

# Web Services em Java com Axis - Teoria e Prática

## Agostinho Campos Neto

*Veja na teoria o que são Web Services e como aplicá-los na prática.*

### Introdução

#### Teoria e definições

A Internet evoluiu muito desde o seu surgimento, no início a tecnologia permitia apenas que alguém se conectasse a uma página e baixasse seu conteúdo para um navegador. Hoje em dia é possível transferir imensos fluxos de mídia através da rede. As últimas criações de maior impacto na rede (além do Gmail, claro) que estão recebendo toda a atenção é a linguagem XML e os Web Services (WS).

Os Web Services é uma tecnologia que tem por objetivo integrar sistemas distintos através da Internet usando protocolos padronizados que garantem a independência de plataforma e de linguagem de programação em que esses sistemas foram escritos. Assim, é possível disponibilizar uma coleção de métodos em um servidor remoto e permitir que sejam acessados por programas clientes.

XML é uma linguagem semelhante ao HTML. A diferença é que ela é usada para definir e armazenar dados, enquanto HTML serve para exibir. Na prática, XML serve para a troca de dados entre programas diferentes. XML está fora do escopo do artigo (veja mais no quadro abaixo).

Todo o sucesso e aceitação dos Web Services está justamente na sua padronização, onde programas escritos em linguagens diferentes podem fazer o intercâmbio de dados. Tenha em mente que os WS são uma opção para a criação de aplicações distribuídas na Internet. WS é uma forma de tentar distribuir serviços na rede. Existem modelos outras formas de se criar aplicativos distribuídos, (como CORBA) mas que não é tão que é muito mais complicado de se trabalhar.

Atenção: recentemente surgiu a Arquitetura Orientada a Serviços (AOS) que é uma **maneira de se projetar e criar** sistemas empresariais distribuídos oferecendo serviços e os WS são uma **opção** tecnológica para se programar esse aplicativos. Não confunda WS com AOS.

Para compreender tudo melhor, veja na seção seguinte explicações mais práticas.

#### Mais detalhes

Você pode encontrar material formal sobre WS e XML em:

- <http://www.w3.org/2002/ws/> (Web Services)
- <http://www.w3c.org/XML> (XML)
- <http://www.w3schools.com/xml/default.asp> (tutorial de XML)

### Protocolos envolvidos

Vamos definir os principais termos utilizados aqui:

- **SOAP:** é um “envelope” que contém dados XML e é utilizado nas requisições e respostas entre os métodos;
- **XML:** é a linguagem padronizada para troca de dados multiplataforma;
- **WSDL:** é semelhante ao XML e é gerada a partir de um WS em particular, a WSDL é uma interface padronizada que permite os programas clientes encontrarem os métodos remotos;

Quase nunca se precisa programar em SOAP, XML ou WSDL diretamente. Basta se preocupar com as regras de negócio que o resto é produzido por ferramentas auxiliares.

### Arquitetura

#### Cenário I

Imagine que você tem um WS feito em Java com métodos que implementam uma calculadora comum (soma, subtração, multiplicação e divisão). Seu serviço está rodando em um servidor remotamente.

Imagine agora que alguém ache que é complicado fazer um programa desse tipo. Então ele descobre que alguém já fez isso pra ele (no caso, você) e deseja usar seu serviço no próprio programa dele, que está escrito, por exemplo, em Delphi ou C#. Então esses programas clientes irão consumir seu serviço (invocando os métodos) e devolver uma resposta. Humm...isso lhe parece familiar? Deveria. Isso é **reutilização de software** elevado a 10! Veja a Fig. 1. para ver em detalhes o que acontece.

