Servicios Web ligeros: alternativas al protocolo SOAP para la creación de servicios distribuidos

Noé Fernández Iglesias
UO67858@uniovi.es

Noviembre 2006

Resumen

Los servicios web se han convertido en uno de los paradigmas de programación distribuida más ampliamente usados en la actualidad gracias a la utilización de tecnologías y protocolos estándar como XML o SOAP. Su potencial les permite conectar servicios y clientes heterogéneos de forma eficiente. Sin embargo, esta funcionalidad tiene un coste asociado a la complejidad del protocolo SOAP y la gran sobrecarga de red que se produce en el envío de los mensajes en XML. En este trabajo se pretende realizar una pequeña introducción a varios protocolos distintos a SOAP que mejoran el problema de la sobrecarga vinculado a la creación de servicios web con este protocolo. Asimismo, se expondrá una comparativa entre los protocolos en la que se darán unas pautas a seguir en la elección del protocolo más adecuado en función del contexto de desarrollo.

Palabras clave: Servicios web, SOAP, XML, Hessian, Burlap, HTTP invoker, Caucho, Spring, framework

1. Introducción

Durante los últimos años, la popularización de protocolos como HTTP (HyperText Transfer Protocol) [4] han propiciado la interoperabilidad entre aplicaciones, gracias a que este tipo de protocolos facilitan y mejoran la eficiencia en la distribución de servicios a través de una red de comunicaciones. Basados en ellos han surgido los *Servicios Web*: aplicaciones modulares atómicas que pueden ejecutarse a través de la red Internet con el fin de realizar una tarea [9].

Se pueden considerar a los servicios web como el paso natural dado desde las tecnologías precursoras en el desarrollo de sistemas distribuidos, como puede ser CORBA o RMI. Los servicios web toman el

potencial de éstas, mejoran sus deficiencias y se afianzan como la tecnología actual por excelencia para el desarrollo de servicios distribuidos. Como prueba de esta hegemonía, está el hecho de que son muchos los fabricantes y empresas desarrolladoras de software las que apuestan por esta tecnología y construyen aplicaciones y frameworks para su uso. Como ejemplo del rápido desarrollo que han tenido los servicios web, se debe destacar la creación y uso del protocolo SOAP para su implementación. SOAP (Simple Object Access Protocol) fue diseñado en 1998 por Dave Winer, Don Box, Bob Atkinson y Mohsen Al-Ghosein y se convirtió en estándar en 2003, siendo actualmente el W3C (World Wide Web Consortium) el encargado de su mantenimiento y desarrollo [3] [8].

El protocolo SOAP define un mecanismo de intercambio de mensajes en XML sobre HTTP para la comuniación entre procesos [2]. El hecho de que todo el mecanismo de comunicación se base en mensajes XML es responsable del éxito del protocolo y también de uno de sus mayores problemas: la complejidad de su uso y la sobrecarga de red que dichos mensajes provocan. Se hace, por tanto, necesaria la búsqueda de soluciones que, sin mermar el potencial, solucionen parcial o totalmente dicha desventaja. Esta búsqueda proviene de la necesidad de disponer de mecanismos lo suficientemente ligeros para desarrollar servicios distribuidos en dispositivos móviles, en redes con ancho de banda limitado o, simplemente, en aplicaciones en las que las características de SOAP no justifican la complejidad que conlleva su uso.

Existen diversos protocolos alternativos a SOAP que solucionan en mayor o menor parte los problemas de este. En este trabajo se van a ver tres de ellos, el protocolo Hessian y Burlap desarrollados por Caucho Technology y el protocolo HTTP invoker implementada por el framework Spring. Todos ellos solucionan el problema de la complejidad de SOAP manteniendo una funcionalidad similar a éste [6][5][1][7].

Este trabajo tiene la siguiente estructura: la sección 2 introduce de forma somera el protcolo Hessian, explicando sus características y un ejemplo de uso; en la sección 3 se proporciona una visión general del protocolo Burlap; del mismo modo, en la sección 4 se presenta el protocolo HTTP invoker del framework Spring. Finalmente se aportan las conclusiones del trabajo en las que se expondrán unas pautas para la elección del protocolo más adecuado en función de las necesidades de la aplicación. Como anexos, se encuentran las definiciones formales de los protocolos Hessian y Burlap.

