Monografía: Descubrimiento de servicios para la evolución dinámica de sistemas software mediante transformación de modelos

WSDLSniffer 1.0 es una API desarrollada en eclipse para el descubrimiento de servicios en documentos .wsdl de servidores remotos o locales.
El API desarrollada devolverá las operaciones de los servicios con sus respectivos atributos.

Tutores:
Jennifer Pérez Benedí
Agustín Yagüe Panadero
Autores:
Pablo Ortiz Sanchidrián
Lourdes Riestra Alba

Índice del capítulo

1. ¿Qué son los servicios Web?

Una definición formal sobre servicios Web la aporta el W3C[1], "*un servicio Web es una aplicación software identificada por una URI[2], cuyas interfaces se pueden definir, describir y descubrir, mediante documentos XML[3]. Un servicio Web soporta interacciones directas con otros agentes software utilizando mensajes XML intercambiados mediante protocolos basados en Internet.*"

En general, el modelo de servicios Web sigue un esquema de tres capas bien diferenciadas que se especifican gráficamente en la figura 1.

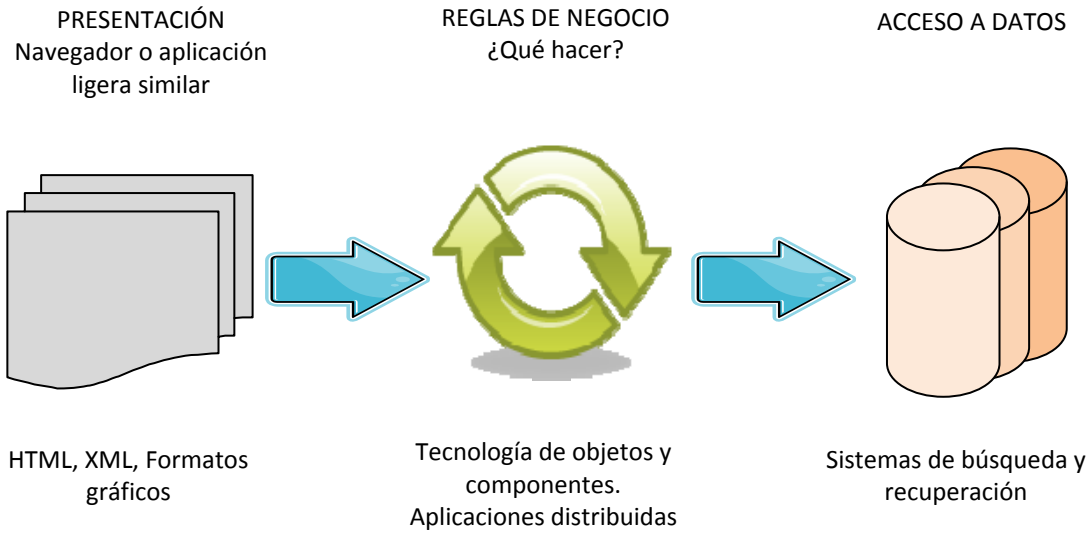

Figura 1 Modelo de tres capas para el desarrollo de servicios Web

Después de esto se puede decir que los servicios Web proporcionan herramientas de comunicación estandarizadas entre diferentes aplicaciones y servicios informáticos, que interactúan entre ellos para dar información dinámica, al usuario que lo solicite, ya

---

[1] W3C: World Wide Web Consortium, es un consorcio internacional que produce estándares para la World Wide Web.

[2] URI: cadena corta de caracteres que identifica inequívocamente un recurso.

[3] XML: Extensible Markup Language («lenguaje de marcas ampliable»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

sea éste una persona física o un proceso informático. En general, se utilizan los servicios Web, para poder proporcionar interoperabilidad y extensibilidad entre diversos servicios y aplicaciones informáticas, y que al mismo tiempo sea posible su combinación y comunicación entre las mismas. Para poder realizar operaciones complejas entre aplicaciones y servicios dispares, es necesaria una arquitectura de referencia estandarizada.

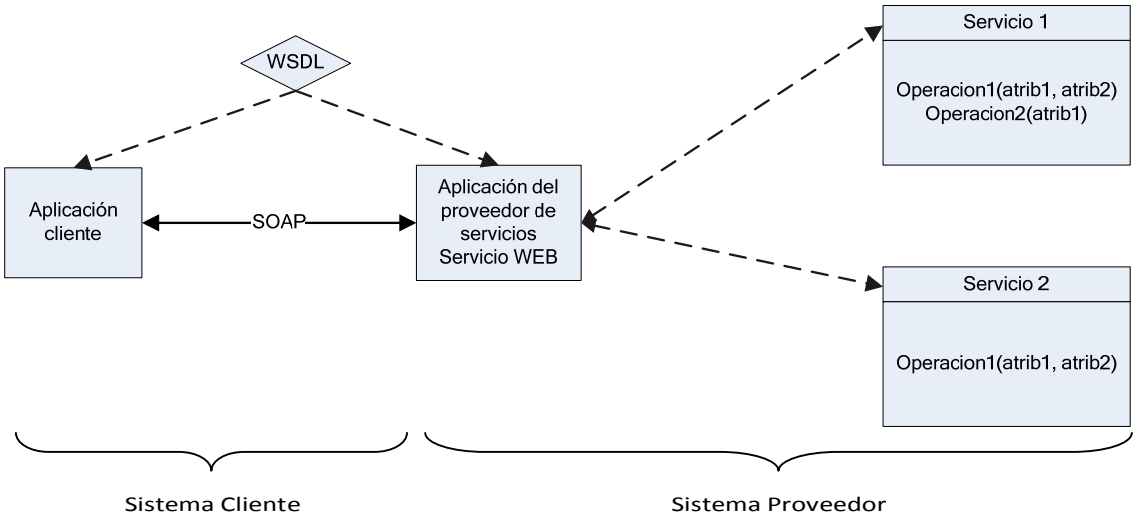

Figura 2 Ejemplo de arquitectura Servicios Web

En el ejemplo de la Figura 2, un cliente, que necesita cierta información de un determinado proveedor, se conecta, a través de una aplicación informática, a un servidor remoto, que posee diversos servicios Web, relacionados con las operaciones sobre la información que ofrece el sistema del proveedor. En este ejemplo, las operaciones sobre la datos serían servicio 1 y servicio 2. El cliente, podrá operar a través de la aplicación del proveedor con los datos que este le permita.

Durante el progreso de las aplicaciones basadas en Servicios Web, se han desarrollado tecnologías que permiten mejorar las descripciones de las operaciones que realiza un servicio mediante anotaciones semánticas y con orientaciones que definen el comportamiento de las aplicaciones. Esto permitiría encontrar los Servicios Web que mejor se ajusten a las necesidades creadas. Además, ante la complejidad de los procesos de las grandes aplicaciones empresariales, existe una tecnología que permite una definición de estos procesos mediante la composición de varios Servicios Web individuales, lo que se conoce como *coreografía*.

1. a. Motivación desde el comercio electrónico (2)

Hoy en día, la mayoría de las empresas ofrecen servicios a sus clientes a través de servicios Web. Un ejemplo concreto sería una agencia de viajes, en la que se gestiona la contratación de los diferentes paquetes vacacionales a través de internet. Un cliente se conecta a un servidor de una agencia de viajes, en la que puede elegir diferentes destinos, fechas, aviones y hoteles. El servidor de viajes, por ejemplo, se tiene que conectar al servidor de horarios y rutas de aviones, en el que se proporciona la información necesaria para poder ofrecer el servicio deseado por el cliente de la agencia. Una vez procesada la información de los aviones se da la información al cliente.

En la siguiente figura se puede generalizar el ejemplo anterior y ver lo que sucede con los servicios Web dentro de un cliente y dos empresas genéricas que se solicitan y proporcionan servicios.

