Universidade de Coimbra

• U  C •

Universidade de Coimbra

Sistemas Distribuídos

Relatório de Projeto
iBei: Leilões Invertidos
Meta 1

Alexandre Ferreira Costa
2014206463 - afc@student.dei.uc.pt

Ana Inês Mesquita Fidalgo
2013134819 - aimf@student.dei.uc.pt

Pedro Filipe Matos Godinho Gabriel Coelho
2009116949 - pfcoelho@student.dei.uc.pt

29 de Outubro de 2016

Conteúdo

Capítulo 1

Introdução

Este projecto foi realizado no âmbito da cadeira Sistemas Distribuídos (SD) inserida no plano de estudos da Licenciatura em Engenharia Informática da Universidade de Coimbra, lecionada pelos Professores Doutores Raul André Brajczewski Barbosa e Alcides Miguel Cachulo Aguiar Fonseca, no ano lectivo de 2016/2017.

O projecto acima referido tem como objetivo mostrar a aplicabilidade de um Sistema *Remote Method Invocation*, ou RMI, de modo a criar uma camada de persistência de dados, numa arquitectura Cliente-Servidor. A comunicação entre Clientes e Servidores e feita através de uma aplicação de sockets *TCP/IP* e segue-se ainda um modelo *multithread* para a estruturação de Servidores.

1.1 Contexto

No propósito do trabalho é-nos pedido para implementarmos um sistema de Leilão Invertido, onde um vendedor declara qual o preço máximo que está disposto a pagar por um artigo e os outros utilizadores licitam o preço mais baixo que aceitariam receber por esse mesmo artigo.

A par deste método de Leilão é-nos pedido para implementarmos várias funcionalidades que permitam a construção desta plataforma como a criação dos servidores, dos utilizadores, dos leilões, das licitações e as funcionalidades de administradores - dado que somos um grupo constituído por 3 pessoas - e todas as eventuais funções necessárias para o decorrer do trabalho.

Capítulo 2

Arquitectura de Software

O nosso projecto está dividido em três packages: RMI, TCPserver e TCP-Client. Sempre que é feito um pedido pelo TCPClient, o TCPserver recebe o pedido e é criada uma nova instância de uma classe auxiliar, à qual se vai fazer lookup ao rmi e chamar as classes genéricas que estão guardadas no RMIServer. No caso do projecto, quando existem três servidores TCP e um cliente java, disponibilizado pelo professor, este cliente liga-se a um dos servidores, sendo este adicionado a um array de sockets do servidor. Os três servidores vão estar a comunicar entre eles por udp enviando a sua lotação de clientes uns aos outros. Por sua vez o cliente irá estar a receber esta lotação de 60 em 60 segundos. Caso aconteça alguma avaria no RMIServer, os servidores TCP iram-se-ão ligar a um servidor RMIServer secundário, ficando esse o primário. Caso o RMIServer primário original supere a avaria, este passa a assumir o papel de RMIServer secundário, permitindo que possa haver outra vez uma ligação, caso haja uma falha no servidor principal. Os clientes, neste processo, não detectam esta avaria.

2.1 TCPserver

Quando o TCPserver se inicia irá adicionar os clientes à escuta de um determinado porto numa lista de sockets - como mencionado acima - onde estão todos os clientes ligados a ele. Sempre que é adicionado um socket cliente e criada a ligação é inicilizada uma thread Menu num *while(true)* de modo a estabelecer esta conexão cliente-servidor para se poder enviar pedidos e receber respostas.

2.2 TCPClient

Classe fornecida pelo professor. Esta classe cria um socket para um determinado porto e contem uma thread que permitirá ao cliente realizar pedidos e obter respostas do TCPserver a que se ligou.

2.3 RMIServer

Sempre que o RMIServer é iniciado vai verificar a informação contida nos ficheiros para preencher os arraylists de leilões, users e admins. Para além disto, a partir de uma thread, é verificado contantemente o término dos leilões para mantendo os ficheiros actualizados.