2. Protocolo Hessian

2.1. Introducción

Hessian es un protocolo binario compacto y ligero que permite la conexión entre servicios web sin la necesidad de utilizar un framework pesado y evitando el aprendizaje de nuevos protocolos para tal fin. Debido a que se trata de un protocolo binario (no está basado en XML), es muy apropiado para el envío de datos en binario sin necesidad de extender la especificación del mismo. La transferencia de objetos y

datos en binario evita el gran ancho de banda que necesitan los protocolos basados en XML para realizar la transferencia de información. Por ello y gracias a su pequeño tamaño, resulta muy practico en dispositivos móviles, como PDA's o teléfonos móviles. Sin embargo, el hecho de que sea un protocolo ligero y compacto no le resta potencial ni funcionalidad, pudiendo usarse también para la conectividad con EJBs (Enterprise Java Beans).

Otras características destacables son que es independiente de lenguaje (existen implementaciones para .NET, C, Java, PHP y Ruby) y, a diferencia de otros protocolos, como SOAP, Hessian elimina la necesidad de ficheros externos de definición como IDL WSDL. Por otro lado, Hessian implementa su propio algoritmo de serialización para la invocación de servicios.

2.2. Características

El protocolo Hessian fue desarrollado como una alternativa más ligera a los protocolos basados en XML para la creación de servicios web.

Los objetivos marcados en la creación del protocolo fueron los siguientes:

- No debe utilizar ficheros externos de definición (IDLs, schemas, etc.)
- Debe ser lo suficientemente potente para trabajar con objetos serializados.
- Debe poder dar soporte a EJBs.
- Debe permitir usar los servicios a clientes escritos en lenguajes distintos a Java.
- Debe permitir desplegar servicios web como si se tratasen de servlets.
- Debe ser simple y rápido.
- Debe poder soportar transacciones.

Hessian posee los siguientes tipos de datos:

- **boolean**
- 32-bit **int**
- 64-bit **long**
- 64-bit **double**
- 64-bit **date**
- UTF8-encoded **string**
- UTF8-encoded **xml**
- raw **binary** data
- **remote** objects
- **list** para listas y arrays
- **map** para tablas hash.
- **null** para valores nulos
- **ref** para referencias a objetos

2.3. Creación de un servicio Hessian

La creación de un servicio mediante el protocolo Hessian usando el lenguaje Java consta de 4 pasos:

1. Creación de una interfaz Java a modo de API pública.
2. Creación de un cliente del servicio mediante *HessianProxyFactory*.
3. Creación de la clase de implementación del servicio.
4. Despliegue y configuración del servicio en un contenedor de servlets.

2.3.1. API del servicio

El API del servicio en Hessian es simplemente una interfaz Java como la mostrada a continuación.

```
public interface HWService {
    public String helloWorld();
}
```

Ya que el protocolo es independiente de lenguaje, las clases de los interfaces Java no son obligatorias para el resto de lenguajes que implementen Hessian, sirviendo estas solamente como documentación de los métodos del servicio.

2.3.2. Creación del cliente

A continuación se muestra la implementación de un cliente Hessian *standalone*:

```
public class BasicClient {
  public static void main(String []args)
  throws Exception {

    String url =
        "http://localhost/helloWorld";

    HessianProxyFactory factory =
            new HessianProxyFactory();

    HWService service = (HWService) factory
        .create(HWService.class, url);

    System.out.println("Hello: " +
                       basic.hello());
  }
}
```

El cliente crea una factoría *HessianProxyFactory* y la usa para obtener un stub de cliente con la implementación del API del servicio.

2.3.3. Implementación del servicio

Los servicios Hessian pueden ser implementados como una simple clase de Java (POJO) o bien como un servlet, debiendo en este caso extender *HessianServlet*. Si se elige esta última forma, se puede usar el servicio de modo análogo al que se realiza con los tradicionales servlets.

Todo método público definido en la implementación es tratado como un método del servicio.

```
public class MyService
    implements HWService {

    public String hello() {
        return "Hello, world";
    }
}
```

2.3.4. Despliegue del servicio

Debido a que HessianServlet es una extensión de los servlets standard, el despliegue y configuración de los servicios Hessian se realiza del mismo modo que los servlets. A continuación se muestra un ejemplo del fichero de despliegue para el servicio:

```
<web-app>
  <servlet>
   <servlet-name>hello</servlet-name>
   <servlet-class>
        com.framework.HessianServlet
   </servlet-class>
    <init-param>
      <param-name>home-class</param-name>
      <param-value>MyService</param-value>
    </init-param>
```

```
    <init-param>
      <param-name>home-api</param-name>
      <param-value>HWService</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <url-pattern>/helloWorld</url-pattern>
    <servlet-name>hello</servlet-name>
  </servlet-mapping>
</web-app>
```

3. Protocolo Burlap

3.1. Introducción

Burlap es la versión XML del protocolo Hessian. A pesar de ello, Burlap sigue manteniendo las principales características de Hessian: protocolo compacto y ligero que reduce el ancho de banda necesario para la invocación de servicios, lo que le hace muy atractivo para el desarrollo de aplicaciones distribuidas en dispositivos móviles.

