Web Services:
Metodologias de Desenvolvimento

Carlos J. Feijó Lopes José Carlos Ramalho

Fevereiro de 2004

Resumo

Os *Web Services* são uma tecnologia emergente, sobre a qual muito se tem especulado. No decorrer deste artigo efectua-se uma primeira contextualização, aproveitanto ao mesmo tempo para apresentar a arquitectura de funcionamento de um qualquer *Web Service*. De seguida, e aproveitando a implementação de um caso de estudo em quatro plataformas de desenvolvimento distintas, propomos metodologias de desenvolvimento para *Web Services* e respectivas aplicações cliente.

1 Contextualização

Os *Web Services* [PRV+01, Cer02] 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 normas *standard*, de entre as quais se destacam: XML, SOAP, WSDL e UDDI.

Segue-se uma breve descrição das funcionalidades de cada uma destas normas:

XML - metalinguagem de anotação na qual estão definidas todas as outras normas que servem de base aos *Web Services*[Con97, JH02].

SOAP - linguagem de anotação com a qual se pode descrever o protocolo de comunicação, responsável pela troca de mensagens de e para os *Web Services*[SOA] (uma mensagem SOAP é um documento XML).

WSDL - linguagem de anotação definida em XML e que tem como objectivo descrever a API de um *Web Service*[WSD, New02].

UDDI - linguagem de anotação definida em XML com a qual se cria a metainformação característica de um *Web Service*; vários registos UDDI são agrupados em repositórios; estes repositórios possuem uma interface/API de pesquisa para permitir a uma aplicação cliente pesquisar e localizar um serviço [UDD].

2 Arquitectura de funcionamento de um *Web Service*

O ciclo de vida de um *Web Service* compreende quatro estados distintos: `Publicação`, `Descoberta`, `Descrição` e `Invocação`. Vejamos com mais pormenor cada um deles:

`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 repositório de *Web Services* (UDDI).

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

`Descrição`: Processo pelo qual o *Web Service* expõe a sua API (documento WSDL); desta maneira 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 um destes estados corresponde uma das normas anteriormente referidas, nomeadamente:

```
Publicação, Descoberta -> UDDI,

Descrição              -> WSDL,

Invocação              -> SOAP.
```

A conjugação destes quatro estados permite constituir o ciclo de vida de um *Web Service*, o qual 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;
- Após a conclusão dos dois primeiros passos, o fornecedor regista o serviço no UDDI;
- O utilizador (aplicação cliente) pesquisa num repositório UDDI e encontra o serviço;
- A aplicação cliente estabelece a ligação com o *Web Service* e estabelece um diálogo com este, via mensagens SOAP.

3 Metodologias de desenvolvimento

Com o objectivo de se definirem metodologias de desenvolvimento de *Web Services* e aplicações cliente, foram escolhidas quatro plataformas distintas: o módulo SOAP-Lite para Perl [RK02, Kul01], o NuSOAP para PHP [PHP], o WASP Server para Java [JC02] e a .Net da Microsoft com recurso à linguagem C#.

Como caso de estudo escolhemos um *Web Service* muito simples capaz de gerar um nº aleatório entre dois valores limite indicados por uma aplicação cliente.

No decorrer desta secção, tanto o desenvolvimento do *Web Service* como da respectiva aplicação cliente é efectuado na mesma plataforma de desenvolvimento, dada a intenção de explorar de forma abrangente cada uma das plataformas escolhidas. No entanto, e como indica o próprio conceito de *Web*

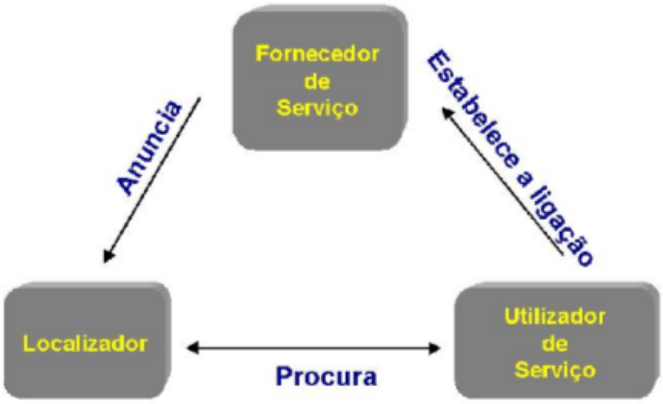

Figura 1: **Ciclo de vida de um *Web Service***

Service, um *Web Service* desenvolvido por exemplo em PHP pode obviamente ser acedido por um cliente em C#. Note que todos os clientes desenvolvidos podem trabalhar com qualquer um dos servidores havendo apenas que mudar o ponto de acesso ao serviço.