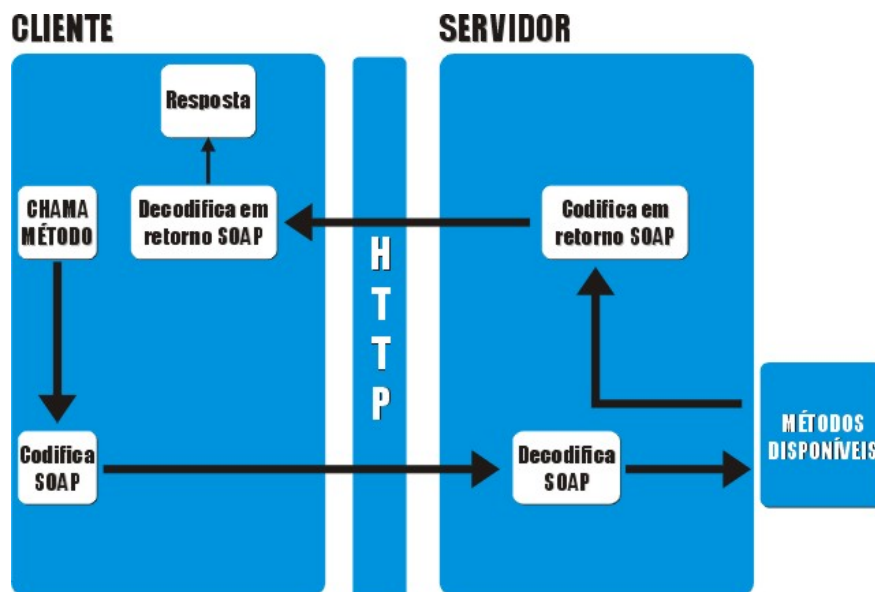


Fig. 1. Chamada de métodos remotos.

As chamadas do cliente são codificadas em SOAP e transmitidas por HTTP (ou seja, o Firewall de qualquer empresa vai liberar), quando o seu serviço recebe a requisição acontece o inverso, as chamadas são decodificadas executadas e a resposta é codificada novamente para ser enviada para o cliente. Como eu disse, você não programa em um nível tão baixo.

### Cenário II

Uma livraria de médio porte possui diversos fornecedores e sempre precisa estar em contato (por telefone ou e-mail) com os mesmos para saber os últimos preços e prazos de entrega dos livros. Com base nessas informações ela decide de que fornecedor realizar a compra. Esse é um cenário ideal para WS.

- **O fornecedor:** criaria um WS disponibilizando um método que acesse sua base de dados com os preços e prazos dos seus livros;
- **A livraria:** criaria um programa que chame esse método remoto e consulte (por exemplo, através do ISBN do livro) e obter os preços e prazos em tempo real, sempre atualizados;
- **Um ainda cenário melhor:** a livraria poderia utilizar WS de vários fornecedores paralelamente (veja Fig. 2.) e no fim ter um relatório dos melhores fornecedores para determinado livro;

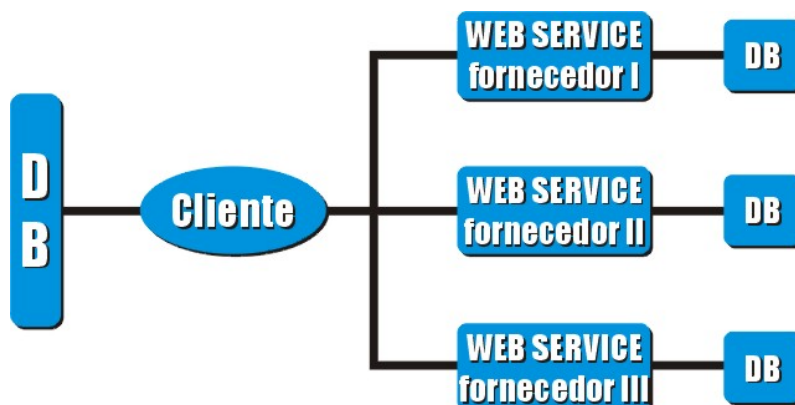


Fig. 2. Programa cliente acessando vários WS.

## WSDL

Como um programa cliente descobre que métodos existem no Web Service? Através da WSDL (Web Service Definition Language, Linguagem de Definição de Serviço). WSDL é uma linguagem padronizada que fornece uma **interface** comum para os aplicativos clientes poderem chamar os métodos remotos.

Em geral, quando você implanta um WS em um servidor, geralmente alguém acessa esse WS pelo navegador e baixa WSDL do seu serviço. Como a WSDL tem todos os métodos do WS registrados nela, é possível usá-la para criar um proxy local para ser usado pelo seu programa cliente.

Na prática, você usa um programa para gerar esse proxy. O Axis (veja abaixo) tem uma ferramenta chamada WSDL2Java que permite criar esse arquivo, mas também existe o plugin para Eclipse Lombok.

