

18 de Maio de 2011

*Instituto Superior de Engenharia de Lisboa*

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Sistemas Distribuídos 2010/2011

Relatório do 1º Trabalho Prático

**Discentes:**  
Ana Correia, 31831

João Silvestre, 32466

**Discentes:**  
A

**Docente:**  
Luís Assunção

.

Índice

[1. Introdução 5](#_Toc293507625)

[1.1 Objectivos e Descrição do Trabalho 5](#_Toc293507626)

[1.2 Organização do documento 5](#_Toc293507627)

[2. Estrutura do Trabalho 6](#_Toc293507628)

[2.1 Introdução 6](#_Toc293507629)

[2.2 Estrutura 6](#_Toc293507630)

[2.2.1 Pesquisa de Músicas 6](#_Toc293507631)

[2.2.2 Divulgação e Actualização da Rede de *Peers* 7](#_Toc293507632)

[2.3 Problemas Identificados 7](#_Toc293507633)

[3. Trabalho Desenvolvido 9](#_Toc293507634)

[3.1 Introdução 9](#_Toc293507635)

[3.2 Implementação do Trabalho 9](#_Toc293507636)

[3.2.1 Objecto *Peer* 9](#_Toc293507637)

[3.2.2 Contentor de *Peers* (*PeerContainer*) 9](#_Toc293507638)

[3.2.3 Motor de Pesquisa (*SearchEngine*) 10](#_Toc293507639)

[3.2.4 Sponsors e Time-to-Live 10](#_Toc293507640)

[3.3 Tratamento de Erro 10](#_Toc293507641)

[4. Conclusões 12](#_Toc293507642)

[Bibliografia 13](#_Toc293507643)

# 1. Introdução

Este documento consiste no relatório do primeiro trabalho da disciplina de Sistemas Distribuídos. Neste relatório aborda-se a arquitectura da aplicação, detalhes da sua implementação e os problemas encontrados.

## 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 localmente e, caso não encontre, procura remotamente através de *peers* conhecidos.

A aplicação deverá também conseguir procurar por novos *peers* de forma a extender a rede de *peers* conhecidos. Esta pesquisa é feita através dos *peers* que são conhecidos de momento.

## 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 as técnicas usadas durante a implementação e os problemas encontrados.

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 da aplicação, assim como todos os problemas detectados.

## 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* disponibiliza portanto dois serviços distintos, um serviço para divulgação e actualização da sua rede de *peers* e outro serviço para pesquisa de músicas.

## 2.2.1 Pesquisa de Músicas

O *peer* ao atender um pedido de pesquisa, verifica se contém essa música, caso encontre irá então responder ao *peer* que efectuou o pedido directamente, identificando a qual pedido a resposta pertence. Caso não encontre a música pretendida, irá então reencaminhar o pedido para a sua rede de *peers*. No entanto, o facto de reencaminhar o pedido para a sua rede poderá criar situações em que o pedido irá navegar eternamente na rede. Para evitar isto foi criado mecanismo de controlo de profundidade da pesquisa, baseado num contador, que por cada reencaminhamento irá ser decrementado. O contador ao chegar a 0 o pedido não irá mais ser reencaminhado.

No entanto, esta solução poderá evitar que o pedido chegue a um *peer* que contenha a música, no entanto a distância lógica a este não permite que o pedido lá chegue.

## 2.2.2 Divulgação e Actualização da Rede de *Peers*

A decisão tomada para este serviço foi que este seria um serviço completamente autónomo do anterior, pois uma das primeiras soluções encontradas seria a de sempre que uma resposta a uma pesquisa fosse de um *peer* desconhecido, este seria adicionado a rede de *peers*. No entanto, esta solução iria criar um aumento da rede de *peers* muito lento e dependente da utilização da aplicação para pesquisas e da quantidade de respostas à mesma.

Decidiu-se então que este sistema deveria periodicamente pesquisar por novos *peers* para a sua rede, sendo este efeito conseguido através da rede de *peers* actuais. Ou seja, vai ser pedido a todos os *peers*, actualmente na rede, a sua rede de *peers* e usando esta rede obter-se-á todos os *peers* que ainda não eram conhecidos pelo *peer* que efectuou o pedido, e adicioná-los-á à sua rede.

## 2.3 Problemas Identificados

Dado que esta é uma arquitectura de um sistema distribuído terão que se ter em conta todos os problemas que são inerentes a um sistema desta natureza. Os problemas identificados foram portanto:

1. Um *peer* poderá tentar comunicar com um outro que já não se encontra ligado.
2. Um *peer* poderá tentar obter a rede de *peers* de um outro enquanto esta se encontra a ser modificada.

A solução identificada para o primeiro problema passa por, identificar quando é feito um pedido se este não foi possível ser efectuado, e caso isto se verifique remover o *peer* da rede de *peers*, de forma a evitar que futuros pedidos sejam feitos ao mesmo.

A solução identificada para o segundo problema passa por, 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 as técnicas utilizadas na implementação da arquitectura *Peer-to-Peer*.

Todo este trabalho foi implementado voltado 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 informação é sempre usada, enviá-la para o peer se liga pareceu ser a escolha óbvia. É também aqui que se encontram, no caso de ser informação de um *peer* remoto, os dois *proxy* 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* (*PeerContainer*)

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.

#### 3.2.2.1 Sincronismo

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.

Isto é conseguido através de *locks* nas leituras e escritas da lista para que apenas uma operação aconteça sobre a lista. Pois não se deve iterar uma lista que se encontra a ser alterada, nem o contrário.

### 3.2.3 Motor de Pesquisa (*SearchEngine*)

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.

### 3.2.4 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 se quer 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 (*Time-to-Live*) 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 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 deve-se 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

Após a conclusão da arquitectura foram detectadas algumas características da mesma, que podem ser vistas como falhas, que na altura da concepção não foi possível identificar uma solução óptima para as mesmas. As soluções adoptadas, embora com algumas falhas, foram as melhores identificadas.

Devido a alguns detalhes da arquitectura, nomeadamente à circulação de uma pesquisa na rede, é provável que um pedido circule em “*loop*” na rede, gerando várias respostas para o mesmo pedido pelo mesmo *peer*. No entanto, a arquitectura não previne isso pois o custo de manter estado sobre os pedidos respondidos por um determinado *peer* tornar-se-ia muito pesado em termos de memória ao fim de algumas centenas de pesquisas ou para evitar isto, guardando apenas um limitado número de pedidos, correríamos sempre o risco que isto volta-se acontecesse. Também existe a possibilidade de com esta arquitectura um pedido de pesquisa, que poderia ser respondido por um *peer* na rede, nunca o chegar a atingir pois a distância lógica ao *peer* ser demasiado elevado. No entanto estas limitações foram aceites como uma medida necessária para conseguir uma aplicação mais robusta e rápida.

# Bibliografia

Assunção, L. (2011). Slides de Apoio à Cadeira de Sistemas Distribuídos.

Microsoft. (s.d.). *.NET Remoting Overview*. Obtido em 05 de 2011, de MSDN: http://msdn.microsoft.com/en-us/library/kwdt6w2k(v=vs.71).aspx

# 