Universidade de Coimbra

Licenciatura Engenharia Informática
2014-2015

Sistemas Distribuídos - Projeto 1

Autores:
Bruno Martins - Nº2007183389

26 de Outubro de 2014

Conteúdo

1 Introdução

O presente relatório destina-se a apresentar a descrição do projecto 1 da cadeira de Sistemas Distribuídos intitulado *Meeto – Collaboration and Social Networking*.

Reuniões são necessárias quando diversas pessoas trabalham em conjunto. Alunos reunem-se para coordenar o seu esforço nos trabalhos práticos; trabalhadores reunem-se para discutir o estado dos projectos e gestores reunem-se para tomar decisões chave relacionadas com as suas organizações e por aí fora. É necessário um sistema que impeça as pessoas de gastarem tempo desnecessário em reuniões. Afinal de contas, tempo é o nosso recurso mais valioso.

Este projecto consiste então numa implementação de um sistema de gestão de reuniões, onde os utilizadores podem criar reuniões, convidar participantes, ter uma sala de chat para cada tópico da reunião e no final adicionar decisões chave a cada reunião. É também possível relacionar pessoas a certas tarefas - uma vez que uma reunião sem atribuição de tarefas concretas aos seus elementos de pouco serve.

2 Internal architecture

Esta aplicação encontra-se subdividida numa série de classes, cada uma responsável por integrar esta aplicação num todo coerente, onde se destacam:

RMIServer:

Dentro da classe **RMIServer** encontram-se os métodos essenciais da aplicação: registo, login, createMeeting, listAllMeetings, acceptMeeting, meetingOverview, addAgendaItem etc. É aqui que é criado o registry e onde é instanciado o objecto para ligação à base de dados MySQL. Dentro do RMIServer são também chamados vários métodos do TCPServer (callback), entre eles o **msgToMany** que trata de enviar uma mensagem a vários destinatários e o **msgsToOne** que envia diversas mensagens para o mesmo utilizador. Relativamente a threads, o RMI faz uma gestão interna automática das mesmas.

RmiInterface:

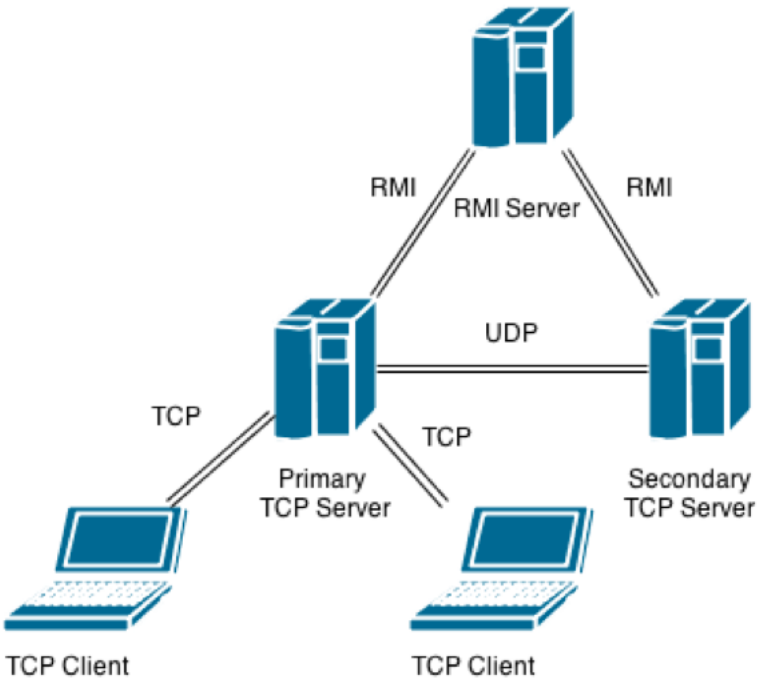

Figura 1: Visão Global da Arquitectura

O RmiInterface, tal como o próprio nome indica, é a interface dos métodos existentes dentro da RMIServer.

MySQL:

Classe onde se encontram os métodos para ligação à base de dados MySQL e para correcta execução dos diferentes tipos de queries.

TCPServer:

Interface dos métodos existentes dentro do TCPServerImpl

TCPServerImpl:

Implementa os métodos da interface e é aqui onde os clientes se ligam (abrindo dois sockets e mais duas threads para cada um). Possui 2 threads internas (UDP Sender e UDP Receiver) e guarda a relação entre users e threads numa HashTable. Correndo o ficheiro duas vezes simula a arquitectura desejada (tcp server primário e tcp server de backup).

