da ideia | Número de Ações |	1/-1	Sucesso/Insucesso – saldo insuficiente ou não há ações disponíveis
13	- Tornar ações disponíveis para venda	|ID do pack | Número de Ações |	1/-1	Sucesso/Insucesso – número de ações inválido
14	- Tornar ações indisponíveis para venda	|ID do pack | Número de Ações |	1/-1	Sucesso/Insucesso – número de ações inválido
15	- Listar minhas ideias		-3	Mensagem com a informação das ideias
16	- Ver packs de ações	|ID da ideia |	-3	Mensagem com informação dos packs
17	- Transitional trading	|ID da ideia | Número de ações | Preço de compra|	1	Sucesso

Especificação dos Protocolos de Comunicação (TCP, UDP, RMI)

UDP - Protocolo usado apenas na comunicação entre servidores para reconhecer a existência de um servidor primário já em funcionamento.

TCP - Protocolo usado na ligação entre o cliente e o servidor. O servidor fica à escuta no seu ServerSocket TCP pela ligação de um novo cliente, gerando automaticamente uma nova thread para gerir a comunicação servidor-cliente, voltando de novo ao seu estado de escuta.

RMI – Este protocolo é usado para a comunicação entre o servidor a correr RMI e o servidor TCP que executa código semelhante a um cliente RMI. É usado para enviar ao servidor os pedidos processados e para que este retorne os resultados que, por sua vez, serão enviados ao cliente. O cliente RMI invoca métodos através da interface que contém os métodos implementados no servidor.

Arquitetura da Transmissão de Ficheiros

Para que um ficheiro possa ser enviado para o servidor é carregado no programa cliente. O utilizador deve introduzir o caminho para o ficheiro que pretende anexar à ideia. O ficheiro é carregado, convertido para um *byte array* e enviado pelo *Output Stream* como objeto. Já do lado do servidor lê-se o *Input Stream* e guarda-se o array de *bytes* correspondente ao ficheiro e por último grava-se o ficheiro em modo binário (*.bin*) numa pasta do servidor.

Tratamento de Exceções

RMI Exceptions:

Todo o ambiente parte do principio da boa funcionalidade do Servidor RMI e de todas as ligações feitas a este. Se acontecer alguma exceção nos pedidos ao Servidor RMI é da total responsabilidade da ligação fisica à rede, cujo bom funcionamento não é da nossa responsabilidade.

TCP Exceptions:

A principal, e mais importante, exceção encontra-se no ficheiro "ClientRead.java" (linha 50). Quando fazemos "catch (IOException e)" ao "answer = (Data) input.readObject();" estamos a certificar-nos não só que estamos a receber dados do Servidor TCP mas também que este ainda se encontra ligado! Quando a exceção ocorrer, procedemos ao chamamento do método "client.connect(true);", em que a flag "true" significa que é uma Reconnecção (ao contrario de quando o cliente se liga usando "connect(false);").

Esta é a exceção mais importante, pois está implementada numa thread com pouco trabalho e que acaba por estar constantemente a verificar se o Server TCP está ligado.

Como a thread está sempre a fazer esta verificação, mesmo que o utilizador esteja a meio de escrever uma descrição de uma ideia (por exemplo), a thread vai encarregar-se de redireccionar os sockets para o Servidor BackUp. (Sendo todo este processo 100% transparente para o utilizador que continua a trabalhar normalmente).

Outra exceção de referenciar é quando fazemos "catch (IOException e)" ao "oos.writeObject(data);" no ficheiro "Utils.java" (linha 35). Esta exceção encarrega-se simplesmente de mandar o objecto Data para o Servidor TCP. Atenção que um trabalho tão simples não faria sentido sem o trabalho da exceção referenciada acima! Como a thread do "ClientRead" se encarrega de se ligar ao Servidor BackUp, este método apenas tenta o envio do pacote até ter sucesso.

A próxima exceção encontra-se no ficheiro “TCPServer.java” (linha 40) e corresponde à linha “catch (SocketTimeoutException e)” ao fazermos “aSocket.receive(ack);”. Esta exceção também tem um trabalho muito importante pois é ela que conta quantas vezes falharam os pacotes UDP (usados para saber se o servidor principal ainda está ligado).

Quando esta exceção ocorre 5 vezes, partimos do principio que o servidor principal está “down” e a partir daí o servidor BackUp recebe o papel principal, dando continuidade ao bom estado do serviço.

No ficheiro “TCPClient.java”, uma das exceções mais importantes é quando fazemos “catch (SocketException e)” para caso a ligação ao socket dar erro, isto é, o servidor correspondente ao IP que estamos a tentar ligar está desligado. Caso isto aconteça (o servidor esteja desligado e o programa vá para o catch) vamos tentar ligar ao servidor de BackUp. Este procedimento é repetido até haver uma ligação com sucesso ou então até falhar 6 vezes (3 em ambos os servidores).

A última exceção relevante é no ficheiro “TCPClient.java” (linha 712) quando fazemos “catch (FileNotFoundException e)” que serve simplesmente para ter a certeza que o ficheiro que o utilizador anexou existe. Caso esta verificação não existisse, e como o servidor TCP ficaria à espera de receber um ficheiro, podiam aparecer problemas, tanto para o cliente como para o Servidor.

