MAC 440/5759 - Sistemas de Objetos Distribuídos - Primeiro Semestre de 2006

EP 3: Serviço de Fachada Baseado na Tecnologia Web Services

Agradecimento: ao Ivan Neto, que escreveu a quase totalidade deste texto.

1 Introdução

Este exercício-programa é uma continuação do trabalho que vocês fizeram nos EPs 1 e 2. Naqueles EPs vocês construíram uma aplicação servidora, voltada para vídeo-locadoras, que oferece sua funcionalidade através de um serviço de fachada CORBA. Nesta etapa do trabalho, vocês criarão um novo serviço de fachada para o mesmo sistema, agora tornando-o acessível como um web service. A idéia é partir do sistema construído nos EPs 1 e 2 e adicionar a ele novas classes, que implementarão a fachada web service.

Como web services implementados em Java rodam dentro de servidores de aplicações (ou, mais precisamente, dentro de servidores web com suporte para servlets[1]), sua fachada web service terá de rodar dessa forma. Por outro lado, queremos que a fachada web service interaja com o restante do sistema (os objetos do EP2) através de chamadas locais. Portanto o sistema do EP2 e a fachada web service deverão ser executados pelo mesmo processo, o servidor de aplicações. Sua primeira tarefa será alterar o EP2 para que ele rode como um módulo implantado num servidor de aplicações. As alterações serão mínimas. Todos os objetos do EP2, incluindo o serviço de fachada CORBA, continuarão a existir dentro do servidor de aplicações. Depois que vocês tiverem o EP2 rodando dentro de um servidor de aplicações, poderão atacar o problema real desta etapa, que é a adição da fachada web service.

2 O que são web services?

Web services é o nome dado à tecnologia que permite a comunicação entre aplicações de uma maneira independente de sistema operacional e de linguagem de programação. Como vocês já sabem, CORBA também possibilita interoperabilidade em ambientes distribuídos heterogêneos. Por esse motivo, alguns acusam web services de ser uma "reinvenção da roda". Entretanto, web services possuem algumas características que os tornam bastante atraentes:

- *Uso intenso de XML*: XML é uma linguagem para representação de dados que é extensível e naturalmente independente de plataforma, além de ser amplamente utilizada pela indústria. Em web services, tando a descrição de um serviço quanto a comunicação entre serviços é feita usando XML.
- *Grande apoio da indústria*: web services têm recebido grande apoio da indústria, particularmente por parte da IBM e da Microsoft. Recentemente, web services se tornatam parte das plataformas J2EE (Java Enterprise Edition, da Sun) e .NET (Microsoft). É possível, por exemplo, que um web service implantado num servidor de aplicações J2EE acesse outro web service implantado numa plataforma .NET.
- *Baseado em padrões abertos*: web services são baseados em uma série de padrões abertos e amplamente difundidos, tais como XML, HTTP, SOAP, WSDL e UDDI. Isso assegura que implementações compatíveis com as especificações sejam interoperáveis.
- *Amigável a firewalls*: as mensagens trocadas entre web services tipicamente usam HTTP como protocolo de transporte, o que em boa parte dos casos evita problemas com firewalls.
- *Uso de URIs para identificação*: os web services são identificados por uma URI (que tipicamente é uma URL), um formato muito utilizado devido à popularização da web e de fácil assimilação pelos humanos.

Estas são apenas algumas das características de web services que tipicamente são mencionadas como positivas.

[1] Isto não é uma necessidade absoluta e sim uma conseqüência da arquitetura (baseada em servlets) que é empregada pelas infraestruturas de web services atualmente disponíveis para Java.

3 Definição do serviço de vídeo-locação

O serviço de vídeo-locação é definido utilizando a Web Service Definition Language (WSDL). O papel da WSDL em web services é análogo ao papel da IDL em CORBA. Para maiores detalhes, veja o arquivo `VideoRental.wsdl`. (Note que a descrição do web service é um documento XML.) Ele define o web service de fachada, análogo à fachada CORBA (interface IDL `RentalService`) dos EPs 1 e 2.

