

18 de Maio de 2011

Relatório do 1º Trabalho

Sistemas Distribuídos 2010/2011

*Instituto Superior de Engenharia de Lisboa*

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

**Discentes:**  
Ana Correia, 31831

João Silvestre, 32466

**Discentes:**  
A

**Docente:**  
Luís Assunção

.

..................

Índice

[1. Introdução 4](#_Toc293482163)

[1.1 Objectivos e Descrição do Trabalho 4](#_Toc293482164)

[1.2 Organização do documento 4](#_Toc293482165)

[2. Estrutura do Trabalho 5](#_Toc293482166)

[2.1 Introdução 5](#_Toc293482167)

[2.2 Estrutura 5](#_Toc293482168)

[2.3 Problemas conhecidos 5](#_Toc293482169)

[3. Trabalho Desenvolvido 7](#_Toc293482170)

[3.1 Introdução 7](#_Toc293482171)

[3.2 Implementação do Trabalho 7](#_Toc293482172)

[3.2.1 Objecto Peer 7](#_Toc293482173)

[3.2.2 Contentor de *Peers* 7](#_Toc293482174)

[3.2.3 Motor de Pesquisa 8](#_Toc293482175)

[3.2.2 Sponsors e Time-to-Live 8](#_Toc293482176)

[3.3 Tratamento de Erro 8](#_Toc293482177)

[4. Conclusões 10](#_Toc293482178)

[Referências 11](#_Toc293482179)

# 1. Introdução

Este documento consiste no relatório do primeiro trabalho da disciplina de Sistemas Distribuídos. Aborda-se o modo que foi resolvido o trabalho

## 1.1 Objectivos e Descrição do Trabalho

O trabalho consiste na criação de arquitectura Peer-to-Peer de procura de referências musicais. A arquitectura consiste em que um utilizador(Peer) consiga efectuar pesquisas de músicas pelos Peers que conhece. Caso o Peer que foi efectuado essa pesquisa não conter a música em questão, então reencaminha o pedido para os Peers que conhece.

## 1.2 Organização do documento

Este documento está dividido em 4 secções.

Na secção 2 consta a descrição do planeamento do trabalho, nomeadamente a sua estrutura, problemas detectados e as soluções apresentadas.

Na secção 3 apresenta-se a implementação do trabalho realizado, nomeadamente a implementação realizada, tratamento de erro.

Por fim a secção 4 contém as conclusões do trabalho realizado.

# 2. Estrutura do Trabalho

## 2.1 Introdução

Nesta secção vai ser abordada a estrutura planeada que foi efectuada para resolução do trabalho.

## 2.2 Estrutura

Um Peer pode efectuar as seguintes operações:

* Pesquisa de uma música;
* Responder a pedidos de músicas;
* Pedir a listas de utilizadores(peers) aos peers que conhece;

O peer irá ter uma estrutura de dados para guardar todas as música que este contem e também irá ter uma estrutura de dados para guardar todos os peers que conhece.

O Peer pode pedir aos seus peers a lista de peers conhecidos, para acrescentar à sua lista, de modo a aumentar o seu conhecimento de peers existentes na arquitectura.

## 2.3 Problemas conhecidos

O peer pode efectuar uma operação apresenta anteriormente, a um dado peer que não se encontra ligado.

Ao pedir a lista de peers aos seus peers conhecidos, terá problemas de concorrências nas respostas de cada peer, porque irá ser modificada a estrutura de dados onde se encontra os peers conhecidos, desse modo a solução será introduzir sincronização na leitura e escrita dessa estrutura.

# 3. Trabalho Desenvolvido

## 3.1 Introdução

Nesta secção irá ser explicada as decisões tomadas durante a implementação nomeadamente o tratamento dado às excepções de .NET Remoting, assim também como as técnicas utilizadas para implementa da arquitectura Peer-to-Peer.

Todo este trabalho foi implementado virado para a interface, de forma a permitir a alteração de qualquer um dos componentes mantendo os outros a funcionar.

## 3.2 Implementação do Trabalho

### 3.2.1 Objecto Peer

O objecto Peer é o “representante” de um *peer*, este é serializável pois este contém informação sobre o mesmo e dado que esta é pouca, o *overhead* de fazer pedidos ao *peer* remoto para a obter é desnecessário. Sendo que esta é sempre usada enviá-la para quem se liga pareceu ser a escolha óbvia. É também aqui que se encontram, no caso de ser informação de um *peer* remoto, os dois proxies para o contentor de *peers* e para o motor de pesquisa. Esta abordagem permite que toda a aplicação seja implementada quase sem se saber que é uma aplicação distribuída, quase pois é necessário tratar os casos de erro nos devidos locais.

