Web Services:
Metodologias de Desenvolvimento

Carlos J. Feijó Lopes José Carlos Ramalho

Fevereiro de 2003

Resumo

Temos de fazer o abstract

1 Contextualização

Os *Web Services* são uma tecnologia emergente, sobre a qual muito se tem especulado. Uns apontam-na como o caminho a seguir no desenvolvimento de aplicações distribuídas, enquanto que outros vêm nelas apenas mais uma evolução de um conceito antigo.

Sendo aplicações modulares, auto-descritivas, acessíveis através de um URL, independentes das plataformas de desenvolvimento e que permitem a interacção entre aplicações sem intervenção humana, os *Web Services* apresentam-se como a solução para os actuais problemas de integração de aplicações.

Estas suas características devem-se em grande parte ao facto de se basearem em protocolos *standard*, de entre os quais se destacam: XML, SOAP, WSDL e UDDI.

Segue-se uma breve descrição das funcionalidades de cada um dos protocolos referidos:

XML - tecnologia que serve de base a todos os protocolos utilizados nos *Web Services*.

SOAP - protocolo de comunicação, responsável pela troca de mensagens de e para os *Web Services*.

WSDL - protocolo responsável pela descrição de toda a API do *Web Service*.

UDDI - trata-se de um registador/localizador de *Web Services*, responsável pela procura, divulgação dos *Web Servides*, assim como pela forma como uma aplicação cliente invoca e interage com um *Web Service*.

2 Arquitectura de funcionamento de um *Web Service*

O funcionamento de qualquer *Web Service* compreende quatro etapas distintas: `Publicação`, `Descoberta`, `Descrição` e `Invocação`. Vejamos com mais pormenor cada uma delas:

`Publicação`: Processo, opcional, através do qual o fornecedor do *Web Service* dá a conhecer a existência do seu serviço, efectuando o registo do mesmo no reservatório de *Web Services* (UDDI).

`Descoberta`: Processo, opcional, através do qual uma aplicação cliente toma conhecimento da existência do *Web Service* pretendido.

`Descrição`: A aplicação cliente tem acesso a toda a interface do *Web Service*, onde se encontram descritas todas as funcionalidades por ele disponibilizadas, assim como os tipos de mensagens que permitem aceder às ditas funcionalidades.

`Invocação`: Processo pelo qual cliente e servidor interagem, através do envio de mensagens de *input* e de eventual recepção de mensagem de *output*.

A cada uma destas etapas corresponde um dos protocolos anteriormente referidos (figura 1).

3 Ciclo de vida de um *Web Service*

Um *Web Service* apresenta um ciclo de vida característico que passamos a descrever:

- O fornecedor constrói o serviço utilizando a linguagem de programação que entender;
- De seguida, especifica a interface/assinatura do serviço que definiu em WSDL;

Figura 1: **Tecnologias associadas aos *Web Services***

- Após a conclusão dos dois primeiros passos, o fornecedor regista o serviço no UDDI;
- O utilizador (aplicação cliente), encontra o serviço procurando no UDDI;
- A aplicação cliente estabelece a ligação com o *Web Service* e estabelece um diálogo com este, via mensagens SOAP.

4 Metodologias de desenvolvimento

Suponhamos que pretendemos desenvolver um *Web Service*, muito simples, capaz de gerar um nº aleatório entre dois valores limite indicados pela aplicação cliente. Com o intuito de permitir comparações, e posterior geração de metodologias, este serviço, e respectiva apliação cliente, será desenvolvido em quatro plataformas distintas: o módulo SOAP-Lite para Perl, o NuSOAP para PHP, o WASP Server para Java e a .Net da Microsoft.

Como iremos ver no decorrer desta secção, o desenvolvimento de *Web Services* é distinto em cada uma das quatro plataformas escolhidas, pelo que apresentar-se-ão de seguida as principais diferenças detectadas, quer na criação do serviço propriamente dito, quer na criação da respectiva aplicação cliente.

