Web Services usando RPC y PHP

Laura Berger[†], **Matias Borgeaud**[†], **Javier Echaiz**[††]
[†]Facultad de Ingeniería, Universidad Nacional de La Pampa, General Pico (6360), Argentina
{berger,borgeaud}@ing.unlpam.edu.ar
[††]Laboratorio de Investigación en Sistemas Distribuidos (LISiDi)
Universidad Nacional del Sur, Bahía Blanca (8000), Argentina
je@cs.uns.edu.ar

Resumen

Por más que lo parezcan, los Web Services no son una cosa muy complicada. Básicamente estos servicios permiten el intercambio de datos entre un servidor y un cliente, "empaquetados" según algún formato estándar de forma que ambos sistemas puedan "entenderse". PHP (PHP Hypertext Preprocessor) es uno de los lenguajes más utilizados en los desarrollos para la Web y es además una buena alternativa para experimentar con Web Services. Por otra parte, las RPCs (Remote Procedure Calls) son un mecanismo muy sencillo (pero potente) de comunicación y especialmente útiles para implementar Web Services. En este trabajo se presenta una serie de conceptos básicos para entender cómo funcionan los Web Services y una implementación sencilla usando, en combinación, RPC, PHP y XML.

Palabras claves: Web Services, RPC, XML, PHP.

1. INTRODUCCIÓN

En primer lugar, presentaremos una introducción de los conceptos básicos detrás de los Web Services. Luego mostraremos dos alternativas para estandarizar el formato de los mensajes que son pasados entre el cliente y el servidor (y viceversa), de modo que el diálogo entre ambos se haga "en un mismo idioma": SOAP y XML-RPC. Aplicando esta última, haremos una implementación en PHP de un Web Service para resolver una expresión algebraica.

Usaremos el siguiente modelo para representar una expresión:

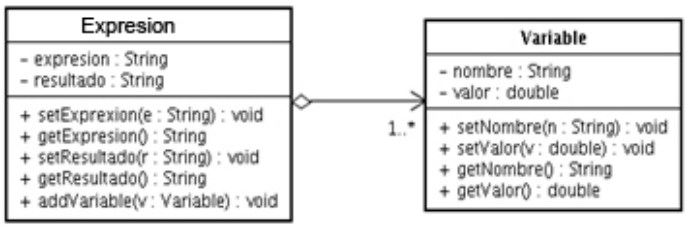

Figura 1. Diagrama de clases (modelo de expresión/variable)

La implementación (simplificada) en PHP del modelo de la Figura 1 (`solver.php`) sería:

```
<?php
class Expresion {
  var $expresion;
  var $resultado;
  var $variables;
  function Expresion($str="") {...}
  function getExpresion(){...}
  function getResultado(){...}
  function addVariable($v){...}
}

class Variable{
  var $nombre;
  var $valor;
  function Variable(){...}
  function setNombre($n){...}
  function setValor($v){...}
  function getNombre(){...}
  function getValor(){...}
}
?>
```

2. ¿Qué es un Web Service?

Por más que lo parezcan, los Web Services no son una cosa muy complicada. Básicamente, estos servicios permiten el intercambio de datos entre un servidor y un cliente, "empaquetados" según algún formato estándar (posiblemente XML), de forma que ambos sistemas puedan "entenderse". El servidor podría ser un servidor web y el cliente, un usuario (a través de un browser). O podría

tratarse de dos servidores web, o incluso cualquier dispositivo electrónico con conexión a la red [10] [11].

Dicho de otra manera, los Web Services operan básicamente de la misma forma que un navegador Web cuando hacemos el POST de un formulario y recibimos la respuesta del sitio en donde está el form (en forma de una página HTML); los datos son intercambiados a través del puerto 80/TCP usando los estándares del protocolo HTTP para ello. La única diferencia es que, en lugar de HTML, los Web Services, usan XML (eXtensible Markup Language) [1].

