



ISEL

INSTITUTO SUPERIOR
DE ENGENHARIA DE LISBOA

Relatório

SD – Sistemas Distribuídos

[SV-2014/2015]

Trabalho Prático I

Carlos Serra [36907]

Adriano Sousa [38205]

Marta Nascimento [38222]

INDICE

○ Introdução_____	3
○ Desenvolvimento_____	4
○ Ficheiro XML de configuração de um Peer_____	5
○ SingleCall ou Singleton_____	5
○ Pesquisa_____	6
○ Interface IPeer_____	6
○ Classe Peer_____	7
○ TTL_____	7
○ Conclusão_____	8

INTRODUÇÃO

Um sistema distribuído executa-se em vários computadores e é composto por vários componentes de *software*. Estes comunicam através de protocolos distribuídos para assistirem a execução coerente de atividades distribuídas e cujo resultado pode ser visto como se só existisse um único computador. Um sistema distribuído partilha um estado global e coopera para a obtenção de um objetivo comum.

O objetivo deste trabalho é a realização de um sistema onde cada utilizador (Peer) gere uma coleção de referências musicais.

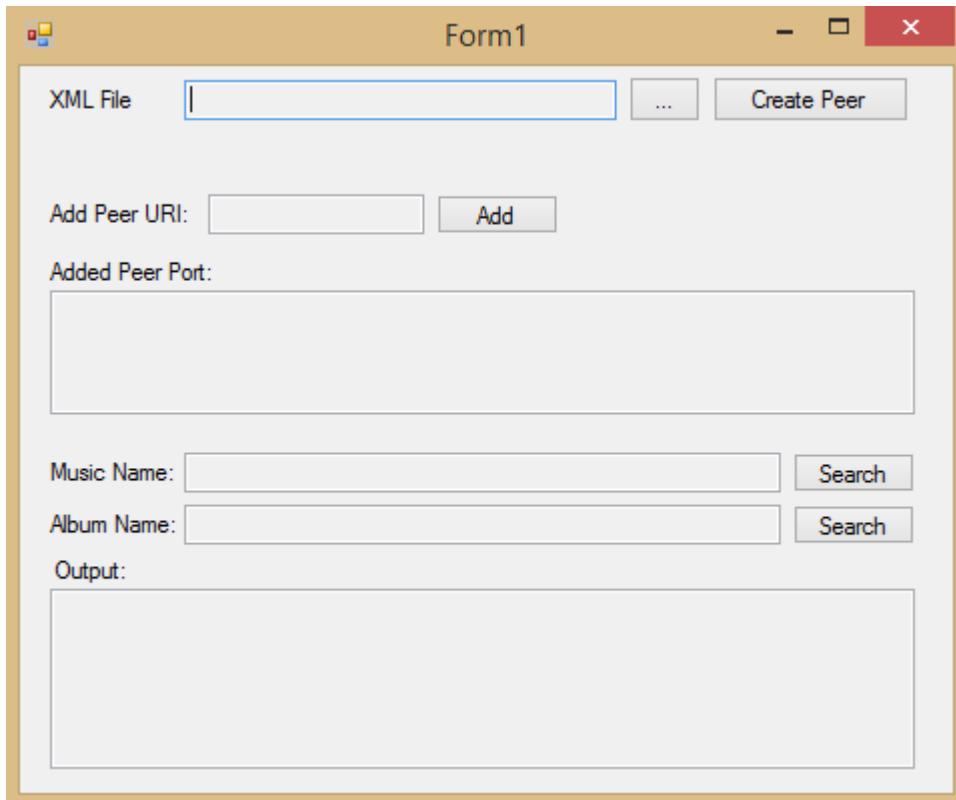
Cada Peer é responsável por:

- Aceitar pedidos, processá-los e devolver um resultado. Ou seja, aceitar e processar pedidos de procuras de músicas de outros Peers, e responder com o resultado da procura, que é representado pelo seu uri.
- Realizar pedidos e obter os resultados. Isto é, fazer pedidos aos Peers “vizinhos” sobre uma música e aguardar, assincronamente, pelo seu resultado.
- Cooperar com outros pares de forma simétrica por forma a executar uma tarefa.

Isto irá permitir ganhar capacidade no desenvolvimento de sistemas distribuídos usando objetos distribuídos na plataforma .NET.

DESENVOLVIMENTO

A aplicação começa com uma simples interface onde é possível carregar um ficheiro XML, onde conste toda a informação necessária para a criação de um Peer, isto é, um PeerUri (uri para si próprio), ConnectionType (tipo de conexão utilizada), um Port (porto), uma lista de músicas, álbuns e os URI's de todos os Peers que este pode estar conectado.



