Implementação de exemplos de uso de Web Services como um tutorial

Alexandra C. P. de Aguiar, Andriele do Carmo, Carlos Petry, Flaviano Luzzato

[1]Programa de Pós Graduação em Ciência da Computação – PPCGG
Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Porto Alegre – RS – Brazil

{alexandra.aguiar,andriele.carmo, carlos.petry, flaviano.luzzato}@pucrs.br

Abstract. *In the last years the technology of Web Services have been increasingly used by both academical and comercial institutes. One of its main objectives is to allow the exchange of information, in a transparent way, among heterogeneous and distributed systems. One important thing in this context is to provide documents such as tutorials which should be complete enough to allow the creation and publication of simple Web Services. Thus, this work purposes the creation of a simple tutorial in which simple Web Services are created through wide known languages, such as Java and PHP.*

Resumo. *Nos últimos anos a tecnologia de Web Services tem sido cada vez mais utilizada por diversas instituições, tanto no meio acadêmico quanto no meio comercial, com o objetivo de trocar informações entre sistemas heterogêneos e distribuídos de forma mais transparente. Com o advento de tal recurso torna-se importante a criação de tutoriais que sejam capazes de exibir de maneira simples a criação de Web Services e sua publicação na Internet. Dessa forma, este trabalho visa apresentar um tutorial acerca da criação e disponibilização de Web Services através de linguagens amplamente conhecidas como Java e PHP.*

1. Introdução

Nos últimos anos, com o advento dos Sistemas Distribuídos (SD) [Becker 2007], cada vez mais heterogêneos, e da utilização cada vez mais instrínseca da Internet por inúmeras instituições surgiu a necessidade de se realizar a interconexão entre ambientes distintos de maneira mais transparente. Dessa forma, o desenvolvimento de novas tecnologias para facilitar essa integracão e também permitir a disponiblidade de informações e processos foi cada vez mais intenso.

A tecnologia de Web Services (WS) representa uma recente evolução nos padrões de desenvolvimento de aplicações distribuídas. A proposta é facilitar o compartilhamento das informações existentes em diferentes instituições proporcionando a criação de ambientes remotamente cooperativos que sejam independentes de plataforma operacional ou linguagem de programacão. Esta evolução tem a potencialidade de alterar a forma como as aplicações são desenvolvidas, como a informação é apresentada e compartilhada e, também, como o software é comercializado [SOAP 2007].

Os novos paradigmas para sistemas de Tecnologia de Informacão (TI) vêm exigindo uma maior interação entre plataformas e serviços de forma mais flexível e que

possa evoluir facilmente. Um desses paradigmas é a linguagem de descrição eXtended Markup Language (XML), utilizada como a linguagem universal para implementar essa interoperabilidade.

Os WS são formados por aplicativos que usam codificações e protocolos padronizados para troca de informações. Assim, são amplamente utilizados em aplicacões de cunho comercial permitindo que sistemas de computação interajam através de diferentes ambientes. Os WS dispõem, além de facilidades nas transacões distribuídas, de bom nível de segurança e de serviços de mensagens confiáveis [WebServices 2007]. Portanto, alguns problemas como distribuição, transporte, segurança, autenticação e controle de transações são amenizados através dessa tecnologia.

Este trabalho tem o objetivo de apresentar demonstrações práticas e simples do uso de WS em ambientes heterogêneos. São utilizados para este fim sistemas operacionais distintos e diferentes linguagens de programacão representando as aplicacões com necessidades de cooperação.

O artigo está organizado como segue. Na próxima Seção são abordadas as definições dos diversos protocolos necessários para o funcionamento dos Web Services, seguido por uma Seção que apresenta a metodologia utilizada para a criação de Web Services. Na Seção 4. são abordados os detalhes da implementação realizada e finalmente, na Seção 5. são apresentadas as considerações finais do trabalho.

2. Web Services - uma abordagem conceitual

Os WS são serviços distribuídos que processam mensagens que:

1. obedecem ao protocolo Simple Object Access Protocol (SOAP);
2. sejam codificadas em XML;
3. enviadas através de requisições Hyper Text Markup Language (HTTP) e;
4. descritas através de Web Services Description Language (WSDL).

Nesse contexto, é possível definir o papel de cada um desses itens durante a transação entre Web Services:

1. XML fornece a sintaxe comum para a representação de dados;
2. o protocolo SOAP fornece a semântica para a troca de dados;
3. a WSDL proporciona um mecanismo para descrever as capacidades de um serviço da Web.