XML permite que el servicio pueda ser portado a cualquier plataforma, sistema operativo, software de servidor y de cliente, e incluso, por ser sólo texto, atravesar dispositivos de control (como por ejemplo firewalls) de la misma forma en la que lo hacen las páginas Web.

Si vemos a un Web Service como una arquitectura en capas, la capa superior (la capa más externa) denominada capa de descripción, ofrece información que describe la interface (API, Application Program Interface) del servicio. Esta capa permite que los clientes usen el servicio y que se desarrollen otros programas que puedan acceder a ellos. El lenguaje estándar en el que se programa esta capa se llama WDSL (Web Service Description Language). Una segunda capa es la capa de descubrimiento, que permite describir la naturaleza del servicio en sí mismo, de manera de poder ser categorizado y encontrado en sitios de ubicación de Web Services (directorios). En esta capa, el estándar usado se denomina UDDI (Universal Description, Discovery and Integration). Estas dos capas, se programan básicamente en XML con un formato particular. Otra capa, quizá la más importante, es la denominada capa de empaquetado, que realiza funciones de conversión de los datos intercambiados en un formato estándar y común a ambos, cliente y servidor [9].

SOAP es el estándar propuesto por la W3C (World Wide Web Consortium) para este propósito. Un estándar Open Source, alternativo a SOAP, es XML-RPC. Este último, fue desarrollado por la empresa Useful Inc. junto a Microsoft en 1998, y a diferencia de SOAP, cuenta con 1500 especificaciones de lenguaje (contra las más de 11000 de SOAP), lo que lo hace mucho más sencillo. Entre otras ventajas, quizás la más importante es la facilidad de integración que tiene con lenguajes de alto nivel, como, por ejemplo, PHP (Hypertext Preprocessor) [2] [3] [4] [5].

Más allá de las ventajas y desventajas, claro está que un estándar no reemplaza al otro y, por el contrario, son alternativas que han podido convivir como tales. Incluso, se pueden transformar solicitudes de servicio SOAP a XML-RPC (y viceversa) usando XSLT (XML-SOAP Language Transformation) y así intercomunicar a las diferentes tecnologías [6].

Este trabajo, estará centrado en XML-RPC y en cómo funciona, para luego abordar una implementación de un Web Service en PHP.

Un detalle que no debemos dejar pasar por alto es la sigla "RPC" (Remote Procedure Call, llamada a procedimiento remoto) en XML-RPC. Un cliente hace una "llamada" a un "procedimiento" (o método) en un servidor "remoto". XML sólo es el estándar para intercambiar datos en una operación RPC.

Otro detalle importante, tiene que ver con las decisiones de implementación respecto de las capas de descripción y descubrimiento en XML-RPC. Aquí uno, virtualmente, puede hacer lo que quiera con ellas. Básicamente, en la capa de descripción se deberá proveer información acerca de los métodos (y parámetros) que ofrece el servidor. La capa de descubrimiento, por el momento, no se mantiene en XML-RPC y no hay un registro central (directorio) en donde se publiquen los servidores XML-RPC con sus servicios respectivos. Sin embargo, el estándar UDDI podrá implementarse de todos modos cuando comience a ser necesario [7].

En síntesis, el funcionamiento de XML-RPC, se diseñó en torno al "mensaje" RPC que intercambian cliente y servidor. Las dos principales componentes de un mensaje son los métodos y los parámetros. Los métodos son una especie de función, que recibe parámetros a los cuales podemos asociar con las variables de la función. La cantidad y tipo de parámetros puede ser cualquiera: strings, enteros, arreglos, etc. Hasta aquí todo parecería ser similar a la definición de funciones y argumentos, por ejemplo, en PHP.