2.3.1 Login

Quando o TCPserver recebe um pedido de login do cliente, é criada uma resposta a partir da classe Login que irá ligar o servidor TCP ao RMI e chamar a função login, presente no servidor RMI, para obter uma String[] onde estarão colocados os parâmetros de resposta necessários para escrever o comando que o TCPClient vai receber. Na função login da classe RMIserver procuramos um user no ArrayList users do RMI que contenha o mesmo username que o desejado e por fim da-nos uma resposta de true ou false caso encontre um user válido ou não. Este User que fez login é adicionado a um ArrayList de Users ligados até acabar a ligação.

2.3.2 Registar User normal e Admin

Sempre que é recebido um comando do tipo registar na linha de comandos do TCPserver, tal como na classe de Login, é criada uma instância da classe Registar, onde é feito lookup ao RMI e é gerada uma lista de strings de resposta para os comandos a partir da função Registar ou RegistarAdmin, ambos presentes no RMIServer. Quando são registados, estes são adicionados ao ArrayList de Users e/ou Admin respectivamente, e no final os ficheiros são actualizados.

2.3.3 Criar Leilao

Quando é recebido este pedido no TCPserver este irá criar uma thread, Menu2 ou MenuAdmin, na classe Connection. No Menu é realizado um split da string do pedido para para obtermos os dados necessários à criação da resposta. O processo é o mesmo das classes de login e registo, temos uma classe CriarLeilao onde é realizado o look up ao RMI para obtermos uma resposta ao pedido. Depois disto é enviada uma resposta ao cliente no servidor TCP. Quando é criado um leilão este é adicionado ao ArrayList global de leilões do RMI e são actualizados os ficheiros.

2.3.4 Procurar Leilão

A procura de um leilão é feita através do seu código. Este código não é único, sendo apenas o código do artigo a leiloar, podendo existir vários leilões com o mesmo código. É criada uma nova instância da Classe SearchLeilao no Menu (thread) e chamada a função SearchLeilao do RMI, onde vamos procurar no ArrayList leiloes um leilão com o id que foi inserido. É criada ainda uma lista de strings onde estão os outputs a aparecer no comando de resposta e, de seguida, é enviada a resposta ao TCPClient a partir do TCPserver, sendo esta impressa na consola do cliente.

2.3.5 Detalhe de Leilão

Esta procura é semelhante à anterior em termos de implementação mas é uma procura por id, o atributo único do leilão. Além disto, devolve ao cliente uma resposta mais detalhada deste.

2.3.6 Listar as Licitações de um User

O RMI contem uma função chamada ListarLicitacoesUser() que irá percorrer o ArrayList de Leilões e o ArrayList de licitações de cada leilão. Esta procura é feita com o username do cliente em questão, sendo este o seu atributo único. O processo é sempre semelhante, criação de uma instância de uma classe que extende pedido, recebe o pedido do cliente e liga-se ao RMI para chamar a função que irá directamente buscar a informação aos ficheiros, e enviar a resposta na thread menu ao cliente.

2.3.7 Editar Leilão

O cliente faz o pedido do que vai ser editado e qual o leilão a ser editado. No menu (TCPServer) é feita a gestão do pedido para obter os parâmetros necessários, é criada uma resposta a partir da classe EditarLeilao, feito o lookup ao RMI e vai buscar a resposta à função correspondente no RMIServer. Nesta função primeiramente é guardado uma cópida do leilão a editar, para posterior visualização. No final são actualizados os ArrayLists e de seguida os ficheiros.

2.3.8 Listar Versões Anteriores de Leilão

Um leilão contem um ArrayList de versões anteriores de si próprio. Sempre que este é alterado é guardada uma cópia neste ArrayList. O cliente a partir do comando e do id de um determinado leilão consegue receber uma lista de todas as versões antigas de determinado leilão.