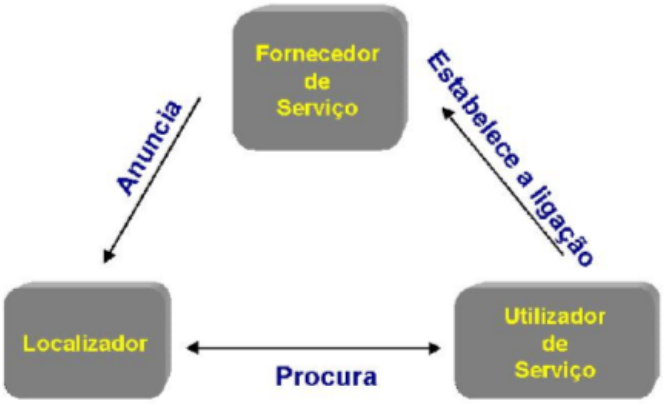

Figura 2: **Ciclo de vida de um *Web Service***

4.1 O Servidor

O primeiro passo no desenvolvimento de qualquer *Web Service* consiste na criação de um servidor, ou seja, do serviço propriamente dito.

Ao falarmos na criação de um servidor, estamos a contemplar duas ideias fundamentais: a implementação das funcionalidades pretendidas e a criação duma instância de um servidor SOAP com o qual pretendemos interagir.

É aqui que começam a surgir as primeiras diferenças entre as plataformas escolhidas.

Se por um lado temos o caso das plataformas NuSOAP e .Net, onde esta distinção é relativamente ténue (ver exemplos 1 e 2), para SOAP-Lite e WASP Server a distinção é bastante notória.

Exemplo 1: Servidor - PHP

```
1 <?php
2
3 require_once('nusoap.php');
4
5 $s = new soap_server;
6 $s->register('geraNumAleatorio');
```

```
7
8  function geraNumAleatorio($min, $max) {
9      srand((double)microtime()*1000000);
10     $numAleatorio = rand($min, $max);
11     return $numAleatorio;
12 }
13
14 $s->service($GLOBALS["HTTP_RAW_POST_DATA"]);
15 ?>
```

Exemplo 2: Servidor - .Net

```
1  ...
2  namespace Aleatorios
3  {
4      [WebService(Name="Números Aleatórios",
5       Description="Web Service que gera um nº"+
6       " aleatório entre um valor mínimo e um valor
7       máximo dados pela aplicação cliente",
8        Namespace="urn:CarlosLopes")]
9      public class Aleatorios : System.Web.Services.WebService
10     {
11         ...
12         [WebMethod(MessageName="geraNumAleatorio",
13          Description="método que gera um nº aleatório de acordo"+
14          " com os valores mínimos e máximos indicados")]
15         public int geraNumAleatorio(int min, int max)
16         {
17             Random r = new Random();
18             return r.Next(min,max);
19         }
20         ...
21     }
22 }
```

Neste caso basta indicar que a classe é um *Web Service* através do atributo `WebService`, e colocar o atributo `WebMethod` nos métodos que passarão a constituir a API do serviço.

Tal como foi indicado anteriormente, nas restantes plataformas a situação é um pouco diferente, isto é, existe uma clara separação entre o que é a implementação das funcionalidades e a sua associação ao servidor SOAP.

No caso do Perl com SOAP-Lite, para construirmos um serviço é necessário antes de mais criar uma instância de um servidor SOAP (exemplo 3).

Exemplo 3: Instância de Servidor SOAP - Perl

```
1 #!c:/perl/bin/perl.exe -w
2 use SOAP::Transport::HTTP;
3 SOAP::Transport::HTTP::CGI
4     -> dispatch_to('aleatorio')
5     -> handle;
```

No elemento `dispatch_to` indicamos qual o módulo onde estão implementadas as funcionalidades que se pretendem disponibilizar.

Estando na posse do nosso servidor SOAP, basta agora criar um módulo com as funcionalidades pretendidas (exemplo 4).

Exemplo 4: Implementação das funcionalidades - Perl

```
1 #!c:/perl/bin/perl.exe -w
2 package aleatorio;
3 sub geraAleatorio
4 {
5     my ($self, $min, $max) = @_;
6     $random = int(rand($max-$min)) + $min;
7     return $random;
```

```
}
1;
```

No caso da plataforma WASP Server para Java, a situação apesar de manter esta noção de separação, é um pouco diferente do que se passou com o Perl. Neste caso, e em primeiro lugar, é necessário criar uma classe (aleatorio.java) com as funcionalidades pretendidas e só depois registar essa classe no servidor WASP, criando assim uma classe *proxy* que representará o *Web Service* (exemplo 5 e figura 3 respectivamente) .

