**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

Engenharia Informática e de Computadores

**Sistemas Distribuidos**

**3º Trabalho Prático**

**Web Services**

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

**Análise do Problema**

**Arquitectura**

De acordo com o enunciado, pretende-se o desenvolvimento de um sistema que, no final, permita a consulta de informação relativa a filmes em exibição em vários cinemas.

A fim de o permitir, deverão ser disponibilizados vários *webservices* (um por cinema) que disponibilizem toda a informação relativa ao próprio cinema e que, adicionalmente, ofereçam funcionalidades de reserva de filmes.

Para que todos os *webservices* disponibilizados pelos cinemas possam ser encontrados, deverá ser desenvolvido um serviço de directório, designado por Broker, que disponibilize essa informação, bem como, possibilite o registo e cancelamento de registo de cinemas.

**Tipos de Repositórios**

No desenvolvimento da solução optou-se por distinguir entre dois tipos de locais para armazenamento de dados: repositórios de leitura e repositórios de leitura e escrita.

No que respeita aos repositórios de leitura, a cada um está associado um tipo Singleton que expõe as funcionalidades apropriadas aos dados. Este tipo é responsável por garantir que as operações de consulta são feitas de forma *thread safe*.

Os repositórios de leitura e escrita foram implementados com recurso a objectos distribuidos oferecidos pela framework .NET Remoting. Assim, o acesso aos dados é feito através da activação no modo Singleton de um objecto remoto, implementado num servidor dedicado a esta tarefa.

**Broker *Webservice***

**Descrição**

O Broker é o serviço cuja localização deverá ser conhecida por todos os Clientes que pretendam obter informações acerca de endpoints para *webservices* disponibilizados por cinemas, bem como, os nomes destes.

Este deverá ter acesso a um repositório de leitura e escrita que guarde os registos, i.e., dados relativos a cada cinema.

**Arquitectura**

A arquitectura escolhida para o processo relativo ao Broker foi a seguinte:

**Repositório - Acesso**

No sentido de centralizar o acesso ao directório de cinemas, optou-se por implementar a entidade *RegistryServer* como sendo a entidade responsável por aceder ao repositório de leitura e escrita, que guarda informações relativas aos registos de cinemas.

Conforme descrito anteriormente, este tipo é um objecto que será activado remotamente por cada Cliente que pretenda ter acesso aos dados e, a sua interface é a seguinte:

|  |
| --- |
| Interface IBrokerRegistryServer  Interface responsável por definir as operações expostas pelo objecto remoto responsável por aceder e persistir dados relativos a registos de cinemas. |
| AddCinema(string name, string url)  Método que permite o registo de um cinema, recebendo o seu nome e endereço onde o *webservice* é disponibilizado. |
| RemoveCinema(string name)  Método que permite apagar o registo de um cinema. |
| GetCinemas() : Dictionary<string, string>  Método que retorna um dicionário com informação relativa a todos os nomes de cinemas registados, bem como, respectivos urls. |

Inicialmente considerou-se a criação de um tipo Singleton em que, por cada instância do *webservice*, se garantiria uma única instância do mesmo.

Assim, no mesmo processo do *webserver* existiria apenas uma instância para acesso ao repositório, contudo, esta solução apresentava desvantagens no caso de se querer por exemplo fazer um *scaling* horizontal ao sistema. Neste caso iriam haver várias instâncias Singleton em cada processador o que iria provocar anomalias no resultado das leituras e escritas.