Dado o estado prematuro do UDDI, as fases de Publicação e de Descoberta não foram tidas em conta na elaboração das metodologias. O atraso na adopção do UDDI, prende-se também com a inexistência de uma interface normalizada para os *Web Services* de determinada espécie, por exemplo, as agências de viagens podiam disponibilizar várias operações, no entanto, se tiverem diferentes assinaturas para a mesma operação o UDDI poderá indicar ao cliente um *Web Service* que não disponha dum método a invocar com o nome pretendido, mas sim um método com a mesma funcionalidade mas com nome diferente, o que poderá originar erros na aplicação cliente.

O primeiro passo na obtenção de metodologias de desenvolvimento de *Web Services*, consistiu na abordagem ao problema sob dois pontos de vista distintos: a criação do servidor, ou seja, do *Web Service* propriamente dito, e a criação da aplicação cliente.

Esta será também a abordagem a utilizar neste artigo, pelo que vamos se seguida apresentar a análise à criação do *Web Service* propriamente dito.

3.1 O Servidor

Ao falarmos na criação de um *Web Service*, 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 ?>
```

Como se pode observar, de uma só vez implementamos as funcionalidades e criamos uma instância de um servidor SOAP.

Exemplo 2: Servidor - .Net

```
...
namespace Aleatorios
{
    [WebService(Name="Números Aleatórios",
     Description="Web Service que gera um nº"+
     " aleatório entre um valor mínimo e um valor
     máximo dados pela aplicação cliente",
      Namespace="urn:CarlosLopes")]
    public class Aleatorios : System.Web.Services.WebService
    {
        ...
        [WebMethod(MessageName="geraNumAleatorio",
         Description="método que gera um nº aleatório de acordo"+
         " com os valores mínimos e máximos indicados")]
        public int geraNumAleatorio(int min, int max)
        {
            Random r = new Random();
            return r.Next(min,max);
        }
        ...
    }
}
```

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

```
#!c:/perl/bin/perl.exe -w
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;
8 }
9 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*[1] que representará o *Web Service* (exemplo 5 e figura 2 respectivamente) .

Exemplo 5: Implementação das funcionalidades - WASP Server

---

[1] O conceito de classe *proxy*, será abordado com maior detalhe na secção 3.2

```
1  package aleatorio;
2  ...
3  public class aleatorio {
4      ...
5
6      // métodos a disponibilizar remotamente
7      public int geraAleatorio(int minimo, int maximo)
8      {
9          Random rand = new Random();
10         return (rand.nextInt(maximo-minimo) + minimo);
11     }
12 }
```