Exemplo 5: Implementação das funcionalidades - WASP Server - aleatorio.java

---

```
package aleatorio;
...
public class aleatorio {
    ...

    // métodos a disponibilizar remotamente
    public int geraAleatorio(int minimo, int maximo)
    {
        Random rand = new Random();
        return (rand.nextInt(maximo-minimo) + minimo);
    }
}
```

E estas são as grandes diferenças entre as plataformas quanto à criação do *Web Service* propriamente dito.

4.2 Aplicação cliente

Quando se fala na construção de uma aplicação cliente para um determinado *Web Service*, é comum confundir dois conceitos distintos: a interface de

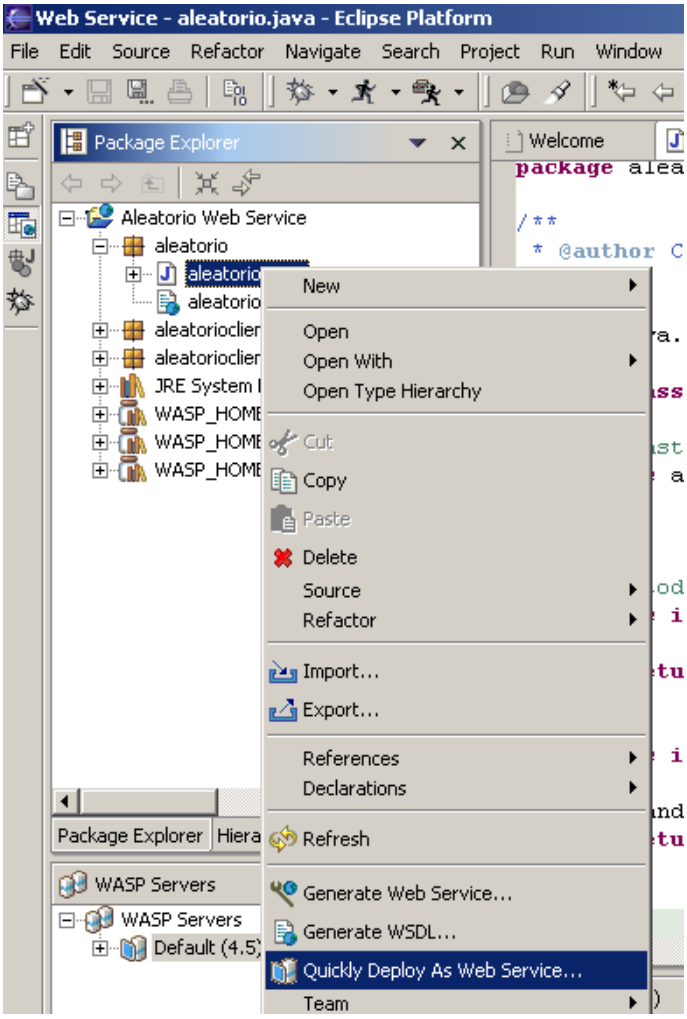

Figura 3: **Criação de um *proxy* a partir da classe em Java**

interacção com o utilizador, e uma outra classe responsável pela ligação ao *Web Service* e respectivo tratamento dos dados.

Se por um lado a interface de interacção com o utilizador é bastante linear em todas as plataformas abordadas, uma vez que estamos a falar de aplicações *windows* ou *web* que correm do lado do cliente, o mesmo não se passou com a classe responsável pelo tratamento dos dados, pois é esta que terá de comunicar com o *Web Service*.

Aqui a separação deve ser feita em plataformas *Open Source* e plataformas proprietárias ou comerciais.

