ANEXO III

Servicios Web Basados en Colecciones de Objetos de Negocio con Perl

(Perl Web Services based on Business

Object Collections)

[1]Derechos de Autor © 2.005. Diego Andrés Asenjo González <dasenjo@unicauca.edu.co>, Alejandro Ríos Peña <alerios@unicauca.edu.co> y Mario Fernando Solarte Sarasty <msolarte@uniauca.edu.co>.

Resumen

Los objetos de negocio representan un concepto de negocio altamente cohesivo, como un pedido, un contacto de negocios, o una empresa. Un objeto de negocio es de naturaleza singular, pero frecuentemente se colecciona en grupos, como lo es una colección de pedidos, para ser expuestos con Servicios Web. Este artículo presenta de manera resumida los patrones de diseño para Servicios Web “Objeto de Negocio” y “Colección de Objetos de Negocio”, acompañados de un ejemplo de su implementación en Perl.

Palabras Clave: Thëwala, Servicios Web, Software Libre, Perl, Patrones de Diseño, SOAP::Lite, WSDL::Generator, Class::DBI, Objeto de Negocio, Colección de Objetos de Negocio.

Abstract

Business Objects

Keywords: Thëwala, Web Services, Free Software, Perl, Design Patterns, SOAP::Lite, WSDL::Generator, Class::DBI, Business Object, Business Object Collection.

1. Introducción

El presente artículo resume uno de los aspectos más importantes del trabajo realizado por sus autores con Servicios Web durante el proyecto de grado titulado “**Thëwala: *Sistema de Información de Productos y Servicios para las Empresas de la Corporación Incubadora de Empresas de Software de Popayán***”, realizado para optar al título de Ingeniero en Electrónica y Telecomunicaciones de la Universidad del Cauca, Colombia.

La construcción del sistema de información se planteó desde un principio, como la oportunidad de explorar las posibilidades tecnológicas que le pudieran permitir a un sistema como Thëwala una futura integración con otros sistemas de información relacionados o similares, es por esto que se utilizaron los servicios web como referencia para la concepción técnica del proyecto:

Durante el desarrollo del proyecto mencionado, se pudo comprobar que un punto crítico y, por lo tanto, muy importante en el proceso de desarrollo, es el paso del análisis al diseño y se pudo constatar la valiosa ayuda de los Patrones de Diseño

como una herramienta poderosa para afrontar esta etapa. Fue así como se aplicaron con éxito tres patrones de diseño para Servicios Web utilizando el lenguaje de programación Perl: Objeto de Negocio, Colección de Objetos de Negocio y Capas Físicas. En el presente artículo se presentan con detalle los primeros dos, acompañados de un ejemplo de su implementación en Perl.

2. Los Servicios Web y Perl

Las herramientas de software libre disponibles actualmente están orientadas principalmente a comsumir servicios web y no a proveerlos. Las dos opciones más comunes son Apache-Axis y SOAP::Lite, para java y perl, respectivamente. Dado que no existe una buena implementación libre del lenguaje de programación Java, por lo tanto no resultaba elegible dentro del marco de desarrollo de este proyecto, a diferencia de perl.

Perl, el Lenguaje Práctico para Extracción y generación de Reportes o "Practical Extraction and Report Language", uno de los lenguajes "de scripting" libres por excelencia, cuenta principalmente con dos módulos que pueden ser usados para crear servicios web de manera fácil y rápida, desde cero o reutilizando aplicaciones de Perl ya existentes. Éstos módulos son SOAP::Lite[2] y WSDL::Generator[3] [Rio04].

3. Patrones de Diseño para Servicios Web

Según Paul B. Monday en su libro "Web Services Patterns: Java Edition" [Mon03], los patrones de diseño se relacionan con los Servicios Web de tres maneras diferentes: primero, el paradigma de los Servicios Web está soportado en tres patrones de diseño; segundo, el desarrollo de implementaciones basadas en Servicios Web puede hacerse más fácil aplicando patrones de diseño existentes; y finalmente, varios de los patrones para Servicios Web son simplemente adaptaciones de patrones creados para el paradigma de la orientación a objetos, y en algunos casos pueden llegar a ser más útiles, educativos e importantes si se aplican al paradigma de los Servicios Web.

En el presente artículo se muestran en forma resumida los patrones "Objeto de Negocio" y "Colección de Objetos de Negocio", acompañados de un ejemplo de su implementación en Perl.

3.1. El Patrón "Objeto de Negocio"

Problema: ¿Cómo exportar Objetos del Negocio autocontenidos desde el Paradigma Orientado a Objetos hacia el paradigma de los Servicios Web?