**Repositório - Dados**

Os dados escolhidos para cada cinema foram o seu nome e o endereço de acesso ao seu *websevice* e, a forma escolhida para a persistência desses dados, foi em formato XML, de acordo com o seguinte exemplo:

<cinemas>

<cinema>

<name>BBC - BlockBusterCalifornia</name>

<url>http://localhost:3354/WSCinema.asmx</url>

</cinema>

</cinemas>

**Webservice**

A fim de especificar as operações disponibilizadas pelo *webservice* do Broker, definiu-se a interface IWSBroker com a seguinte estrutura:

|  |
| --- |
| Interface IWSBroker  Interface responsável por definir as operações expostas pelo webservice de Broker. |
| GetCinemas() : List<CinemaRegistry>  Método que devolve uma lista de objectos representativos de cinemas e respectivos *endpoints*. |
| RegisterCinema(string name, string url)  Método que permite o registo de um cinema, recebendo o seu nome e endereço onde o *webservice* é disponibilizado. |
| UnregisterCinema(string name)  Método que permite apagar o registo de um cinema |

O tipo genérico da colecção devolvida pelo método GetCinemas(), *CinemaRegistry*, não é mais do que um tipo que permite encapsular o nome do cinema e o seu url, ficando assim definido em WSDL:

<s:complexType name="CinemaRegistry">

<s:sequence>

<s:element minOccurs="0" maxOccurs="1" name="Name" type="s:string"/>

<s:element minOccurs="0" maxOccurs="1" name="Url" type="s:string"/>

</s:sequence>

</s:complexType>

**Webservice – Tratamento de Falhas**

O tratamento de falhas é feito com recurso a SoapExceptions. De entre os pontos onde podem surgir falhas, optou-se por dar atenção à ligação com o *RegistryServer*.

No caso de não ser possível a ligação ao mesmo, é lançada uma SoapException com a informação relativa ao erro.

|  |
| --- |
| catch (SocketException) |
| { |
| throw new SoapException("Cinema Registry Server down", |
| SoapException.ServerFaultCode, "BBBroker", |
| GetSoapExceptionDesc("No registration capabilities possible.")); |
| } |
| return ret; |

Do lado do Cliente que pretende inserir e remover registos de cinemas, terá que ser feita a detecção da excepção, invalidando assim a possibilidade de aceder às funcionalidades relativas a registos de serviços de Cinemas.

O Cliente final, que terá como objectivo comunicar com os *webservices* dos Cinemas, terá que detectar a excepção, invalidando assim a comunicação com qualquer cinema, devido à indiponibilidade do serviço de directório.

**Cinema *Webservice***

**Descrição**

O Cinema é o serviço que disponibilizará informação relativa a filmes em exibição, sessões e salas. Adicionalmente terá que suportar reservas de filmes e anulações de reservas.

Este deverá ter acesso a um repositório que disponibilize dados relativos a filmes, sessões e salas, sendo estes apenas de leitura (considerando a inexistência de uma aplicação que actualize esta informação) e dados relativos a reservas, sendo estes de leitura e escrita.

**Arquitectura**

A arquitectura escolhida para o processo relativo ao Cinema foi a seguinte:

**Repositório - Acesso**

O desenvolvimento inicial da solução levou-nos à implementação de uma solução que se baseava na criação de um tipo Singleton para acesso a dados, contudo, revelou as mesmas fragilidades que as indicadas no acesso ao repositório de registos de cinemas.

Uma vez que, na consulta de dados, estas fragilidades não constituem um problema, optou-se por manter o tipo Singleton no acesso ao repositório apenas para leitura, considerando para o exemplo que, a informação nele presente (filmes, sessões e salas) está persistida no mesmo suporte, acessível por todas as instâncias deste tipo e que, actualizações não serão feitas em alturas de disponibilidade do serviço.

No que respeita ao repositório com informação relativa a reservas, optou-se por seguir a estratégia adoptada anteriormente no acesso a dados de leitura e escrita, i.e., criar um objecto activado remotamente no modo Singleton.

**Repositório – Dados**

No que respeita aos dados relativos a filmes, sessões e salas optou-se por estrutura-los de acordo com o seguinte modelo.