TCPClient:

Este cliente estabelece a ligação com o servidor através de dois Sockets TCP (uma para pedidos request-reply e outro para eventos em "tempo real") promovendo não só a construção dos menus (menu de login, inicial, etc.) como também a arquitetura da aplicação melhorando, deste modo, a comunicação com o servidor. É responsável por providenciar informação e dados através da criação de mensagens, como também requisita serviços à aplicação, como por exemplo listar meetings existentes, ou comprar aceitar convites. Possui internamente uma thread para ler dados que surjam sem ser necessário uma interacção do utilizador com o sistema.

Estas classes são responsáveis por estabelecer grande parte da interoperabilidade desta aplicação embora sejam necessárias as restantes que suportam a estrutura por detrás deste sistema.

Relativamente ao armazenamento de dados na Base de Dados, foram criadas as tabelas:

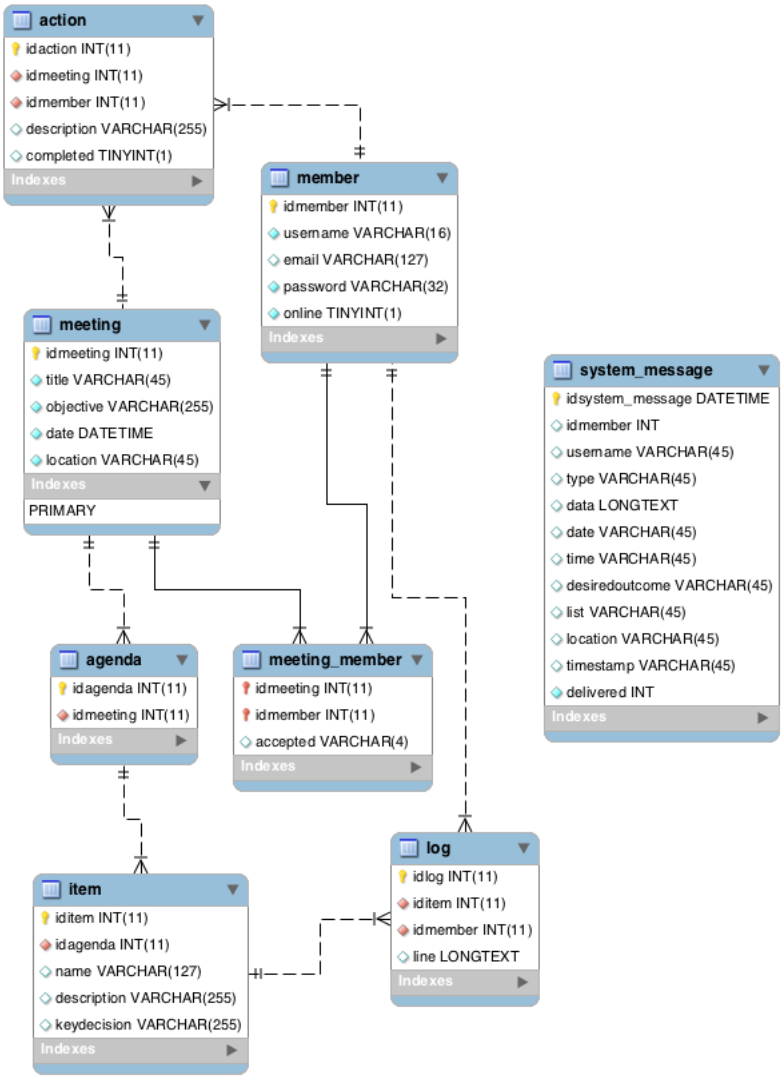

Figura 2: Diagrama Entidade Relacionamento

3 Data model

O seguinte diagrama representa o nosso esquema de Data Model.

Todo o tipo de requests, acções e triggers no projecto são acionados a partir do instanciamento de variáveis da classe "Message- são estes os objectos que circulam na rede. Possui diversos construtores que se podem usar consoante a acção a tomar. O tratamento correcto das mensagens é feito através da inspecção do parâmetro "type".

