

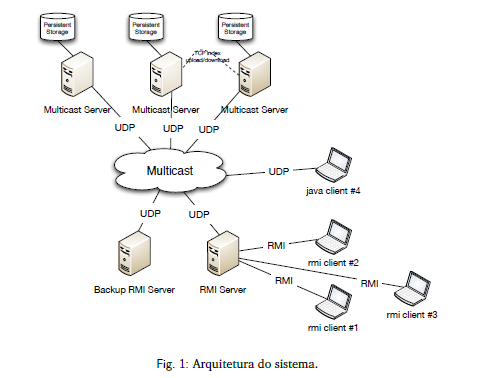
**ucBusca: Motor de pesquisa de páginas Web**

**Projeto de Sistemas Distribuídos – Meta 1**

Carlos Miguel da Luz Lima – 2017266922

Gabriel Nunes Saraiva - 2017256436

* **Arquitetura do projeto**



A arquitetura do nosso projeto consiste em vários clientes que, por RMI, comunicam com um servidor RMI, estando outro servidor a servir de backup para uma eventual falha do primeiro. Estes servidores posteriormente comunicam utilizando o protocolo UDP com servidores de multicast que processam o pedido e enviam de volta uma mensagem ao RMI Server que, por sua vez, envia para o cliente, concluindo assim o seu pedido. Na nossa implementação utilizámos apenas um servidor multicast para comunicação com o cliente, contudo, temos outro multicast secundário que vai receber informação do multicast principal com os ficheiros de utilizadores e atualiza no seu próprio ficheiro, servindo desta forma como um backup de ficheiros para o caso de ocorrer alguma falha no multicast principal.

* **Constituição do nosso projeto**

O nosso projeto é composto pelos seguintes ficheiros:

* MulticastServer.java
* User.java
* MulticastServer2.java
* RMIServer.java
* RMIServer\_Aux.java
* RMIClient.java
* Aux\_TCP.java
* Conecao1.java
* NewThread.java
* Thread\_Multicast.java
* users.txt
* urls.txt

Os ficheiros users.txt e urls.txt são ambos ficheiros de texto que guardam, respetivamente, a lista de utilizadores e os URLs anexados.

* **Funcionamento do Servidor Multicast**

O ficheiro MulticastServer.java corre o servidor Multicast.

O servidor Multicast tem como atributos um array com a lista de utilizadores que é preenchido no início da programa com a chamada da função “le\_ficheiros()”. Tem o HashMap Index que vai ser constituído por chave e valores, sendo as chaves a palavra encontrada num determinado url e os valores esses mesmos urls. Outro ArrayList “urls” que vai ter todos os urls da base de dados. Os HashMaps “titles” e “descricao” vão ser dois HashMaps com os respetivos valores <URL, título> e <URL, descrição> que vão ser necessários para enviar ao cliente quando ele faz uma pesquisa. Por último temos os dois Maps UrlCountMap e WordCountMap que vão ser consituidos respetivamente por <Url, popularidade> e <palavra, popularidade> necessários para depois enviar ao administrador a informação do sistema.

O servidor tem um ciclo while que esta sempre a correr à espera de uma mensagem enviada pelo RMI Server através de socket Multicast usando o protocolo UDP (Datagram Packets). O servidor recebe uma mensagem na forma de string “função:informação:”; A função é um número que , através de um switch, serve para indicar que tipo de função foi pedida. A informação consiste mesmo na informação que foi enviada pelo RMIServer. Ao saber que tipo de informação é pedida o servidor processa e envia o pedido através do mesmo socket na forma de string de volta para o RMIServer.

O servidor antes do ciclo “while” cria uma thread (Conecao1.java), esta thread serve para comunicar através de uma ligação TCP com um segundo servidor multicast que cria também ele mesmo uma thread para isso. Este servidor secundario, como não tivemos tempo suficiente, apenas serve de backup para guardar a lista de utilizadores num segundo ficheiro.

O servidor cria também uma outra thread (Thread\_Multicast.java), esta thread cria um novo socket com endereços e um porto diferente do socket inicial. Este novo socket comunica com o RMI Server e estão em persistente comunicação. Esta comunicação serve para saber se algum cliente recebeu alguma notificação enquanto está online.

* **Funcionamento do Servidor RMI**

O ficheiro RMIServer.java corre o servidor RMI.

O servidor RMI começa por criar o registo, o qual vai partilhar com o cliente. Para além disso, cria também um Multicast Socket que, usando o protocolo UDP, comunica com o MulticastServer.

A classe RMIServer contem vários métodos que podem ser chamados pelo RMI Client, ou seja, métodos remotos. Estes métodos são todos declarados na interface Rmi. Cada um destes métodos cria um socket que comunica com o servidor multicast, enviando uma string “função:informação:” e fica à espera da informação para depois enviar ao cliente.

No caso do servidor RMI principal falhar, temos um servidor secundário (RMIServer\_Aux.java) que contem os mesmos métodos que o principal, mas que cria uma ligação UDP através de outro porto.

* **Tarefas de cada elemento**

As tarefas foram divididas de forma equivalente, sendo que o Carlos ficou mais responsável da parte da pesquisa e da utilização da biblioteca Jsoup e o Gabriel tratou da interface e ligação entre os servidores, sendo o resto feito em conjunto pelos dois membros.

* **Testes realizados**

Os testes realizados foram correr o programa e testar cada função individualmente e garantir que funcionava, desligar propositadamente um servidor RMI para perceber se o secundário funcionava como devia.