The screenshot shows a Windows-style window titled "Form1". Inside the window, there is a form with several input fields and buttons. At the top, there is a text box labeled "XML File" followed by a browse button "...". To the right of the "XML File" text box is a button labeled "Create Peer". Below this, there is a text box labeled "Add Peer URI:" followed by an "Add" button. Underneath that is a text box labeled "Added Peer Port:". Further down, there are two rows, each with a text box and a "Search" button. The first row is labeled "Music Name:" and the second row is labeled "Album Name:". At the bottom of the form is a large text box labeled "Output:".

Figura 1 – Interface Gráfica

Assim que um dado Peer é criado, deixa de ser permitido que se crie outro Peer, abrindo assim as opções onde se pode adicionar o URI de um outro Peer ou procurar por uma música. Assim que é feita a criação desse novo Peer, são associados todos os Peers correspondentes, mas apenas aqueles que se encontrem *online*. Para garantir a não existência de associações com Peers *offline*, é sempre realizado um teste de conexão, informando assim o utilizador sobre o estado do mesmo.

Ficheiro XLM de configuração de um Peer

Para carregar um Peer deve é necessário existir um ficheiro com extensão. XML com um elemento na raiz que representa o Peer (PeerConfig). Este deve conter obrigatoriamente o URI do Peer, podendo ter, opcionalmente, informação sobre os Peers associados, musicas, álbuns, porto e tipo de conexão.

O URI do Peer (PeerUri), o porto (Port) e o tipo de conexão (ConnectionType) são atributos do elemento PeerConfig enquanto os Peers associados (AssociatedPeers), os álbuns (Albuns) e as músicas (Musics) são elementos. Na imagem seguinte representa um exemplo de um ficheiro .XML para a criação de um Peer.

```
<?xml version="1.0" encoding="utf-8"?>
<PeerConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            PeerUri="http://localhost:1237/RemotePeer.soap"
            ConnectionType="http" Port="1237">
  <Musics>
    <string>music7</string>
    <string>music8</string>
  </Musics>
  <Albuns>
    <string>album7</string>
    <string>album8</string>
  </Albuns>
  <AssociatedPeers>
    <string>http://localhost:1236/RemotePeer.soap</string>
    <string>http://localhost:1234/RemotePeer.soap</string>
  </AssociatedPeers>
</PeerConfig>
```

Figura 2 – Exemplo de um ficheiro .XML

SingleCall ou Singleton

Quando registado o tipo de serviço, existe a possibilidade deste ser singlecall ou singleton. Singlecall implicaria que por cada registo fosse criado um novo, o que não seria verdade, para um dado URI, apenas pode existir um único Peer, então o registo é realizado com singleton, onde existe um e um só Peer por URI. Caso se tente criar dois Peers com o mesmo URI irá então aparecer uma mensagem de erro para que o utilizador possa aperceber-se da situação.

Pesquisa

Cada Peer dispõe assim de três listas, uma para os Peers associados, as músicas que este contém e uma outra lista de músicas que sabe onde se encontram. Esta última lista irá crescer conforme vai conhecendo a localização de novas músicas que fez a procura, facilitando assim a procura caso um Peer peça essa mesma.

Quando iniciada a procura de uma música, será validado se o próprio não contém a música. Caso não tenha, é então feita uma procura pela segunda lista, verificando se não conhece quem tenha essa música. Esta lista irá crescer conforme possíveis pesquisas com resposta afirmativa. Isto é, se um Peer pesquisar por N músicas e todas elas forem encontradas, então este Peer irá colocar na sua lista de músicas conhecidas as referências para os Peers que dispõem das músicas, para que futuramente facilite a procura a novos pedidos.

Caso a procura não tenha sido encontrada em nenhuma das listas é então iniciado um pedido a todos os IPeers encontrados na terceira lista, dos associados.

Caso exista uma resposta da parte de algum destes, a resposta será entregue diretamente a quem criou o pedido inicial, nunca passando novamente pelos intermediários.

Interface IPeer

Para a comunicação entre os Peers, foi criada a interface IPeer onde constam os seguintes métodos:

- **bool TestConnection():** permite que quando criada uma associação de Peers seja feito um teste pois o URI pode pertencer a um Peer offline.
- **void SearchMusic(IPeer p, string musicName, int ttl):** utilizado quando um Peer faz pedidos a outro Peer, devendo assim passar o IPeer que fez o pedido inicial, o nome da música que está a ser procurada e o time to leave. A variável 'p' será utilizada mais tarde, caso a música procurada seja encontrada. Desta forma o Peer que souber a localização da música deverá informar dessa ação, com p.MarkAsFound(...). O nível de profundidade deve então decrescer conforme faz pedidos, deixando de pesquisar quando atinge o valor 0.