## Prática

### Como criar um WS em Java?

De maneira geral, você precisa:

1. Criar uma classe Java com os métodos do seu serviço;
2. Implantar essa classe no servidor (no nosso caso, Tomcat);

### Como consumir um WS criado?

1. Descobrir onde o WS está hospedado;
2. Recuperar o WSDL do serviço;
3. Criar um programa cliente usando o WSDL;

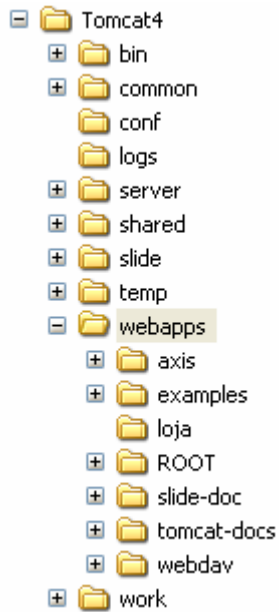
### Softwares necessários

- Tomcat 4.1 instalando e funcionando (a instalação do Tomcat não é descrita nesse tutorial pois o Google é suficiente pra isso);
- Faça o download da versão mais recente do Axis (usei a 1.2) em <http://ws.apache.org/axis>
- Download da versão mais recente do Xerces (usei a 2.6.2) <http://www.apache.org/dist/xml/xerces-j/> ele contém dois arquivos .JAR que o Axis precisa pra poder interpretar XML.

### Instalação do Axis

O Axis é uma ferramenta que auxilia na criação de WS. Para implantar o WS no Tomcat, o Axis precisa ser instalado e ativado. A instalação é exibida para Windows XP, mas é semelhante para os outros sistemas.

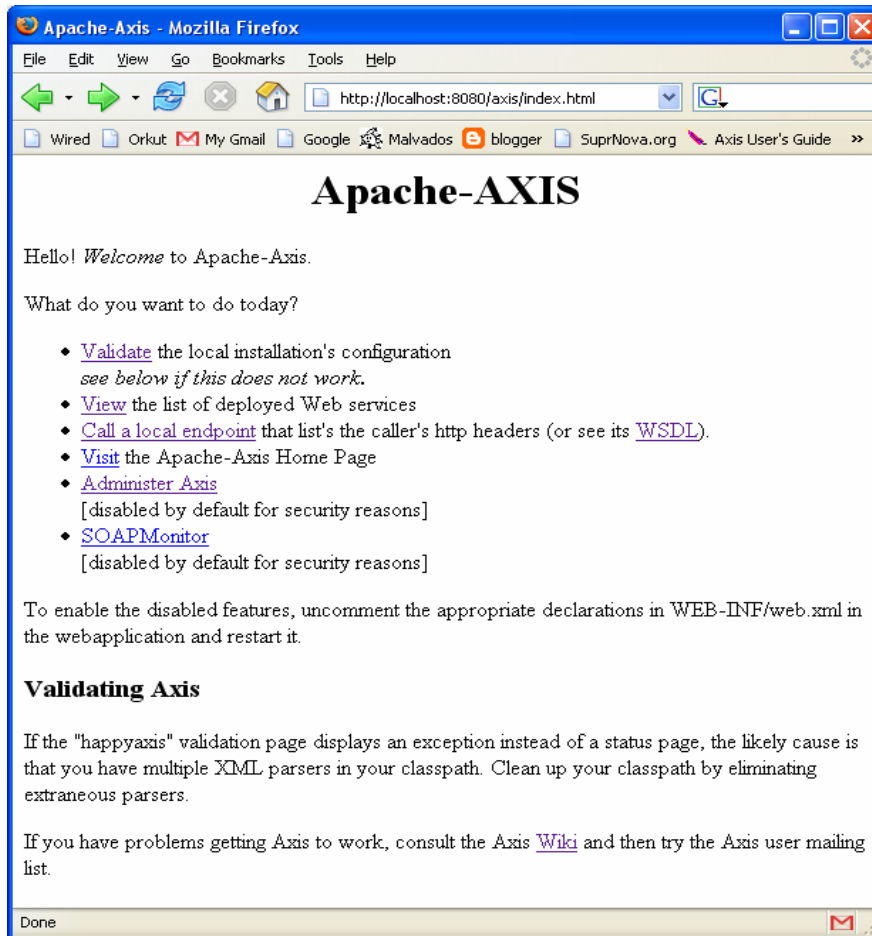
1. Descompacte o .zip do Axis e deixe ele em: "C:\axis"
2. Descompacte o .zip do Xerces no disco e copie os arquivos: `xml-apis.jar` e `xercesImpl.jar` para o diretório "C:\axis\lib"
3. Dentro de "C:\axis\webapps" tem um diretório chamado "axis". Copie esse diretório para dentro do seu "webapps" do seu Tomcat (no meu caso: "C:\Tomcat4\webapps")
4. Seu Tomcat vai ficar assim:



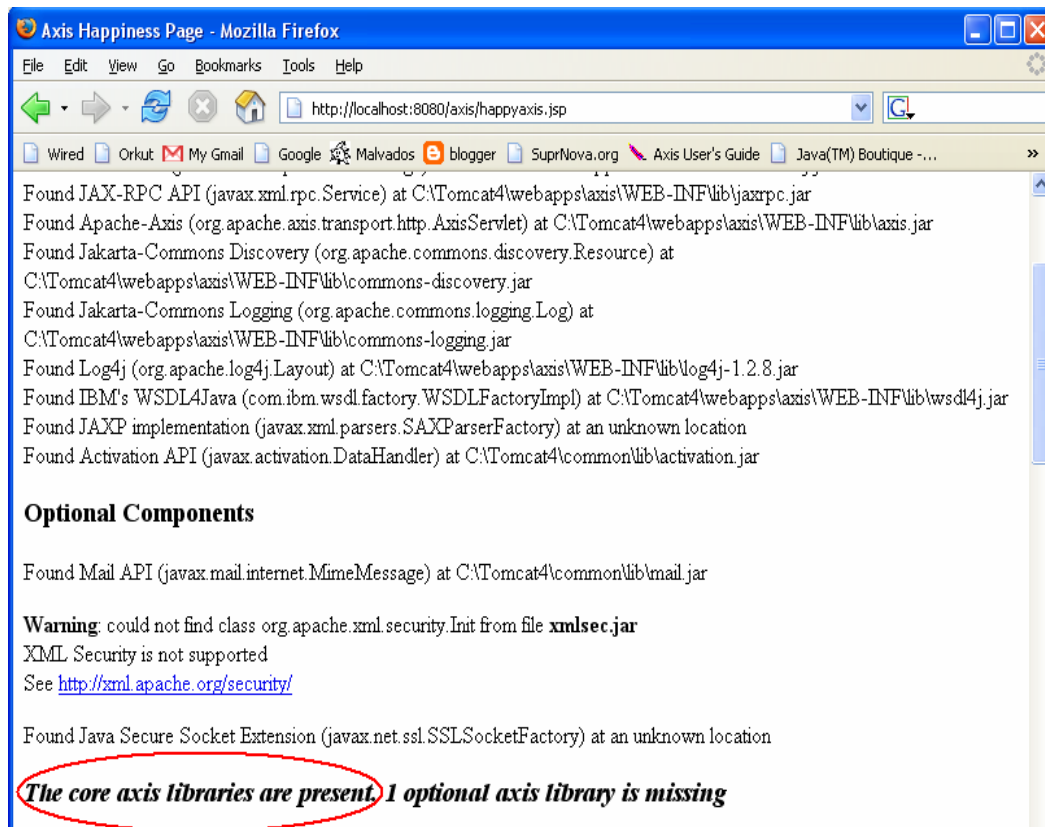
5. Nas variáveis de ambiente do sistema, configure da seguinte forma:

Crie uma nova variável `AXIS_HOME = C:\axis`  
Crie outra variável `AXIS_LIB = %AXIS_HOME%\lib`  
Crie uma variável `AXISCLASSPATH =`  
`%AXIS_LIB%\axis.jar;%AXIS_LIB%\commons-discovery.jar;%AXIS_LIB%\commons-`  
`logging.jar;%AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\saaj.jar;%AXIS_LIB%\log4j-`  
`1.2.8.jar;%AXIS_LIB%\xml-apis.jar;%AXIS_LIB%\xercesImpl.jar`

6. O Axis está instalado. Para verificar se tudo está funcionando inicie o Tomcat e digite:  
<http://localhost:8080/axis>



7. Clique em Validate e você deverá ver um resumo geral. Se ele der algum erro (como falta de algum .JAR), você tem que corrigir antes de continuar. Observe que as **core axis libraries** devem ser encontradas.



Agora vamos criar um WS. Crie esse arquivo Hello.java

```
public class Hello {  
  
    public String alo() {  
        return "Ola, meu primeiro WS...";  
    }  
  
    public int somar(int a, int b) {  
        return (a + b);  
    }  
  
}
```

Agora copie esse arquivo pra dentro de "C:\Tomcat4\webapps\axis" e troque a extensão dele de .java para .jws. Agora digite: <http://localhost:8080/axis/Hello.jws?wsdl> e você vai ver a descrição do seu arquivo WSDL. Fácil não? O servlet do Axis converte a nossa classe automaticamente para WSDL. Essa é a forma mais simples de se implantar um WS, mas existe outra forma (como vou mostrar). Agora já podemos criar um cliente para usar o serviço.