É importante salientar que a WSDL é uma linguagem baseada em XML e é utilizada para descrever um WS. No documento WSDL estão definidas todas as interfaces, operações, esquemas de codificação, entre outros [WSDL 2007]. Assim sendo, um documento WSDL define um esquema XML que descreve um WS.

O protocolo SOAP é construído com base em XML e HTTP, sendo largamente aceito para uso com WS. Esse protocolo permite a intercomunicação entre diferentes sistemas através da utilização de linguagens que permitam a implementação de aplicações transportadas por SOAP.

3. Metodologia

Os sistemas operacionas utilizados são Microsoft Windows XP [Microsoft 2007] e Ubuntu [Ubuntu 2007] Linux. As linguagens testadas nos exemplos de aplicacões WS são PHP5 [5 2007], JSP [JSP 2007] e Java [Java 2007].

Em cada máquina cliente devem ser instalados os itens necessários para o funcionamento da linguagem escolhida para a aplicação cliente. No caso do presente trabalho existem duas possibilidades (ambas implementadas em Ubuntu Linux):

- **Cliente WS em PHP:** necessária a instalação do servidor HTTP Apache [Apache 2007] e do PHP5;
- **Cliente WS em Java:** necessária a instalação do Java Development Kit (JDK) e a utilização da Application Programming Interface (API) Axis [Axis 2007].

Por outro lado, neste trabalho, os servidores Web Services foram implementados em sistemas operacionais (SO) distintos:

- **Microsoft Windows XP.** Nesse SO o servidor WS foi descrito em JSP acessando uma base de dados em MySQL [MySQL 2007]. Para isso foi necessária a instalação: do servidor HTTP Apache Tomcat [Tomcat 2007], do JDK, do MySQL e da API Axis.
- **Ubuntu Linux.** Nesse SO o servidor WS foi descrito em PHP5 utilizando uma base de dados MySQL necessitando da instalação do servidor HTTP Apache, do PHP5 e da base de dados MySQL.

Através da utilização dos servidores (criadores de WS) e clientes é possível realizar a intercomunicacão entre linguagens e SO's heterogêneos, como descrito nas Figuras 1 e 2. Nessas Figuras são exibidas duas transações. A primeira, determinada de *Aluno*, tem o cliente em PHP e o servidor em JSP. Já a segunda, determinada de *RU*, tem o cliente em Java e o servidor em PHP.

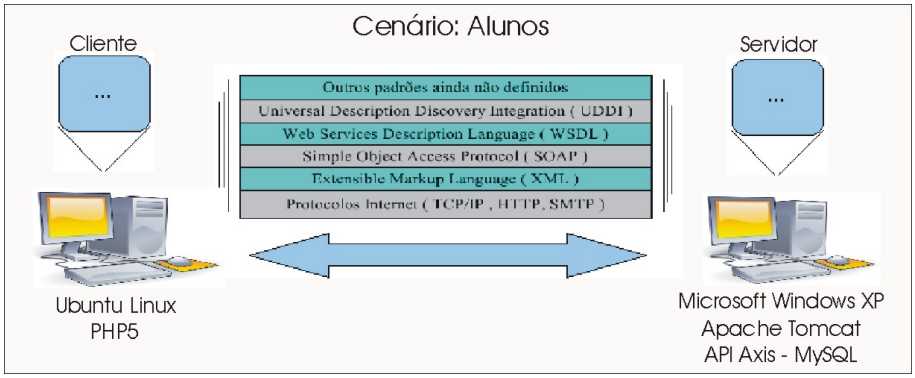

Figure 1. WS operando em ambiente heterogêneo - Cenário Aluno

4. Implementações

Na próxima Seção são exibidos os detalhes de implementação de cada um dos cenários propostos como estudo de caso.

4.1. Cenário: *Aluno*

O objetivo principal deste cenário é possibilitar a consulta, por parte de qualquer cliente, a base de dados *Aluno*. Um exemplo de uso deste cenário é um restaurante universitário (cliente) que solicite a um determinado estudante seu número de matrícula e consulte uma base de dados central da universidade (disponibilizada através de um servidor WS) para verificar se o mesmo é ou não aluno da instituição. As próximas Seções detalham as implementações necessárias para este cenário: cliente e servidor.