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

3.2 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 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
```

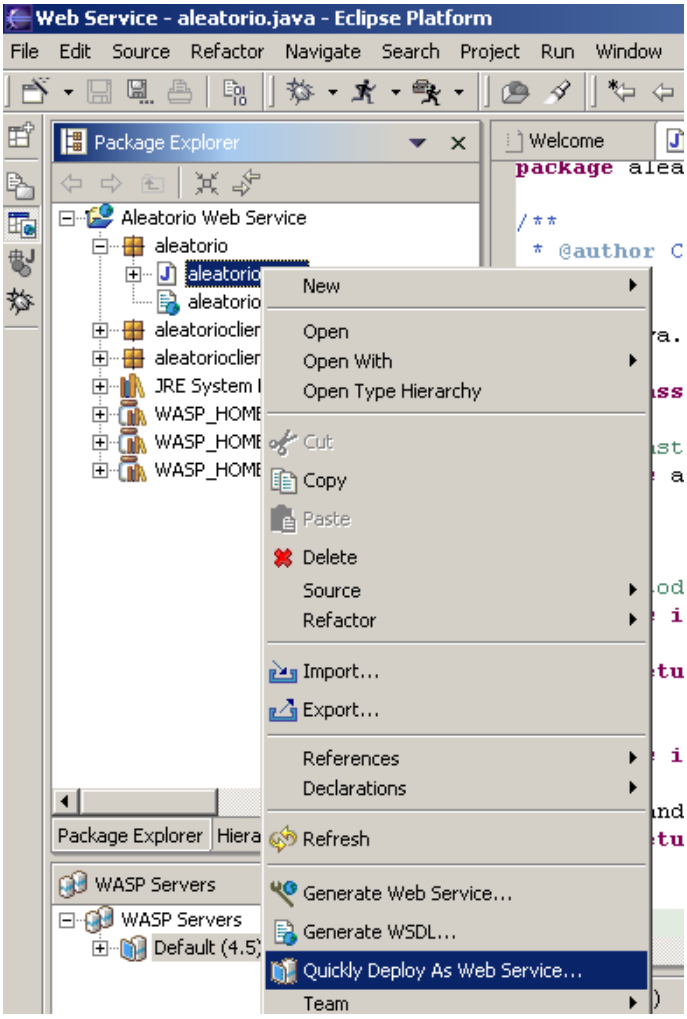

Figura 2: **Criação de um *proxy* a partir da classe em Java**

```
require_once('nusoap.php');
$min = $_REQUEST["minimo"];
$max = $_REQUEST["maximo"];

$localizacaoServidor = 'http://localhost:8080/nusoap/
                       aleatorio/aleatorioserver.php';

$parameters = array('min'=>$min,
                    'max'=>$max);

$soapclient =& new soapclient($localizacaoServidor);
$resultado=$soapclient->call('geraNumAleatorio',$parameters);
...
?>
```

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 criação do cliente é muito semelhante (exemplo 7).

Exemplo 7: Cliente - Perl

```
...
use SOAP::Lite;
use CGI qw(:standard);

my $soapserver = SOAP::Lite
    -> uri('http://localhost:8080/aleatorio')
    -> proxy('http://localhost:8080/cgi-bin/aleatorio.cgi');

my $min = param('minimo');
my $max = param('maximo');

my $resultado = $soapserver->geraAleatorio($min,$max);
...
```

Como podemos ver, para criar a aplicação cliente basta criar uma instância de uma classe para comunicação com o 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 em relação às duas plataformas comerciais analisadas. Efectivamente, nestas últimas é introduzido o conceito de *proxy class* [Lop04]. 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

```
1 ...
2
3 AleatorioClient.localhost.NúmerosAleatórios ws =
4    new AleatorioClient.localhost.NúmerosAleatórios();
5
6 int aleatorio = ws.geraNumAleatorio(minimo, maximo);
7 ...
8 }
```

Como podemos verificar a criação da aplicação cliente é de facto muito simples. Após a criação da 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