El protocolo Burlap utiliza un subconjunto de XML: SML (Simple Markup Language) en su definición. Sus características son:

- Sólo se permiten elementos y caracteres.
- No existen namespaces, atributos, comentarios así como tampoco DTDs y secciones CDATA.
- No se pueden mezclar contenido: un elemento contiene otros elementos o caracteres, pero no ambos.
- Sólo se permite la codificación UTF-8
- El espacio en blanco es un carácter significativo.

Al igual que Hessian, Burlap implementa su propio algoritmo de serialización. Sin embargo, a diferencia del anterior, en la actualidad sólo existen implementaciones de Burlap en el lenguaje Java.

3.2. Características

El protocolo Burlap fue creado para resolver un problema específico: *Permitir a servidores y clientes no basados en Java interoperar con EJBs (Enterprise Java Beans) usando un protocolo basado en XML.*

Los objetivos marcados en la creación del protocolo fueron los siguientes:

- Debe usar el menor subconjunto posible de XML.
- No debe necesitar de IDLs o schemas para su definición.
- Debe tener potencial para serializar objetos Java.
- Debe soportar Enterprise Java Beans (EJBs)
- Debe permitir a clientes escritos en lenguajes distintos a Java invocar los servicios web.
- Debe permitir desplegar los servicios web como si se tratasen de servlets.
- Debe ser sencillo y rápido.
- Debe poder soportar transacciones.

Burlap posee los siguientes tipos de datos:

- **boolean**
- 32-bit **int**
- 64-bit **long**
- 64-bit **double**
- ISO-8609 **date**
- UTF8-encoded **string**
- UTF8-encoded **xml**
- base64 **binary** data
- **remote** objects
- **list** para listas y arrays
- **map** para tablas hash.
- **null** para valores nulos
- **ref** para referencias a objetos

3.3. Creación de un servicio Burlap

La creación de un servicio mediante el protocolo Burlap es análoga a la realizada con Hessian: definición del interfaz del servicio, creación del cliente, implementación del interfaz del servicio y despliegue del mismo. La única diferencia está en el cliente, ya que éste debe usar un proxy específico para la llamada a servicios Burlap. De este modo, el cliente visto en 2.3.2 quedaría como sigue:

```
public class BasicClient {
  public static void main(String []args)
  throws Exception {

    String url =
        "http://localhost/helloWorld";

    BurlapProxyFactory factory =
            new BurlapProxyFactory();

    HWService service = (HWService) factory
        .create(HWService.class, url);

    System.out.println("Hello: " +
                       basic.hello());
  }
}
```

De modo similar, la implementación del servicio debe extender de *BurlapServlet* en vez de *HessianServlet*.

4. Spring HTTP invoker

4.1. Introducción

Spring es un framework para el desarrollo de aplicaciones J2EE que provee del mecanismo HTTP invoker para la implementación de aplicaciones remotas de forma transparente en Java. El protcolo HTTP invoker usa la serialización Java de la misma forma en la que lo hace RMI, pero su configuración y despliegue es tan sencilla como en los protocolos Hessian y Burlap del framework Caucho.

En comparación se puede concluir que el protocolo HTTP invoker es más potente y flexible que Hessian y Burlap pero manteniendo la sencillez de configuración de estos, lo que le confiere una ventaja respecto a RMI.

4.2. Creación de un servicio mediante HTTP invoker

La creación de un servicio mediante HTTP invoker del framework Spring consta de los siguientes pasos:

- Creación del interfaz del servicio y de la implementación del mismo. Este paso se realiza del mismo modo que para los protocolos Hessian y Burlap.
- Exportación del servicio mediante la definición de un bean de Spring:

```
<bean name="/MyService"
 class="org.springframework.remoting.
 httpinvoker.HttpInvokerServiceExporter">
      <property name="service">
          <ref bean="myService"/>
      </property>
      <property name="serviceInterface">
          <value>
           com.httpInvoker.example.MyService
          </value>
      </property>
</bean>
```