Como veremos em classe, há dois estilos de interação com um web service: RPC e troca de documentos. O arquivo `VideoRental.wsdl` define duas fachadas, uma para interação via RPC (`RentalServiceRPC`) e outra para interação por troca de documentos (`RentalServiceDoc`). Isso não é uma coisa típica, pois quem oferece um web service normalmente escolhe um dos estilos de interação. No arquivo WSDL deste EP temos os dois tipos de fachada apenas para vocês tomarem contato com os dois estilos de interação.

4 Obtenção e instalação de um servidor de aplicações J2EE

Este trabalho será desenvolvido utilizando um servidor de aplicações J2EE open-source, o JBoss, que pode ser obtido em `http://www.jboss.org`. No site do JBoss há uma série de produtos para download. O que vocês devem baixar para esse EP é a última versão (4.0.4) do "J2EE application server". Baixem o `jboss-4.0.4.GA-installer.jar`, pois a outra alternativa de download (arquivo `.zip`) exige algumas configurações manuais para ser usada neste EP. Baixando o instalador, vocês não precisarão mexer na configuração do servidor de aplicações.

Para instalar o JBoss, basta executar

```
java -jar jboss-4.0.4.GA-installer.jar
```

e seguir as instruções. Escolha o perfil "default".

Para iniciar o servidor de aplicações, basta executar o script

```
$JBOSS_HOME/bin/run.[sh|bat]
```

5 Fazendo o EP de CORBA rodar dentro do servidor de aplicações

O EP de vocês precisa de duas modificações para rodar dentro do JBoss:

- Troquem a chamada a `ORB.init(String[] args, Properties props)` por

  ```
  (ORB) new javax.naming.InitialContext().lookup("java:/JBossCorbaORB");
  ```

- O código que inicializa o servidor de vocês (que estava no main) deve ir para o método `startService()` de um MBean. A chamada a `ORB.run()` deve ser removida desse código.

Um MBean é um componente plugável do JBoss. Vocês não precisam entender perfeitamente o que é um MBean, basta seguir esta "receita de bolo" que o EP de vocês deve funcionar dentro do JBoss. Criem uma interface (vazia mesmo):

```
public interface VideoRentalStoreServiceMBean extends org.jboss.system.ServiceMBean
{
}
```

Embora vocês possam alterar o nome desta interface, ela tem que ter o sufixo "MBean". Escrevam uma classe que implementa essa interface (deve ter o mesmo nome da interface, mas sem o sufixo "MBean"):

```
public class VideoRentalStoreService extends org.jboss.system.ServiceMBeanSupport
    implements VideoRentalStoreServiceMBean
{
   protected void startService() throws Exception
   {
```

```
        // O código que inicia o servidor vem aqui (sem ORB.run()).
    }

    protected void stopService()
    {
        // Se alguma coisa for executada antes de parar o servidor, coloque a aqui.
    }
}
```

Dica: para compilar estas classes vocês irão precisar do arquivo `$JBOSS_HOME/lib/jboss-system.jar` no classpath. Vocês precisam também criar um arquivo chamado `jboss-service.xml`, com o seguinte conteúdo:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <mbean code="videorentalstore.corba.impl.VideoRentalStoreService"
         name="jboss.ws:service=VideoRentalStoreService">
    <depends>jboss:service=CorbaNaming</depends>
    <depends>jboss:service=TransactionManager</depends>
  </mbean>
</server>
```

Troquem a classe em `code` pela classe (não a interface) do seu MBean. Agora basta empacotar tudo num .sar (service archive), que nada mais é do que um JAR com a extensão .sar. Apenas como sugestão, seu .sar pode ser organizado do seguinte modo:

```
videorental.sar
 |-- videorental-corba.jar            (classes do EP de CORBA + MBean)
 |-- META-INF
      |-- jboss-service.xml
```

Uma dica: usem o ant para automatizar o empacotamento. Para implantar o .sar basta copiá-lo para o diretório de implantação do JBoss. Na configuração default do servidor de aplicações, esse diretório é `$JBOSS_HOME/server/default/deploy`. Vocês podem fazer isso com o servidor JBoss rodando (*hot-deployment*) ou não. Após a implantação, o seu EP de CORBA rodará dentro do servidor de aplicações e suas classes/objetos ficarão localmente acessíveis para um web service que também rode dentro do servidor.

6 Geração de artefatos Java e XML a partir do arquivo WSDL

Assim como em CORBA vocês geraram artefatos Java a partir do arquivo IDL, em web services vocês também terão que gerar alguns artefatos Java e XML, só que a partir do arquivo WSDL.

No caso do JacORB você usaram um compilador IDL chamado `idl`. O JBoss vem com um "compilador" WSDL chamado `wstools`. Infelizmente o `wstools` possui um bug que impossibilita o seu uso neste EP. Portanto, vocês terão que baixar outro "compilador" WSDL.

6.1 Obtenção do Java Web Services Developer Pack

O Java Web Services Developer Pack (JWSDP) é um pacote da Sun para desenvolvimento de web services. Nó só queremos o "compilador" WSDL do JWSDP, mas infelizmente temos que baixar todo o pacote. Faça o download em

`http://java.sun.com/webservices/downloads/webservicespack.html`

Quase todas as etapas da instalação são triviais. Quando perguntado sobre qual web container utilizar, escolha "No web container". Isso porque nós só queremos o "compilador" WSDL, então não vale a pena perder tempo configurando um web container.

6.2 Geração de código Java e do arquivo de mapeamento a partir do WSDL

Após instalar o JWSDP, entre no seguinte diretório:

```
$JWSDP_HOME/jaxrpc/bin
```

É aí que está o "compilador" WSDL, que é chamado de `wscompile` (a ferramenta é um script, isto é, um `.sh` ou um `.bat`, dependendo do seu sistema operacional). A sintaxe da linha de comando do `wscompile` é a seguinte:

```
wscompile [options] configuration_file
```

Como é possível ver, o parâmetro para o `wscompile` não é um arquivo WSDL, mas sim um arquivo de configuração. Isso ocorre porque o `wscompile` faz mais coisas além de "compilar" WSDLs. Como no nosso caso nós só queremos "compilar" o WDSL, nosso arquivo de configuração será:

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
    <wsdl location="VideoRental.wsdl" packageName="videorentalstore.webservice.gen"/>
</configuration>
```

O arquivo de configuração é bem simples. O atributo `location` deve indicar a localização do arquivo WSDL. Já o atributo `packageName` indica qual o nome do pacote Java em que as classes Java serão geradas. Para gerar os artefatos a partir do WSDL, basta executar:

```
$ mkdir output
$ ./wscompile.sh -import -f:norpcstructures,wsi -mapping output/jaxrpc-mapping.xml
    -d output -s output -keep config.xml
```

Vocês não precisam entender todos os parâmetros passados ao `wscompile`. Se vocês olharem na documentação, vão ver que o `wscompile` aceita uma grande quantidade de parâmetros. Vou explicar os parâmetros que usei acima de maneira sucinta (isso é apenas para os curiosos, vocês não precisam entender isso para fazer o EP).

A primeira coisa feita foi criar um diretório `output`, onde os artefatos gerados serão colocados (o `wscompile` não cria o diretório automaticamente). O primeiro parâmetro (`-import`) é usado para indicar ao `wscompile` que ele deve "importar" um arquivo WSDL. O segundo parâmetro (`-f:norpcstructures,wsi`) habilita as features `norpcstructures` e `wsi`. O parâmetro `-mapping` define o arquivo de mapeamento entre WSDL e Java (este arquivo será gerado pelo `wscompile`). Os parâmetros `-d` e `-s` definem onde serão gerados os arquivos `.class` e os fontes, respectivamente. Por fim, o `-keep` diz ao `wscompile` para não apagar os fontes gerados.

Depois de rodar o comando acima, serão geradas classes e interfaces Java, além de um arquivo de mapeamento chamado `jaxrpc-mapping.xml`. Tudo isto estará no diretório `output`. Se vocês olharem para o arquivo `jaxrpc-mapping.xml`, verão que ele relaciona partes do WSDL com entidades Java. A partir deste ponto você não precisam mais do JWSDP.