```
1  ...
2  import aleatorioclient.iface.Aleatorio;
3  ...
4  String wsdlURI = "http://carloslopes:6060/aleatorio/wsdl";
5  String serviceURI = "http://carloslopes:6060/aleatorio/";
6  ServiceClient serviceClient = ServiceClient.create(wsdlURI,
7                                                     Aleatorio.class);
8  serviceClient.setServiceURL(serviceURI);
9  serviceClient.setWSDLServiceName(
10         new QName("urn:aleatorio.aleatorio", "aleatorio"));
11 serviceClient.setWSDLPortName("aleatorio");
12 service = (Aleatorio) Registry.lookup(serviceClient);
13
14 int aleatorio = service.geraAleatorio(min, max));
15 ...
16 }
```

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.

Estando apresentadas as principais diferenças, quer na criação do *Web Service* quer na criação da respectiva aplicação cliente, entre as quatro plataformas escolhidas, é tempo de apresentar um quadro resumo com as metodologias daí retiradas [Lop04].

Tabela 1: Metodologias para desenvolvimento de *Web Services*

	Servidor	**Cliente**
PHP	Criação de uma instância de um servidor; Registo das funções a invocar remotamente; Especificação das funções;	Criação de um array com os parâmetros necessários à função a aceder remotamente; Criação de instância de cliente SOAP; Efectuar chamada ao método ao qual pretendemos aceder, e obtenção dos resultados;
Perl	Criação do módulo com as funcionalidades pretendidas; Criação da CGI com o servidor;	Criação de uma instância de uma classe para comunicação com o servidor SOAP; Invocação dos métodos pretendidos e obtenção de resultados;
.Net	Criação de classe funcional + extensão .asmx + tag <% Web Service ... %> **ou em alternativa** criar um novo projecto do tipo *Web Service*; Indicação dos métodos a expôr remotamente com o atributo `WebMethod`;	Criação de um *proxy*; Invocação dos métodos expostos pelo WS e obtenção de resultados;
WASP Server	Criação da classe funcional; Registo da classe como sendo um *Web Service* e consequente obtenção de classe *proxy*;	Inicialização de uma instância da classe *proxy*; Invocação dos métodos pretendidos e obtenção dos resultados;

De um ponto de vista mais abstracto podem-se retirar da tabela 1, alguns pontos comuns que convém salientar:

Servidor :

1. Criação de classe onde se implementam as funcionalidades que se pretendem disponibilizar para acesso remoto;
2. Identificação dessa classe como sendo um *Web Service*.

Cliente :

1. Criação de uma classe cliente;
2. Indicação à classe cliente da localização do serviço;
3. Invocação dos métodos pretendidos.

4 Conclusões

Os *Web Services* apresentam-se como a solução para muitos dos problemas associados aos sistemas distribuídos, nomeadamente nas questões relacionadas com a integração de sistemas heterogéneos, razão pela qual têm estado rodeados por uma euforia nem sempre benéfica.

No entanto o facto de serem componentes de *software* modulares, auto-descritivas, que se baseiam em protocolos *standard* e que permitem a interacção entre aplicações sem intervenção humana, torna-os sem dúvida o futuro dos sistemas distribuídos.

A existência de plataformas de desenvolvimento que tornam transparente ao programador todo esforço de criação/interacção de mensagens SOAP e criação/utilização dos documentos WSDL que descrevem o *Web Service*, facilitam a sua adopção comercial, fomentando assim o seu desenvolvimento.

Dada a relevância dos *Web Services* no futuro das tecnologias de informação, torna-se oportuna a proposta de metodologias para o seu desenvolvimento. Para tal, aproveitou-se a implementação de um caso de estudo, para apresentar uma espécie de "receita"pronta a utilizar no desenvolvimento dos mesmos.

Referências

[Cer02] Ethan Cerami. *Web Services Essentials*. O´Reilly, 2002.

[Con97] W3C: World Wide Web Consortium. Extensible markup language (xml): version 1.0. http://www.w3.org/XML, Dezembro 1997.

[JC02] Tyler Jewell and Davis Chapel. *Java Web Services*. O'Reilly, 2002.

[JH02] J.C.Ramalho and Pedro Henriques. *XML & XSL Da Teoria à Prática*. FCA, 2002.

[Kul01] Paul Kulchenko. Quick start guide with soap and soap::lite. http://guide.soaplite.com, 2001.

[Lop04] Carlos Jorge Feijó Lopes. Web services: Aplicações distribuídas sobre protocolos internet. Master's thesis, Universidade do Minho, Janeiro 2004.

[New02] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP and UDDI*. Addison-Wesley, 2002.

[PHP] Site oficial do projecto php. http://www.php.net.

[PRV+01] P.Cauldwell, R.Chawla, V.Chopra, G.Damschen, C.Dix, T.Hong, F.Norton, U.Ogbuji, G.Olander, M.Richman, K.Saunders, and Z.Zaev. *Professional XML Web Services*. Wrox Press, 2001.

[RK02] Randy J. Ray and Pavel Kulchenko. *Programming Web Services with Perl*. O´Reilly, 2002.

[SOA] Simple object access protocol (soap). http://www.w3.org/TR/SOAP.

[UDD] Universal description, discovery, and integration (uddi). http://www.uddi.org.

[WSD] Web services description language (wsdl). http://www.w3.org/TR/WSDL.