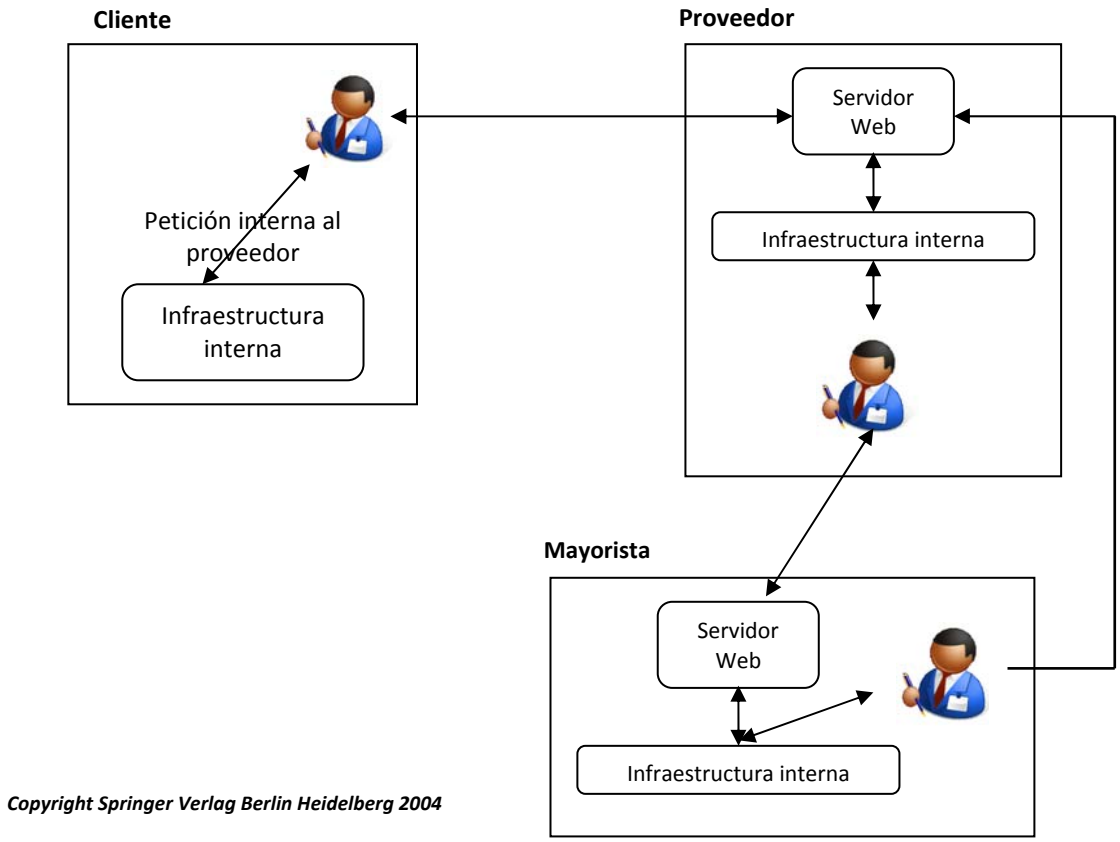

Figura 3 Gráfico Servicios Web

Las interacciones en comercio electrónico, se realizan principalmente a través de páginas Web, rellenando formularios, o vía mail. Salvo excepciones, los procesos internos están automatizados, pero las relaciones con el exterior no. Es por esta razón que surge la necesidad de crear unos protocolos generalizados para la comunicación entre las diversas plataformas de envío de mensajes, como son UDDI[4], WSDL[5] o XML.

1. b. Middleware[6]: paso anterior a los servicios Web

Anteriormente de que surgieran los servicios Web, se trabajaba con aplicaciones y componentes que utilizaban un middleware, como podrían ser ejemplos de estas aplicaciones DCOM, CORBA y Java RMI.

En las relaciones entre organizaciones no hay un lugar concreto donde colocar un middleware. Normalmente el middleware se sitúa entre las aplicaciones a integrar, no siendo un estándar para todas las empresas. Las aplicaciones están distribuidas, pero el middleware se centraliza y controla por una determinada compañía, lo cual supone un enorme problema a la hora de estandarizar los protocolos de comunicación de servicios entre empresas. La aceptación de una misma solución supone que el cliente que accede a un determinado servicio web facilitado por un proveedor, ha de acordar con éste el utilizar una determinada plataforma de middleware.

Se hizo un intento de solucionar las limitaciones, que se cernían sobre los middleware, mediante un modelo denominado *stateless programing(3).* Esta solución no consiguió su propósito ya que estas tecnologías son bastante pesadas y el restablecimiento de la conexión con un servidor remoto resulta muy costoso. A continuación se muestran algunas de las limitaciones sobre los middleware:

- Una de las limitaciones más importantes que se dan en los middleware es la seguridad, ya que para poder comunicar diferentes aplicaciones, necesitan un

[4] UDDI: Universal Description Discovery and Integration. Describe cómo organizar la información sobre y servicio y cómo construir repositorios dónde se pueda registrar e interrogar cierta información.

[5] WSDL: Web Services Description Language. Pasaremos a describirlo con detalle más adelante.

[6] Middleware: Software de conectividad.

puerto libre, por encima del 1024. Por este motivo, los middleware establecen y gestionan sus políticas de seguridad de forma eficaz, haciendo que la comunicación entre un cliente y un proveedor, a través de internet tenga muchas barreras que superar. Para poder sobrepasar estos impedimentos, los administradores de red se encargan de configurar dispositivos, como routers[7] y firewalls[8] con el objetivo de garantizar la seguridad y no permitir ciertas comunicaciones con internet. Todo esto hace que los protocolos que utilizan middleware no sean útiles para trabajar con servicios Web.

- El hecho de que los protocolos que utilizan middleware estén orientados a la conexión crea varios inconvenientes a la hora de ser utilizados en servicios Web.
    - Las aplicaciones han de estar integradas en un determinado centro de datos.
    - Hacen difícil mantener una infraestructura balanceada en su carga que permita lograr una alta escalabilidad, ya que cuando una comunicación entre un cliente y un servidor se rompe no se puede cambiar y enviar la siguiente petición a otro servidor.
    - No se gestionan de forma adecuada las interrupciones en la comunicación. Es un gran inconveniente ya que internet no está bajo el control del administrador de red, luego no se puede asegurar ni la calidad, ni la fiabilidad de la comunicación.

---

[7] Router: es un dispositivo de hardware para interconexión de red de computadoras que opera en la capa tres (nivel de red). Este dispositivo permite asegurar el enrutamiento de paquetes entre redes o determinar la ruta que debe tomar el paquete de datos.

[8] Firewall: es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido la organización responsable de la red.

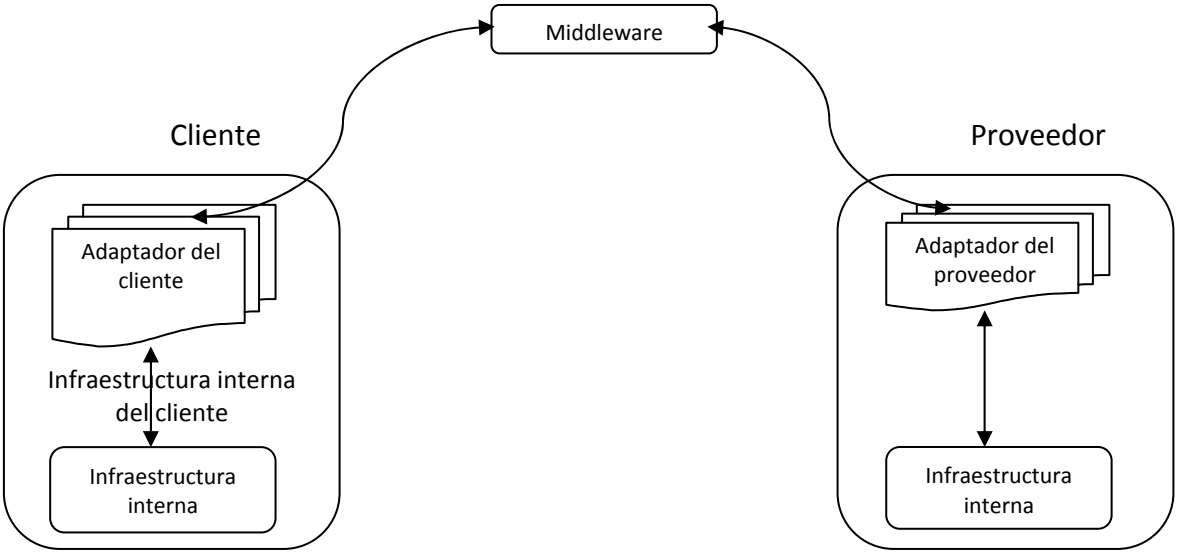

Figura 4 Limitaciones de los Middleware

Una posible solución al problema podría ser la comunicación punto a punto, significa que cuando dos compañías quieren comunicarse, acuerdan utilizar cierta infraestructura middleware. Con lo que implica que si un proveedor tenga varios clientes, tenga que acordar con cada uno de ellos la forma de comunicación.

