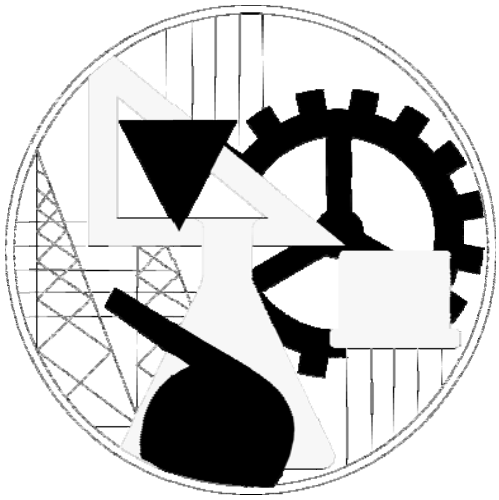
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

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.

Canal

O canal escolhido para a implementação da solução é do tipo TCP, com vista a diminuir o tráfego na rede (considerando a alternativa disponível HTTP), e o tipo de formatação é binária, uma vez que oferece a possibilidade de transporte de genéricos.

Servidor

Descrição

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

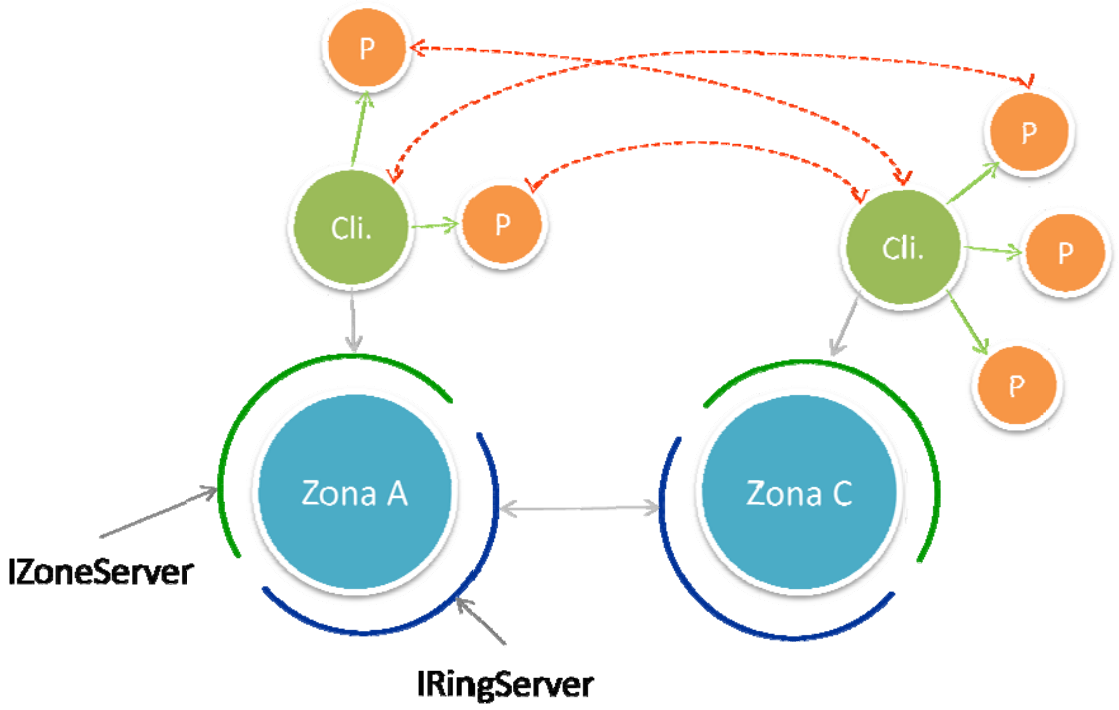
A escolha para a distribuição da configuração recaíu sobre a criação de uma secção própria no ficheiro de configuração da aplicação servidora, secção essa denominada “RingServers”.

```
<configSections>
  <section
    name="RingServers"
    type="System.Configuration.NameValueFileSectionHandler,System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
</configSections>

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

Quando um servidor é instanciado, este identifica qual o próximo servidor, de acordo com a configuração do anel, que deverá contactar assim que lhe for ordenada tal acção.

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.
GetSponsor() : ITriviaSponsor Método *factory* chamado pelo cliente para obter uma instância de sponsor que terá a função de prolongar o tempo do objecto remoto que encapsula o servidor do Cliente.

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.
GetSponsor() : ITriviaSponsor Método *factory* chamado por um servidor para obter uma instância de sponsor que terá a função de prolongar o tempo de vida do objecto remoto que encapsula o próximo servidor disponível no anel.

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.

Tempo de vida

A gestão do tempo de vida das referências para o servidor terá que ser garantida pelas duas entidades que o podem utilizar, Cliente e Servidor de Anel.

A fim de poder oferecer uma estrutura de *sponsor* que fosse conhecida pelas várias entidades, decidiu-se criar a interface ITriviaSponsor (que implementa Isponsor) com a seguinte estrutura:

Interface ITriviaSponsor Interface responsável por definir o comportamento que um *Sponsor* deverá oferecer
SetNotRenew() Método chamado por uma entidade que pretenda impedir a renovação do tempo de vida do objecto "*sponsorizado"* por este tipo.
Renewal (ILease) : TimeSpan Método de ISponsor que determinará qual a duração de tempo a ser adicionada ao tempo de vida do objecto "*sponsorizado"* por este tipo.

Registo de Sponsor por Cliente

Com vista a permitir que o Cliente continue a ter disponível o objecto que activou inicialmente, este terá que registar um *sponsor* no Servidor, manifestando assim interesse em que o tempo de vida da instância seja renovado.

Assim, no momento em que o Cliente activa o Servidor, solicita uma instância de ITriviaSponsor, guardando-a e registando-a através do serviço de lifetime associado à instância de servidor.

Desta forma, o CLR que guarda a instância local do servidor, ao detectar que este objecto foi acedido remotamente, inicia o processo de verificação do tempo de vida, fazendo uma chamada ao(s) *sponsor(s)* registados no mesmo, prolongando ou não, conforme respostas obtidas.

A partir do momento em que o Cliente já não necessita do Servidor, tipicamente quando se vai desligar, com base na instância que guardou para o *sponsor*, faz a chamada ao método SetNotRenew e anula o registo no lease da instância de Servidor.

Registo de Sponsor por Servidor

Uma vez que um Servidor necessita de interagir com outro Servidor, normalmente o seguinte (pelo encaminhamento de registos/anulações de peritos), é necessário que consiga garantir o tempo de vida deste último.