- Despliegue del servicio en un contenedor de servlets:

```
<servlet>
  <servlet-name>
      myServiceHttpInvoker
  </servlet-name>
  <servlet-class>
      org.springframework.web.
      servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>
  3
  </load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>
      myServiceHttpInvoker
  </servlet-name>
  <url-pattern>
      /remoting/*
  </url-pattern>
</servlet-mapping>
```

- Creación del bean de Spring para el cliente:

```
<bean id="service"
class="org.springframework.remoting.
httpinvoker.HttpInvokerProxyFactoryBean">
  <property name="serviceUrl">
      <value>
          http://localhost/remoting/MyService
      </value>
  </property>
  <property name="serviceInterface">
      <value>
          com.httpInvoker.example.MyService
      </value>
  </property>
  <property name="httpInvokerRequestExecutor">
      <bean class="org.springframework.remoting.
      httpinvoker.
      CommonsHttpInvokerRequestExecutor"/>
  </property>
</bean>
```

- Invocación del servicio:

```
MyService service =
  (MyService)context.getBean("service");
service.hello();
```

5. Conclusiones

En este trabajo se han visto protocolos alternativos a SOAP para el desarrollo de servicios web. Dichos protocolos son más ligeros y sencillos, pero mantienen hasta cierto punto el mismo potencial de SOAP. Sin embargo, cada protocolo puede enmarcarse dentro de un contexto derterminado para su uso. A continuación se mostrarán unas pautas que pueden servir de ayuda en la elección del protocolo más adecuado en función de la aplicación en la que se desean aplicar:

- Hessian: es un protocolo binario sobre HTTP, por lo que es muy rápido y reduce considerablemente el ancho de

banda necesario para la comunicación. Como desventaja está el hecho de usar un algoritmo propio de serialización, lo que, combinado con el hecho de ser un protocolo binario, limita su uso al no ser genérico. Hessian es un protocolo adecuado para dispositivos móviles, debido a la transferencia de datos en binario, su ligereza y sencillez. Asimismo, puede resultar apropiado para aplicaciones en las que tanto la parte servidora como la cliente están bajo el control del desarrollador, ya que de este modo, la compatibilidad entre ambas está asegurada.

- Burlap: es un protcolo basado en XML sobre HTTP. En cierto modo puede considerarse la versión XML de Hessian. Como ventajas cabe destacar el hecho de realizar la comunicación a través SML (un conjunto restringido de XML), con lo que el modelo es más próximo a SOAP. Del mismo modo, el hecho de utilizar XML no resta simplicidad ni rapidez al protocolo. Sus desventajas provienen del escaso soporte que este protocolo posee por lenguajes distintos a Java y, al igual que Hessian, por el empleo de su propio algoritmo de serialización. Del mismo modo que para Hessian, el protocolo Burlap resulta apropiado para entornos de desarrollo en los que se controla la parte servidora y cliente. Por contra, no es tan apropiado para entornos móviles debido al uso de XML.

- HTTP invoker: protocolo del framework Spring para la implementación de servicios remotos sobre HTTP. Como ventajas destaca el uso del mecanismo de serialización de Java, lo que soluciona el problema que presentan Hessian y Burlap en este aspecto. Asimismo, es destacable la sencillez del protocolo para la configuración y despliegue de servicios. Su principal desventaja es la vinculación al lenguaje Java por el uso de su sistema de serialización de objetos. HTTP invoker resulta apropiado para aplicaciones Java desarrolladas con el framework Spring en las que se desea evitar los inconvenientes de los protocolos Hessian y Burlap.

A pesar de todos las virtudes de estos protocolos ligeros, existen ciertos contextos de desarrollo para los que no son apropiados. En el caso de complejas aplicaciones que deban operar con sistemas y protocolos heterogéneos y en los que la calidad y la funcionalidad priman sobre la velocidad, el protocolo SOAP supera al resto, siendo por tanto la elección óptima.

A. Definición formal del protocolo Hessian

A.1. Gramática

```
top     ::= call
        ::= reply

call    ::= c x01 x00 header* method object* z

reply   ::= r x01 x00 header* object z
        ::= r x01 x00 header* fault z

object  ::= null
        ::= boolean
        ::= int
        ::= long
```

```
        ::= double
        ::= date
        ::= string
        ::= xml
        ::= binary
        ::= remote
        ::= ref
        ::= list
        ::= map

header  ::= H b16 b8 header-string object
method  ::= m b16 b8 method-string

fault   ::= f (object object)* z

list    ::= V type? length? object* z
map     ::= M type? (object object)* z
remote  ::= r type? string

type    ::= t b16 b8 type-string
length  ::= l b32 b24 b16 b8

null    ::= N
boolean ::= T
        ::= F
int     ::= I b32 b24 b16 b8
long    ::= L b64 b56 b48 b40 b32 b24 b16 b8
double  ::= D b64 b56 b48 b40 b32 b24 b16 b8
date    ::= d b64 b56 b48 b40 b32 b24 b16 b8
string  ::= (s b16 b8 string-data)* S b16 b8
                                   string-data
xml     ::= (x b16 b8 xml-data)* X b16 b8
                                   xml-data
binary  ::= (b b16 b8 binary-data)* B b16 b8
                                   binary-data
ref     ::= R b32 b24 b16 b8
```