### 3.2.2 Contentor de *Peers*

O contentor de peers é o serviço que permite aos *peers* remotos obter novos *peers* aos quais se ligarem, através de um outro. É também aqui que é possível obter um representante do *peer* ao qual a aplicação se pretende ligar. O contentor é portanto um objecto MarshalByRef para que possa ser acedido remotamente e é *singleton* pois este deve ser o mesmo para todos os pedidos dado que guarda estado.

Foi então necessário garantir que o acesso a esse estado partilhado é *thread safe* através da utilização de monitores. Dado que este estado passa por uma lista que ao mesmo tempo que alguém consultava pode estar a ser alterada pelo próprio *peer*, temos que garantir que isso não acontece.

### 3.2.3 Motor de Pesquisa

O motor de pesquisa é o serviço sobre o qual são feitas as pesquisas de músicas, este está implementado como *SingleCall* pois não guarda estado e o custo de criação é pequeno. As chamadas a este serviço, dado que a sua natureza é de ser uma operação que pode demorar algum tempo, são assíncronas. A resposta a uma pesquisa é feita por uma função de *callback*, que obriga a ter um canal “*full serializable”*, e não por retorno da função porque senão o retorno iria andar de *peer* em *peer* até a fonte do pedido, um *overhead* desnecessário. Portanto o *peer* que tiver resposta a uma pesquisa irá avisar directamente o interessado.

### 3.2.2 Sponsors e Time-to-Live

Na implementação temos dois tipos de chamada, um singleton para o PeerContainer e singlecall para o SearchEngine. No caso do PeerContainer, dado que este é singleton e a verdade é que queremos manter a instância criada durante toda a aplicação pois a sua criação é pesada dado ao contentor com todos os peers que o peer actual conhece, e que podem ser bastantes decidimos tornar o seu TTL infinito. Evitando assim que ele seja reciclado e garantido que realmente ele só é construído uma vez.

Para o *SearchEngine* dado que é *singlecall* é necessário manter o objecto vivo durante o tempo de vida dos clientes portanto foi optado colocar o *sponsor* do lado do cliente, ou seja, do lado do *peer* que se ligou. Pois esta é a melhor forma de evitar que o objecto se mantenha vivo depois do cliente se desligar/cair. No entanto, isto obriga a um canal que seja “*full serializable*” para que o “*peer servidor*” possa fazer pedidos ao “*peer cliente*”.

## 3.3 Tratamento de Erro

Como já foi identificado podem acontecer diversos erros numa aplicação distribuída, sendo neste caso concreto o mais evidente é o *peer* cair. Neste caso a opção tomada passa sempre por remover esse *peer* da nossa lista de peers conhecidos, caso este exista, pois se ele não se encontra a responder devemos evitar que mais uma vez a aplicação tente comunicar com ele pois isso resulta em recursos computacionais desperdiçados.

Embora existam *sponsors* a garantir que o tempo de vida de um *peer* não se esgota existe também prevenido o facto de uma falha na aplicação deixar o tempo de vida do *peer* esgotar-se, sendo mais uma vez a solução remover este da lista de *peer* disponíveis.

# 4. Conclusões

# Referências

x

|  |  |
| --- | --- |
| [1] | ARM. The Architecture for the Digital World. [Online (02-05-2011)]. <http://www.arm.com> |
| [2] | R. Schafer A. Oppenheim, *Discrete-Time Signal Processing 2nd edition*.: Prentice Hall , 1999. |
| [3] | Keil. LPC2294 User Manual. [Online (02-05-2011)]. <http://www.keil.com/dd/docs/datashts/philips/user_manual_lpc2119_2129_2194_2292_2294.pdf> |
| [4] | Andrew G. Dempster, Izzet Kale Robert Beck, "Finite-Precision Goertzel Filters Used for Signal," vol. VOL. 48, no. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING, 2001. |
| [5] | Kevin Banks. The Goertzel Algorithm. [Online (02-05-2011)]. <http://www.eetimes.com/design/embedded/4024443/The-Goertzel-Algorithm> |
| [6] | R. Schafer A. Oppenheim, *Discrete-Time Signal Processing 2nd edition*.: Prentice Hall, 1999. |
| [7] | Gene Small. Detecting CTCSS tones with Goertzel's algorithm. [Online]. <http://www.eetimes.com/design/embedded/4025660/Detecting-CTCSS-tones-> |

x

# 