2.3.9 Escrever Mensagem em Leilao

Um leilão contem um ArrayList de mensagens de Mural. Para escrever uma mensagem num leilão, tem de se usar o comando message e dar um id. Este pedido é tratado com splits quando é recebido no servidor TCP e é criada uma resposta a partir da classe EscreverMural. Esta classe faz o lookup ao RMI e obtem uma resposta obtida a partir da função EscreverMensagem do RMI. Quando é enviada a mensagem, esta é armazenada num ArrayList do tipo Mensagem e caso o dono ou os licitadores desse leilão não estejam online na altura do envio esta é armazenada num ArrayList de mensagens pendentes. Quando estes utilizadores fazem login, recebem uma notificação de mensagem e esta é eleminada do ArrayList de mensagens pendentes.

2.3.10 Ver Quem está Online

Para verificar quem está online, sempre que é estabelecida a conexão entre um cliente e um servidor, este cliente é adicionado a um ArrayList do tipo utilizadores, que estão online. Quando esta conexão é terminada, ou seja, quando o socket cliente é removido da lista de sockets de cada servidor TCP, dá-se entrada num catch onde se remove o utilizador da lista de utilizadores online.

2.3.11 Listar os Meus Leilões

Quando é realizado este pedido, cria-se a classe genérica de listar os leilões por procura de utilizador. É realizado mais uma vez o lookup, e é criada uma lista de Strings que contem os parâmetros necessários a formar o comando de resposta.

2.3.12 Licitar num Leilão

Quando é feito um pedido para licitar um leilão, este só é feito se o valor introduzido for inferior ao valor base introduzido pelo dono do leilão ou ao da licitação melhor. Caso o valor introduzido seja superior, é retornado um comando com o parâmetro "ok"a false.

Caso seja realizada a licitação, é enviada uma notificação tanto ao dono como aos outros licitadores, a informar que determinado utilizador fez uma melhor licitação. Esta notificação fica no ArrayList de mensagens pendentes de cada utilizador e, quando estes utilizadores realizam o login, recebem a notificação de licitação, removendoa mensagem do ArrayList. São actualizados os ficheiros em tempo real.

2.3.13 Cancelar Leilão

Quando um leilão é cancelado, este é removido do ArraList de leilões, sendo que este leilão nunca obtem um vencedor.

2.3.14 Banir Utilizador

Função específica do Administrador; O cliente admin realiza um pedido para banir um utilizador e é criada uma resposta a partir da classe BanirUser. Este utilizador é identificado no arraylist de utilizadores e é removido, assim como todos os seus leilões e todas as suas licitações. É enviada uma resposta ao cliente admin de "ok: true"caso tenha encontrado o utilizador que este desejava banir ou "ok: false"caso este não tenha sido encontrado. No final são actualizados os ficheiros em tempo real.

2.3.15 Listar Estatisticas

Existem três pedidos diferentes de estatísticas diferentes, top 10 utilizadores que já ganharam leilões, top 10 criadores de leilões e os leilões criados nos

últimos 10 dias. No top 10 utilizadores que já ganharam, é realizada uma pesquiza nos ArrayLists de Leilões acabados e ganhos em todos os utilizadores presentes no ArrayList de utilizadores. Caso os ficheiros contenham utilizadores com zero leilões ganhos e/ou contenham menos de dez utilizadores, estes são também considerados no top 10.
No top 10 de criadores de leilões, é feita uma pesquisa no ArrayList de leilões geral onde se verificam quais os que criaram mais leilões.
A função de dizer o número de leilões criados nos últimos dez dias, vai verificar a diferença de dias entre a data de criação (guardada aquando da criação do leilão) e a data actual, incrementando um contador, é no final então retornada um comando de resposta ao utilizador com o tipo de pedido e o número de leilões criados.

Capítulo 3

Funcionamento do Servidor TCP

Neste projeto é pedido para executar três servidores TCP's que vão comunicar entre si através do protocolo UDP, para terem a informação da lotação de cada uma. Por sua vez vão também comunicar com o cliente através do protocolo TCP. Para isso foi necessário criar várias classes:

- *TCPServer*;
- *Listenner*;
- *RecebeUDP*;
- *EnviaUDP*;
- *Connection*;
- *EnviaCliente*;
- *Menu*;
- *Menu2*;
- *MenuAdmin*;
- *Data*;

3.1 Detalhe das classes

3.1.1 TCPServer

Nesta classe está presente a main, ou seja, para executar o server é este ficheiro que tem de ser corrido. Aqui encontra-se toda a informação relativa a ip's das maquínas, ports TCP dos server's e ports UDP ao qual os Servers se ligam para comunicar entre si.