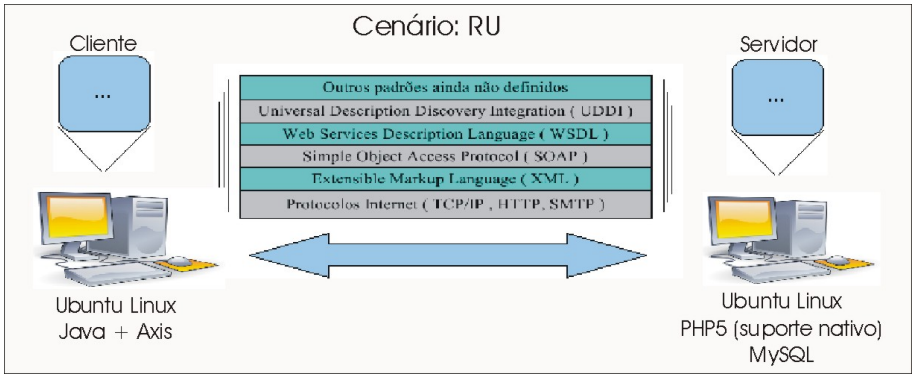

Figure 2. WS operando em ambiente heterogêneo - Cenário RU

4.1.1. Cliente

Conforme descrito na Seção 3. onde apresenta-se a implementação dos clientes, o cliente deste cenário foi desenvolvido em PHP5.

Inicialmente é necessária a criação de um objeto cliente de WS no arquivo php:

```
$client = new SoapClient(null, array('
'location' => "http://192.168.145.128:8090/axis/Servico.jws?wsdl",
'uri' =>"http://192.168.145.128:8090/axis/Servico.jws?wsdl"));
```

Nesse trecho de código pode-se observar a criação de um objeto cliente (*$client*) que é uma instância da classe (nativa ao PHP5) *SoapClient*. Para realizar essa chamada deve-se passar o enredeço do WS servidor, nesse caso disponibilizado através da API Axis, no servidor Apache Tomcat.

Após a criação do objeto cliente basta realizar a chamada à função disponibilizada pelo servidor WS, nesse caso, uma função denominada *retornaAlunoPorMatricula* cujo parâmetro de retorno é um vetor contendo os dados de um determinado aluno e parâmetro de entrada é a matrícula do estudante. Essa chamada é realizada de maneira muito simples no PHP dessa forma:

```
$vetor = (array)$client->retornaAlunoPorMatricula("071902");
```

Uma maneira genérica de realizar essa chamada é através da utilização de formulários HTML que solicite ao usuário que está acessando a página para que digite a matrícula e clique em um botão do tipo *enviar*, como pode-se observar a seguir:

```
<form name="input" action="'.\$_SERVER['PHP_SELF'].
'?action=get_data" method="POST">
Matrícula: <input type="text" name="matricula">
<input type="submit" value="Buscar"></form>
```

A ação deste formulário pode ser a entrada para a criação do cliente WS (da mesma maneira como já exibido) mas agora a chamada à função *retornaAlunoPorMatricula* tem como entrada o dado digitado pelo usuário no formulário: `$vetorDadosAluno = (array)$client->\—retornaAlunoPorMatricula($_POST["matricula"]);`

Uma vez tendo chamado a função de maneira correta, basta exibir os dados através de uma função que imprime os dados de um vetor:

```
print_r($vetorDadosAluno);
```

Dessa forma está montado um cliente WS em PHP que solicita uma informação ao usuário, realiza a consulta a um WS e exibe os dados de retorno. A próxima Seção aborda a parte servidora deste cenário.

4.1.2. Servidor

Conforme descrito na Seção 3. onde apresenta-se a implementação dos servidores, o servidor deste cenário foi desenvolvido em JSP no Windows.

Como o JSP não possui um suporte nativo a WS é necessária a instalação de uma API externa para que seja possível a disponibilização desses serviços. Para isso deve-se realizar o download da API Axis [Axis 2007] e após descompactar o arquivo baixado deve-se localizar em seu conteúdo um diretório de nome *axis*. Deve-se então copiar esse diretório inteiramente para o diretório *WEBAPPS* do servidor Apache Tomcat (tipicamente, esse diretório está localizado em `C:\Arquivos de programas\Apache Software Foundation \Tomcat 6.0\webapps`). Cada WS deve ser uma classe Java salva em um arquivo com extensão modificada de `.java` para `.jws` (Java Web Service - JWS). Essa classe deve ser salva no recém reposicionado diretório `axis` e já estará disponível para acesso dos clientes.