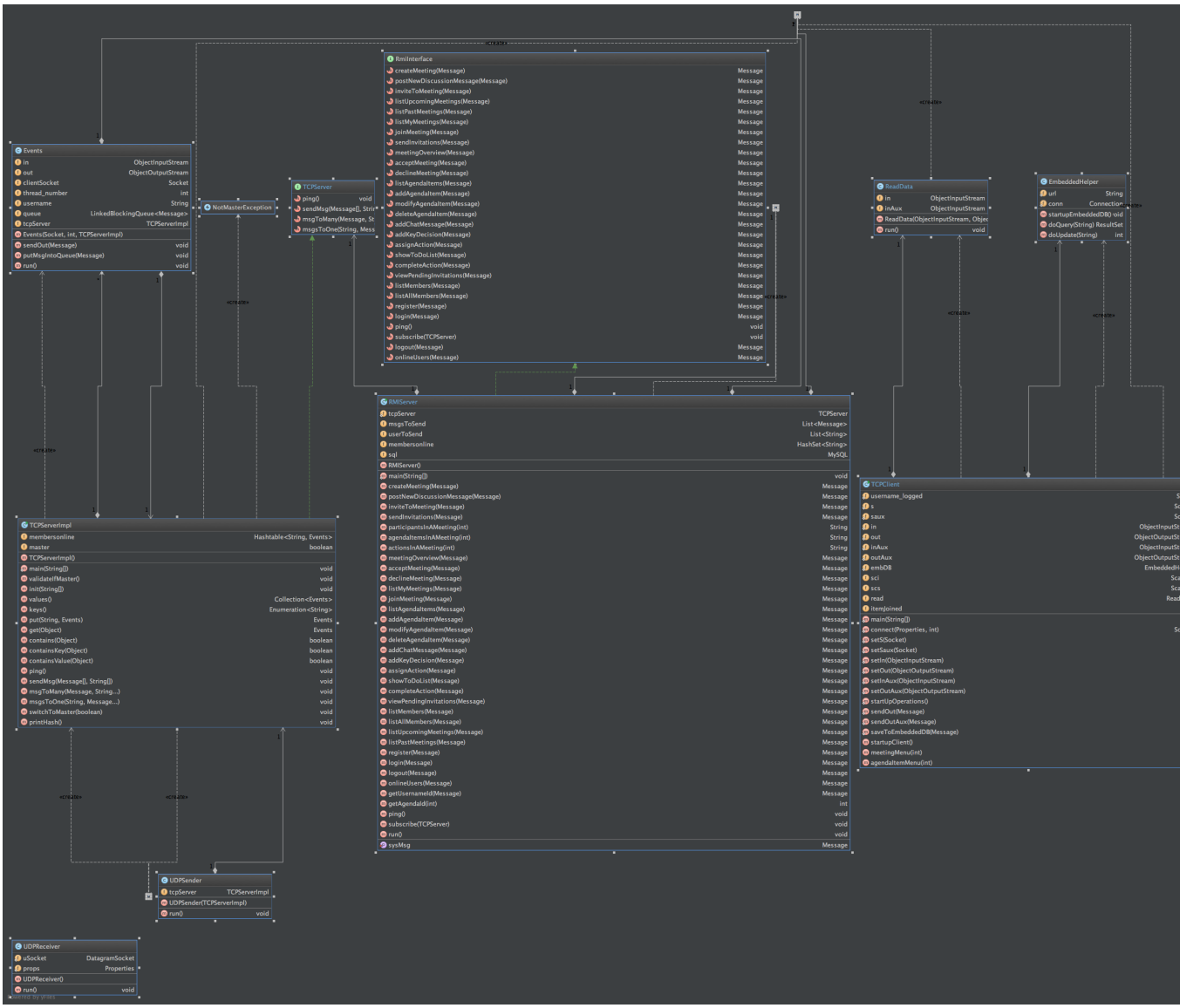

Figura 3: Data Diagram

4 Protocol Specification (TCP, UDP, RMI)

4.1 TCP

Ao serem iniciados, os servers TCP criam dois ServerSockets e ficam à espera de ligações de clientes. Assim que estes se ligam, são criadas duas threads que irão tratar dos pedidos dos e para os clientes: A thread Connection, que irá servir prin-

cipalmente para tipo de pedidos request reply (listagem de meetings, por exemplo), e a thread Events, que irá servir para enviar informação em "real-time"(thread para enviar mensagens que não são necessáriamente solicitadas pelos clientes). Estas threads façam o lookup do Registry do RMI e chamam os seus métodos consoante necessário.

