Introducción

Además de como gestor de contenido, Drupal también puede funcionar sirviéndose de contenido externo a través de otras aplicaciones Web, tales como Twitter, Facebook o Google. Esta comunicación entre Drupal y aplicaciones externas es lo que hace del primero una potente plataforma de gestión de contenido, tanto propio como ajeno. Por ejemplo, un desarrollador Drupal puede alimentar su aplicación con contenido a través de *Aggregation* o *feeds RSS*.

Uno de los métodos más robustos para utilizar contenido externo es el módulo *FeedAPI (Application Programming Interface)* ya que permite fácilmente utilizar RSS para ser usado desde Drupal. Otro módulo importante para la integración de Drupal con Servicios Web es *Service* ya que nos ayudará a conectarnos con Servicios Web utilizando diferentes protocolos, como XMLRPC, JSON, JSON-RPC, REST, SOAP y AMF. En este sección revisaremos cómo interactuar nuestro sitio Drupal utilizando estas interfaces.

En esta sección además aprenderemos:

- ¿Qué son los Servicios Web y para qué se utilizan?
- ¿Para qué y cómo utiliza Drupal Servicios Web?
- Drupal como proveedor y consumidor de Servicios
- Normas para desarrollar y utilizar Servicios Web.

1. ¿Qué son los Servicios Web?

Para conseguir que nuestro sitio Drupal se comunicque e interactue con otras aplicaciones web como Facebook, Twitter o Google, necesitamos utilizar una ser de protocolos estándares de comunicación, llamados Servicios web. Los Servicios web proporcionan interactuación y comunicación entre aplicaciones web tanto remota como localmente. Es decir, podemos utilizar Servicios web de Flickr para importar todas nuestras fotografías de estas vacaciones, en nuestra aplicación Drupal.

Las llamadas a Servicios Web ocurren a través de protocolos cifrados y son traducidos en un lenguaje estándar que son comprendidos por ambas partes, utilizando normalmente XML para este propósito. XML es un formato basado en texto, por lo que casi todos los sistemas informáticos y las aplicaciones pueden trabajar con dicho formato.

El protocolo de Servicios Web está basado en un concepto conocido como **Remote Procedure Calling (RPC)** que permite a una aplicacion inicializar o "llamar" a una función dentro de una aplicación que se encuentra en un servidor remoto. Por ejemplo, nuestro sitio Drupal puede llamar a un servicio Web de Twitter para obtener nuestros últimos tweets y publicarlos automáticamente en nuestro sitio, o moverlos a nuestro tablón de Facebook, utilizando los Servicios Web de este último.

1.1. XML y Servicios Web

Como se ha mencionado, los Servicios Web utilizando el lenguaje de marcado XML para comunicarse. Nuestra aplicación Drupal y el proveedor de Servicios Web utilizando el protocolo estándar de comunicación HTTP para enviarse mensajes utilizando el lenguaje XML. Se utiliza XML porque es un lenguaje común entre todos las aplicaciones Web y éste reemplaza al lenguaje propietario en el que el servicio Web fue escrito y el protocolo de comunicación que utilice. Así las aplicaciones se entienden más fácilmente. Quizás se entienda mejor con esta analogía: si dos personas, un Español y un Ruso hablan entre si y ninguno conoce el lenguaje del otro, lo usual es hablar un lenguaje que hablen ambos, como el Inglés.

Algunos ejemplos de protocolos de comunicación con Servicios Web que veremos en el capítulo son:

- SOAP (Simple Object Access Protocol),
- UDDI (Universal Description, Discovery and Integration),
- WSDL (Web Services Description Language),
- XML-RPC (XML Remote Procedure Call),
- JSON (JavaScript Object Notation),
- JSON-RPC,
- REST (Representational State Transfer) y
- AMF (Action Message Format)

1.2. El protocolo REST

El protocolo REST (*Representational State Transfer* o *Transferencia de Estado Representacional*) es una técnica para intercomunicar diferentes aplicaciones Web. En él se definen un conjuntos de bases por las cuales se diseñan Servicios Web haciendo teniendo en cuenta los recursos del sistema, el acceso al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. La principal característica de REST es la simplicidad, calidad por la que logró desbancar a SOAP y las interfaces basadas en WSDL.