No presente trabalho criou-se uma classe JWS chamada de `Servico` através do código Java: `public class Servico {}`. Dentro dessa classe, sem necessidade de construtores ou atributos privados, cada função já estará disponível para ser acessada por um determinado cliente. Para os devidos fins foi implementada uma função denominada `retornaAlunoPorMatricula` que retorna um vetor e recebe como entrada a matrícula de um aluno. O código comentado dessa função está exibido a seguir e realiza uma consulta MySQL na tabela `alunos` buscando registro de alunos com uma determinada matrícula.

```
    public ArrayList retornaAlunoPorMatricula(String matricula) {
Statement statement;
ResultSet resultSet;
String url = "jdbc:mysql://localhost:3306/webservice";
String username = "root";
String password = "root";
String query = "";
Connection conn;
ArrayList lista = new ArrayList();
String [] result = new String[4];
result[0] = "codigo";
result[1] = "matricula";
result[2] = "nome";
result[3] = "curso";
try {
  \\pega driver mysql
```

```
   Class.forName("com.mysql.jdbc.Driver");
   \\conecta com BD
   conn = DriverManager.getConnection(url, username, password );
   \\monta query baseado no parametro matricula
   query = " SELECT codigo,matricula,nome,curso from " +
   "alunos where matricula =  "+matricula;
   \\prepara statement
   PreparedStatement stmt = conn.prepareStatement(query);
   \\executa query
   ResultSet rs = stmt.executeQuery();
   int i=0;
   \\coloca resultados da query no retorno
   if (rs.next()){
      result[0] = rs.getString(1);
      result[1] = rs.getString(2);
      result[2] = rs.getString(3);
      result[3] = rs.getString(4);
      lista.add(i++,result);
    }
    rs.close();
    stmt.close();
   }
   catch ( ClassNotFoundException cnfex ) {
    result[0]="not found";
    System.err.println("Failed to load JDBC/ODBC driver." );
   }
   catch ( SQLException sqlex ) {
    result[0]="not connect";
   }
 \\retorna valores
 return lista;
}
```

Assim está criado e disponibilizado um WS através da linguagem JSP que utiliza na verdade classes descritas em Java na sua implementação.

4.2. Cenário: *RU* - Restaurante Universitário

O objetivo principal deste cenário é possibilitar a consulta, por parte de qualquer cliente, a base de dados *RU*. Um exemplo de uso deste cenário é uma aplicação cliente disponibilizada por uma universidade em todos os computadores de seu parque tecnológico que faça consulta aos diversos restaurantes localizados em suas dependências que disponibilizem WS contendo o cardápio do dia. As próximas Seções detalham as implementações necessárias para este cenário: cliente e servidor.

4.2.1. Cliente

Conforme descrito na Seção 3. onde apresenta-se a implementação dos clientes, o cliente deste cenário foi desenvolvido em Java.

Inicialmente foi descrita uma classe Java denominada de Cliente e dentro dessa classe foi desenvolvido um método chamado `callWebService` que tem como parâmetros de entrada o local do servidor WS e o parâmetro a ser enviado. Para a utilização das chamadas de WS em Java deve-se instalar no `classpath` do sistema o caminho para a API *Axis*. Inicialmente no método citado é realizada a criação de um serviço e a criação de uma chamada para o servidor WS:

```
     static void callWebService( String url,  String param) throws
ServiceException, RemoteException {
Service  service = new Service();
Call  call  = service.createCall();
```

Após realizar a criação da chamada deve-se configurar alguns itens: endereço do WS, nome da função a ser executada (disponibilizada pelo servidor), adicionar o nome e tipo do parâmetro requerido pela função, ajustar o tipo de retorno esperado. O código Java responsável por realizar essas configurações está exibido a seguir:

```
     call.setTargetEndpointAddress( url );
call.setOperationName( new QName("retornaCardapio" ) );
call.addParameter( "dia", XMLType.XSD_STRING, ParameterMode.IN );
call.setReturnType(new QName("ArrayOfString"),ArrayOfString.class);
```