- **void MarkAsFound(IPeer p, string musicName):** este método é chamado por outro Peer para informar que encontrou a música procurada pelo atual. Esta resposta pode vir de vários Peers ao mesmo tempo, como tal devem ser tomados cuidados conforme a implementação. Caso se queira armazenar unicamente o primeiro a responder, deve ser feita uma verificação pelo nome da música, se esta já foi encontrada ou não, que é a implementação feita no trabalho. Caso fosse pedido outro tipo de implementação, também o seria possível (exemplo: apenas armazenar o último, armazenar todos, etc.). O IPeer 'p' recebido por parâmetro serve para poder obter o URI de qual o Peer que tem a música procurada, podendo assim armazenar e ficar com a sua referência.
- **string GetPeerURI():** devolve o URI de um dado IPeer.

Classe Peer e interface IPeer

A interface IPeer é o esqueleto de um Peer que é conhecido e pode ser utilizado no exterior.

A classe Peer é quem vai tratar dos pedidos efetuados por parte de outros Peers, como tal será o nosso objeto partilhado, devendo estender MarshalByRefObject e implementar IPeer.

Esta classe é instanciada quando criado o novo Peer e apenas será conhecida pelo formulário que a criou. Isto apenas pode ser feito porque “quem cria, sabe o que criou”. Assim quando pesquisada, por exemplo, uma música, o formulário pode então utilizar métodos da própria classe Peer, não apenas de IPeer, métodos como AddMusic(...), AddAlbum(...), SetUri(...), SetForm(...), AddAssociatedPeer(...), etc.

Time to Leave (TTL)

Quando uma música não é encontrada num dado Peer, este tem como função de reencaminhar o pedido para todos os seus Peers associados, o que pode trazer problemas.

Poderiam ser utilizadas duas técnicas, TTL por contador ou por tempo tendo cada uma vantagens e desvantagens.

No caso da utilização do TTL por tempo, a procura dependeria de fatores como a ligação da rede, caso esta fosse lenta, os pedidos seriam realizados mais lentamente, fazendo menos pesquisas. No pior caso não seria feita qualquer pesquisa. Para que a interface gráfica não

parasse de aceitar pedidos, teria também de ser criada uma thread cuja função seria a de verificar se esse tempo já teria acabado ou não para que a thread da componente gráfica pudesse prosseguir.

Por outro lado informa o utilizador quando o tempo acabou e a procura deixou de fazer efeito.

No caso da utilização de TTL por profundidade, já não dependeria da rede (lenta ou rápida) o que irá sempre fazer a pesquisa pela profundidade desejada. Não é também necessária a criação de *threads* extra, pois a interface gráfica realiza o pedido e não fica bloqueada, o mesmo se passará com o atendimento do pedido, recebe o pedido, verifica se tem ou não a música e caso necessário faz os pedidos necessários, acabando de imediato.

Por outro lado tem a desvantagem de não se saber quando teve uma procura sem resultados pois não sabe quando a contagem chega a 0. Na realidade é possível informar, quando a contagem do TTL chegasse a 0. O Peer corrente poderia informar o Peer que iniciou a procura, mas isto poderia resultar em grandes quantidades de resposta. Por exemplo, se cada Peer tiver 10 Peers associados, e o nível de profundidade seja 2, então teria 100 respostas a dizer que não encontrou a música, o que seria confuso para o utilizador que fosse ler as mensagens. Outro caso ainda pior seria se dessas 100 respostas houvesse 1 positiva, recebendo 99 'não' e 1 'sim'. Com tantas respostas, o utilizador poderia não se aperceber que a música foi encontrada.

CONCLUSÃO

Durante a realização deste projeto, foram necessários bons conhecimentos em cadeiras anteriores, sem as quais o projeto se tornaria mais complicado de realizar, nomeadamente:

- RCp (Redes de Computadores) – protocolos de comunicação, nomeadamente HTTP, baseado em ligações TCP.
- PC (Programação Concorrente) – noção de *thread* e contexto de sincronização.

A cada fase, conseguiu-se superar todas as dificuldades que foram surgindo, o que requer uma boa análise do problema.

Atingiu-se o objetivo deste trabalho que era ganhar capacidade no desenvolvimento de sistemas distribuídos usando objetos distribuídos na plataforma .NET.