Los pilares de REST son:

- utiliza métodos de HTTP para comunicarse.
- no mantiene estado.
- utiliza URIs como si se tratase de directorios.
- se comunica a través de mensajes escritos tanto en XML, JSON o ambos.

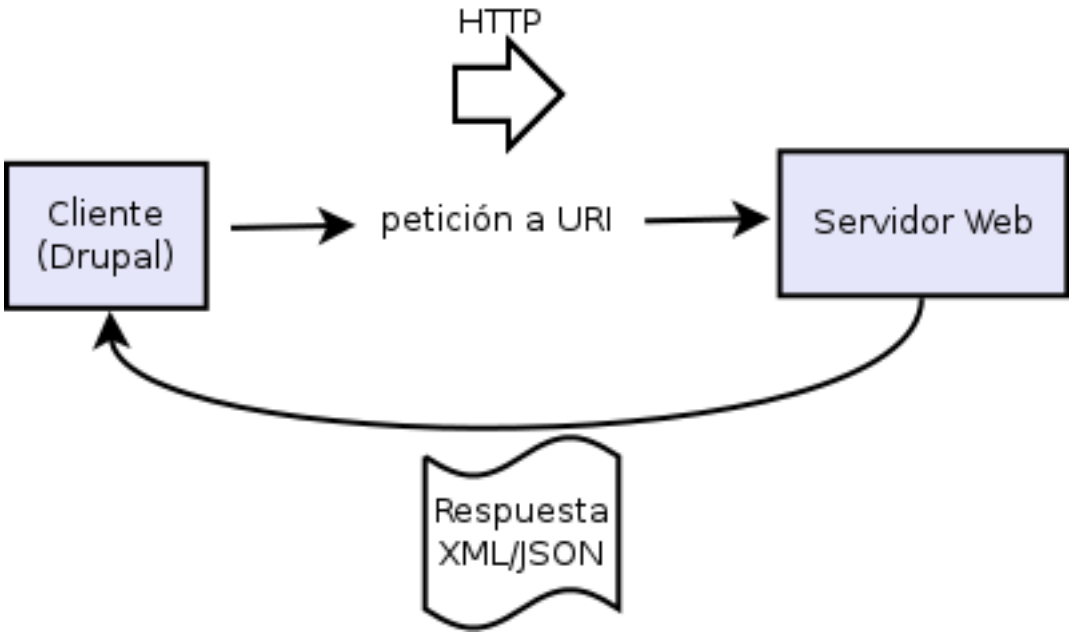


Figura 1: Ejemplo de Rest

Un ejemplo de la potencia y facilidad con la que trabajamos utilizando REST la podemos encontrar en la integración de Drupal con Twitter: podemos pedir los últimos *tweets* de una cuenta, recibirlos y mostrarlos en nuestra aplicación Drupal. Para ello, pediremos este contenido a través de una petición GET a una URI de Tweet, que nos responderá con el contenido en formato XML o JSON, el cual ya podremos procesar y utilizar en nuestra aplicación.

2. ¿Son útiles los Servicios Web para Drupal?

Está claro que si lo que necesitamos es utilizar contenido de otras aplicaciones Web, necesitamos hacer uso de Servicios Web. Debido a la interoperabilidad de éstos, podemos comunicar nuestra aplicación Drupal con cualquier otra aplicación, sea Drupal o no. Utilizando Drupal como proveedor de Servicios Web, además podremos compartir nuestro contenido con el resto del mundo y no será específico de nuestro sitio.

Integrando, por ejemplo, nuestra aplicación Drupal con Picassa, podemos subir automáticamente todas las fotos que utilicemos en nuestro sitio a un álbum público, consiguiendo ser indexados también a través de nuestras imágenes. Otro ejemplo lo podemos tener con Linkedin: utilizando los Servicios de la famosa red de profesionales podremos compartir contenido de nuestra aplicación a nuestro perfil, directamente

Además, podemos servirnos de datos externos para hacer nuestro sitio más

rico: podemos utilizar Servicios de un centro metereológico o utilizar la herramienta de traducción o los mapas de Google para convertir nuestra aplicación en un sitio robusto y funcional.

3. ¿Cómo utiliza Drupal Servicios Web?