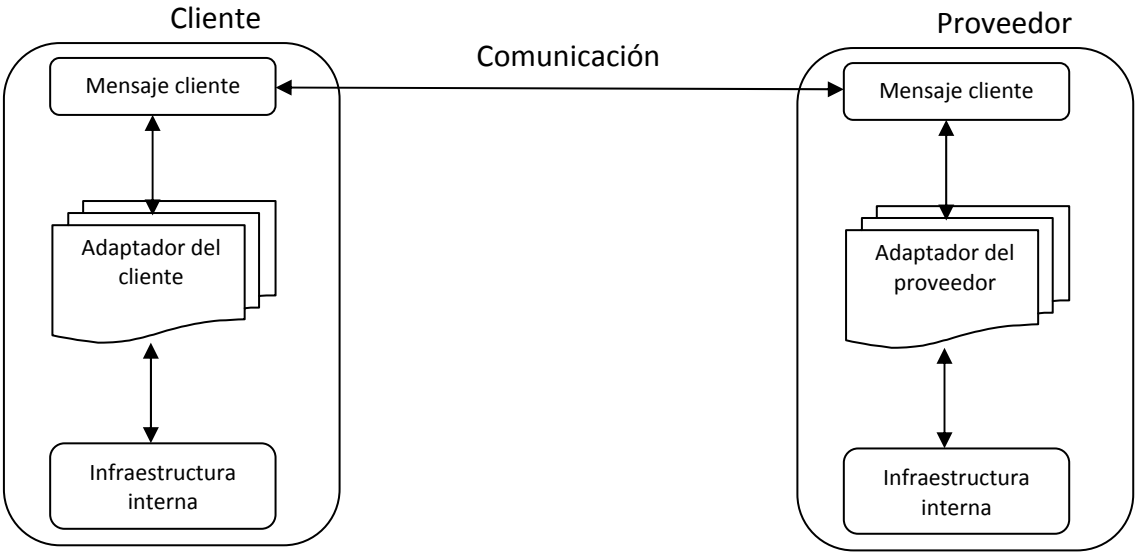

Figura 5 Integración punto a punto entre compañías

Con los middleware tradicionales se entiende que las relaciones entre clientes y proveedores duran poco, por lo que la seguridad de la infraestructura de ambos, no importa mucho. El problema es cuando entran en juego más clientes y más compañías proveedoras, donde se empieza a desconfiar de la falta de seguridad del resto, lo cual

implica introducir cierta seguridad en la infraestructura de los implicados, como sería la autentificación, la encriptación de mensajes y cualquier método de restricciones de accesos a diversas partes de la infraestructura.

El comercio electrónico antes de los Servicios Web, estaba regido por estándares como EDIFACT[9], pero no ha sido ampliamente aceptado por la falta de infraestructura adecuada. Tengamos en cuenta que la Web introdujo ciertos protocolos de iteración HTTP[10] y formato de datos como XML, adoptados como una infraestructura común.

El comercio electrónico desde la aparición de estándares de servicios Web, adopta tres soluciones que resuelven las limitaciones del los middleware tradicionales:

- **Arquitecturas orientadas a servicios**:

  Describiendo un servicio a través de la perspectiva de middleware, un servicio es un procedimiento con una interfaz estable y pública (tengamos en cuenta que el servicio **no** es un conjunto de páginas Web), que es accesible por un cliente. Como los servicios Web, pueden estar implementados por diversas organizaciones, independiente unos de otros. Esto lleva a aplicaciones desacopladas y modulares.

- **Rediseño de los protocolos middleware**:

  Rediseño de los protocolos para trabajar uno a uno y entre compañías.

- **Estandarización**:

  Los diversos estándares han ayudado siempre a la integración de distintas tecnologías dentro de una misma infraestructura y diseñar aplicaciones portables, permitiendo el desarrollo de middleware de bajo coste.

Una de las mayores ventajas que ha puesto por delante a los servicios Web frente a los middleware, es que como las comunicaciones entre empresas se dan mayoritariamente a través de internet y ésta se limita a un conjunto reducido de protocolos y lenguajes (xml, http entre otros). El aprovechamiento de los recursos

---

[9] EDIFACT: Electronic Data Interchange for Administration, Transport and Commerce. Reglamento de las Naciones Unidas para el Intercambio Electrónico de Datos para la Administración, el comercio y el transporte. Comprenden un conjunto internacional de normas acordadas, directorios y directrices para el intercambio electrónico de datos estructurados relacionados con el comercio de bienes y servicios independientes entre sistemas de información computadorizados.

[10] http: protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web.

establecidos de internet, es lo que aprovechan los servicios Web para poder establecer la comunicación entre las empresas.

1. c. Características de los servicios Web

La interfaz de un Servicio Web se puede hacer pública en un servidor mediante un lenguaje de descripción de interfaces, como es WSDL, explicado más adelante en este mismo capítulo. Dicha interfaz ofrece un conjunto de actividades que un cliente puede invocar. El cliente accede al servicio Web usando los estándares de internet.

Para acceder a un servicio Web se utiliza un protocolo especialmente diseñado para ello: SOAP[11]. SOAP se basa en la utilización de http para el transporte de mensajes y el lenguaje XML para la estructura del cuerpo de los mensajes. Todo esto permite que la comunicación entre diversas aplicaciones sea estándar, independientemente de la plataforma en la que esté implantada.

Los servicios Web se caracterizan entre otras cosas por:

- **Interoperabilidad**: A los servicios web, se puede acceder desde cualquier plataforma.
- **Acceso externo desde internet**: Los servicios Web realizan una buena gestión para los accesos que provienen de clientes de internet.
- **Tipos de datos de las interfaces**: Los tipos de datos que utilizan los servicios Web son los que se utilizan en la mayoría de los lenguajes de programación.
- **Uso de los estándares de internet**: Los servicios Web, reutilizan protocolos que ya están en uso, con lo que se ahorra en inventar nuevos estándares que los soporten.
- **Soporte de cualquier lenguaje**: El desarrollo de un servicio Web, no tiene porqué estar ligada a un lenguaje determinado.
- **Soporte para cualquier infraestructura de componentes distribuidas**: Los servicios web, se pueden desarrollar en arquitecturas muy dispares. Por esto, por lo único que hay que preocuparse es del desarrollo y la utilización de los servicios Web.

[11] SOAP: es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

2. Plataformas y protocolos utilizados en servicios Web

Una vez que el servicio ha sido definido, se necesita un conjunto de abstracciones que soporten las comunicaciones que se dan dentro de los servicios web. Éstas, se definen como estándares que regulan las comunicaciones entre servicios, tratándose como protocolos entre diferentes capas. Los bloques necesarios para construir una aplicación remota son los que se muestran en la figura 6.

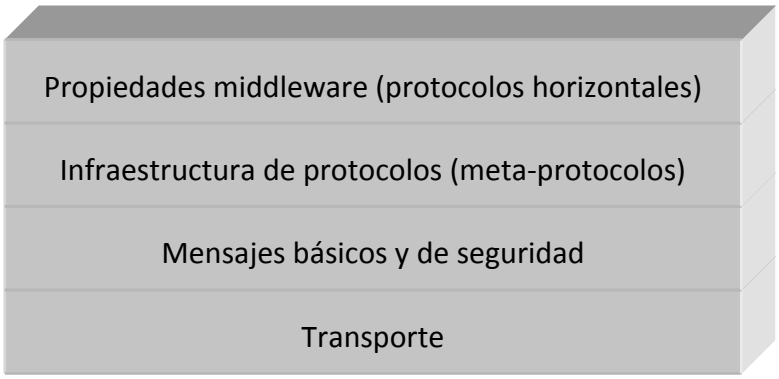

Figura 6 Abstracciones de comunicación mediante servicios Web

- **Middleware (protocolos horizontales):** Idealmente el middleware de los servicios Web debería dar un soporte similar a los middleware convencionales. Como los servicios Web y la infraestructura que los soporta están distribuidos, la infraestructura necesaria más allá de la comunicación se logra por medio de los protocolos. Se denominan protocolos horizontales porque son aplicables a muchos servicios Web.
- **Infraestructuras de protocolos (meta-protocolos):** Los servicios Web se caracterizan por un interfaz y unos protocolos de negocio, son específicos de aplicación, pero el software requerido para desarrollar estos protocolos puede ser implementado como una infraestructura genérica de componentes.
- **Mensajes**: Una vez que se cuenta con un protocolo de transportes se precisa un formato para la información. Este papel lo juega SOAP, aunque no detalla propiedades como se está encriptado. Existen estándares adicionales de cómo utilizar SOAP para implementar características particulares.
- **Transporte**: La red de comunicaciones se oculta detrás de un protocolo de transporte, el más común http.

