Instituto Superior de Engenharia de Lisboa

Sistemas Distribuídos

Relatório da terceira série

Autoria:
32632 Pedro Pedroso
33724 David Raposo
33404 Ricardo Mata

6 de Julho de 2014

Conteúdo

1 Introdução

Neste terceiro trabalho da disciplina de Sistemas Distribuídos, foi requerido que se implementasse as diferentes componentes de um sistema de pesquisa de automóveis e respetiva reserva.

Este sistema é composto pelos seguintes intervenientes:

- Cliente: Aplicação gráfica onde serão feitas as pesquisas e respetivos pedidos de reserva.
- Stands: Aplicações que contêm um conjunto de carros, onde incidem as pesquisas e reservas dos clientes. Mantém dados persistentes usando ficheiros *XML* para obter/guardar a informação necessária.
- Broker: Aplicação que o cliente comunicará para fazer a sua pesquisa e onde os *stands* se podem registar para mais tarde ser propagada as pesquisas dos clientes.

2 Arquitetura

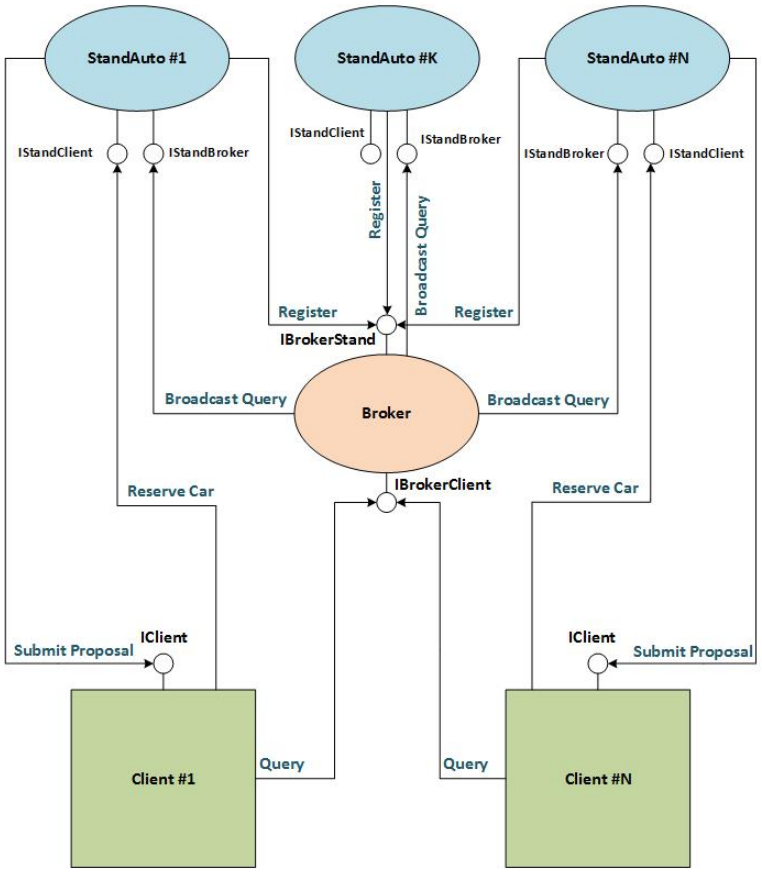

Figura 1: Arquitetura

As diversas aplicações foram desenvolvidas em *c#*. Na componente gráfica do cliente foi usado *Windows Forms*, enquanto para a componente *web* das diversas aplicações foi usado *WCF*.

Para o desenvolvimento dos serviços, existem restrições em relação aos *bindings* que podem ser usados (*Binding wsHttpBinding* e *basicHttpBinding*). Tendo em conta que não foi possível usar *bindings* com suporte para *callbacks*, optou-se por passar os endereços durante as diversas comunicações, para executar posteriormente a entrega/aceitação de propostas.

3 Componentes

3.1 *Broker*

Componente intermédio para fazer *broadcast* ás pesquisas dos clientes para os stands e para registar os mesmos.

3.1.1 Serviços disponibilizados

- *IBrokerClient:* Serviço que permite aos clientes submeterem as suas pesquisas ao *Broker*.
  - Modo de instanciação: Singleton
    Visto não haver alterações de objetos neste serviço e admitindo que haverá vários clientes a fazer várias pesquisas então não seria razoável criar um objeto por chamada. Um só objeto consegue lidar com os pedidos de forma eficaz.
  - Modo de controlo de concorrência: Multiple
    Ao receber vários pedidos será razoável que o processamento desses pedidos seja *multi-thread*. Visto não haver alterações nenhumas em variáveis no *Singleton* então não é necessário sincronismo adicional por parte do *Broker*.
  - Binding utilizado: basicHttpBinding
    Visto a informação enviada para o *Broker* não ter dados importantes, é utilizado *basicHttpBinding* para simplificar a mensagem enviada. Visto também que o cliente não obtém *feedback* nesta comunicação, os métodos estão declarados como *IsOneWay*. Esta opção faz com que o *binding* seja imediatamente libertado após a receção da mensagem por parte do *Broker* e liberta este *binding* para ser utilizado pelos stands para submissão de propostas.
- *IBrokerStand:* Serviço que permite que um *stand* se registe no *Broker* para receber as pesquisas dos clientes.
  - Modo de instanciação: Singleton
    Devido a existência de dois serviços, foi necessário usar *partial classes*, logo, o modo de instanciação teria de ser igual a *IBrokerClientService*.
  - Modo de controlo de concorrência: Multiple
    Com a possibilidade de vários stands se registarem ao mesmo tempo, utilizamos um modo de concorrência *Multiple*. Isto cria problemas em relação ao sincronismo do objeto que está a ser usado para guardar o registo dos *stands*, em que se optou por usar mecanismos de *lock* que, visto o registo de *stands* ser algo que ocorrerá ocasionalmente, não vai afetar a *performance* por ser um mecanismo bloqueante.
  - Binding utilizado: wsHttpBinding
    Binding utilizado como requisito não funcional.