Drupal puede utilizar los protocolos que antes mencionamos, incluyendo SOAP, REST y XML-RPC, así como servirse de Servicios cuyos mensajes hayan sido codificados usando RSS y XML. Para ello, se pueden escribir clientes de comunicación propios o utilizar el módulo *Services* o cualquier otro módulo de integración con Servicios. En este capítulo veremos cómo funciona Drupal como un consumidor de Servicios Web, es decir, cómo consume Drupal recursos de un servidor Web externo. Además, haremos un repaso de cómo implementar que Drupal sirva contenido a otras aplicaciones Web, es decir, cómo hacer que Drupal sea un proveedor de Servicios Web.

Drupal como consumidor de Servicios Web

Anteriormente hemos citado que nuestro sitio Drupal puede utilizar contenido externo a través de Servicios Web, utilizando los protocolos XML-RPC, REST, JSON, XML y RSS. Primero revisaremos con detalle un protocolo que puede ser usado por Drupal para comunicarse: el protocolo **SOAP (*Simple Object Access Protocol*)**. Utilizaremos el módulo **Web Service Client**, ya que nos proporciona una interfaz para integrarnos con Servicios Externos. Se centra en la integración de operaciones de Servicios Web como acciones de reglas, pero también se puede utilizar como proveedor de datos para otros módulos. Debido a estas acciones, necesitaremos además el módulo **Rules**. La principal característica del módulo *Web Service Client* es que viene con una interfaz de usuario que permite a los administradores del sitio crear descripciones de servicios REST y SOAP con sus respectivas operaciones y tipos de datos. Además, gracias a este módulo, podemos utilizar el **WSDL (*Web Service Description Language*)** del proveedor de Servicios Web.

WSDL es un lenguaje basado en XML que se utiliza para describir Servicios Web. Especifica dónde está el Servicio Web y el tipo de operaciones que éste implementa. Incluye especificaciones como el puerto o *endpoint* dónde se encuentra el Servicio, el tipo de puerto o interfaz, operaciones permitidas y cómo es el mensaje permitido para cada operación.

4. SOAP en Drupal

SOAP (Simple Object Access Protocol) es un protocolo orientado al acceso a objetos, que nos permite intercambiar información. Al igual que Al igual que XML-RPC y REST, el protocolo SOAP utiliza XML como formato de intercambio. Funciona básicamente como una base para nuestros interfaces de Servicios Web y permite construir un entorno de Servicio Web sobre el protocolo. Una de las principales ventajas de SOAP es que éste se encuentra de manera nativa a partir de la versión PHP5, con lo que no necesitaremos librarías externas.

SOAP funciona haciendo peticiones RPC (Remote Procedure Call) a un servidor externo. En esta sección veremos cómo integrar SOAP en Drupal para hacer peticiones a un WSDL de prueba, como el que provee la W3C para convertir métricas de temperatura.

La manera en la que la llamada se ejecuta es codificando el mensaje en formato XML por el cliente SOAP y ese mensaje es enviado a través del protocolo HTTP al servidor de aplicaciones Web externo. Para una mayor seguridad, se peude utilizar el protocolo HTTPS, por ejemplo, si estamos tratando con un Servicio Web de un banco.

Drupal se sirve de dos maneras diferentes a la hora de consumir utilizando SOAP. Drupal Soporta la extensión SOAP de PHP 5.x como mencionamos y de la extensión NuSOAP. NuSOAP no es una extensión SOAP sino más bien un conjunto de clases PHP que permiten la creación y consumo de Servicios Web. No se necesita instalar ninguna exptensión PHP para trabajar con NuSOAP y

soporta las especificaciones de SOAP 1.1, pudiendo también generar documentos WSDL.

4.1. Mensaje SOAP

SOAP se basa en XML por lo que se considera de lectura humana, pero hay un esquema específico que debe ser respetado. Primero vamos a dividir un mensaje SOAP, eliminando datos para estudiar los elementos específicos que componen un mensaje SOAP.