2. a. WSDL

"WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

Así, WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar que funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL."

WSDL permite describir la interfaz ofrecida por un servicio Web, las operaciones que el servicio puede soportar y los parámetros de entrada y salida.

2.a.i. Estructura de un documento WSDL

Un documento WSDL, está formado por el siguiente esquema general, **una parte abstracta** donde se definen los tipos, los mensajes y las operaciones, y **una parte concreta** donde se define el protocolo de transporte y otras informaciones.

La parte principal de un documento WSDL es el bloque de definiciones, que está compuesto por cinco bloques diferenciados:

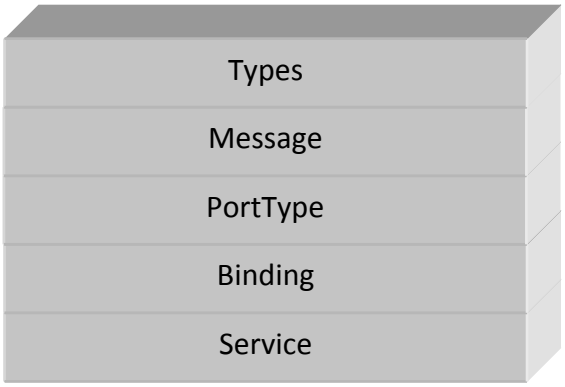

Figura 7 Bloque de definiciones

- **Types**: Contiene información de esquema referenciado en el documento WSDL. El sistema de tipos predeterminado que admite WSDL es de esquema de XML. Si se usa esquema de XML para definir los tipos que contiene el elemento Types

el elemento schema aparecerá inmediatamente como elemento hijo. Se pueden utilizar otros sistemas de tipo tipos por extensión. Si desea, utilizar otro sistema de tipo pude aparecer un elemento de extensibilidad bajo el elemento Types. El nombre de este elemento debería identificar el sistema de tipos utilizados.

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitionstargetNamespace="http://calculator"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://calculator"                xmlns:intf="http://calculator"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <wsdl:types>
     <schema elementFormDefault="qualified"
   targetNamespace="http://calculator"
   xmlns="http://www.w3.org/2001/XMLSchema">

   <element name="divide">
    <complexType>
     <sequence>

      <element name="x" type="xsd:int"/>
      <element name="y" type="xsd:int"/>

     </sequence>
    </complexType>
   </element>

  </schema>

 </wsdl:types>
```

- **Message**: proporciona una abstracción común para el paso de mensajes entre el cliente y el servidor. Como puede utilizar múltiples formatos de de definición de esquema en documento WSDL es necesario de disponer de un mecanismo común de identificar los mensajes. El elemento Message proporciona este nivel común de abstracción al que se hará referencia en otras partes del documento WSDL.

  Pude Aparecer, y normalmente aparecerán, múltiples elementos Message en un documento WSDL, uno para cada mensaje que se comunica entre el cliente y el servidor. Cada mensaje contiene uno o más elementos "Part" que describen las piezas del contenido del mensaje.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
 <wsdl:types>
     ...
 </wsdl:types>

<wsdl:message name="divideRequest">
      <wsdl:part element="impl:divide" name="parameters"/>
</wsdl:message>
<wsdl:message name="divideResponse">
      <wsdl:part element="impl:divideResponse" name="parameters"/>
</wsdl:message>
```

- **PortType**: Contiene un conjunto de operaciones abstractas que representan los tipos de comunicación que pueden producirse entre el cliente y el servidor. Para los Servicios Web se puede pensar en un portType como una definición interna en donde cada método se pude definir como una operación.

  Un portType se compone de un conjunto de operaciones que definen una determinada acción. Los elementos operación se componen de mensajes definidos en el documento WSDL. WSDL define los siguientes tipos de operaciones:

  - **Request-response(petición-respuesta):** Comunicación en la que le cliente realiza una petición y el servidor envía la correspondiente respuesta.
  - **One-way (Unidireccional):** Comunicación del estilo documento en la que el cliente envía un mensaje pero no recibe una respuesta del servidor indicando el resultado del mensaje procesado.
  - **Solicit-response (solicitud-respuesta):** La contraria a la operación petición-respuesta. El servidor envía una petición y el cliente le envía de vuelta una respuesta.
  - **Notification (Notificación):** La contraria a la operación unidireccional. El servidor envía una comunicación del estilo documento al cliente.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
 <wsdl:types>
     ...
 </wsdl:types>

<wsdl:message name="divideRequest">
      ...
</wsdl:message>

<wsdl:portType name="calculadora2">
  <wsdl:operation name="divide">
    <wsdl:input message="impl:divideRequest" name="divideRequest"/>
    <wsdl:output message="impl:divideResponse"
   name="divideResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

- **Binding:** Contiene las definiciones de la asociación de un protocolo como SOAP a un determinado bindingType. Las definiciones binding especifican detalles de formatos del mensaje y el protocolo.

  Las definiciones binding también indican el número de comunicaciones de red que se requieren para realizar una determinada acción. Por ejemplo, una llamada SOAP sobre HTTP podría involucrar un intercambio de comunicación HTTP, pero esa misma llamada sobre SMTP podría involucrar dos intercambios de comunicaciones de SMTP.

  La asociación se logra utilizando elementos de extensión que contiene cada protocolo, para especificar los detalles del protocolo y el formato de los mensajes. Para un determinado protocolo los elementos de extensión se suelen utilizar para decorar las acciones individuales de una operación y la propia operación con la información de asociación del protocolo. A veces los elementos de extensión se utilizan en el propio nivel portType.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
 <wsdl:types>
      ...
 </wsdl:types>