A.2. Hessian call

```
call ::= c x01 x00 header* m b16 b8
         method-string (object)* z
```

A.3. Hessian reply

```
valid-reply ::= r x01 x00 header* object z
fault-reply ::= r x01 x00header* fault z
```

B. Definición formal del protocolo Burlap

B.1. Gramática SML

```
tag-list ::= (S tag)* S

tag  ::= <name> tag-list </name>
     ::= <name> cdata </name>

S    ::= (' ' | '\t' | '\r' | '\n')*

cdata ::= ([^<&] | &# [0-9]+ ; | &lt;
          | &gt; | &amp;)*
name  ::= [a-zA-Z:_] [a-zA-Z0-9.-_:]*
```

B.2. burlap:call

```
<!DOCTYPE burlap:call>

<!ENTITY % object "null | boolean | int
                 | double | string | xml
                 | base64 | date | ref
                 | map | list | remote">

<!ENTITY %
    header "(header, (%object;))*">

<!ELEMENT burlap:call - -
    ((%header;), method, (%object;)*)>

<!ELEMENT header - - #CDATA>

<!ELEMENT method - - #CDATA>

<!ELEMENT null - - EMPTY>
<!ELEMENT boolean - - #CDATA>
<!ELEMENT int - - #CDATA>
<!ELEMENT double - - #CDATA>
<!ELEMENT string - - #CDATA>
<!ELEMENT xml - - #CDATA>
<!ELEMENT base64 - - #CDATA>
<!ELEMENT date - - #CDATA>

<!ELEMENT ref - - #CDATA>
<!ELEMENT map - -
    (type, ((%object;), (%object;))*)>
```

```
<!ELEMENT list - -
    (type, length, (%object;)*)>
<!ELEMENT type - - #CDATA>

<!ELEMENT remote - - (type, string)>
```

B.3. burlap:reply

```
<!DOCTYPE burlap:reply>

<!ENTITY % object "(null | boolean | int
                | double | string | xml
                | base64 | date | ref
                | map | list | remote)">

<!ENTITY % header "(header, (%object;))*">

<!ELEMENT burlap:reply - -
    ((%header;), (%object; | fault))>

<!ELEMENT header - - #CDATA>

<!ELEMENT fault - - (string, (%object;))*>

<!ELEMENT null - - EMPTY>
<!ELEMENT boolean - - #CDATA>
<!ELEMENT int - - #CDATA>
<!ELEMENT double - - #CDATA>
<!ELEMENT string - - #CDATA>
<!ELEMENT xml - - #CDATA>
<!ELEMENT base64 - - #CDATA>
<!ELEMENT date - - #CDATA>

<!ELEMENT ref - - #CDATA>
<!ELEMENT map - -
    (type, ((%object;), (%object;))*)>
<!ELEMENT list - -
    (type, length, (%object;)*)>
<!ELEMENT type - - #CDATA>

<!ELEMENT remote - - (type, string)>
```

Referencias

[1] Caucho Technology. Disponible online: http://www.caucho.com.

[2] Extensible Markup Language (XML). Disponible online: http://www.w3.org/XML.

[3] Hypertext Transfer Protocol (HTTP). Disponible online: http://www.w3.org/TR/soap.

[4] imple Object Access Protocol (SOAP). Disponible online: http://www.ietf.org/rfc/rfc2616.txt.

[5] Protocolo Burlap. Disponible online: http://www.caucho.com/resin-3.0/protocols/burlap.xtp.

[6] Protocolo Hessian. Disponible online: http://www.caucho.com/hessian/index.xtp.

[7] Spring framework. Disponible online: http://www.springframework.org.

[8] (World Wide Web Consortium). Disponible online: http://www.w3.org.

[9] J. Gergic, J. Kleindienst, Y. Despotopoulos, J. Soldatos, G. Patikis, A. Anagnostou, and L. Polymenakos. An approach to lightweight deployment of web services. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 635–640, New York, NY, USA, 2002. ACM Press.