Assim, para a classe CinemaModel, responsável pelo acesso aos dados indicados, foi definida a seguinte interface:

|  |
| --- |
| Interface ICinemaModel  Interface responsável por definir as operações expostas pelo tipo responsável por aceder aos dados de leitura relativos a filmes, sessões e salas. |
| GetMovies() : List<Movie>  Método que devolve a lista de filmes. |
| GetMovies(List<String> keywords) : List<Movie>  Método que devolve a lista de filmes que contém no seu titulo as palavras chave recebidas. |
| GetMovies(DateTime start, DateTime end) : List<Movie>  Método que devolve a lista de filmes que são exibidos em sessões que começam dentro do periodo recebido. |

Os dados escolhidos para as reservas foram o seu código, que a identificará inequivocamente, o nome do titular, o código da sessão e o número de lugares, ficando persistidos em formato XML, de acordo com o seguinte exemplo:

<contents

<reservation code="2eaab201-af12-451e-bfad-3c901b2fc9a3"

name="Zé Manel Pirata" sessionId="13H-R2" seats="20"/>

<reservation code="7e6ad399-5dee-4310-8c13-88ec740dd0ae"

name="Carlos Lopes" sessionId="13H-R2" seats="10"/>

</contents>

Sendo o acesso realizado por um tipo a ser activado remotamente definiu-se a seguinte interface:

|  |
| --- |
| Interface ICinemaReservationServer  Interface responsável por definir as operações expostas pelo objecto remoto responsável por aceder e persistir dados relativos a reservas. |
| AddReservation(string name, string sessionId, int seats) : Guid  Método que permite a inserção de uma reserva, retornado o se o registo de um cinema, recebendo o seu nome e endereço onde o *webservice* é disponibilizado. Caso a reserva não tenha sido efectuada então é retornado um Guid vazio. |
| RemoveReservation(Guid code) : bool  Método que permite apagar uma reserva. |
| GetTotalReservations(String sessionId) : int  Método que devolve o número de reservas inseridas para o código de sessão recebido. |

**Webservice**

A fim de especificar as operações disponibilizadas pelo *webservice* do Broker, definiu-se a interface IWSCinema com a seguinte estrutura:

|  |
| --- |
| Interface IWSCinema  Interface responsável por definir as operações expostas pelo webservice de Cinema. |
| GetMovies() : List<Movie>  Método que devolve a lista de filmes. |
| GetMoviesByTitle(List<string> keywords) : List<Movie>  Método que devolve a lista de filmes que contém no seu titulo as palavras chave recebidas. |
| GetMoviesByPeriod(DateTime start, DateTime end) : List<Movie>  Método que devolve a lista de filmes que são exibidos em sessões que começam dentro do periodo recebido. |
| AddReservation(String name, String sessionId, int seats) : Guid  Método que permite adicionar uma reserva. |
| RemoveReservation(Guid code) : bool  Método que permite remover uma reserva. |

Toda a lógica de negócio inerente à gestão de reservas fica centralizada no *webservice* do Cinema. Este, ao receber um pedido de inserção de reserva, verificará junto do *ReservationServer* qual a o número de reservas para determinada sessão e, junto do *CinemaModel* confirmará a capacidade da sala relativa ao código de sessão indicado. Assim, confirmará a possibilidade de acrescentar uma nova reserva para o número de lugares recebidos.

**Webservice – Tratamento de Falhas**

O tratamento de falhas é feito com recurso a SoapExceptions. De entre os pontos onde podem surgir falhas, optou-se por dar atenção à ligação com o *ReservationServer*.

No caso de não ser possível a ligação ao mesmo, é lançada uma SoapException com a informação relativa ao erro.

O Cliente final deverá detectar a excepção no momento em que tenta fazer operações relativas a reservas, invalidando este tipo de operações, contudo, podendo continuar a comunicar com o *webservice* para obtenção de informações provenientes de *CinemaModel*.