Antes de continuar é necessário explicar que o tipo de retorno aqui utilizado é um tipo próprio sendo que podem ser usados tanto os tipos já conhecidos quanto tipos próprios. Como para tipos próprios não é necessário setar explicitamente o tipo de retorno, escolheu-se exemplificar o "pior caso", onde a função do servidor WS retorna um tipo "não padrão" de dado. Para isso, deve-se criar uma segunda classe, nesse caso *ArrayOfString* que conterá métodos serializadores e desserializadores responsáveis por controlar o parâmetro de retorno. Essa classe é conhecida por ter o metamodelo do novo tipo de dado a ser retornado. O código está comentado e exibido a seguir.

```
     public class ArrayOfString {
// Type metadata
private static org.apache.axis.description.TypeDesc typeDesc = new
org.apache.axis.description.TypeDesc(ArrayOfString.class);
//cria explicitamente o novo tipo.
//do servidor WS está retornando um "item" cujo tipo é uma string.
static {
   typeDesc.setXmlType(new javax.xml.namespace.QName(
"", "ArrayOfString"));
   org.apache.axis.description.ElementDesc elemField = new
   org.apache.axis.description.ElementDesc();
   elemField.setFieldName("item");
```

```
    elemField.setXmlName(new javax.xml.namespace.QName("", "item"));
    elemField.setXmlType(new javax.xml.namespace.QName("", "String"));
    typeDesc.addFieldDesc(elemField);
}
//'Array' contem varios 'items'
private String[] Array;
private String item;
private java.lang.Object __equalsCalc = null;
private boolean __hashCodeCalc = false;
//metodos basicos
public ArrayOfString() {
    Array = null;
    item = null;
}
public String[] getArray() {
    return Array;
}
public void setArray(String[] array) {
    this.Array = array;
}
public String getItem() {
    return item;
}
public void setItem(String item) {
    this.item = item;
}
//metodos necessarios para serializaacao e desserializacao
public synchronized boolean equals(java.lang.Object obj) {
    if (!(obj instanceof ArrayOfString)) {
        return false;
    }
    ArrayOfString other = (ArrayOfString) obj;
    if (obj == null) {
        return false;
    }
    if (this == obj) {
        return true;
    }
    if (__equalsCalc != null) {
        return (__equalsCalc == obj);
    }
    __equalsCalc = obj;
    boolean _equals;
    _equals = true && (item == other.getItem())
        && (Array == other.getArray());
    __equalsCalc = null;
```

```
   return _equals;
}
public synchronized int hashCode() {
   if (__hashCodeCalc) {
      return 0;
   }
   __hashCodeCalc = true;
   int _hashCode = 1;
   _hashCode += item.hashCode();
   _hashCode += Array.hashCode();
   __hashCodeCalc = false;
   return _hashCode;
}
//retorna o objeto do tipo de dado
public static org.apache.axis.description.TypeDesc
 getTypeDesc() {
   return typeDesc;
}
//metodo serializador
public static org.apache.axis.encoding.Serializer \—
getSerializer(
   java.lang.String mechType, java.lang.Class _javaType,
   javax.xml.namespace.QName _xmlType) {
   return new org.apache.axis.encoding.ser.BeanSerializer(\—
 _javaType,_xmlType, typeDesc);
}
//metodo deserializador
public static org.apache.axis.encoding.Deserializer \—
getDeserializer(
   java.lang.String mechType, java.lang.Class _javaType,
   javax.xml.namespace.QName _xmlType) {
   return new org.apache.axis.encoding.ser.BeanDeserializer(\—
 _javaType,_xmlType, typeDesc);
}
}
```

Voltando ao método da classe Cliente deve-se então invocar a função disponibilizada pelo servidor e exibir os resultados pertinentes:

```
ArrayOfString resu = (ArrayOfString)call.invoke(
new Object[] {param} );
System.out.println(resu.getItem());
```

Assim, um cliente Java que aceita tipos definidos pelo usuário como retorno foi criado. É importante destacar novamente que optou-se por exibir o caso mais complicado relacionado ao retorno do tipo de dados principalmente pela escassez de material acerca desse item na Internet. A seguir é apresentado o servidor para este cenário.

4.2.2. Servidor

Conforme descrito na Seção 3. onde apresenta-se a implementação dos servidores, o servidor deste cenário foi desenvolvido em PHP5 no Linux. Como o PHP5 possui suporte nativo a WS a criação e disponibilização de um servidor WS é bastante simplificada. Inicialmente deve-se explicitar a localização do WS a ser criado pelo PHP:
`$server = new SoapServer(null, array('uri' => 'http://localhost/webservice/PHP_Server/'));`