Un "diálogo" XML-RPC entre dos sistemas, comienza con un requerimiento del cliente, al cual, un servidor responde con una respuesta. Un requerimiento consistirá de un método y, tal vez, de parámetros requeridos por él. La respuesta devuelve uno o más parámetros con los datos requeridos. Y también hasta aquí, el proceso parecería ser similar a un script PHP: se llama a una función pasándole algunos parámetros. La función, luego, responde retornando algún resultado.

El siguiente es un ejemplo de un servidor XML-RPC que acepta como entrada un modelo de datos que permite representar una expresión matemática conteniendo la expresión a evaluar, incluyendo las variables y sus correspondientes valores. Por ejemplo, debe ser posible calcular la expresión "x^2+ 3y – 5", especificando las variables "x" e "y", cada una con sus valores.

El cliente se conecta con el servidor a través del portal del Web Service (la interface del servicio) e interactúa, a través de ésta, enviando y recibiendo mensajes. Para el ejemplo, esta interface podría ser:

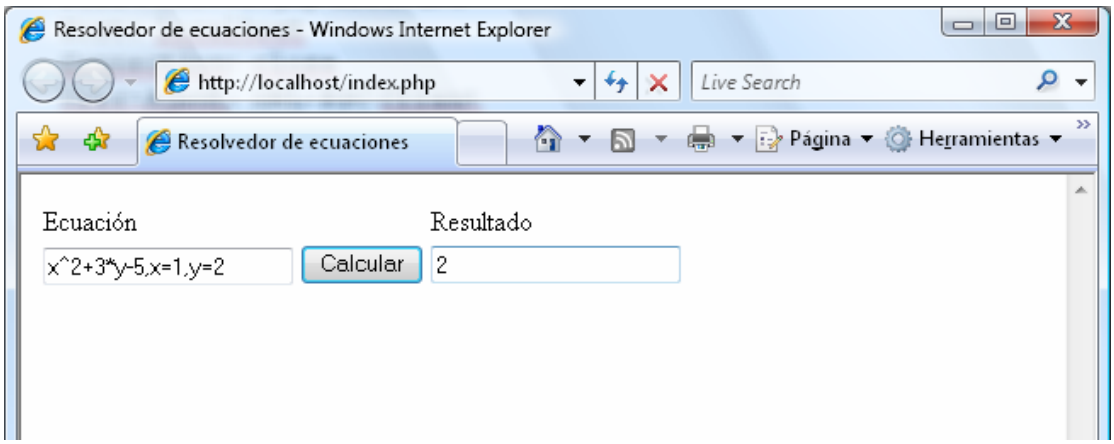

Figura 2. Interface del servicio

Cuando el cliente hace el "post" del formulario, la librería XML-RPC prepara el mensaje que enviará al servidor. Suponiendo que el cliente envía la expresión x^2+3*y-5, con las asignaciones x=1 e y=2, el mensaje "empaquetado" en XML, se verá como:

```
POST /server.php HTTP/1.0
Host: localhost
Connection: close
User-Agent: XML-RPC Client
Content-Type: text/xml
Content-Length: 206
<?xml version="1.0" ?>
<methodCall>
  <methodName>solve</methodName>
    <params>
    <param>
      <value>
        <string>x^2+3*y-5,x=1,y=2</string>
      </value>
    </param>
    </params>
</methodCall>
```

En respuesta, el servidor XML-RPC, le enviará al cliente lo siguiente:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 154
content-Type: text/xml
Date: Wed, Apr 23 2008 09:56:04 GMT
Server: XML-RPC Server
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
          <int>2</int>
        </value>
    </param>
  </params>
</methodResponse>
```

3. Implementación del Web Service

Presentaremos a continuación una implementación simplificada para el ejemplo antes mostrado. Definida la clase `Expresion` con los métodos que permiten manejar expresiones y variables que la integran dispondremos del método `Solve`, que instanciará la clase `Expresion` y finalmente obtendrá el resultado.

Para publicar el servicio, hemos instalado un servidor web con las librerías necesarias para soporte de PHP (versión 4.1+) y XML-RPC (versión 2.5f de Keith Devens [8]).

3.1. El Servidor XML-RPC

Como hemos dicho, el servidor XML-RPC se encarga básicamente de recibir los parámetros de entrada (parseando el request) y de ejecutar la llamada al procedimiento. Para ello, hemos implementado el siguiente código (`server.php`):