**Aplicação Cliente - WinForms**

No sentido de retirar toda a lógica associada à comunicação com os *webservices* do form da aplicação, foi criada a interface ICoordinator, implementada pela classe Coordinator, que tem a seguinte estrutura:

|  |
| --- |
| Class Coordinator  Componente responsável por comunicar de forma assíncrona com os webservices de Broker e Cinema. |
| GetMovies()  Método que obtém de forma assíncrona a lista de filmes do Cinema. |
| GetMoviesByTitle(string keywords)  Método que obtém de forma assíncrona a lista de filmes do Cinema que contêm no seu titulo as palavras chave recebidas. |
| GetMoviesByPeriod(DateTime start, DateTime end)  Método que obtém de forma assíncrona a lista de filmes do Cinema que serão exibidos em sessões que começam dentro do periodo recebido. |
| SendReservation(SessionInfo resInfo)  Método que envia de forma assíncrona um pedido de inserção de reserva para determinada sessão. |
| RemoveReservation(Guid code) : bool  Método que envia de forma assíncrona um pedido de remoção de reserva para determinada sessão. |

A fim de poder notificar interessados no fim de uma operação assíncrona, este componente expõe os seguinte eventos:

|  |
| --- |
| Class Coordinator  Descrição dos eventos expostos |
| ReceivedMovieListDelegate MoviesReceived  Este evento notificará os interessados de que o pedido de uma lista de filmes foi terminado.  A assinatura exigida ao método interessado deverá ser a definida pelo seguinte delegate:  *void ReceivedMovieListDelegate(string cinemaName, List<Movie> movies)* |
| ErrorOccuredDelegate ErrorOccured  Este evento notificará os interessados de que ocorreu um erro.  A assinatura exigida ao método interessado deverá ser a definida pelo seguinte delegate:  *void ErrorOccuredDelegate(string message)* |
| ReservationProcessedDelegate AddReservationProcessed  Este evento notificará os interessados de que o processo de inserção de uma reserva terminou.  A assinatura exigida ao método interessado deverá ser a definida pelo seguinte delegate:  *void ReservationProcessedDelegate(SessionInfo resInfo)* |
| ReservationProcessedDelegate RemoveReservationProcessed  Este evento notificará os interessados de que o processo de remoção de uma reserva terminou.  A assinatura exigida ao método interessado deverá ser a definida pelo delegate em cima detalhado. |

Assim, as aplicações que pretendam tirar partido do tipo Coordinator deverão registar-se nos eventos por este expostos, que é o caso do form da aplicação implementada.

O tipo SessionInfo, utilizado nas operações de reservas, que apresenta a estrutura em baixo apresentada, visa diminuir os pedidos de obtenção de informação relativa a determinada sessão, ficando, na implementação proposta, associado à propriedade *Tag* do nó de sessão correspondente.

|  |
| --- |
| public class SessionInfo |
| { |
| public string SessionID { get; set; } |
| public DateTime StartTime { get; set; } |
| public string Cinema { get; set; } |
| public string MovieTitle { get; set; } |
| public string Name { get; set; } |
| public Guid Code { get; set; } |
| public int Seats { get; set; } |
| } |

**Instruções para Utilização**

No sentido de se utilizar a aplicação Cliente, disponível no projecto *ClientFormsApplication*,foi incluido o ficheiro “*BlockBuster Example.rar”* que se apresenta dividido da seguinte forma:

|  |  |
| --- | --- |
| **BBBroker** | *Webservice* do Broker |
| **BBBrokerRegistryServer** | Servidor de registos de cinemas |
|  |  |
| **BBCalifornia** | *Webservice* de exemplo #1 |
| **BBCReservationServer** | Servidor de reservas #1 |
|  |  |
| **BBSidney** | *Webservice* de exemplo #2 |
| **BBSReservationServer** | Servidor de reservas #1 |
|  |  |
| **BBZambujeira** | *Webservice* de exemplo #3 |
| **BBZReservationServer** | Servidor de reservas #1 |

As configurações de remoting do servidor de registo, bem como, dos servidores de reservas já estão parametrizadas no *web.config* de cada *webservice*.

Os *endpoints* que ficaram estabelecidos para os *webservices* são os seguintes, podendo os mesmos serem re-configurados:

<http://localhost/BBBroker/WSBroker.asmx>

<http://localhost/BBCalifornia/WSBroker.asmx>

<http://localhost/BBSidney/WSBroker.asmx>

<http://localhost/BBZambujeira/WSBroker.asmx>