Após realizar a instanciação de um objeto da classe SoapServer deve-se descrever a função a ser disponibilizada no WS. Essa função `retornaCardapio` requer como parâmetro de entrada uma string que descreve o dia da semana que deseja-se saber o cardápio. Então, utilizou-se uma classe extra `dboperations2.php` (a fim de exemplificar que pode-se modularizar as funcionalidades) responsável pela conexão com um banco de dados MySQL e pelo retorno das informações. Após consultar o banco, a função retorna um vetor com os dados encontrados.

```
     function retornaCardapio($dia){
include("../db/dboperations2.php");
$dias = array("domingo" => "0", "segunda" => "1", "terca" => "2",
  "quarta" => "3", "quinta" => "4","sexta" => "5", "sabado" => "6");
 $dbOperations = new DBOperations;
  $dbOperations->connect();
  $arrayCardapio = $dbOperations->getRecordsFromTableClause(
   "*", "ru"," codigo >= 1 and dia_semana = ".$dias[$dia].""","");
  $dbOperations->close();
return  $dbOperations->retornaVetor(&$arrayCardapio);
}
```

Uma vez que a função tenha sido criada de maneira correta deve-se adicioná-la explicitamente ao WS através do código:
`$server -> addFunction("retornaCardapio");`

Finalmente, faz-se um controle onde somente é executada essa função (através de `handle`) quando existir um método de requisição:

```
if($_SERVER["REQUEST_METHOD"] == "POST"){
 $server -> handle();
}else{
  $functions = $server->getFunctions();
  foreach($functions as $func ){
     print $func;
  }
}
```

Assim o servidor está disponibilizado e pode ser consultado por quaisquer clientes

WS que se conectem a ele. Na próxima Seção são apresentadas as conclusões do trabalho.

5. Conclusões

Os WS trazem um conjunto de tecnologias de uso distribuído com soluções facilitadoras para aplicacoes em TI. Essas aplicacões vão desde a obtencão de informacões específicas sobre um dado servidor em uma intranet até a manutencão de bases de dados distantes através da internet.

No presente trabalho foram implementados dois cenários distintos utilizando sistemas e linguagens de programação heterogêneos. No primeiro cenário, cujo cliente WS foi implementado em PHP e servidor WS foi implementado em JSP, visava-se a consulta - através do número de matrícula - acerca de um determinado estudante (se pertencia ou não a instituição e se tinha permissão ou não para utilizar as dependências do Restaurante Universitário). Já no segundo cenário, cujo cliente WS foi uma aplicação Java e servidor WS foi implementado em PHP, visava-se exemplificar uma aplicação que estivesse presente em todos os computadores da universidade que fazia consulta a diversos WS disponibilizados pelos seus restaurantes que forneciam informações sobre o cardápio do dia.

Por fim, através da implementação de WS é possível observar que os são solucões relevantes para os problemas de comunicacão entre SD's uma vez que permitem a interoperabilidade de sistemas heterogêneos distribuídos em diferentes locais e que podem ser acessados de diversas formas (aplicações web e não-web).

References

5, P. (2007). Php. Disponível por www em: http://www.php.net/. Acesso em 7 de abril.

Apache (2007). Apache. Disponível por www em: http://www.apache.org/. Acesso em 1 de abril.

Axis (2007). Webservices - axis. Disponível por www em: http://ws.apache.org/axis/. Acesso em 7 de abril.

Becker, A. (2007). Web services e xml. um novo paradigma da computacao distribuída. Disponível por www em: http://www.inf.ufsc.br/ danclaro/. Acesso em 8 de abril.

Java (2007). Java. Disponível por www em: http://java.sun.com/. Acesso em 9 de abril.

JSP (2007). Jsp. Disponível por www em: http://java.sun.com/products/jsp/. Acesso em 9 de abril.

Microsoft (2007). Windows xp. Disponível por www em: http://www.microsoft.com/windowsxp/default.asp. Acesso em 7 de abril.

MySQL (2007). Mysql. Disponível por www em: http://www.mysql.com/. Acesso em 10 de abril.

SOAP (2007). Webservices, soap e aplicações web. Disponível por www em: http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html. Acesso em 2 de abril.

Tomcat, A. (2007). Tomcat. Disponível por www em: http://tomcat.apache.org/. Acesso em 10 de abril.

Ubuntu (2007). Linux. Disponível por www em: http://www.ubuntu.com/. Acesso em 2 de abril.

WebServices (2007). Webservices. Disponível por www em: http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html. Acesso em 6 de abril.

WSDL (2007). Wsdl. Disponível por www em: http://www.w3schools.com/wsdl/wsdl_intro.asp. Acesso em 29 de marco.