3.1.2 Listenner

Nesta classe temos acesso a todas as portasTCP e portasUDP e vamos criar duas variáveis, *ServerSocket e lsocket* que vão conter as ligações do Servidor num determinado Port. Essas ligações, por sua vez, encontram-se no array de portasTCP e *DatagramSocket aSocket* que vai possuir a ligação UDP num determinado port para os servidores poderem comunicar uns com ou outros e ainda vamos criar um *Arralist* de clientes que esse servidor possui.

Ao vir para esta classe é criado uma Thread que vai fazer as ligações todas dos servidores TCP e as respetivas ligações UDP. Depois é executado,o*new RecebeUDP(aSocket, clientesligados,portasUDP)* que vai receber mensagens de outros serviores e*new EnviaUDP(udpport, portasUDP, clientesligados)* que vai enviar mensagens para os servidores ligados.

Ao mesmo tempo que isso executa é criado uma nova Thread que está sempre à espera que novos clientes se liguem aos servidores.

Quando ocorre essa ligação é criado um *new Connection(ssocket,clientesligados)*, ja no fora dessa Thread ainda é criado um
new EnviaCliente (recebe.getLotacao(),clientesligados).Envialotacao() que envia a lotação dos servidores a todos os clientes.

3.1.3 RecebeUDP

Esta classe tem uma função única que é receber as lotações de outros servidores e colocar essas lotações num Array. Para que isto receba todas as mensagens da lotações de outros servidores é criado uma Thread nova que recebe as mensagens destes e coloca-as no array.

3.1.4 EnviaUDP

Esta classe tem uma única função que é enviar a lotação do servidor a outros servidores ligados. Para isso quando inicia é criada uma Thread nova que envia mensagem da lotação a outros servidores a cada 5 segundos.

3.1.5 Connection

Esta classe recebe a mensagem do cliente e transforma-o num pedido. Consoante o pedido ele vai para menus diferentes: se o pedido for login ou register vai para a classse *Menu* depois dependendo se for cliente normal ou admin vai para a classe *Menu* e *MenuAdmin* respectivamente.

Para isto é necessário criar uma thread que está sempre à espera de mensagem do cliente para lhe enviar mensagem de volta.

3.1.6 EnviaCliente

Esta classe envia para todos os clientes daquele servidor a lotação de todos os servidores de 60 em 60 segundos. Isto é feito através da criação de uma Thread que envie essa mensagem.

3.1.7 Menu

Esta classe vai receber o pedido do cliente.

Se for login ou register ele vai criar uma Thread para tratar o pedido. Se for login vai comunicar com o RMIServer atráves de *resposta = (Response) new Login("login",username2,password2).execute(rmiserver)*, se for register comunica através de
resposta = (Response) new Registar("register",username2,password2).execute(rmiserver).
Ao fazer isto vai receber uma resposta do RMI que comunica com o server que por sua vez envia ao cliente essa resposta.

3.1.8 Menu2

Este é o menu que trata todos os pedidos de um cliente normal, tais como: Create an auction, Search an auction, Detail auction, List all my bids, Edit my auctions, List older versions of edited auction, Write message on Auction, See who's online, List all my created auctions e Make a bid on an action.

Ao receber um pedido vai tratá-lo com trata a classe do Menu.

3.1.9 MenuAdmin

Este é o menu que tratade todos os pedidos de um administrador, tais como: Create an auction, Search an auction, Detail auction, List all my bids, Edit my auctions, List older versions of edited auction, Write message on Auction, See who's online, List all my created auctions, Make a bid on an action, Cancel an auction, Ban an user e See statistics.

Capítulo 4

Funcionamento do Servidor RMI

No propósito do trabalho é-nos pedido a implementação de um servidor *Remote Method Invocation* (RMI), de modo a criar uma camada de persistência de dados. Um RMI é um método de programação orientada a objectos onde os dados objectos conseguem interagir entre diferentes computadores numa determinada rede distribuida.