3.2 *Client*

Componente que faz a submissão das propostas e reserva dos carros. O cliente tem uma interface gráfica para melhor ajudar a visualização das propostas e submissão das várias pesquisas.

3.2.1 Serviços disponibilizados

- *IClient:* Serviço que permite aos stands submeteram as propostas aos clientes.
  - Modo de instanciação: PerCall
    Um cliente só receberá propostas depois de efetuar uma pesquisa. Isto faz com que usar um objeto *Singleton* não seja razoável no sentido que não há necessidade de manter um objeto de serviço alocado quando não se espera mais propostas. Por esta razão foi escolhido o método de instanciação *PerCall* que cria um objeto de serviço que tem a duração da submissão da proposta.
  - Modo de controlo de concorrência: Multiple
    Para resolver os problemas de concorrência causados pelo controlo *multiple* utilizamos o método *beginInvoke* para obrigar a que as alterações à *listbox* sejam feitas sempre na *UI thread*.
  - Binding utilizado: basicHttpBinding
    Binding utilizado como requisito não funcional.

3.3 *StandAuto*

Componente que mantém persistente dados de carros e submete as propostas aos clientes com base nas pesquisas feitas.

3.3.1 Serviços disponibilizados

- *IStandBroker:* Serviço que permite ao *Broker* propagar a pesquisa aos *stands*.
  - Modo de instanciação: Singleton
    Admitindo que este serviço irá estar frequentemente a receber pedidos vindos do *Broker* é razoável a utilização de um objeto de serviço *Singleton*.
  - Modo de controlo de concorrência: Multiple
    Vários pedidos implicam um ambiente *multi-thread* portanto é usado o controlo de concorrência *multiple*. Não é alterado nenhum objeto no *stand* portanto não é necessário controlo de concorrência adicional.
  - Binding utilizado: wsHttpBinding
    Binding utilizado como requisito não funcional.
- *IStandClient:* Serviço que permite aos clientes fazerem as suas reservas de carros.

- Modo de instanciação: Singleton
  Admitindo que este serviço irá estar frequentemente a receber reservas vindos dos clientes é razoável a utilização de um objeto de serviço *Singleton*.
- Modo de controlo de concorrência: Multiple
  Várias reservas implicam um ambiente *multi-thread* portanto é usado o controlo de concorrência *multiple*. Visto haver mudança em variáveis (alterar o estado do carro para reservado) foi necessário implementar um mecanismo adicional de concorrência. Por isso utilizamos o mecanismo *Interlocked.CompareExchange* para fazer a mudança de forma *thread-safe*.
- Binding utilizado: wsHttpBinding
  Binding utilizado como requisito não funcional.

3.3.2 Persistência de dados

Um dos requisitos da aplicação, era que os carros que cada *stand* disponibiliza, fossem persistentes através do uso de ficheiros *.xml*.

Como tal, definiu-se que cada carro teria a seguintes estrutura:

```
<car>
   <id> identificador dentro do stand em concreto</id>
   <price> preço </price>
   <brand> marca </brand>
   <yearRegistration> ano do registo de
matricula</yearRegistration>
   <isAvailable> Se está reservado de momento</isAvailable>
</car>
```

Figura 2: Estrutura de cada entrada

Não foi adicionado nenhum mecanismo de verificação da validade destes ficheiros (*.xsd*), sendo apenas possível passar de não reservado para reservado. Qualquer outra alteração terá de ser feita manualmente sobre estes ficheiros. Em relação a problemas gerados devido a concorrência, optou-se por assegurar apenas que não é possível múltiplas reservas no mesmo carro, informando o cliente desta situação.

4 Pesquisas

Foram implementadas três tipos de pesquisas que o cliente pode usar para procurar pelos carros.

- Por marca
  Retorna todos os carros da marca escolhida.
- Por ano
  Retorna todos os carros que tenham data de criação igual ou superior à data escolhida.
- Por preço
  Retorna todos os carros que tenham um preço menos ou igual ao preço escolhido.

5 Tratamento de erros

Para lidar com erros que poderão eventualmente acontecer utilizamos *FaultContracts*. No momento da reserva de um carro, se o carro já estiver reservado então lançamos uma exceção nossa *AlreadyReservedFault* para dizer ao cliente que o carro já foi reservado sem ser necessário a criação de mais um *binding* de comunicação.

6 Ficheiros de configuração

Cada componente está suportado por um ficheiro de configuração (*app.config*) onde são definidos os *endpoints* dos serviços disponibilizados assim como os *endpoints* dos serviços dos outros componentes assim como todas as configuração associadas aos mesmos (tipos de *binding*, contractos, etc). Assim é fácil mudar os *endpoints* sem ser necessário o compilador.