<wsdl:message name="divideRequest">
       ...
</wsdl:message>

<wsdl:portType>
      ...
</wsdl:portType>

<wsdl:binding name="calculadora2SoapBinding"
       type="impl:calculadora2">
<wsdlsoap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="divide">
          <wsdlsoap:operation soapAction=""/>
          <wsdl:input name="divideRequest">
             <wsdlsoap:body use="literal"/>
          </wsdl:input>
          <wsdl:output name="divideResponse">
             <wsdlsoap:body use="literal"/>
          </wsdl:output>
       </wsdl:operation>
 </wsdl:binding>
```

- **Service:** Un servicio es un grupo de puertos relacionados y se definen en el elemento service. Un puerto es un extremo concreto de un Servicio Web al que se hace referencia por una dirección única. Los puertos que se definen en determinado servicio son independientes. Por ejemplo, la salida de un puerto que no puede utilizarse como una entrada de otro.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
 <wsdl:types>
       ...
 </wsdl:types>
<wsdl:message name="divideRequest">
       ...
</wsdl:message>
<wsdl:portType>
       ...
</wsdl:portType>
<wsdlsoap:binding>
       ...
</wsdl:binding>

<wsdl:service name="calculadora2Service">
       <wsdl:port binding="impl:calculadora2SoapBinding"
             name="calculadora2">
          <wsdlsoap:address location="http://localhost:10510
           /Ejemplo/services/calculadora2"/>
       </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

WSDL permite que un servicio y un cliente constituyan un pacto en lo que se refiere al transporte de mensajes y su contenido, a través de un documento procesable por distintos terminales. WSDL simboliza una especie de acuerdo entre el proveedor y el cliente. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes entre ambos actores.

2. b. SOAP

En todo el proceso de creación de servicios Web, se mezclan una serie de tecnologías que hacen posible esta comunicación entre aplicaciones cliente y servicios web. A parte de WSDL, estaría SOAP, que se trata de un protocolo basado en XML, que permite la iteración entre varios terminales y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etc.

SOAP es un protocolo diseñado para acceder remotamente a los servicios Web, pero se puede utilizar para invocar aplicaciones remotas de distintos tipos. SOAP se diferencia de otros protocolos de invocación de aplicaciones por las siguientes características:

- **SOAP no está asociado a un solo lenguaje o a una sola plataforma**. No especifica ninguna API[12], lo que permite abstraer el lenguaje sobre el que pueda estar implementado el objeto invocado, lo cual, facilita la reutilización de código.
- **No se encuentra fuertemente asociado a ningún protocolo de transportes**. SOAP describe su transporte en http. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- **No está atado a ninguna infraestructura de middleware**. La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.

[12] API: Application Programming Interface - Interfaz de Programación de Aplicaciones. Es el conjunto de funciones y que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- **Aprovecha los estándares ya existentes**. A la hora de desarrollar SOAP, se evitó reinventar las cosas. Se optó por extender los estándares existentes para que coincidieran con las necesidades surgidas en el desarrollo.
- **Permite la interoperabilidad entre múltiples entornos.** SOAP se ha desarrollado sobre estándares existentes, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse, mediante mensaje SOAP, con aplicaciones que se ejecuten en distintas plataformas.

2.b.i. Estructura de un mensaje SOAP

SOAP especifica el formato de los mensajes que se van a transmitir. El mensaje SOAP está formado por un envelope (envoltura), cuya estructura está formada por los elementos header y body. (1)

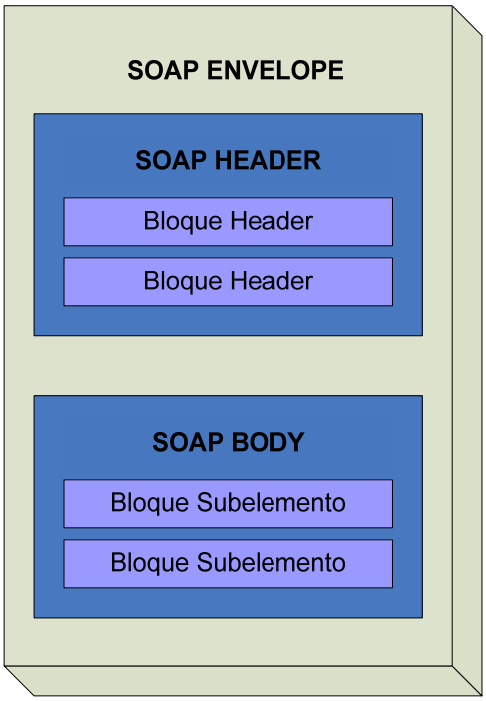

Figura 8 Estructura de un mensaje SOAP

La *cabecera* o *header*, es opcional y sirve para dar información adicional al cuerpo del documento. Es en esta parte donde se ha de decir las peculiaridades del documento SOAP, como por ejemplo, si el documento estuviera cifrado. Como el campo cabecera es opcional, muchas aplicaciones lo ignoran, para evitar que sea descartada se debe indicar con la etiqueta “mustUnderstand=1”.

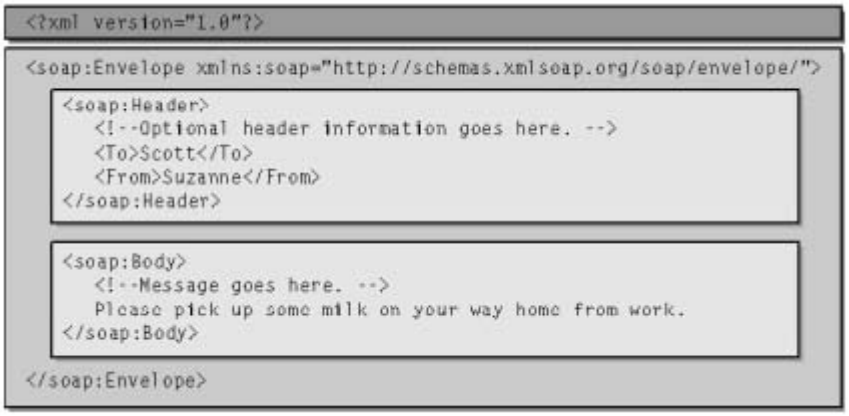

Figura 9 Esqueleto de un mensaje SOAP

Después de la información adicional que proporciona la cabecera, se encuentra el *body* o *cuerpo*, el cual, si es obligatorio para todo tipo de aplicaciones SOAP.

2.b.ii. Tipos de datos en SOAP

El bloque body de divide en diferentes apartados. Uno de estos apartados, siempre se ha de definir con el nombre del método, y dentro de este, tantos bloques como parámetros necesite la invocación. Los parámetros deberán tener el mismo nombre y posición que en la definición del método.

En los mensajes SOAP, los parámetros que se utilizan, se agrupan en dos tipos diferenciados:

- **Tipos simples**: Los tipos simples, son un conjunto de tipos estándar para muchos lenguajes de programación. Son por ejemplo los tipos int, boolean, float…, una instancia de estos tipos es codificada como un elemento XML. Un ejemplo de tipos simples es el siguiente:

  Invocación:

```
public int Suma (int x, int y)
{
      return x+y;
}
```

Mensaje SOAP equivalente a la invocación:

```
<?xml versión=”1.0”?>
<soap:Envelope
	xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”>
	<soap:Body>
		<Suma>
			<x>1</x>
			<y>3</y>
		</Suma>
	</soap:Body>
</soap:Envelope>
```

Mensaje SOAP obtenido tras la invocación:

```
<?xml versión=”1.0”?>
<soap:Envelope
	xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”>
	<soap:Body>
		<SumaResult>
			<Result>4</ Result >
		</SumaResult>
	</soap:Body>
</soap:Envelope>
```

- **Tipos Estructurados o compuestos**: SOAP define tipos de datos compuestos correspondientes a patrones estructurales presentes con frecuencia en lenguajes de programación. Los tipos de datos compuestos a su vez pueden contener tipos de datos compuestos. La especificación de SOAP no limita la serialización de otros tipos de datos compuestos. Se codifica la estructura como un elemento XML y dentro del mismo, cada uno de sus campos.

  Los tipos de datos compuestos, se dividen en dos:

  - **Estructuras**: Una estructura es un tipo de datos compuesto en la cual los nombres de acceso de sus valores miembros es la única distinción entre ellos.

    Invocación:

```
public struct angulo
{
	public int grados;
	public int minutos;
	public int segundos;
}
```

Mensaje SOAP equivalente a la invocación:

```
<?xml versión=”1.0”?>
<soap:Envelope
xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”>
	<soap:Body>
		<girar>
			<a>
				<grados>10</grados>
				<minutos>13</minutos>
				<segundos>43</segundos>
			</a>
		</girar>
	</soap:Body>
</soap:Envelope>
```

- **Arrays**: Es un tipo de dato compuesto donde la posición de sus miembros sirve como única distinción entre ellos. Los arreglos SOAP están definidos como "SOAP-ENC:Array".

  Ejemplo de vectores: int valores = [4,3,2]

  media(valores);

  Mensaje SOAP obtenido tras la invocación:

```
<?xml versión=”1.0”?>
<soap:Envelope
xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:soap-enc=”http://schemas.xmlsoap.org/soap/encoding/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
	<soap:Body>
		<Media>
		<valores soap-enc:arrayType=”xsi:int[3]”>
			<int>4</int>
			<int>3</int>
			<int>2</int>
		</valores>
		</Media>
	</soap:Body>
</soap:Envelope>
```

Para mejorar el rendimiento de las aplicaciones basadas en Servicios Web, se han desarrollado diversas tecnologías complementarias a SOAP, que aceleran el envío de los mensajes como por ejemplo el mecanismo MTOM[13] y los recursos y operaciones que se transmiten dentro de estos mensajes.

[13] MTOM: Mecanismo para transmitir los datos adjuntos binarios grandes con mensajes SOAP como bytes sin formato, permitiendo los mensajes menores.

2. c. Axis (4)

Axis es esencialmente una plataforma para construir procesos SOAP, como pueden ser aplicaciones cliente, etc. Axis también incluye entre otras funcionalidades, un servidor independiente, soporte extendido para el Lenguaje de descripción de servicios (WSDL) y una herramienta de monitorización de paquetes TCP/IP.

Axis trata la comunicación de objetos a datos SOAP cuando se envía o recibe resultados a través de la red. Axis también implementa la API JAX-RPC, uno de los estándares para programar servicios Web.

2.c.i. La ruta del mensaje en Axis

En general, Axis maneja cualquier proceso que tenga que ver con el tratamiento de mensajes entre aplicaciones. Cuando el procesador lógico del núcleo de Axis está en ejecución, una serie de gestores internos son invocados en un orden predeterminado. Éste orden, viene fijado por dos factores muy importantes, la configuración y si el motor es un cliente o un servidor.

El objeto enviado a cada una de las invocaciones de los gestores internos es un MessageContext o mensaje de contexto. Un mensaje de contexto es una estructura definida que contiene varias partes diferenciadas: Un mensaje de petición, un mensaje de respuesta y un conjunto de propiedades específicas.

Hay dos formas básicas en las que Axis puede ser invocado:

- **Como un servidor**: se creará un mensaje de contexto y se invocará a la plataforma de procesamiento del mensaje.
- **Como un cliente**: El código, normalmente asistido por el modelo de programación cliente de Axis, generará un mensaje de contexto e invocará a la plataforma de procesamiento del mensaje.

De cualquier forma, el trabajo de la plataforma es simplemente pasar el resultado del mensaje de contexto a través del conjunto de gestores internos, que tratan al mensaje de contexto según como hayan sido programados.

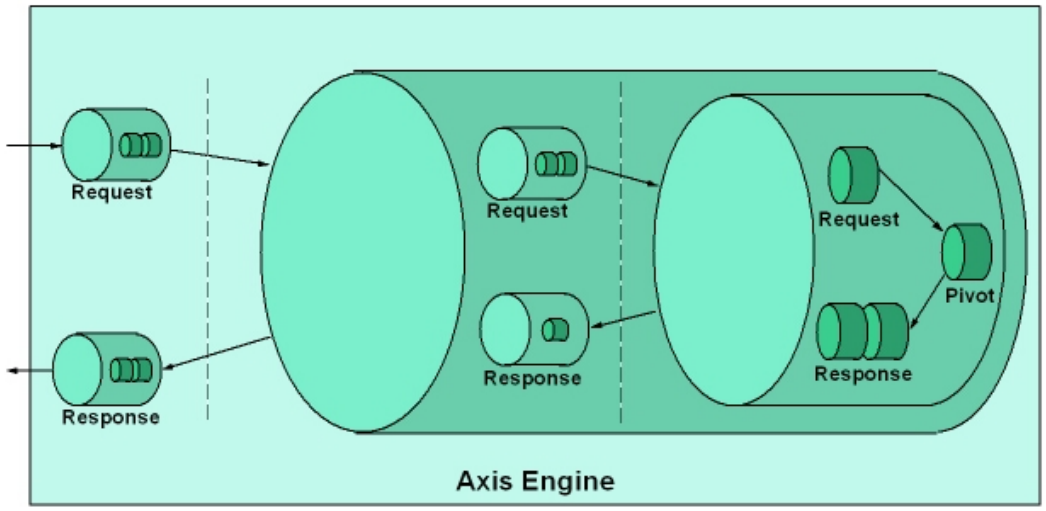

Figura 10 Esquema del motor de Axis

2.c.ii. Subsistemas de Axis

Axis, está compuesto por varios subsistemas que trabajan juntos con el objetivo de mantener la modularidad del sistema. Éstos subsistemas, están divididos en capas, las cuales, permiten que únicamente, se utilicen las partes del sistemas que sean necesarias en ese preciso momento.

La figura 11 muestra las diferentes capas de los sub sistemas Axis. Las capas inferiores son independientes de las capas superiores. Las cajas apiladas representan alternativas, mutuamente independientes, pero no excluyentes entre sí. Por ejemplo http, SMTP y JMS son protocolos muy independientes pero pueden tener una función común que los utilice a todos.

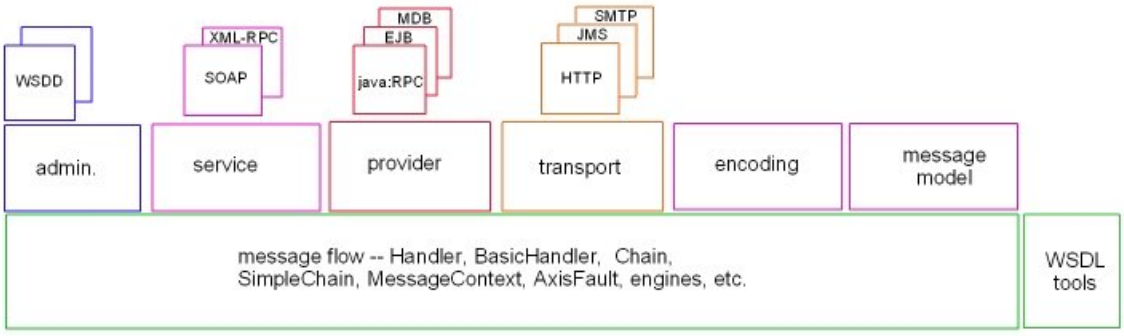

Figura 11 Sistema de capas y sub-sistemas de Axis

2.c.iii. Visión general de Axis

Para el desarrollo de WSDLSniffer, Axis resulta altamente esencial ya que da a la aplicación el acceso a los servicios y su descripción dentro de un índice.

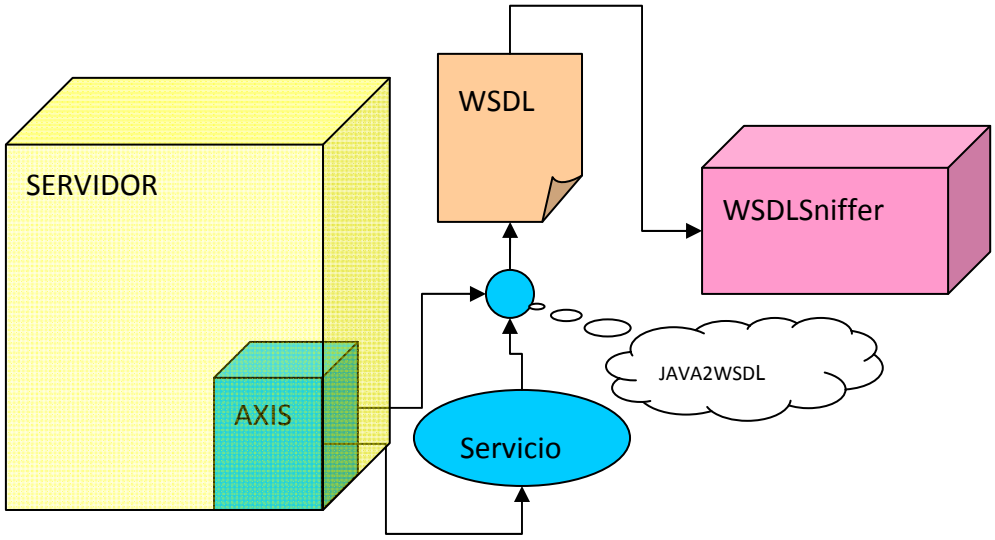

Figura 12 Esquema de dependencia de Axis

Dentro del servidor, una pequeña parte corresponde a AXIS. AXIS, da soporte a los servicios que están definidos en el servidor. Esos servicios, se representa mediante documentos WDSL, explicados anteriormente. Se utiliza este documento WSDL como documento base para el tratamiento con la aplicación WSDLSniffer. Se puede observar en la figura 12 como AXIS proporciona los servicios y mediante un programa que hace corresponder el servicio implantado con un documento WSDL.

3. API´s estudiadas para el desarrollo de la aplicación WSDLSniffer 1.0 [14]

Para el desarrollo de la aplicación, se han utilizado diversas API´s que, para el correcto funcionamiento de WSDLSniffer 1.0, han sido minuciosamente investigadas para averiguar los pros y los contras de cada una de éstas. Las API´s estudiadas, se utilizarán en el desarrollo o se descartarán según unos puntos a tener en cuenta, como por ejemplo, si el desarrollo de la API estudiada sigue en marcha.