4.1 Detalhe das Classes

4.1.1 RMI/RMIServer

A implementação do RMI passa pela criação de uma classe *RMIServer.java* - com o respectivo *header* de interface *RMI.java* - onde são implementadas todas as funções de ligação directa dos pedidos efectuados pelo utilizador, com a informação - Users, Leilões, Licitações - necessária à resposta dos seu pedido, informação essa guardada temporariamente em ArrayLists. Dentro desta classe encontram-se então as funções de Login/Logout e listagem de clientes e administradores, acesso e escrita de ficheiros, criação, edição, procura, cancelamento e término de leilões, escrita de mensagens no mural do leilão, licitação sobre os leilões e ainda funções de administrador, como recolha de estatísticas.

Na função main desta classe encontra-se um *try/catch* para criar a ligação

a um *RMIServer* (tenta ligar-se a um e, caso não consiga, tenta a outro) - e a leitura dos ficheiros para carregar a informação necessária para o decorrer do projecto.

4.1.2 Connection

O ficheiro *Connection.java* faz a ligação entre o pedido executado pelo utilizador e o menu de execução. Mediante o pedido, a informação é retornada, através da classe *Pedido.java* e envia-se para os ficheiros de *Menu*.java*.

4.1.3 Menus/Pedido/Outras

As funções *Menu.java, Menu2.java e MenuAdmin.java* recebem toda a informação necessária para a resposta a um pedido feito pelo utilizador de modo a estruturar a resposta. Através das funções do ficheiro *Pedido.java* os scripts Menu recebem a ordem executada pelo utilizador e, através das suas classes filho, recebem as informações de resposta ao pedido.

Estas classes-filho de *Pedido* são todas as classes com o mesmo nome que as funções chamadas em *RMIServer.java* em que se faz o Registry e o lookup e envia para as classes *Menu*.java* a informação sob a forma de *Array de strings*. Exemplos dessas classes-filho são *Registar.java, RegistarAdmin.java, TopTenLeiloesGanhos.java,* etc.

Por último, nas classes *Menu*.java*, mediante a ordem dada - primeira casa do *Array* recebido - a ordem é executada, chamando a função correspondente.

4.2 Fail-Over

Neste projecto é-nos ainda pedido para protegermos o programa no caso de haver uma falha de servidor. A protecção pedida é um acesso a um outro servidor secundário que, caso seja acedido, torna-se servidor principal, fazendo com que o original se torne secundário, assim que se encontre disponível outra vez.

O acesso ao RMI tem uma condição *try/catch* para garantir - tal como é o seu papel - que caso a ligação corra mal haja algum tipo de resposta dentro do *catch*.

Para garantir que existe essa protecção decidimos, no *catch*, implementar

um ciclo *while(check ==0)* onde a variável *check* é uma flag de sucesso e, dentro desse ciclo voltamos a fazer um *try/catch* de ligação ao RMI. Caso o *try/catch* de origem não tenha sucesso o programa volta a tentar aceder ao RMI usando o mesmo port.

Capítulo 5

Distribuição de Tarefas

Tratamento de Exceções e Balanceamento de Carga	
Avaria de um servidor RMI não tem qualquer efeito nos clientes	Inês Fidalgo
Não se perde/duplica licitações se os servidores RMI falharem	Alexandre Costa e Inês Fidalgo
Avarias temporárias no RMI são invisíveis para clientes	Alexandre Costa
Servidores TCP trocam via UDP o número de clientes ligados	Alexandre Costa
Clientes são informados a cada 60s da carga dos servidores TCP	Alexandre Costa

Failover	
O servidor RMI secundário testa periodicamente o primário	Alexandre Costa
Em caso de avaria longa os servidores TCP ligam ao secundário	Alexandre Costa
Servidor RMI secundário substitui o primário em caso de avaria	Inês Fidalgo
Os dados são os mesmos em ambos os servidores RMI	Alexandre Costa e Ines Fidalgo
O failover é invisível para utilizadores (não perdem o login)	Inês Fidalgo
O servidor original, quando recupera, torna-se secundário	Alexandre Costa