Para criar o cliente, vamos usar a ferramenta WSDL2Java que vem com o Axis. Vá até o diretório "<seu\_tom\_cat>\webapps\axis\WEB-INF\lib". No prompt, digite:

```
java org.apache.axis.wsdl.WSDL2Java http://localhost:8080/axis/Hello.jws?wsdl
```

### Mais detalhes

Para o comando anterior funcionar, certifique-se de os seguintes arquivos estejam no seu **classpath**: axis.jar, commons-logging.jar, commons-discovery.jar, saaj.jar, wsdl4j.jar, jaxrpc.jar.

Se não, adicione-os da seguinte forma:

```
%AXIS_LIB%\axis.jar;%AXIS_LIB%\commons-logging.jar;%AXIS_LIB%\commons-  
discovery.jar;%AXIS_LIB%\saaj.jar;%AXIS_LIB%\wsdl4j.jar;%AXIS_LIB%\jaxrpc.jar
```

Observe que é gerado um diretório chamado "localhost". Dentro dele existem várias classes Java, que são seu proxy local sobre o qual vamos trabalhar. Crie o seguinte programa fora desse diretório:

```
import localhost.axis.Hello_jws.*;  
  
public class MeuCliente {  
    public static void main(String[] args) throws Exception {  
  
        HelloService hservice = new HelloServiceLocator();  
        Hello h = hservice.getHello();  
  
        System.out.println( "! Iniciando chamada a metodos remotos !" );  
  
        System.out.println( "METODO ALO: " + h.alo() );  
        System.out.println( "METODO SOMAR: " + h.somar(2, 6) );  
  
    }  
}
```

O mais importante do código são as linhas que fazem conexão com seu Web Service:

```
HelloService hservice = new HelloServiceLocator();  
Hello h = hservice.getHello();
```

HelloService é uma interface que é implementada pela classe HelloServiceLocator. HelloService tem um método ( getHello() ) que devolve um objeto para a interface Hello, que possui os métodos do WS criado, assim, podemos usar o objeto **h** como se fosse um objeto local normalmente. Veja o código fonte do diretório localhost pra ficar mais claro. Isso também lembra RMI, a diferença fundamental é que RMI só funciona com programas escritos em Java e WS não depende de linguagem.

Agora compile essa classe e rode o programa. Você verá os métodos sendo chamados.

### Implantação do WS

Além dessa forma instantânea de obter o WSDL de um WS, existe uma outra forma, que é usando um descritor de implantação. Se usar essa forma de implantação, você verá na página do navegador todos os seus WS registrados com seus respectivos WSDLs disponíveis (é uma forma mais profissional).

Vamos fazer isso agora. Dentro de um diretório qualquer, crie uma classe simples como essa:

```
public class TesteWSDD {  
  
    public String testar() {  
        return "testando implantacao...";  
    }  
  
}
```