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:Header>
  ...
 </soap:Header>
 <soap:Body>
  ...
  <soap:Fault>
   ...
  </soap:Fault>
 </soap:Body>
</soap:Envelope>
```

Esto podría parecer sólo un archivo XML normal, pero lo que lo convierte en un mensaje SOAP es el elemento raíz *Envelope* con el espacio de nombres con http://www.w3.org/2001/12/soap-envelope. El atributo *soap: encodingStyle* determina los tipos de datos utilizados en el archivo, pero SOAP en sí no tiene una codificación predeterminada.

soap:Envelope es obligatorio, pero el elemento siguiente, *soap: Header*, es opcional y por lo general contiene información relevante para la autenticación y gestión de sesiones. El protocolo SOAP no ofrece ninguna *built-in* de autenticación, pero se lo permite a los desarrolladores incluirlo a través de esta etiqueta de cabecera.

A continuación está el elemento requerido *soap:Body* que contiene el mensaje real RPC, incluyendo los nombres de método y, en el caso de una respuesta, los valores devueltos del método. El elemento *soap:Fault* es opcional, si está presente, se tiene algún mensaje de error o información de estado para el mensaje SOAP y debe ser un elemento secundario de *soap:Body*.

Ahora que ya entendemos los fundamentos de lo que constituye un mensaje SOAP, echemos un vistazo a la solicitud SOAP y los mensajes de respuesta. Vamos a comenzar con una petición.

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:Body xmlns:m="http://www.yourwebroot.com/stock">
  <m:ObtenerPreciosMercado>
   <m:NombreMercado>IBM</m:NombreMercado>
  </m:ObtenerPreciosMercado>
 </soap:Body>
</soap:Envelope>
```

El anterior es un ejemplo de mensaje SOAP de solicitud para obtener el precio de las acciones de una empresa en particular. Dentro de *soap:Body* se observa el elemento *ObtenerPreciosMercado* que es específico de la aplicación. No es un elemento SOAP, y toma su nombre de la función en el servidor que se llamará para esta solicitud. *NombreMercado* también es específico de la aplicación y es un argumento para la función.

El mensaje de respuesta es similar a la solicitud:

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:Body xmlns:m="http://www.yourwebroot.com/stock">
  <m:ObtenerPreciosMercadoResponse>
   <m:Precio>183.08</m:Precio>
  </m:ObtenerPreciosMercadoResponse>
 </soap:Body>
</soap:Envelope>
```

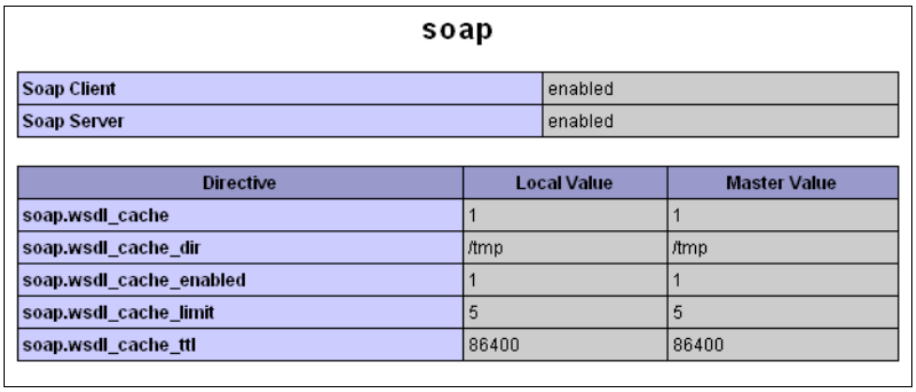
En el elemento *soap:Body* hay un elemento *ObtenerPreciosMercadoResponse* con un hijo *Precio*, que contiene los datos de retorno. Como es de suponer, tanto *ObtenerPreciosMercadoResponse* tanto *Precio* son específicos para esta aplicación.

Ahora que ya hemos visto un ejemplo de petición y respuesta y entendemos la estructura de un mensaje SOAP, vamos a instalar NuSOAP y construir un cliente SOAP y el servidor para probar la generación de este tipo de mensajes.

4.2. Instalando SOAP en PHP

Antes de utilizar SOAP en frameworks de PHP, como Drupal, hay que asegurarse de que la extensión SOAP está instalado y habilitada en nuestro servidor. SOAP se ejecuta como una extensión de PHP. Podemos revisar el archivo de configuración PHP, o pulse en el link para comprobar su versión de PHP, en su la página de informes de Drupal para comprobar la configuración de PHP y ver si SOAP es habilitado.

En muchos servidores compartidos o dedicados, la extensión debe ser autoactivada, por lo que no tendrás que hacer nada especial para que funcione. Sin embargo, en caso de que la extensión no está activada, tendremos que habilitarla. Podemos abrir el fichero de configuración de PHP (*php.ini*) y buscar el siguiente



soap

Soap Client	enabled
Soap Server	enabled

Directive	Local Value	Master Value
soap.wsdl_cache	1	1
soap.wsdl_cache_dir	/tmp	/tmp
soap.wsdl_cache_enabled	1	1
soap.wsdl_cache_limit	5	5
soap.wsdl_cache_ttl	86400	86400

Figura 2: Extensión SOAP

texto desde en el apartado *Configure Command*: `--enable-soap`. Este comando le muestra si SOAP está habilitado.

A continuación, podemos acceder a la página de información de PHP (utilizando el método `phpinfo()`) y buscar la sección específica para la extensión SOAP y deberíamos tener algo como lo siguiente:

Esto nos muestra que el cliente SOAP y el servidor SOAP están activados en nuestra configuración de PHP. Teniendo en cuenta que tenemos el paquete SOAP en nuestra máquina, podemos activarlo desde un Terminal de la siguiente manera, teniendo en cuenta que la distribución de Linux es Fedora/CentOS y que el usuario tiene permisos para hacerlo (*root*):

