**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

Engenharia Informática e de Computadores

**Sistemas Distribuidos**

**1º Trabalho Prático**

**.NET Remoting**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Membros do Grupo** | | | | | |
| **31768** | **Ricardo Romão** | **31923** | **Nuno Sousa** | **26657** | **Ricardo Neto** |

**Análise do Problema**

**Arquitectura**

Copnsiderando o enunciado, identificaram-se as entidades que irão intervir na solução, nomeadamente: Servidor de Zona, Cliente e Perito.

Para cada uma destas entidades avaliou-se a sua função no sistema e o conjunto de funcionalidades que deveriam oferecer, bem como, a quem oferecer essas mesmas funcionalidades.

**Servidor**

**Descrição**

O Servidor encontra-se inserido num anel de servidores em que todos os servidores conhecem a configuração desse anel.

A escolha para a distribuição da configuração recaíu sobre a criação de uma secção própria no ficheiro de configuração da aplicação servidora, secção essa denominada “RingServers”.

<configSections>

<section

name="RingServers"

type="System.Configuration.NameValueFileSectionHandler,System, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>

</configSections>

<RingServers>

<add key="Zone2" value="tcp://localhost:9002/RemoteServer.rem"/>

<add key="Zone3" value="tcp://localhost:9003/RemoteServer.rem"/>

<add key="Zone4" value="tcp://localhost:9004/RemoteServer.rem"/>

<add key="Zone5" value="tcp://localhost:9005/RemoteServer.rem"/>

<add key="Zone1" value="tcp://localhost:9001/RemoteServer.rem"/>

</RingServers>

Relativamente ao papel de um servidor de zona, este deverá oferecer funcionalidades aos seus clientes, bem como, a outros servidores do anel, conforme a ilustração que se segue.

De acordo com o observado, decidiu-se criar duas interfaces distintas, uma distribuída a clientes e outra distribuida a servidores, sejam elas IZoneServer e IRingServer, respectivamente.

|  |
| --- |
| Interface IZoneServer  Interface responsável por definir os comportamentos que um servidor deverá expor aos seus clientes. |
| GetExpertList(string) : List<IExpert>  Método que recebe como parâmetro um tema e devolve a lista de peritos capazes de responder perguntas acerca do mesmo. |
| NotifyClientFault(string, IExpert)  Método chamado pelo cliente, a fim de informar o servidor falha na comunicação com um perito para determinado tema. |
| Register(string, IExpert)  Método chamado pelo cliente para registar um objecto perito para determinado tema.  Internamente o servidor irá difundir este pedido de registo ao próximo servidor do anel. |
| UnRegister(string, IExpert)  Método chamado pelo cliente para cancelar o registo de um objecto perito para determinado tema.  Internamente o servidor irá difundir este pedido de cancelamento de registo ao próximo servidor do anel. |

|  |
| --- |
| Interface IRingServer  Interface responsável por definir os comportamentos que um servidor deverá expor aos outros servidores do anel. |
| Register(Guid, string, IExpert)  Método chamado por um servidor que pretende difundir o pedido de registo de perito por parte de um cliente seu. O método recebe o Guid do servidor que iniciou a difusão, permitindo ao mesmo termina-la. |
| UnRegister(Guid, string, IExpert)  Função igual ao método Register, contudo, com o objectivo de cancelar o registo de um perito. |

**Modo de Activação**

Cada servidor é responsável por atender pedidos de uma zona, assim, consideramos que a activação apropriada é Singleton.

Este modo de activação é o mais adequado devido ao facto dos servidores serem *statefull*. Não faria sentido um modo de activação Singlecall uma vez que, nessa modalidade, cada cliente “depositaria” em instâncias diferentes do servidor de zona os seus objectos peritos.

**Cliente**

**Descrição**

O Cliente nesta solução é a entidade que utiliza o servidor e o perito.

Do servidor irá ordenar registo e cancelamento de registo de peritos, bem como, solicitar listas de peritos para determinados temas. Conforme referido atrás, o cliente será conhecedor da interface IZoneServer.

No que respeita ao perito o cliente irá ter referências para dois tipos de peritos: os peritos criados por si (locais) e os peritos devolvidos pelo servidor (remotos).

Dos peritos criados por si o cliente estará interessado em capturar os eventos relativos à colocação de perguntas e devolução de respostas por parte desse perito a qualquer outro cliente. Para tal, na criação de um perito, o cliente subscreve o evento exposto pela interface IExpert, interface esta que que será falada mais á frente.

Dos peritos obtidos do servidor o cliente estará interessado em colocar perguntas e obter respostas. A colocação de perguntas e obtenção de resposta é feita de forma assíncrona, por forma a preservar a resposta da interface gráfica.

O cliente expõe eventos que visam permitir aos seus consumidores (form, consola,...) obterem notificação acerca de erros ocorridos, obtenção de notificações por parte de peritos “locais” relativamente a questões respondidas e informação relativa a resposta de perito “remoto”

**Distribuição**

No que respeita à distribuição desta entidade, inicialmente pensou-se em criar a interface IzoneClient como forma de expor um método (ReceiveAnswer) que pudesse ser chamado por um perito “remoto”, notificando o cliente de uma resposta.

Todavia, conforme indicado em cima, implementou-se o processo de colocar pergunta e obter resposta de um perito de forma assíncrona. Desta forma, o Cliente disponibiliza um callback a ser chamado pelo perito quando tiver uma resposta disponível.

**Detecção de Falhas**

Sempre que um cliente coloca uma questão a um perito e detecta uma falha informa o servidor da mesma, ordenando o cancelamento do registo desse perito.

**Perito**

**Descrição**

O perito é a entidade capaz de responder a questões (materializadas sob uma lista de palavras chave) acerca de determinado tema.

Tendo apenas a função indicada, a interface exposta pelo perito é a seguinte:

|  |
| --- |
| Interface Irepository  Interface responsável por definir as funcionalidades expostas pelo repositório. |
| Evento OnQuestionAnswered : QuestionHandler  Este evento é exposto para que um cliente o possa subscrever, podendo assim ter informação relativa a que perguntas são colocadas a um perito e quais terão sido as respostas dadas. |
| Ask(List<String>) : String  Método que, recebendo uma lista de palavras chave devolve uma resposta adequada, após consulta do repositório. |

**Distribuição**

Uma instância de perito é utilizada localmente por um cliente, quando criada pelo mesmo ou, utilizada remotamente, no caso de ser um perito disponibilizado por um servidor a um cliente.

Em qualquer um dos cenários, ambos os clientes conhecem a interface do perito não havendo portanto uma publicação do serviço por qualquer entidade.

**Repositório**

Um perito é sempre instanciado no cliente e acede a um repositório com a seguinte interface:

|  |
| --- |
| Interface Irepository  Interface responsável por definir as funcionalidades expostas pelo repositório. |
| GetThemes() : List<String>  Método chamado por um cliente que pretende obter informações acerca de quais os temas disponíveis na actual instância de repositório. |
| GetAnswer(List<String>, String) : String  Método que, recebendo uma lista de palavras chave e um tema, devolve uma resposta adequada.  Caso não tenha resposta é devolvida uma mensagem indicando esse facto. |

No que respeita à implementação desta interface, optou-se por, conforme sugerido, utilizar o suporte existente na framework para manipulação de XML. Em baixo mostra-se um exemplo da estrutura do ficheiro XML representativo do repositório.

<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="repSchema.xsd">

<theme name="theme1">

<card>

<question text="question"/>

<answer text="answer"/>

</card>

...

</theme>

<theme name="theme2">

...

</root>