Agora crie um arquivo chamado deploy.wsdd e coloque o seguinte código XML:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"  
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">  
  <service name="TesteWSDD" provider="java:RPC">  
    <parameter name="className" value="TesteWSDD"/>  
    <parameter name="allowedMethods" value="*" />  
  </service>  
</deployment>
```

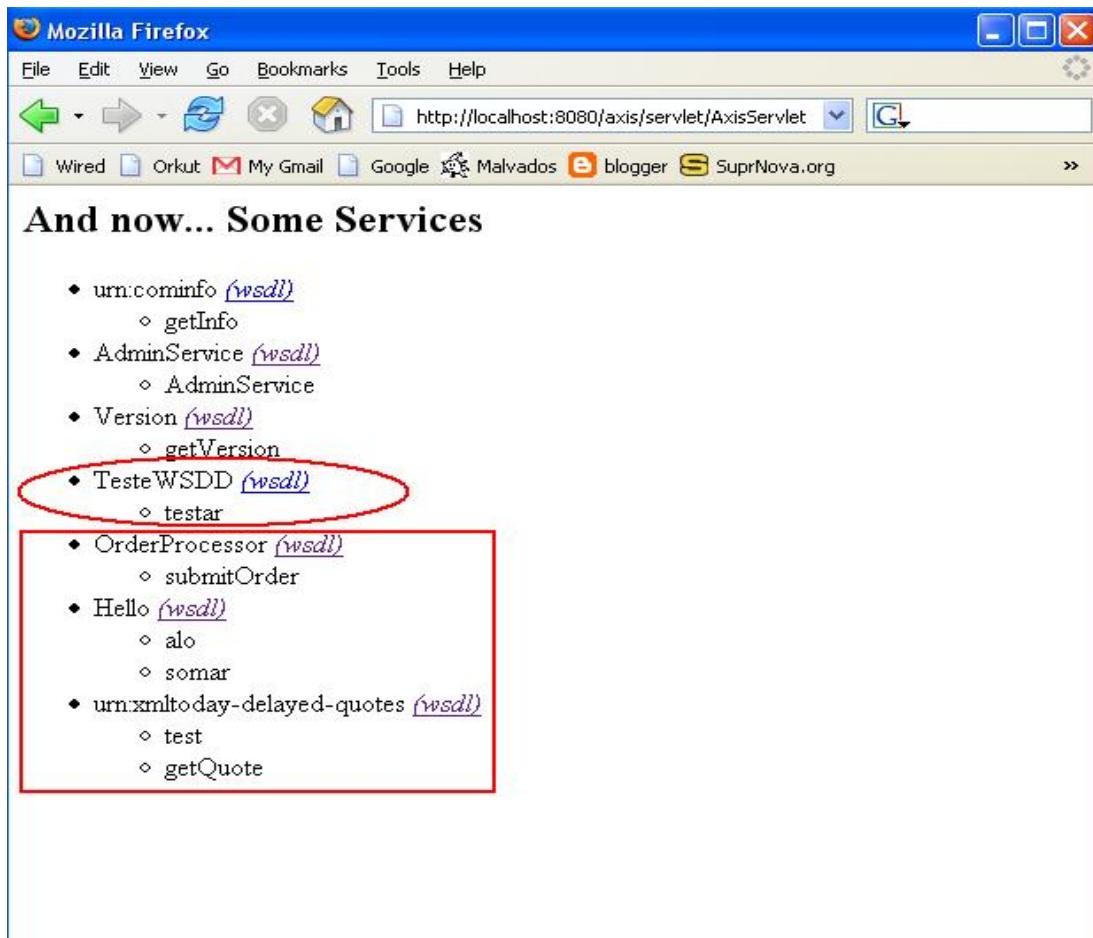
Esse código não tem mistério. Informa um nome para o WS (<service name>), o nome da classe ("className") é definido no primeiro parâmetro e no segundo ("allowedMethods") o asterisco indica que todos os métodos públicos serão acessíveis. Consulte a documentação para saber de detalhes mais específicos.

Compile a classe e mova o arquivo TesteWSDD.class para dentro de <seu\_tom\_cat>\webapps\axis\WEB-INF\classes (isso é obrigatório).

Execute o seguinte comando no prompt:

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

Pronto, seu WS está implantado no servidor. Outra clara vantagem dessa técnica é que se você tiver implantado vários serviços, a pessoa que deseja usar seu serviço, pode acessar seu “catálogo” online e navegar entre os serviços e escolher o que melhor resolve seus problemas. Veja:



A elipse mostra o WS que acabou de ser implantado com seu único método testar(). O quadrado mostra outros WS implantados na minha máquina no momento e seus respectivos métodos. Depois que obtiver seu WSDL, é só criar o programa cliente (possivelmente em outra linguagem).

### Mais detalhes

Visite o site: [www.xmethods.net](http://www.xmethods.net) para obter uma imensa listagem de Web Services disponíveis online. Nesse site, você pode obter o WSDL de cada serviço e criar programas clientes para utilizá-los.

### Conclusão

Web Services são uma alternativa para aplicações distribuídas na Internet e possui inúmeros benefícios em relação a outras tecnologias existentes. Por ser uma tecnologia recente, ainda precisa melhorar em vários pontos, como a especificação de segurança e de transações.

Tentei mostrar como usar WS e o Axis da forma mais clara e simples possível. Também existem outras ferramentas para se trabalhar com WS, mas de forma geral, sempre vai ser a mesma coisa. Uma coisa que não foi abordada aqui (talvez em outro artigo) foi a passagem de tipos complexos nos métodos (como classes e arrays), mas que é fundamental para programas “reais”.

**Agostinho Campos Neto** ([agostinho@gmail.com](mailto:agostinho@gmail.com)) é graduando em Sistemas de Informação pela UnP – Universidade Potiguar e se interessa por assuntos como J2ME, EJBs e aplicações distribuídas.