Requisitos Funcionais	
Classe TCPserver Interface RMI	Inês Fidalgo Inês Fidalgo e Pedro Coelho e Alexandre Costa
Gestão Pedidos e Respostas	Inês Fidalgo
Ligações TCP-RMI	Inês Fidalgo
Menus (threads)	Inês Fidalgo
Ficheiros Registar novo user	Inês Fidalgo Pedro Coelho e Inês Fidalgo
Login protegido com password	Inês Fidalgo
Criar leilão	Pedro Coelho e Inês Fidalgo
Pesquisar leilões por código EAN/ISBN	Inês Fidalgo
Ver todos os detalhes de um leilão (incl. histórico de licitações)	Inês Fidalgo
Listar todos os leilões em que o utilizador está/esteve ativo	Inês Fidalgo
Licitar um valor num leilão	Inês Fidalgo
Editar propriedades de um leilão, guardando versões anteriores	Inês Fidalgo e Pedro Coelho
Escrever mensagem no mural de um leilão	Inês Fidalgo
Utilizadores offline leem novas mensagens assim que se ligam	Inês Fidalgo e Pedro Coelho
Listar utilizadores online	Alexandre Costa e Inês Fidalgo
Notificação imediata de licitação melhor de outro utilizador	Inês Fidalgo
Leilão termina corretamente na data, hora e minuto marcados	Alexandre Costa e Inês Fidalgo
Cancelar um leilão	Inês Fidalgo
Banir utilizador, cancelando leilões e licitações	Pedro Coelho e Inês Fidalgo
Mostrar estatísticas de atividade	Pedro Coelho
Relatório	Pedro Coelho e Inês Fidalgo e Alexandre Costa

Capítulo 6

Testes

Pedido	Comando
Registar um administrador	type: admin, username: Alexandre, password: ola
Registar um cliente	type: register, username: ines, password: ola
Login normal user or admin	type: login, username: pedro, password: lll
Criar leilão	type: create_auction, code: 111, title: SDF, descritption: adoro o dei, deadline: 2017-01-01 00-01, amount: 10
Procurar leilão	type: search_auction, code: 111
Detalhes do leilão	type: detail_auction, id: 65416541
Ver as licitações de um user	type: my_lici, username: pedro
Editar acção	type: edit_auction, id: 65416541, deadline: 2017-01-02 00-01, title: ahah, description: oooh whats you saaay, amount: 13
Ver acção original	type: older_versions, id: 65416541
Inserir mensagem numa acçã	type: message, id: 65416541, text: format?
Listar users online	type: online_users
Listar acções	type: my_auctions
Licitar um leilão	type: bid, id: 65416541, amount: 5
Registar Admin	type: admin, username: costalindo, password: lindoe
cancelar leilão	type: cancel, id: 65416541
banir utilizador	type: ban_user, username: pedro
top 10 win	type: win10
last 10	type: last10
created 10	type: created10

Resposta
type: admin, ok: true
type: register, ok: true
type: login, ok: true
type: create_auction, ok: true
type: search_auction, items_count: 2, items_0_id: 84056363, items_0_code: 111, items_0_title: SDF, items_-1_id: 65416541, items_-1_code: 111, items_-1_title: SDF
type: detail_auction, title: SDF, description: adoro o dei, deadline: 2017-1-1 0-1, message_count: 0, bids_count: 0
type: my_bids, bids_count: 0
type: edit_auction, ok: true
code: 111, title: SDF, description: adoro o dei, date: 2017-1-1 0h-1m
type: message, ok: true
type: online_users, users_count: 1, users_0_username: pedro
type: my_auctions, items_count: 1, items_0_id: 84056363, items_0_code: 111, items_0_title: SDF
type: bid, ok: true
type: admin, ok: true
type: cancelar, ok: true
type: Banir, ok: true
type: statics_toptenwon, ok: bye, nov2, bye2, byebye, costalindo,
type: statics_last10, ok: 1
type: statics_toptencreated, ok: bye, dummy,