Solución: La razón de ser de un objeto de negocio es representar un concepto de negocio altamente cohesivo, como un pedido, un contacto de negocios, o una empresa. [Mon03]. El patrón propone los siguientes componentes principales para la estructura de un objeto de negocio:

- **ObjetoDeNegocio:** Esta es la representación de un concepto de negocio de alto nivel. El objeto de negocio abarca sólo propiedades o, en objetos más complejos, operaciones sobre múltiples propiedades. Las propiedades pueden representar datos primitivos, como cadena, entero o valores booleanos. El componente TipoDeDatoComplejo también puede tratarse como una propiedad del objeto de negocio [Mon03].

- **TipoDeDatoComplejo:** Este es un subcomponente de un concepto de negocio de alto nivel que es más complejo que un tipo primitivo de datos. Los tipos complejos de datos pueden contener a su vez a otros tipos complejos de datos [Mon03].

Consecuencias: Cuando se trata de exportar objetos del negocio autocontenidos desde el paradigma orientado a objetos hacia el paradigma de los servicios web, se obtiene un efecto colateral infortunado, desde el punto de vista de la pérdida de riqueza en el diseño orientado a objetos. Esto se debe a que se debe aplanar el modelo de objetos para que sea compatible con otras arquitecturas y lenguajes [Mon03].

3.2. El Patrón "Colección de Objetos de Negocio"

Problema: ¿Cómo construir colecciones de objetos de negocio, para utilizarlas en servicios web? Las colecciones son muy útiles en todo tipo de aplicaciones, incluyendo los servicios web. Es mucho más común encontrar objetos de negocio reunidos en colecciones que por separado, por ejemplo, cuando se solicitan listados de pedidos, clientes, productos, etc.

Solución: La información almacenada en una colección consiste en objetos de negocio, tal y como se definen en el patrón anterior [Mon03]. El patrón Colección de Objetos de Negocio simplemente agrega una clase adicional de colección a la estructura del objeto de negocio vista en el patrón anterior. La estructura completa se muestra en la siguiente ilustración:

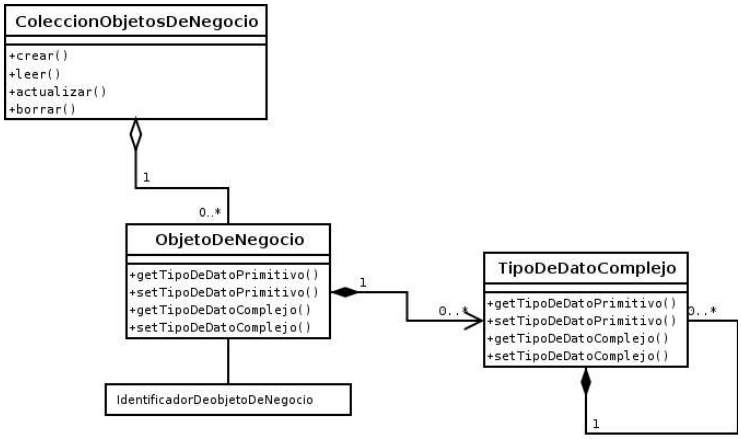

Ilustración 1 - Colección de objetos del negocio.

- **ColeccionObjetosDeNegocio:** La implementación de la colección debe abarcar métodos de ciclo-de-vida y consulta para los objetos que están en la colección. Típicamente, las colecciones ofrecen varias alternativas de métodos de ciclo-de-vida, especialmente los métodos "leer", "crear" y "buscar", dependiendo del dominio que sirve la colección [Mon03].
- **IdentificadorDeObjetoDeNegocio:** Es un identificador único para cada objeto de negocio. Este tipo de identificador es útil en aplicaciones empresariales como un mecanismo para forzar la unicidad. Los identificadores de objetos de negocio también ayudan en la transición de objetos complejos a modelos de persistencia relacional [Mon03].

Consecuencias: Usar colecciones de objetos es una práctica extendida en

aplicaciones empresariales, y crear una interfaz común hará que las colecciones sean más predecibles y, en consecuencia, utilizables [Mon03].

4. Implementación de Ejemplo con Perl

En el ejemplo se implementarán un objeto de negocio, denominado "Item" y una colección de éstos objetos, denominada "ItemCollection". Luego se expondrá la colección como un servicio web y se utilizará un cliente de este servicio para llamar remotamente a uno de sus métodos.

4.1. Creando La Colección de Objetos de Negocio

Para la implementación del ejemplo, se utilizarán los módulos de perl DBI[4] y Class::DBI para realizar la interfaz con el almacenamiento en el motor de base de datos MySQL.