Solução Fail-Over

A implementação desta solução passou por, na inicialização do servidor TCP, enviar 5 pacotes por UDP com um *timeout* entre eles. Caso não seja obtida nenhuma resposta, o servidor inicia as suas funções uma vez que assume que não existe nenhum servidor activo e preparado para responder a pedidos.

Ao inicializar funções, o programa do servidor TCP cria uma *Thread* (UDPThread) que se encarregará de responder a pedidos UDP com o objetivo de, caso algum novo servidor se tente ligar, receber a informação de que já existe um servidor a responder a pedidos. Esta *Thread* fica activa durante todo o tempo de execução do programa. Caso o programa pare, esta *Thread* não enviará resposta a eventuais pedidos e um novo servidor poderá inicializar funções, repetindo todo o preocesso.

Do lado do cliente, caso haja uma falha no servidor ele tentará reconectar-se àquele com quem estava a comunicar. Caso esgote as tentativas de ligação e não obtenha resposta o programa do cliente vai tentar ligar-se ao servidor secundário. Se obtiver resposta, é aberto um novo Socket e criada uma *Thread* para gerir os pedidos do cliente no servidor e de imediato é enviado o *username* do utilizador que estava ligado ao servidor primário para que não seja necessário repetir o processo de login e, deste modo, o utilizador não se apercebe de qualquer falha.

Caso não exista servidor secundário e o primário falhe, o programa cliente tentará a ligação de novo ao primário e, caso não consiga, vai verificar que não consegue ligar-se também ao secundário e receberá a mensagem de que o sistema está em baixo.

Manual de Configuração e Instalação

Para correr a aplicação é necessário executar três programas, o RMIServer, o TCPServer e o TCPClient.

RMIServer: para correr este programa é necessário ter o rmiregistry em execução e ainda indicar o caminho para o *driver* para que seja possível a comunicação com a base de dados. Caso não esteja a executar o rmiregistry utilizar o comando:

rmiregistry & java –cp "./driver.jar:." RMIServer

Nota: não colocar porta após rmiregistry uma vez que o programa está configurado para se ligar à porta default deste serviço.

Depois disto deve ser posto em execução o TCPServer. Para isso, devemos saber o IP da máquina onde vamos correr o *backup server*, e o IP da máquina onde estamos a executar o RMIServer. Sabendo isto basta usar o comando:

java TCPServer [backupServerHostname] [RMIServerHostame]

Agora sim está tudo pronto para executar o programa do cliente e interagir com a aplicação. Neste caso precisamos de saber o endereço IP das máquinas onde correm o servidor principal e o secundário.

java TCPClient [primaryServHost] [primaryServPort] [secondaryServHost] [secondaryServerPort]

Efetuar registo ou iniciar seção com as credenciais: Username: admin e Password: admin.

Com o programa a correr seguir as orientações no programa cliente para executar qualquer função.

Descrição de Testes Feitos à Aplicação

Teste feito	Resultado
Registo (com username livre)	Sucesso e volta para a entrada
Registo (com username em uso)	Erro e volta para a entrada
Login (com user e pass correctos)	Sucesso e entra no menu principal
Login (com user ou pass incorrectos)	Erro e volta para a entrada
Carteira	Mostra o saldo e os packs de acções
Alterar o preço de acções	Pergunta qual o ID do Pack e o novo preço
Tornar acções disponiveis para venda	Pergunta qual o ID do Pack e o numero de acções
Tornar acções indisponiveis para venda	Pergunta qual o ID do Pack e o numero de acções
Histórico de Transações	Mostra todas as transações completas do user
Criar um tópico (com titulo livre)	Sucesso e volta para o menu principal
Criar um tópico (com titulo ja usado)	Erro e volta para o menu principal
Criar idea	Pede todos os parametros necessários à criação de uma ideia e no final pergunta se o user quer anexar um ficheiro à sua ideia.
Eliminar ideia	É recebida uma lista de todas as ideias em que o user é dono de 100% das acções dessa ideia.
Eliminar X em que é dono de 100% das acções	Sucesso e volta para o menu principal
Eliminar X em que não é dono de 100%	Erro e volta para o menu principal
Navegar	É mostrada uma lista de tópicos.
Escolher o ID de um tópico	São mostradas as ideias ligadas a esse tópico
Escolher o ID de uma ideia	São mostrados os promenores dessa ideia e todos os packs de acções relacionados com essa ideia.
Comprar acções desta ideia	Depois de verificado o numero de acções e o saldo do utilizador, é efectuada a compra.
Responder a esta ideia	É criada uma ideia com relação com a actual. É ainda possivel dizer se a nova ideia está contra, a favor ou neutra em relação à ideia principal.
Ver subideias	É retornada uma lista de respostas submetidas por outros utilizadores à ideia actual.
Agendar uma compra mais barata	É pedido ao utilizador o numero de acções que ele pertende comprar e o preço que ele está disposto a pagar. A compra é feita quando os requesitos do comprador forem satisfeitos.