Aqui também se encontra uma Hastable que relaciona os membros online com as suas threads de Events - para o servidor RMI fazer callback do método que envia as mensagens (e.g chat) aos diversos utilizadores e o TCP Server saber que thread corresponde a cada user.

É aqui também que é realizada a verificação do primary server e do backup server através do método "validadeIfMaster"(que lança uma custom exception **NotMasterException** caso não seja.

4.2 UDP

Correndo o TCP server em máquinas diferentes (ou duas vezes na mesma máquina) consegue-se simular o comportamento de um server primário e de um server secundário. Estes comunicam entre sim através de duas threads: UDP Sender e UDP Receiver. Quando o UDP receiver deixa de receber heartbeats, espera alguns segundos até começar a receber clientes.

4.3 RMI

O Java RMI(Remote Method Invocation) usa uma metodologia diferente da do TCP. Ao contrário deste, não é estabelecido directamente um canal de comunicação como no TCP em que o servidor está a escuta num determinado porto e para cada cliente cria um socket. Especificamente, neste projecto o RMI é o nosso main-server, sendo este responsável pelos métodos existentes em toda a sua envolvente. Torna-se assim também o nosso **single point of failure**. Assim que é iniciado é criado o registry para os diversos recursos do programa poderem chamar os seus métodos.

5 Exception Handling

No que toca a tratamentos de excepção, foi assegurado que um Cliente reconecta ao mesmo servidor em caso de falha temporária do TCPServer (através de um tempo de espera), sendo a falha transparente para o usuário final. Através do uso de base de dados, o TCPClient não perde a ideia que está sendo escrita. O TCPServer reconecta em caso de perda de conectividade com o RMIServer e a falha transitória da RMIServer é transparente para o usuário final. Também houve o cuidado da verificação e foi assegurado que não existem operações duplicadas. duplicadas.

5.1 A nível de Base de Dados

O tratamento de erros do nosso MySQL é baseado em condições, que por sua vez, são nomeadas e preparadas para serem executadas mediante ao atendimento de tais condições. Além de tratarmos a violação de chaves primárias, foi tido em conta que os ids não assumam valores nulos (NULL), assim como houve um cuidado para que estes fossem auto-incrementáveis. Ao definir corretamente todas as relações e entidades fracas, foi possível haver um relacionamento directo entre entidades de diferentes campos. Foi dado **cascade** em operações de *update* e *delete*, para que os dados entre tabelas se mantenham coerentes.

5.2 TCP

De modo a tornar esta aplicação ao mesmo tempo transparente a erros e eficiente no seu tratamento procurámos controlar as exceções provenientes dos mecanismos de conexão (Sockets, RMI registry, etc.). Estes problemas advêm de falhas de ligação entre o servidor e o cliente, de uma ruptura do socket TCP, etc. Implementámos diversas rotinas para gerir estas exceções (try/catch (IO Exception, Remote Exception)) em ambas as 'extremidades' da aplicação. As threads responsáveis por estas funções permitem não só isolar o tratamento destas situações, podendo o cliente continuar a usar a aplicação (com algumas restrições), como também restabelecer as ligações entre as entidades executantes ou, em caso de falha, ativar um backup. Quando o servidor deixa de funcionar correctamente, vai ser detectada uma falha. O cliente irá então tentar voltar a conectar-se, sendo

a thread UDP Sender a responsável pelas tentativas. Na tentativa do Cliente se conectar ao Servidor, este efetua uma espera de um intervalos de 3 segundos entre cada tentativa de conexão. Entretanto, caso o cliente tente realizar algum pedido, a thread destinada à escrita de teclado fica bloqueada até este realizar re-connect.

5.3 RMI

No RMI foram dados os mesmos tratamentos de exceção utilizados no TCP.

6 FailOver

O fail-over pressupõe a existência de dois servidores (primary e backup) que comunicam entre si através de um canal UDP. Só assim é possível controlar o estado (ativo/inativo) de cada uma das entidades e efetuar a troca em caso de necessidade. Para tal, operámos um mecanismo de hearbeats (2s).Se depois de 10 segundos o server em utilização não comunicar, os clientes serão redirecionados para o servidor secundário que assumirá o papel de principal.