Lo primero que se debe hacer es crear una tabla en una base de datos de MySQL, llamada "item", con una llave primaria autoincremental de tipo INT, llamada id (identificador del ítem), y una columna de tipo VARCHAR, llamada "name" (nombre).

Luego se debe implementar clase de perl que realiza la clase ObjetoDeNegocio del patrón Objeto de Negocio para los ítems. Esta clase hereda de la clase 'server/data/DBI', que contiene los datos de la conexión a la base de datos, y representa el punto de acceso a la tabla 'item' en la base de datos del ejemplo.

La clase de perl '**server/data/DBI.pm'** hereda de la clase Class::DBI y centraliza la información de conexión a la base de datos de la aplicación:

```
package data::DBI;
use base 'Class::DBI';
use strict;
#'driver:base_de_datos;host=máquina/ip;port=puerto','usario','contraseña'
data::DBI->connection('dbi:mysql:item;host=172.16.1.4','thewala','th3wal4');
1;
```

La clase **'server/data/Item.pm'** tiene como atributos los nombres de las columnas de la tabla que representa ('name' e 'id'), y provee automáticamente los métodos de acceso 'name()' e 'id()', que modifican o devuelven los valores de éstos atributos, dependiendo de como sean invocados:

```
package data::Item;
use base 'data::DBI';
use strict;
#Tabla y columnas
data::Item->table('item');
data::Item->columns(All => qw/id name/);
#Constructor
sub new{
        my $this = shift;
        my $class = ref($this) || $this;
        my $self = {};
```

```
        $self->{id}=shift;
        $self->{name}=shift;
        bless($self,$class);
        return $self;
}
1;
```

La clase '**server/data/ItemCollection.pm**' realiza la clase ColeccionObjetoDeNegocio del patrón Colección de Objetos de Negocio. Tiene como métodos 'add(data::Item)', para agregar un ítem a la base de datos, y 'getItems()', para obtener un arreglo con todos los ítems que se encuentran almacenados en la base de datos:

```
package data::ItemCollection;
use data::Item;
use strict;
#Agregar un Item a la base de datos
sub add{
        shift;
        if(@_) {
                my $param = shift;
                my $data = {name=>$param->name()};
                my $item = data::Item->create($data);
                return $item->id;
        } else {
                return 0;
        }
}
#Obtener todos los Items de la base de datos
sub getItems{
        shift;
        my @sectdbi = data::Item->retrieve_all;
        my @items;
        #Poner en un formato más plano
        for my $s (@sectdbi){
                push @items,data::Item->new($s->id,$s->name);
        }
        return @items;
}
1;
```

4.2. Exponiendo el Servicio Web

Se utilizará el módulo de perl SOAP::Lite para exponer y consumir el servicio web 'ItemCollection', que permite agregar ítems a la base de datos y obtener el listado de los ítems existentes.

El script o guión de perl '**server/server.pl**' utiliza SOAP::Lite para exponer el módulo 'data/ItemCollection' y sus métodos como un servicio web en el puerto tcp 8070 del equipo:

```
#!/usr/bin/perl -w
use SOAP::Transport::HTTP;
use data::ItemCollection;
use strict;
#Declarar e iniciar el demonio del proxy SOAP
my $daemon = SOAP::Transport::HTTP::Daemon
        -> new(LocalAddr => 'localhost', LocalPort => 8070)
        -> dispatch_to('data::ItemCollection');
$daemon->handle();
```

Si se quisiera crear un archivo de descripción WSDL para publicar y enlazar dinámicamente este servicio web, se tendría que escribir un script de perl que utilice el módulo WSDL::Generator para crear el archivo de descripción del servicio web 'ItemCollection.wsdl'.

4.3. Consumiendo el Servicio Web

El siguiente script de perl, llamado '**client/client.pl**', utiliza SOAP::Lite para consumir el servicio web del prototipo y llamar a sus métodos 'add()' y 'getItems ()', imprimiendo mensajes en pantalla con las respuestas obtenidas:

```
#!/usr/bin/perl -w
use SOAP::Lite;
use strict;
#Declaración del acceso al servicio a través de SOAP::Lite
my $soap_col = SOAP::Lite
                -> uri('http://localhost:8070/data/ItemCollection')
                -> proxy('http://localhost:8070/server')
                -> readable(1);
#Agregar un Item con id='1' y name='hola'
my $last_key = $soap_col->add(SOAP::Data->name('data__Item')->
                                    value(bless { id => '1', name => 'hola'},
'namesp1:data__Item'))->result;
print "Last key: ".$last_key."\n";
#Obtener e imprimir la lista de items
my $result = $soap_col->getItems;
for my $s ($result->valueof('//getItemsResponse/data__Item')) {
     print $s->{id} . " - " . $s->{name} . "\n";
}
```