```
<?php
  include("kd_xmlrpc.php");
  include("web_service_api.php");
  $xmlrpc_request = XMLRPC_parse( $GLOBALS['HTTP_RAW_POST_DATA']);
  $methodName = XMLRPC_getMethodName($xmlrpc_request);
  $params = XMLRPC_getParams($xmlrpc_request);
  if(!isset($xmlrpc_methods[$methodName]))
    $xmlrpc_methods['method_not_found'] ($methodName);
  else
    $xmlrpc_methods[$methodName]($params);
?>
```

El código comienza con la inclusión de la librería XML-RPC `kd_xmlrpc.php`, y luego del archivo `web_service_api.php` que contiene la descripción de los métodos exportados. Al ser invocado, este script hace el parse del request HTTP `XMLRPC_parse`, identifica los métodos invocados `XMLRPC_getMethodName` y lo ejecuta `$xmlrpc_methods[$methodName]()`, si existe, o llama al error correspondiente `$xmlrpc_methods['method_not_found']()`.

3.2. Métodos Exportados (API)

Un componente importantísimo de los Web Services es su API. En ella definimos los métodos exportados, o publicados, para que los clientes usen.

```
<?php
  include('solver.php');

  /* Métodos exportados */
  $xmlrpc_methods = array();
  $xmlrpc_methods['solve'] = Solve;
  $xmlrpc_methods['method_not_found'] = XMLRPC_method_not_found;

  function Solve($iparams) {
    $e = new Expresion($iparams[0]);
    $oparam = $e->getResultado();

    $oparams[] = $oparam;
```

```
    XMLRPC_response(
      XMLRPC_prepare($oparams),
      'XML-RPC Server'
    );
  }
  function XMLRPC_method_not_found ($methodName) {
    XMLRPC_error("2", "El método $methodName no existe",KD_XMLRPC_USERAGENT);
  }
?>
```

En la API del Web Service del ejemplo (`web_service_api.php`), hemos definido un arreglo para contener todos los métodos exportados. Además del método `Solve`, que se encargará del cálculo de la expresión, definimos un método `method_not_found`, que se ejecuta para indicar el error "el método invocado no existe", para cualquier método no definido. La diferencia en cómo un método definido resuelve la respuesta y cómo lo hace `method_not_found` está en la invocación de `XMLRPC_response`, en el primer caso y de `XMLRPC_error`, en el segundo.

3.3. El Cliente XML-RPC

Por su parte, el script del cliente (`index.php`) será el responsable de llamar al procedimiento remoto `Solve` para obtener el resultado de una expresión que le pasará al servidor, junto con los valores de las variables que la integran.

```
<?php
  if($_REQUEST['expr']){
    include("kd_xmlrpc.php"); /* Inclusión de la librería */
    $site = "127.0.0.1"; /* Variables de identificación del servidor */
    $location = "/server.php";
    // XMLPRC_request hace el POST XML al servidor llamando al método
    // con sus parámetros adaptados con XMLRPC_prepare
    list($success,$response) =
      XMLRPC_request(
        $site,
        $location,
        'solve', //método
        array(XMLRPC_prepare($_POST['expr'])), //parámetros
        'XML-RPC Client' //user-agent
      );
    if(!$success)  echo $response['faultString'];
}
?>
<html>
<head>
  <title>Resolvedor de expresiones</title>
</head>
<body>
  <form action="" method="post">
```