7 Causal Order Implementation

Para implementar a ordem causal no projecto foi utilziada um LinkedBlockingQueue em cada thread de Events. Assim que o server RMI pretende enviar mensagens para N utilizadores, faz callback do método sendMsg do TCP Server. Este limita-se a ver se os destinatários estão na sua Hashtable e faz **offer** às mensagens para a LinkedBlockingQueue. A thread Events está então responsável por ler o conteúdo da LinkedBlockingQueue e enviar para o seu cliente.

8 Description of tests (Table with Pass/Fail)

Foram feitos diversos testes nas diversas funcionalidades da aplicação, de modo a assegurar a robustez da aplicação garantindo, assim, um sistema seguro e eficiente, procurámos testá-lo minuciosamente.

8.1 Teste de Registo

Test Name: Registar				
Description: Registo do cliente na BD da aplicação				
Prerequisites: O cliente tem de entrar na aplicação e escolhe a opção REGISTAR				
Setup: O cliente cria uma nova conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Escolhe a opção Registar	Sistema abre os campos de registo	São os esperados	PASS
2	Inserir username, password e email	Sistema aceita dados e guarda na BD ou rejeita caso o username escolhido já esteja a ser utilizado	São os esperados	PASS

8.2 Teste de Login

Test Name: Login				
Description: Identificação do cliente				
Prerequisites: Cliente tem de ter uma conta criada na aplicação				
Setup: O cliente insere a sua pass e username para usar a sua conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Inserir username, password	Sistema aceita dados se essa conta existir	São os esperados	PASS

8.3 Teste de criação de meeting

Test Name: Teste de Submissão de Meeting				
Description: É testada a funcionalidade de introduzir meeting				
Prerequisites: O utilizador tem que já estar registado				
Setup: O servidor irá fazer handle das meetings				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**P / F**
1	logar User	Login efectuado com sucesso!	Nada apontar	PASS
2	User acede ao menu e escolhe a opção 1	aparece o passo seguinte	antes de escolher a opção, aparece uma listagem de opções	PASS
3	Introduz os detalhes da meeting	aparece o passo seguinte	nada apontar	PASS
4	Escolhe vários utilizadores para a meeting	Output "Meeting successfully created"!	nada a apontar	PASS
5	Escolhe a opção 2 do menu	A meeting acaba de criar encontra-se na lista	nada a apontar	PASS

8.4 Teste de Ligação ao Servidor Backup

Test Name: Ligação ao Servidor Backup				
Description: Teste de Ligação ao Backup				
Prerequisites: Cliente tem de ter uma conta criada na aplicação e devem estar os dois servidores a correr. Será feita uma ligação normal entre cliente TCP e Servidor TCP. De seguida será quebrada a ligação do servidor principal TCP, esperando assim que o cliente se reconecte ao servidor backup				
Setup: Acesso normal À sua conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Inserir username, password	Sistema aceita dados se a conta existir	São os esperados	PASS
2	O servidor principal é desligado	Existe um período de espera da parte do cliente até este se ligar ao servidor backup, podendo ter existido um curto downtime da parte do principal	O servidor principal não vem up dentro o tempo de espera	PASS
3	—-	Ligação feita ao servidor backup	São os esperados	PASS

9 Manual de Instalação

Para correr a aplicação irá necessitar dos seguintes requisitos:

- Java 8
- Servidor com MySQL com a respectiva base de dados (o script de criação da base de dados encontra-se na pasta support, ficheiro *meetosqlscript.sql*

Após reunidos todos os requisitos necessários, é necessário que o MySQL esteja a correr dentro da própria (localhost), ou no caso que esteja se encontre noutra máquina, que seja definido o respectivo IP e porta. Como foi mencionado anteriormente, deve-se correr o script *meetossqlcript.sql* que contem todas as tabelas criadas bem como alguns dados. Por fim, deve-se executar-se por esta ordem a execução de servidores da máquina:

- 1 - RMIServer.java
- 2 - TCPServer.java (duas vezes para simular o comportamento de servidor primário e secundário)
- 3 - TCPClient.java (quantas vezes necessário para criar vários clientes)

Relativamente ao IP de máquina onde os servidores e o cliente estão a correr, existe um ficheiro intitulado *property* (support/property) onde se pode alterar os endereços e as portas onde estes vão estar a correr.

```
tcpip1=127.0.0.1
tcpip2=127.0.0.1
tcpServerPort=7000
tcpServerPortAux=7001
tcpBackupServerPort=7002
tcpBackupserverPortAux=7003
udpPort=5432
rmiServerip=127.0.0.1
rmiServerPort1=1099
rmiServerPort2=6001
```