7 A implementação do web service

Vocês terão que escrever classes que implementam as interfaces `RentalServiceRPC` e `RentalServiceDoc` geradas pelo `wscompile`. Estas classes apenas delegarão chamadas para a implementação CORBA de vocês. Notem que como o servidor CORBA e o web service vão estar no mesmo servidor de aplicações, não é necessário que o web service faça chamadas CORBA. O ideal é que o web service chame diretamente os serventes (vocês podem adicionar métodos aos serventes).

8 Empacotando o web service

O seu web service deve ser empacotado num .war (web archive). Para maiores detalhes sobre como fazer isso, vejam o capítulo 12 do Admin Guide do JBoss. Vocês também devem criar um .jar com descritores de implantação para clientes J2EE (application-client.xml e jboss-client.xml). Para maiores detalhes vejam:

`http://wiki.jboss.org/wiki/Wiki.jsp?page=WS4EEDOCClientStepByStep`

É importante ressaltar que vocês não vão usar Dynamic Invocation Interface (DII) do lado cliente. O código cliente de vocês deve ser parecido com o da seção 12.3.2 do Admin Guide do JBoss, isto é, vocês devem obter um proxy registrado no JNDI. No caso do Admin Guide, o cliente do web service é um EJB. No caso de vocês, vai ser uma simples aplicação Java. Portanto, ao invés de declarar `service-ref`s nos arquivos `ejb-jar.xml` e `jboss.xml`, vocês vão ter que declará-las nos arquivos `application-client.xml` e `jboss-client.xml`.

9 Tarefas

Seu trabalho consiste em:

- Escrever um web service que implemente a fachada definida no arquivo VideoRental.wsdl (isto é, tanto a fachada RPC/literal quanto a document/literal).
- Escrever um cliente que chame as operações do web service. Seu cliente pode ser interativo ou pode ler de um script (ou os dois), mas ele deve permitir a chamada das operações da fachada, e mostrar os resultados obtidos com estas chamadas. Seu cliente deve de algum modo permitir escolher se a interação com o servidor será feita usando o estilo document/literal ou RPC/literal.
- Examinar o arquivo `$JBOSS_HOME/server/default/log/server.log`, após a execução de operações sobre as duas fachadas (RPC/literal e document/literal). Enre muitas outras coisas, esse arquivo conterá as mensagens SOAP trocadas entre o cliente e o servidor. Observe a diferença entre as mensagens RPC/literal e document/literal.
- **Bônus:** Medir os tempos de execução de chamadas remotas
  - do cliente para a fachada CORBA;
  - do cliente para a fachada web service.

  Em que caso as chamadas remotas são mais rápidas? Tente justificar.

10 Requisitos

- Este trabalho deve ser feito em equipes de até duas pessoas.
- Caso sua equipe tenha mudado, a nova equipe deve se registrar com o professor da disciplina até o dia 01 de junho, através de uma mensagem informando os nomes dos integrantes da equipe. A mensagem deve ser enviada para o email do Prof. Reverbel e deve ter *subject* "registro de equipe de SOD".
- Linguagens de programação aceitas: Java.
- Servidor de aplicações que vocês devem utilizar: JBoss.
- Você deverá entregar um arquivo tar.gz contendo os seguintes ítens:
  - um arquivo tipo txt, pdf ou ps com o seu relatório;
  - os arquivos-fonte em Java;
  - um arquivo `build.xml`, para geração automatizada dos programas Java pelo utilitário `ant` (seu arquivo também deve possibilitar a geração dos arquivos .war, .jar e .sar);

- um arquivo `README` descrevendo o brevemente o conteúdo e a estrutura de subdiretórios do arquivo tar.gz entregue e contendo quaisquer informações adicionais que sejam importantes para gerar e executar o servidor e o(s) cliente(s).

O desempacotamento do seu arquivo tar.gz deverá produzir um <u>diretório</u> contendo esses ítens. O diretório deve ter nome da forma ep3-membros-da-equipe (exemplo: ep3-joao-maria).

11 Dúvidas

Muitas dúvidas devem surgir durante o desenvolvimento deste trabalho. Para esclarecê-las, recorra à lista de discussão da disciplina.
Bom trabalho!