```
yum install php-soap
```

Ahora que tenemos nuestro entorno listo, revisaremos un ejemplo de cómo se puede utilizar SOAP en Drupal, para solicitar datos a través de una interacción con Servicios Web de un servidor externo. En este caso, nos basaremos de un Servicio Web de W3School de conversión, y lo utilizaremos para pasar grados de Celsius a Fahrenheit. Es un ejemplo sencillo pero con él aprenderemos cómo crear cualquier peticiones en SOAP y podremos extrapolarla a nuestros requerimientos.

La idea de este ejemplo es crear un nuevo tipo de contenido con un campo Celsius, en el cual, cada vez que creemos un nodo de este tipo, llame a un Servicio Web SOAP y devuelva la llamada con la temperatura introducida, convertida a grados Fahrenheit.

5. Uso del módulo Web service client

El módulo **Web service client** es un módulo de la comunidad Drupal que contiene una API que incorporará a nuestra instalación de Drupal y que nos permitirá comunicarnos a través de la extensión de PHP 5.x SOAP. Esta API

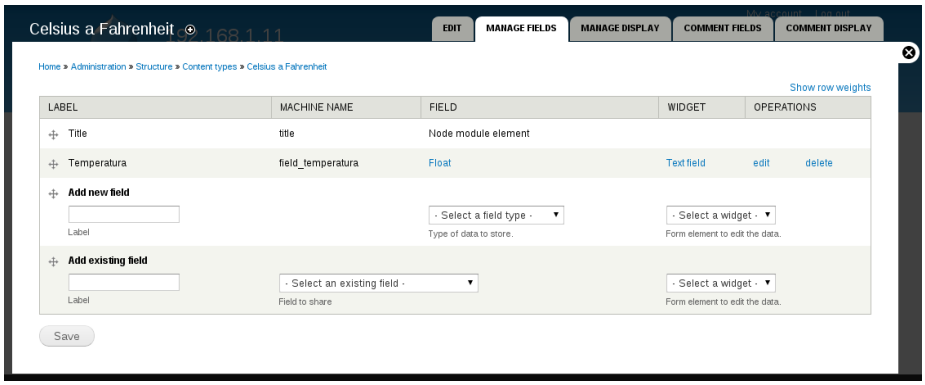


Figura 3: Tipo de contenido "Celsius a Fahrenheit"

nos permitirá además integrarnos con otros Servicios WEB REST, que veremos más adelante. La API se sirve del módulo **Rules**, ya que integra las operaciones de Servicios Web como acciones de **Rules**, aunque también puede servirse de otros módulos.

A continuación, mostraremos cómo instalar y configurar el módulo y, finalmente, cómo usarlo para conectarse a un Servicio Web externo.

5.1. Requisitos

Para comenzar, veamos primero cómo resolver todos los requisitos para llegar a comunicarnos. Para empezar, crearemos el tipo de contenido *Celsius a Fahrenheit*, que nos servirá para almacenar la información en Celsius y activar la llamada al Servicio Web para recoger la misma temperatura en Fahrenheit.

Crearemos este nuevo tipo desde *Structure / Content types* y *Add content type*. Lo llamaremos "Celsius a Fahrenheit" y tendrá un campo *Float* como muestra la siguiente pantalla:

A continuación, instalaremos y configuraremos el módulo *Rules*. Lo descargaremos de su página `http://drupal.org/project/rules`, hasta el momento, utilizando la versión `v.7.x-2.2`.

Lo descomprimimos y lo movemos al directorio de módulos de nuestro Drupal (*/sites/all/modules*). Ahora nos dirigimos a la página de administración de módulos de Drupal y activamos **Rules** y **Rules UI**:

5.2. Instalando y configurando Web service client

Descarge el módulo desde su página: `http://drupal.org/project/wsclient`. En el momento en el que se escribieron estas líneas, el módulo se encontraba en la versión `v.7.x-1.0-beta2`. Una vez que se ha descargado el paquete, tendremos que:

- Descromprimirlo en el directorio *module* de nuestro drupal: `/sites/all/modules`.

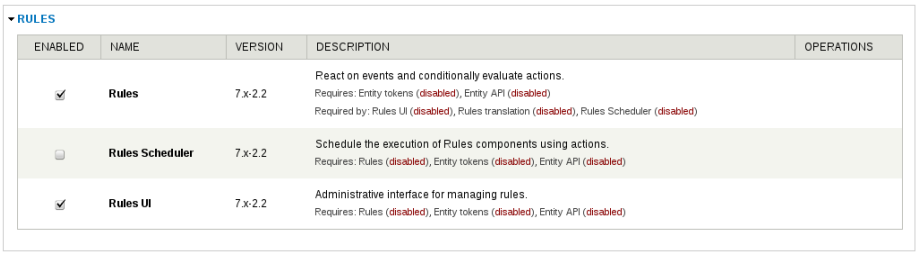


Figura 4: Instalación de los módulos Rules y Rules UI

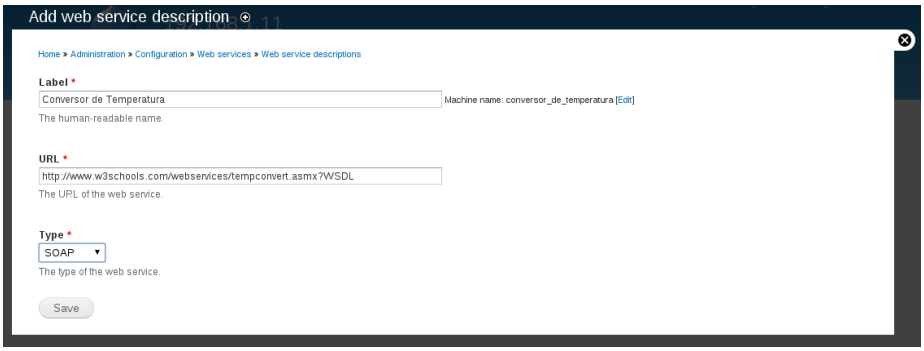


Figura 5: Creación de descriptor del Servicio Web

- Acceder a la página de módulos de nuestro Drupal y activarlo.
- Instalaremos los módulos **Web service client**, **Web service client SOAP** y **Web service client UI**