Vejamos então como criar um cliente em PHP com NuSOAP:

Exemplo 6: Cliente - PHP

```
1  <?php
2
3  require_once('nusoap.php');
4  $min = $_REQUEST["minimo"];
5  $max = $_REQUEST["maximo"];
6
7  $localizacaoServidor = 'http://localhost:8080/nusoap/
8                          aleatorio/aleatorioserver.php';
9
10 $parameters = array('min'=>$min,
11                     'max'=>$max);
12
13 $soapclient =& new soapclient($localizacaoServidor);
14 $resultado=$soapclient->call('geraNumAleatorio',$parameters);
15 ...
16 ?>
```

Como se pode verificar a criação da aplicação cliente não poderia ser mais simples, bastando criar uma instância de um cliente SOAP e de seguida invocar o método pretendido.

No caso do Perl com SOAP-Lite a a criação do cliente é muito semelhante (exemplo 7).

Exemplo 7: Cliente - Perl

```
1  ...
2  use SOAP::Lite;
3  use CGI qw(:standard);
4
5  my $soapserver = SOAP::Lite
6      -> uri('http://localhost:8080/aleatorio')
7      -> proxy('http://localhost:8080/cgi-bin/aleatorio.cgi');
8
9  my $min = param('minimo');
10 my $max = param('maximo');
11
12 my $resultado = $soapserver->geraAleatorio($min,$max);
13 ...
```

Como podemos ver, para criar a aplicação cliente basta criar um instância de um servidor SOAP, onde no método `uri()` é necessário indicar o módulo onde se encontram definidas as funcionalidades disponibilizadas pelo *Web Service* (ver exemplo 4). Ao mesmo tempo é necessário indicar, no método `proxy()`, o endereço do servidor SOAP, i.e., a localização da *script* que efectua o acesso ao módulo (ver exemplo 3).

Como foi possível observar, no caso das plataformas *Open Source* apresentadas, a aplicação cliente acede directamente ao *Web Service*. Esta é uma das principais diferenças com as duas plataformas comerciais analisadas. Efectivamente, nestas últimas é introduzido o conceito de *proxy class*. A classe *proxy* é construída a partir do documento WSDL do *Web Service* em causa, apresentando uma API de interacção com a aplicação cliente, constituída por métodos com os mesmos nomes dos métodos expostos remotamente pelo *Web Service*.

A classe *proxy* para além de permitir o estabelecimento da ligação entre o *Web Service* e a aplicação cliente, encapsula toda a complexidade inerente à utilização do SOAP e dos protocolos de transporte a utilizar. Na prática, a aplicação cliente liga-se à classe *proxy* e é esta que estabelece contacto com o *Web Service* propriamente dito.

Vejamos então o exemplo dum cliente desenvolvido em .Net.

Exemplo 8: Cliente - .Net

```
...

AleatorioClient.localhost.NúmerosAleatórios ws =
   new AleatorioClient.localhost.NúmerosAleatórios();

int aleatorio = ws.geraNumAleatorio(minimo, maximo);
...
}
```

Como podemos verificar a criação da aplicação cliente é de facto muito simples. Após termos criado a classe *proxy* (`AleatorioClient.localhost`), basta utilizá-la para criar uma instância do *Web Service* e a partir daí invocar o método `geraNumAleatorio` definido no exemplo 2

Vejamos agora o caso de um cliente em WASP Server:

Exemplo 9: Cliente - WASP Server

```
...
import aleatorioclient.iface.Aleatorio;
...
String wsdlURI = "http://carloslopes:6060/aleatorio/wsdl";
String serviceURI = "http://carloslopes:6060/aleatorio/";
ServiceClient serviceClient = ServiceClient.create(wsdlURI, Aleatorio.class);
serviceClient.setServiceURL(serviceURI);
serviceClient.setWSDLServiceName(new QName("urn:aleatorio.aleatorio", "aleatorio"));
serviceClient.setWSDLPortName("aleatorio");
service = (Aleatorio) Registry.lookup(serviceClient);

int aleatorio = service.geraAleatorio(min, max));
...
```

```
14 }
15
```

Note-se a referência à classe *proxy* a utilizar para este *Web Service*.

```
import aleatorioclient.iface.Aleatorio
```

Os restantes passos para a criação da aplicação são bastante simples como se pode observar, bastando criar uma instância de um cliente do serviço, indicar a localização do serviço e invocar o método pretendido.

---

5 Conclusões