

**Relatório**

**SD – Sistemas Distribuídos**

**[SV-2014/2015]**

*Trabalho Prático I*

**Carlos Serra [36907]**

**Adriano Sousa [38205]**

**Marta Nascimento [38222]**

Introdução

Desenvolvimento

Um sistema distribuído é um conjunto de computadores autónomos e interconectados por uma rede, com software desenvolvido para uma determinada finalidade. Graças a este sistema é então possível concluir tarefas em comum.

Para este trabalho foi pedido um sistema onde cada utilizador (Peer) gere uma colecção de referências musicais.

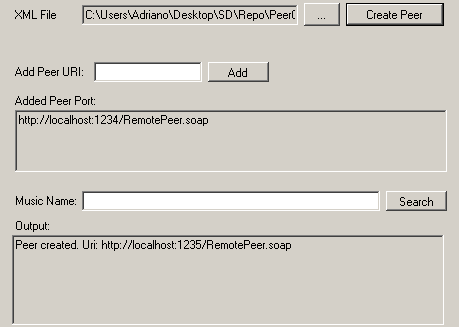
 A aplicação começa com uma simples interface onde é possível carregar um ficheiro XML, onde conste toda a informação necessária para a criação de um Peer, isto é, um PeerUri (uri para si próprio), ConnectionType (tipo de conexão utilizada), um Port(porto), uma lista de músicas, alguns e os URI’s de todos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

Fig.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 feita a criação desse novo Peer, são associados todos os Peer’s 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.

**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 Peer’s 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 contêm e uma outra lista de músicas que sabe onde se encontram. Esta ultima 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 ser o próprio não contem 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 Peer’s que dispõem das músicas, para que futuramente facilite a procura a outros pedidos.

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

Caso exista uma resposta da parte de algum destes, a resposta será entregue directamente 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 Peer’s 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 que fazer pedidos a outro Peer, devendo assim passar um IPeer que fez o pedido inicial, o nome da música que esta a ser procurada e o time to leave.

A vareável ‘p’ será utilizada mais tarde, caso a música procurada seja encontrada. Desta forma o Peer que souber a localização da música deverá informar dessa acção, 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 actual. Esta resposta pode vir de vários Peer’s ao mesmo tempo, como tal devem ser tomados cuidados comforme 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 ultimo, 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 ficar com a sua referência.

**- string GetPeerURI():** Devolve qual o URI de um dado IPeer.

**Classe Peer e interface IPeer**

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

A classe Peer é quem vai tratar dos pedidos efetuados por parte de outros Peer’s, como tal será o nosso objecto partilhado, devendo estender MarchalByRefObject e 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 nos deparamos com o caso que ‘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 Peer’s 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 factores como a ligação da rede, caso esta fosse lenta, os pedidos seriam realizados mais devagar, fazendo menos pesquisas. No pior caso não seria feita qualquer pesquisa. Para que a interface gráfica não parasse de aceitar pedido, 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 é fácil 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 fica despachada, o mesmo se passará com o atendimento do pedido, recebe o pedido, verifica se tem ou não a procura e caso necessário faz os pedidos, 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 currente poderia informar o Peer que iniciou a procura, mas isto poderia resultar em grandes quantidades de resposta. Por exemplo, se cada Peer tiver 10 Peer’s 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’, que com tanta resposta, o utilizador poderia nunca fazer ideia que a música foi encontrada.

CONCLUSÃO