Los atributos presentes en las llamadas a los métodos remotos: 'data__Item', 'namesp1:data__Item' y '//getItemsResponse/data__Item', son los tipos de datos complejos definidos implícitamente por SOAP::Lite en el esquema de XML creado automáticamente al exponer el servicio web en el script 'server/server.pl'. Deben ser escritos de esta forma para que los datos intercambiados entre el servidor y el cliente sean tratados apropiadamente, de acuerdo a su tipo.

Si se quisiera consumir el servicio web a partir de un archivo wsdl (como el que crearía WSDL::Generator), se podría utilizar el programa de SOAP::Lite, stubmaker, que crearía automáticamente un módulo de perl llamado 'ItemCollection.pm', listo para ser incluido en el script 'cliente.pl' y así, a través de él, consumir el servicio web descrito por el archivo 'ItemCollection.wsdl'.

5. Conclusiones y Trabajos Futuros

- Se aplicaron con éxito dos patrones de diseño para Servicios Web utilizando el lenguaje de programación Perl: Objeto de Negocio y Colección de Objetos de Negocio.
- En las pruebas realizadas durante el proyecto Thëwala, se comprobó que el módulo SOAP::Lite funciona en forma excelente al consumir servicios web, y en forma aceptable al proveerlos, y el módulo WSDL::Generator funciona bien cuando se intercambian mensajes con tipos de datos sencillos, pero no respondió a las expectativas que se tenían para el manejo de tipos de datos más complejos.
- El proyecto SOAP::Lite aún está en una etapa de crecimiento, y sería bueno aportar a su desarrollo, al igual que contribuir a mejorar el módulo de perl WSDL::Generator.

6. Bibliografía

[Ase04-3] Asenjo González, Diego Andrés. «Acerca de Perl». http://gluc.unicauca.edu.co/. 2004.

[Rio04] Rios Peña, Alejandro. «Servicios Web con Perl». http://gluc.unicauca.edu.co/. 2004.

[Ale77] Alexander, Christopher. «A Pattern Language». Oxford University Press. 1977.

[Mon03] Monday, Paul B.. «Web Services Patterns: Java Edition». Apress. 2003.

THËWALA: SISTEMA DE INFORMACIÓN DE PRODUCTOS Y SERVICIOS PARA LAS EMPRESAS PERTENECIENTES A LA CORPORACIÓN PARQUE TECNOLÓGICO DE POPAYÁN

Diego Andrés Asenjo González
Alejandro Ríos Peña

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán, Colombia
Mayo de 2.005

ANEXO III

ARTÍCULO DE DIVULGACIÓN: “SERVICIOS WEB BASADOS EN COLECCIONES DE OBJETOS DE NEGOCIO CON PERL”

Diego Andrés Asenjo González

Alejandro Ríos Peña

Director

Mario Fernando Solarte Sarasty

Magíster en Ingeniería Telemática

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Telemática

Popayán, Colombia

Mayo de 2.005

CONTENIDO

1 Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes, sin Texto de Cubierta Frontal y sin Texto de Cubierta Posterior. La versión oficial en línea de esta licencia se puede conseguir en el sitio de Internet http://www.gnu.org/copyleft/fdl.html y una traducción no oficial de ella en el sitio http://gugs.sindominio.net/licencias/fdl-es.html

2 SOAP::Lite es un proyecto que tiene como objetivo el desarrollo de varios módulos de Perl y algunas otras herramientas para facilitar la utilización de una interfaz simple y liviana para el protocolo SOAP en este lenguaje. La URL del sitio es http://www.soaplite.com/

3 WSDL::Generator es un pequeño módulo de perl que sirve para generar un archivo WSDL a partir de un módulo que va a ser expuesto como un servicio web. Este módulo se puede encontrar en el sitio de CPAN, pero la documentación disponible no es muy completa.

4 DBI es una interfaz de acceso a bases de datos independiente del motor. Un módulo basado en DBI, pero que provee un interfaz de más alto nivel, es Class::DBI. Este módulo facilita el manejo de las relaciones entre tablas, la integridad referencial, los procedimientos almacenados, el borrado en cascada de registros, y muchas otras cosas. El sitio web del proyecto DBI esta ubicado en http://dbi.perl.org/