El punto en común entre estas API´s, es la funcionalidad que tienen sobre los documentos WSDL. Estas API´s son capaces de leer cualquier documento WSDL, es decir, mediante la correspondencia de etiquetas, estas API´s son capaces de discernir las partes de un documento WSDL, dando al programa, por ejemplo, las operaciones que tiene este documento, siendo útiles para conocer las operaciones que proporciona un servicio. Estas API´s se basan en la estructura anteriormente comentada del documento WSDL.

A continuación se pasa describir brevemente las API´s que se han investigado a lo largo del desarrollo de la aplicación WSDLSniffer 1.0.

3. a. wsdl4j-1_6_2

wsdl4j-1_6_2 API recupera el documento con extensión .wsdl, para su tratamiento en java y ha sido desarrollado por la comunidad java. Las principales características de esta API son:

- La lectura, escritura, creación y modificación de definiciones WSDL, elementos extensibles.
- La especificación de interfaces para representar los elementos extensibles en el documento wsdl.
- Para definir mecanismos que permitan la lectura, escritura y representación de elementos extensibles.

[14] WSDLSniffer 1.0: Aplicación desarrollada por Pablo Ortiz y Lourdes Riestra, para el trabajo de investigación de fin de carrera.

Esta API presenta un problema por el cual, ha sido descartada para el desarrollo de la aplicación WSDLSniffer 1.0, es que wsdl4j-1_6_2 no provee de soporte para consultar a manipular esquemas XML[15], siendo en este esquema donde se definen los tipos de datos contenidos en las operaciones, necesarios para el objetivo principal De WSDLSniffer 1.0.

3. b. XMLBeans-2.3.0

XMLBeans es una manera de hacer corresponder documentos XML con objetos JAVA. Se han hecho pruebas con esta API, con la intención de poder referenciar el esquema XML de documento WSDL, para su tratamiento por objetos de JAVA. Las pruebas de esta API, se han hecho a la par de las pruebas de la API anteriormente descrita, wsdl4j-1_6_2, ya XMLBeans-2.3.0 resolvía los inconvenientes de esta.

Al probar las diferentes librerías conjuntas de las dos API´s, no se conseguía referenciar de forma correcta el esquema XML del documento WSDL, por lo que se descartó al no prestar soporte para el objetivo principal del desarrollo de la aplicación WSDLSniffer 1.0 . Además de este gran inconveniente, el plugin de eclipse, está parado desde el año 2005, por lo que sería utilizar una herramienta de desarrollo desactualizada, que podría ocasionar inconvenientes en un futuro.

3. c. Castor1.2

Castor es una plataforma de código abierto para JAVA, que provee un binding[16] de JAVA a XML. Igual que en el caso anterior, Castor, no consigue referenciar de forma correcta el esquema XML del documento WSDL, inconveniente principal para el desarrollo de la aplicación WSDLSniffer 1.0.

---

[15] XML: Extensible Markup Language («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

[16] Binding: es una “ligadura” o referencia a otro símbolo más largo y complicado, y que se usa frecuentemente. Este otro símbolo puede ser un valor de cualquier tipo, numérico, de cadena, etc o el nombre de una variable que contiene un valor o un conjunto de valores.

3. d. Woden (4)

El proyecto Woden es un subproyecto de APACHE[17] Web Service en perfeccionamiento, para desarrollar una librería de clases JAVA para leer, manipular, crear y escribir, documentos WSDL. Inicialmente soporta la especificación 2.0 de WSDL.

El objetivo inicial del proyecto Woden es el desarrollo de un procesador que implemente la especificación W3C WSDL 2.0.

Las funcionalidades que ofrece Woden actualmente incluyen:

- Un mecanismo para obtener un lector WSDL.
- Una aplicación DOM[18] del lector basada el APACHE Xerces.
- Configurar las características apropiadas del lector.
- Usar el lector para convertir un documento WSDL de una URL especificada, al modelo de objetos Woden WSDL 2.0
- Existen dos formas de usar el modelo de objetos:
  - Representando el modelo abstracto de componentes de WSDL 2.0
  - Hace corresponder los elementos y atributos XML, en el espacio de nombres WSDL.

Para la utilización de Woden en Eclipse, se importan las librerías descargadas de http://ws.apache.org/woden/. No obstante, el API WSDLSniffer 1.0, lleva integrada Woden, sin la necesidad de importar estas librerías necesarias para el correcto funcionamiento del API.

---

[17] Apache: es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1[1] y la noción de sitio virtual.

[18] DOM: es esencialmente un modelo computacional a través del cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML y XML. Su objetivo es ofrecer un modelo orientado a objetos para el tratamiento y manipulación en tiempo real (o de forma dinámica) a la vez que de manera estática de páginas de Internet.

3.d.i. Mapeo de los elementos WSDL a la API

WSDL element	Element API	Component API
<description>	DescriptionElement	Description
<documentation>	DocumentationElement	
<import>	ImportElement	
<include>	IncludeElement	