A continuación, nos disponemos a crear el descriptor del Servicio Web, utilizando la URL del WSDL del Servicio:

`http://www.w3schools.com/webservices/tempconvert.asmx?WSDL`

Para ello, accedemos a *Configuration / Web service descriptions*. Desde aquí, creamos un nuevo descriptor, el cual llamaremos **Conversor de Temperatura** e indicaremos que el Servicio es SOAP.

Desde la nueva ventana podremos ver los métodos que implementa el servicio a los que podremos acceder desde Drupal, en nuestro caso, vamos a utilizar el método *CelsiusToFahrenheit*. También podremos comprobar qué tipos de datos contempla el descriptor importado.

Una vez salvado el formulario, cumplimos todas las dependencias necesarias para implementar nuestra llamada. Debido a que definimos que sólamente íbamos a llamar al servicio de conversión una vez que guardaramos contenido del tipo *Celsius a Fahrenheit*, nos disponemos a crear una regla que actue de *trigger* una vez que salvemos contenido de dicho tipo.

Figura 6: Diagrama Servicio

Nuestra llamada al servicio se representaría con el siguiente diagrama:

Para ello, desde la interfaz de administración (*Configuración / Rules*) añadimos una nueva regla con las siguientes características:

- Name: *Celsius a Fahrenheit*
- Tags: *celsius, fahrenheit*
- Reacts on event: *After saving new content*

Tras salvar este formulario, se nos presenta otro nuevo formulario en el que podremos incluir condiciones y reacciones de este evento. Tendremos que completarlo con las siguientes condiciones y acciones:

- Condition **content is of given type**:
  - En el bloque *Conditions* clicamos en *Add condition*.
  - En el selector, buscamos *Content is of a type* y en el nuevo bloque que nos aparece con los tipos de contenido, elegimos nuestro tipo: *Celsius a Fahrenheit*.
  - Salvamos el fomulario
- Action **create a data structure**:
  - En el bloque *Actions* clicamos en *Add action*.
  - De este selector, elegimos *Create a data structure* y de la nueva ventana que nos aparece, el valor será *Conversor de Temperatura: CelsiusToFahrenheit*. Salvamos.
  - Elegimos el *Data selector* que sea el campo de nuestro tipo de contenido: *node:field-temperatura*
  - El último paso será crear una etiqueta para el resultado de nuestra llamada al servicio. En el bloque *Provided variables*, nombramos la variable con *Celius* y el nombre de variable, *celsius_input*.

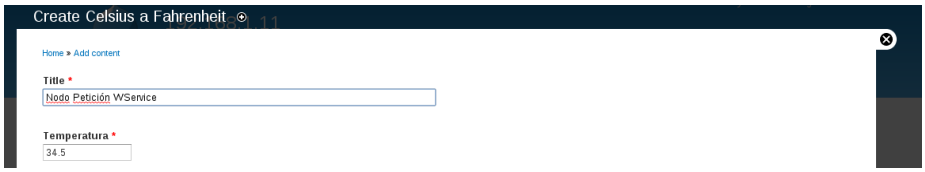


Figura 7: Creando un nodo

Figura 8: Respuesta del servicio

- Action **call web service**
  - Para la llamada al Servicio Web, añadimos una nueva acción *Conversor de Temperatura: CelsiusToFahrenheit*.
  - Tenemos que tener en cuenta que el dato que utiliza el servicio será *celsius_input*.
  - En este caso, en el apartado *Provider variables* vamos a renombrar el nombre de la variable como *fahrenheit_output*.
- Action **display result**
  - Por último, vamos a elegir que nuestra respuesta se muestre por pantalla con la opción *Show a message on the site*.
  - El campo que mostraremos será el definido anteriormente como *fahrenheit-output:CelsiusToFahrenheitResult*.