Assim, e devido à dinâmica que pode existir dentro do anel, foi criado o método privado *SetNextServer* que, em caso de ainda não ter sido feita nenhuma ligação ou a ligação com o servidor *siebeling* tenha sido perdida, este é responsável por conseguir ligação a um servidor.

A tentativa de obter do Servidor de destino uma instância de *sponsor* será determinante, uma vez que, desta operação irá resultar a instância pretendida de *sponsor* ou uma falha, indicando assim que a ligação não foi estabelecida.

A fim de garantir que, quando for destruída localmente a instância do servidor, foi implementado um destrutor que terá a função de, além de libertar recursos, cancelar o registo do *sponsor* ao Servidor seguinte.

Interacção com outros servidores

Por forma a que, por exemplo, um Cliente da Zona 1 possa fazer perguntas a um perito de um Cliente da Zona 2, decidiu-se encaminhar os pedidos de registos de peritos automaticamente.

Assim, por cada vez que um Cliente regista um perito, este registo será encaminhado pelo Servidor para o Servidor seguinte até que chegue a si, altura essa em que o encaminhamento parará.

Esta solução visa minimizar a actividade no anel, evitando por exemplo o cenário em que o Servidor da Zona 1 tenha que obter peritos para o tema X e, dessa forma teria que iniciar um pedido ao Servidor seguinte, pedido esse que seria encaminhado por todos os servidores do anel, onde estes depositariam as referências para os peritos apropriados.

Com a adopção desta solução incorreu-se numa falha que permite que um Servidor que tenha sido inidicado mais tarde que os outros indique não haver peritos disponíveis para nenhum tema.

Instruções para Utilização

No sentido de se poder utilizar um Servidor, deve-se utilizar o projecto **TriviaServer**, configurando a chave *serverId* para um valor válido no anel, bem como, o porto do canal no ficheiro de configuração.

Cliente

Descrição

O Cliente é a entidade que consome o Servidor e o Perito. Do Servidor irá ordenar registo e anulação de registo de peritos, bem como, solicitar listas de peritos para determinados temas sendo, conforme referido atrás, 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).

Peritos Locais vs. Peritos 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.

Eventos expostos a "consumidores"

O cliente expõe eventos que visam permitir aos seus consumidores (form, consola,...) obterem notificação acerca de ocorrências no decorrer da interacção com Peritos e Servidores, conforme seguem.

Interface IClient Interface responsável por definir o comportamento que um Cliente oferece.
ErrorHandler(String) **- OnError** Evento a ser subscrito por quem pretende receber informação acerca de erro ocorrido.
QuestionHandler(IExpert, String, String) **- OnQuestionAnswered** Evento a ser subscrito por quem pretende receber informação acerca de pergunta e respostas respondidas por perito "local"
ThemeHandler(String) **- OnExpertsGetComplete** Evento a ser subscrito por quem pretende receber informação acerca da recepção de lista de peritos para determinado tema
ResponseHandler(Int32, String) **- OnAnswerReceived** Evento a ser subscrito por que pretenda receber informação acerca de pergunta respondida por perito "remoto", referindo o número da pergunta e a resposta.
...

Interface IClient

Inicialmente pensou-se 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.

Essa ideia foi afastada e, a substituição assenta sobre a utilização da APM para recolher retornos de métodos de Peritos chamados assíncronamente.