```
    <table>
    <tr>
      <td>Expresión</td><td>Resultado</td>
    </tr>
    <tr>
      <td>
       <input type="text" name="expr" value="<?=$_REQUEST['expr']?>">
       <input type="submit" value="Calcular">
      </td>
      <td><input type="text" name="result" value="<?=$response[0]?>"></td>
    </tr>
    </table>
  </form>
</body>
</html>
```

En la llamada a `XMLRPC_request` se pasan cinco parámetros. El primero es el sitio en donde se encuentra el servidor (para el ejemplo 'localhost'). El segundo es la ruta al archivo del script del servidor, relativa al root del webserver (en este caso '`/server.php`'). El tercer parámetro corresponde al nombre de la llamada al procedimiento remoto `Solve`. El cuarto es un array que contiene los parámetros que se pasan al procedimiento; cada uno de ellos es antes normalizado por `XMLRPC_prepare()`. El quinto y último parámetro es el User Agent con el que queremos que se identifique el cliente `XML-RPC`, para el ejemplo, '`XML-RPC Client`'.

3.4. La clase kd-XML-RPC

Todo el funcionamiento detrás de la comunicación RPC y de la conversión XML está soportado por la clase `kd_xmlrpc.php`, la cual es incluida por los demás scripts. Entre las funciones más importantes que ofrece, están las siguientes:

- `XMLRPC_prepare`, para normalizar los parámetros.
- `XMLRPC_request`, handle del requerimiento del cliente y dispara la respuesta del servidor.
- `XMLRPC_response`, produce la respuesta del servidor.
- `XMLRPC_getMethodName`, retorna el método invocado en la RPC.
- `XMLRPC_getParams`, retorna los parámetros que se pasaron en la llamada del procedimiento.
- `XMLRPC_error`, retorna un error, en caso de ocurrir.

4. Conclusiones

XML-RPC es una interesante alternativa para desarrollar aplicaciones distribuidas a través de la Internet. En el presente trabajo hemos intentado explicar cómo crear clientes y servidores XML-RPC usando el lenguaje PHP.

Hemos visto que XML-RPC es una implementación del protocolo RPC, usando XML y HTTP como agente de transporte. Por definición, en la arquitectura, los requests y sus respuestas son definidos en XML. De esta forma, el protocolo permite la comunicación transparente sobre diferentes sistemas operativos y diferentes lenguajes.

Si bien el presente es un trabajo introductorio, estas tecnologías pueden ser aplicadas para casos más complejos como una alternativa sencilla y robusta a otras más difundidas como es el caso de SOAP.

En el futuro, prevemos desarrollar nuevas propuestas relacionadas con XML-RPC para abarcar situaciones más complejas, como podría ser el interesante caso de un webservice actuando como cliente de otro webservice. Adicionalmente se planteará una serie de mecanismos que permitan publicar y exponer webservices implementados en XML-RPC al estilo de los directorios UDDI, con las ventajas que ofrece la simplicidad de las tecnologías abarcadas en este trabajo.

REFERENCIAS

[1] XML.org Focus Area community. http://www.xml.org/

[2] The World Wide Web Consortium. http://www.w3c.org

[3] XML-RPC Home Page. http://www.xmlrpc.org

[4] Kate Rhodes, XML-RPC vs. SOAP. http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm

[5] Sitio oficial de PHP. http://www.php.net

[6] XSL Transformations (XSLT), W3C. http://www.w3.org/TR/xslt

[7] Online community for the Universal Description, Discovery, and Integration OASIS Standard http://www.uddi.org/

[8] Keith Deven's XML-RPC Implementation, 2005. http://keithdevens.com/software/xmlrpc

[9] Bård Farstad, eZ xmlrpc. http://www.zez.org/article/articleprint/47/

[10] Coulouris G., Dollimore J., Kindberg T. Distributed Systems Concepts and Design. Third Edition. Addison-Wesley, 2001.

[11] Tanembaum, A. Distributed Operating Systems. Prentice-Hall International, 1995.