Para comprobar cómo funciona nuestra llamada al servicio, ejecutamos la acción, es decir, creamos un nodo del tipo *Celsius a Fahrenheit*:

Tras salvar el nodo anterior obtendríamos la llamada al servicio y su pertinente respuesta, tal como muestra el siguiente pantallazo:

6. XML-RPC

XML-RPC es uno de los protocolos de Servicios Web que citamos en el primer capítulo, y que se puede decir, de los más importantes, ya que Drupal lo soporta de manera nativa. Es decir, este protocolo está implementado en el Drupal sin necesitar módulos o librerías de la comunidad. En este capítulo, aprenderemos a trabajar con él y utilizarlo como un cliente y como un proveedor de Servicios Web.

6.1. ¿Qué es XML-RPC?

Al igual que el protocolo SOAP visto anteriormente, XML-RPC se basa en RPC (*Remote Procedure Call*), es decir, en llamadas a procedimientos remotos, para enviar y recibir mensajes cifrados en XML a través del protocolo HTTP. Dichos mensajes contienen información sobre:

- El procedimiento que se va a ejecutar en el servidor.
- Los parámetros

Además, el servidor devuelve el resultado en formato XML.

6.1.1. Diferencias entre XML-RPC y SOAP

XML-RPC fue el primer mecanismo que surgió para invocar procedimientos remotos vía XML, ofrece una manera muy sencilla de invocar operaciones en sistemas heterogéneos a través de una estructura simple. Sin embargo, SOAP es una implementación más robusta para llevar acabo una intercomunicación en XML, a diferencia de XML-RPC a SOAP se le han integrado diversos mecanismos que le permiten operar en ambientes distribuidos mas complejos tales como:

- Un lenguaje neutro para su descripción (WSDL/"Web Services Description Language").
- Directorios distribuidos para su ubicación (UDDI/"Universal Discovery, Description and Integration")

6.1.2. Prerrequisitos para XML-RPC

Si lo que queremos implementar es un servidor de peticiones, no tenemos que preocuparnos de ningún requisito previo, ya que al ser mensajes enviados a través de HTTP, utilizarán el puerto web estándar (puerto 80). El fichero `xmlrpc.php` del directorio raíz de nuestro Drupal recibirá las peticiones y las enrutará.

Para ser usado como cliente, solo tenemos que tener en cuenta que muchas empresas de alojamiento Web no nos permiten realizar peticiones HTTP por razones de seguridad. Sería conveniente contactar con nuestro proveedor si no estamos seguros si nuestros clientes funcionarán.

6.2. Drupal como servidor XML-RPC

Para configurar nuestro Drupal como un Servidor Web XML-RPC necesitaremos implementar tres puntos:

1. Relación entre nombre de método remoto y función PHP a ejecutarse al ser llamados.
2. La firma del método (*signature*): tipo de retorno y de parámetros requeridos. Un método puede tener varias firmas.
3. Documentación del método (*docstring*).

Como es normal en Drupal, vamos a implementar el ejemplo como un pequeño módulo que responda a peticiones. Procedemos pues a crear un "Hola Mundo" de un Servidor XML-RPC en Drupal, con la estructura y el código básicos. Para empezar, generaremos el fichero de información del módulo en `sites/all/modules/holaxmlrpc/holaxmlrpc.info`:

```
name = Hola Mundo - Servidor XML RPC
description = Módulo que responderá con un "Hola" a cada petición XML-RPC
que se le realice a nuestro Drupal
package = Libro Experto Drupal 7
core = 7.x
```

El código de `holaxmlrpc.module` quedaría como sigue:

```
<?php

/**
 * Implementacion del metodo hook_xmlrpc()
 * Relaciona nombre de procedimiento remoto y función PHP a ejecutarse
 */

function holaxmlrpc_xmlrpc()
{
    $metodos['remotoHola.hola'] = 'xmls_holaxmlrpc_hola';
    return $methodos;
}

/**
 * Saluda a una peticion
 * @param string $nombre
 */
function xmls_holaxmlrpc_hola($nombre) {
    if(!$nombre){
        return xmlrpc_error(1, t("Perdona, no te puedo saludar si no me dices tu nombre"));
    }
```

```
    return t("¡Hola, @nombre! ¿Que tal?", array("@nombre" => $nombre));
}
```

¡Así de simple! Podemos complicar más nuestro servidor añadiendo más métodos en el array servicios de nuestro Drupal y validando los parámetros de entrada de las peticiones que se nos realizan. Primero veamos cómo funciona el `hook\_xmlrpc()` y como testear este servicio.

Este *hook* permite a un módulo registrar mapeos entre métodos de un Servicio Web y funciones de PHP a ejecutar, cuando un cliente invoca a los métodos de nuestro servidor XML-RPC. Devuelve un array con el mapeo así como configuración referida a la firma del método y la documentación del mismo. En nuestro ejemplo, el método `remotoHola.hola` será el método público, y gracias a `hook_xmlrpc()`, cada llamada a este método se manejará a través de la función PHP `xmls_holaxmlrpc_hola()`. Normalmente se utiliza el prefijo `xmls_`[1] para las funciones PHP para indicar que esta función habla con el mundo exterior.

Además, podemos observar que en el código hemos útilizado un método para designar un error en la petición: `xmlrpc_error()`, mediante el cual podemos indicar el código de error y el mensaje que se enviará al cliente. Los códigos de error son específicos de nuestra implementación, así que podremos utilizarlos a nuestro gusto.

Probemos nuestro servicio. Asumiendo que el módulo está alojado en un servidor `ejemplo.com` y que nos encontramos en `localhost`, tendremos que hacer una petición como la que sigue para comprobarlo:

```
$url = 'http://ejemplo.com/xmlrpc.php';
$nombre_metodo = 'remotoHola.hola';
$nombre = t('Fran');
$resultado = xmlrpc($url, array($nombre_metodo => array($nombre)));
```

El resultado de ejecutar el código es:

```
echo $resultado;
>> "¡Hola, Fran! ¿Que tal?"
```

[1] *xmlns* es el atributo que se utiliza en XML para designar el espacio de nombres o *namespace*