Assim, disponibiliza-se uma interface IClient apenas com o objectivo de definir a estrutura para um consumidor, não estando implicita qualquer ideia de distribuição para esta entidade.

Interface IClient Interface responsável por definir o comportamento que um Cliente oferece.
Connect() Método responsável por iniciar uma ligação ao servidor de zona.
Disconnect() Método responsável por terminar a ligação com o servidor de zona.
AddLocalExpert(String) Método que, com base à consulta de repositório, irá criar um Perito local.
RegisterAll() Ordem de registo de todos os peritos locais no Servidor.
UnregisterAll() Ordem de anulação de registo de todos os peritos locais no Servidor.
IsConnected() Informação acerca da existência de conectividade com o servidor.
GetServerUrl() Informação acerca de qual o URL do servidor a que se está ligado.
GetQuestionCount() Informação acerca de qual o número de pergunta colocadas.
Ask(String, List<String>) Método responsável por solicitar resposta a pergunta acerca de determinado tema.
GetExperts(String) Método responsável por solicitar lista de peritos relativos a determinado tema.

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.

Internamente o Servidor deverá encaminhar pelo anel este pedido de cancelamento de registo, evitando a distribuição de um perito inacessível por outros Clientes.

Instruções de Utilização

No sentido de se poder utilizar um Cliente, deve-se utilizar o projecto **TriviaClient**, configurando o caminho para o Servidor de zona pretendido, bem como, a origem de dados, que deverá ser um ficheiro XML que seja validado pelo *squema*.

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.

Tempo de Vida

No que respeita à gestão do tempo de vida das referências para o Perito foram consideradas as seguintes opções:

Cliente criador regista *sponsor*

O Cliente que cria o Perito é o responsável por registar um *sponsor* capaz de prolongar o seu tempo de vida, no momento em que obtem confirmação de registo do Perito no servidor.

Este método tem a desvantagem de, mesmo que ninguém se interesse pelo tema que o Perito representa, o tempo de vida do mesmo será renovado, estando apenas a ocupar recursos

Servidor regista *sponsor*

O Servidor que aceita o Perito, por parte de um Cliente, deve registar um *sponsor.*
Esta solução tem a desvantagem de se estár a dar uma responsabilidade não adequada ao Servidor, sendo este apenas um encaminhador de recursos, neste caso Peritos.

Cliente criador e Cliente consumidor registam *sponsor*

O Cliente que cria o Perito é o responsável por registar um *sponsor* e, regista o Perito no Servidor. Quando esse Perito é distribuido a um Cliente consumidor, este irá registar um outro *sponsor*.
O pretendido seria garantir que este último registo seria escutado pelo Cliente criador e, nesse instante, ele iria retirar o seu *sponsor*.

Cliente consumidor regista *sponsor*

O Cliente que solicita um Perito para determinado tema, ao recebe-lo do Servidor será responsável por registar nesse objecto um *sponsor*, garantido renovação do seu tempo de vida, justificando-se pelo facto de o ter solicitado ao Servidor. Esta foi a solução adoptada pela simplicidade de implementação e garantia de prolongação do tempo de vida apenas quando existe interesse no tema representado pelo Perito.

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>
```

Instruções de Utilização

No sentido de serem disponibilizados alguns dados de exemplo, distribuimos alguns ficheiros XML, cada um relativo a um tema, disponíveis na pasta *data* do projecto **TriviaModel**.