<types>	TypesElement	
<interface>	InterfaceElement	Interface
<fault>	InterfaceFaultElement	InterfaceFault
<operation>	InterfaceOperationElement	InterfaceOperation
<input>	InterfaceMessageReferenceElement	InterfaceMessageReference
<output>	InterfaceMessageReferenceElement	InterfaceMessageReference
<infault>	FaultReferenceElement	InterfaceFaultReference
<outfault>	FaultReferenceElement	InterfaceFaultReference
<binding>	BindingElement	Binding
<fault>	BindingFaultElement	BindingFault
<operation>	BindingOperationElement	BindingOperation
<input>	BindingMessageReferenceElement	BindingMessageReference
<output>	BindingMessageReferenceElement	BindingMessageReference
<infault>	FaultReferenceElement	BindingFaultReference
<outfault>	FaultReferenceElement	BindingFaultReference
<service>	ServiceElement	Service
<endpoint>	EndpointElement	Endpoint
<feature>	FeatureElement	Feature
<property>	PropertyElement	Property

XML Schema element		
<xs:import>	ImportedSchema	
<xs:schema>	InlinedSchema	
<xs:element name="..">		ElementDeclaration
<xs:complexType name="..">		TypeDefinition

4. Introducción a WSDLSniffer 1.0:

WSDLSniffer es una API creada para la generación de descripciones sencillas de Web Services sobre AXIS. Mediante la lectura de archivos WDSL contenidos en un servidor, generaremos todos los servicios contenidos.

WSDLSniffer trabaja con otra API que le da soporte, ésta es Woden, de Apache Software Foundation, anteriormente descrita. Woden está desarrollada para la lectura de documentos WSDL 2.0.

Frente a otras APIs que incluyen la lectura de documentos WSDL Woden proporciona la posibilidad de leer directamente el esquema de tipos de datos complejos que generalmente tiene anidado el documento WSDL. Encontrar una API capaz de leer el esquema XML dentro de un documento WSDL ha resultado relativamente difícil, ya que en la mayoría de las APIs, solo se puede referenciar el esquema XML, sin posibilidad de lectura.

El principal objetivo de WSDLSniffer es la generación de un documento simple XML, con la información de las operaciones ofrecidas por el servicio, filtrando así toda la información relativa a modos de conexión, puertos, etc., que contiene un documento WSDL. Estos documentos WSDL serán recuperados de un servidor AXIS mediante la lectura del documento que vincula los servicios con sus documentos WSDL. El siguiente ejemplo, muestra como trabajo la aplicación WSDLSniffer 1.0. La aplicación recoge un documento WSDL, ésta, busca el esquema XML y selecciona las operaciones y ofrece el servicio. Una vez leídas las operaciones, la aplicación WSDLSniffer 1.0, recoge estas operaciones y los parámetros de las mismas. En la figura 13 se puede apreciar un pequeño esquema explicativo del trabajo de la aplicación WSDLSniffer 1.0.

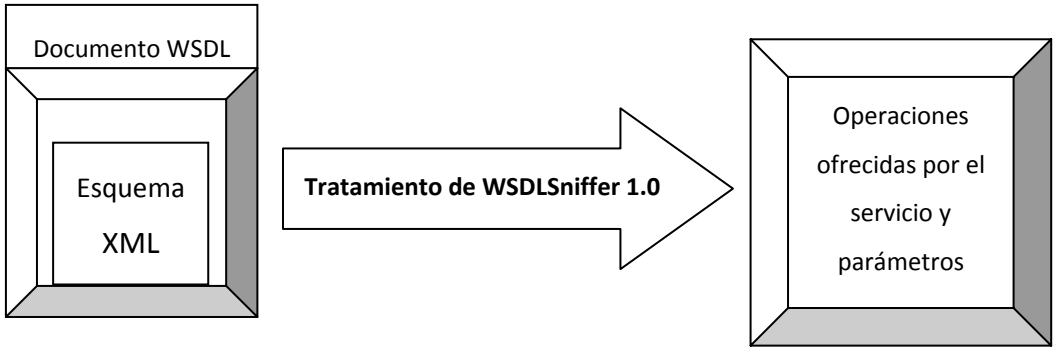

Figura 13 Esquema general de WSDLSniffer 1.0

4. a. Clases desarrolladas en WSDLSniffer

El siguiente gráfico muestra una visión general de las clases desarrolladas para WSDLSniffer y un detalle de sus métodos. Se muestran las clases y el apoyo de unas sobre otras en la ejecución de la API. Se ha de tener en cuenta que la clase principal, es Sniffer, la cual tiene la función main. En orden ascendente, el soporte es el siguiente: Attribute, da soporte a Operation, que da soporte a Service, que da soporte a Sniffer.

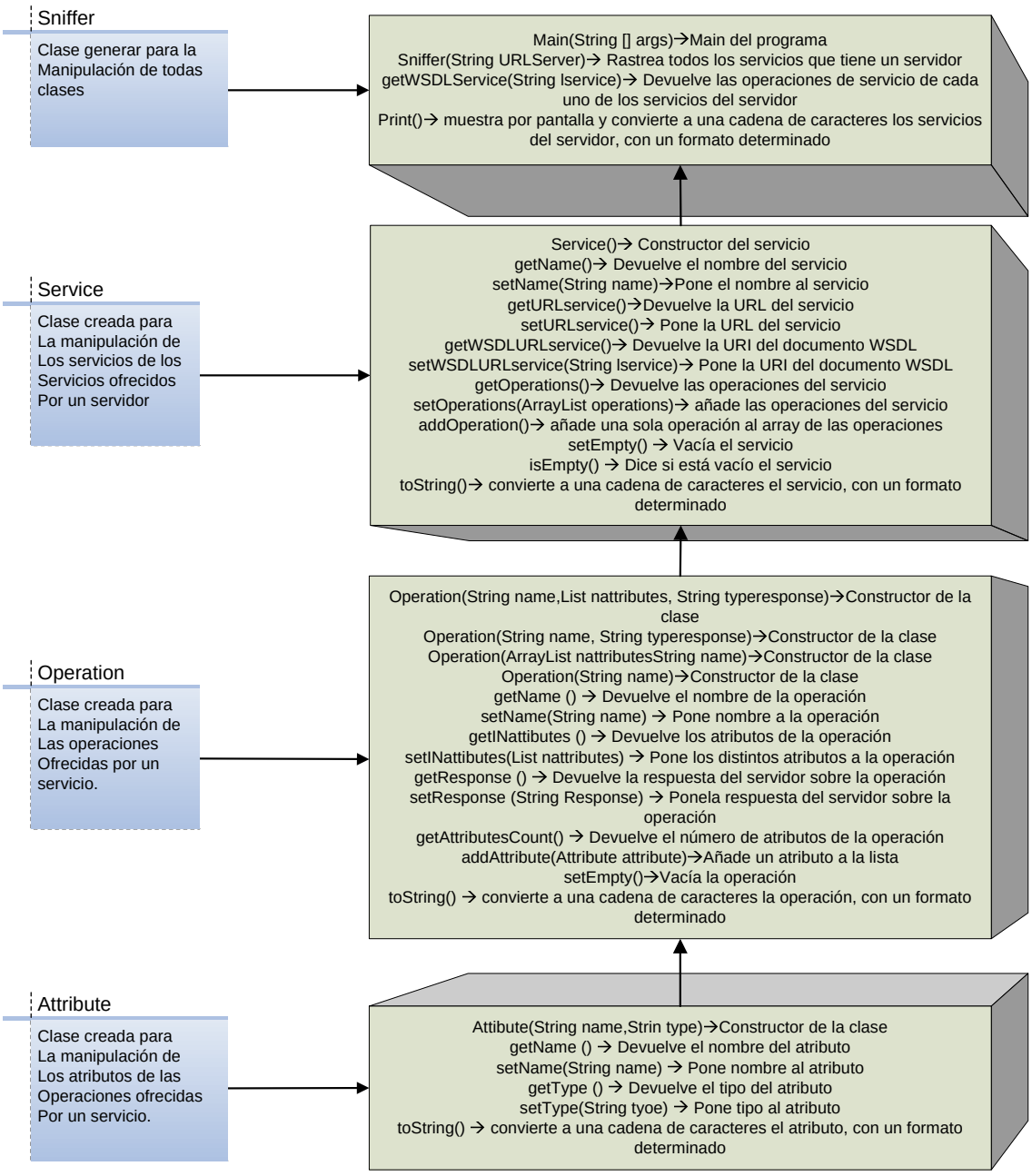

Figura 14 Esquema API WSDLSniffer

4. b. Manual de usuario de WSDLSniffer

El funcionamiento de la API WSDLSniffer es bastante intuitivo. Se puede utilizar la API WSDLSniffer de la siguiente manera:

Los servidores AXIS ofrecen un índice de los servicios contenido en el servidor. Este índice es una página html con el enlace al documento .wsdl de los servicios. También AXIS da una visión general de las operaciones que ofrece cada uno de los servicios del servidor. Se ha de tener en cuenta que los servicios con los nombres de AdminService y Version son unos servicios generados automáticamente por Axis. Por lo que, no será útil analizarlos para un tratamiento de los servicios que se ofrecen en el documento WSDL, que en ese momento se estuviera analizando.

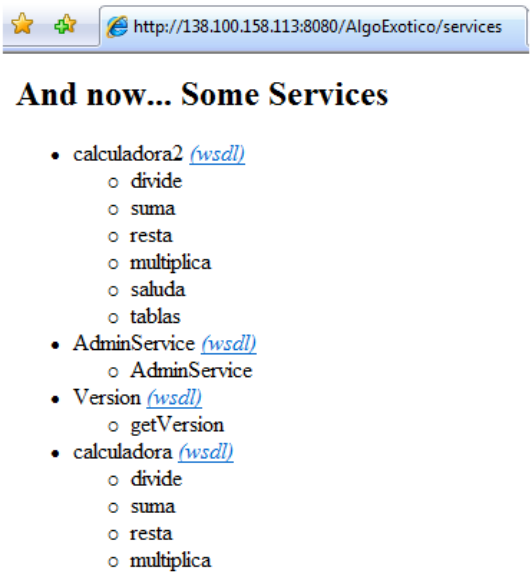

Figura 15 Página generada automáticamente por Axis

El documento a la figura 15, es el documento que la aplicación WSDLSniffer lee automáticamente después de haber introducido la ruta del servidor a analizar. Funciona de la siguiente manera:

Se introduce una ruta para analizar, en el ejemplo que vamos a utilizar la ruta sería, http://138.100.158.113:8080/AlgoExotico/services

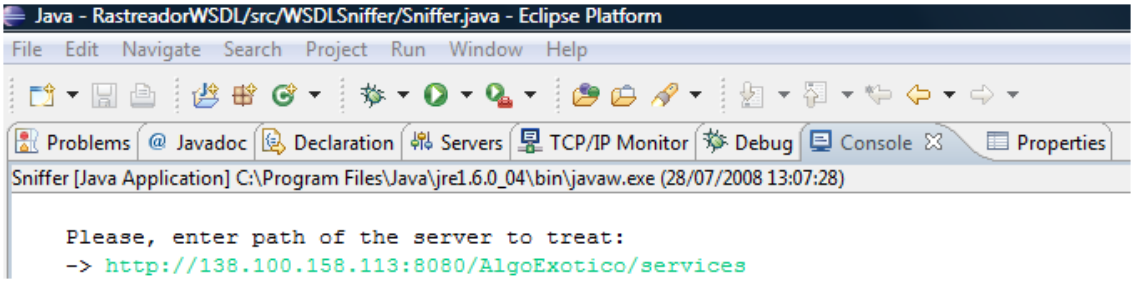

Figura 16 WSDLSniffer ejecución

Una vez introducido la ruta del documento WSDL a tratar, el resultado obtenido es una pantalla como la que se muestra en la figura 17, la cual, da la información requerida de cada uno de los servicios de la ruta introducida.

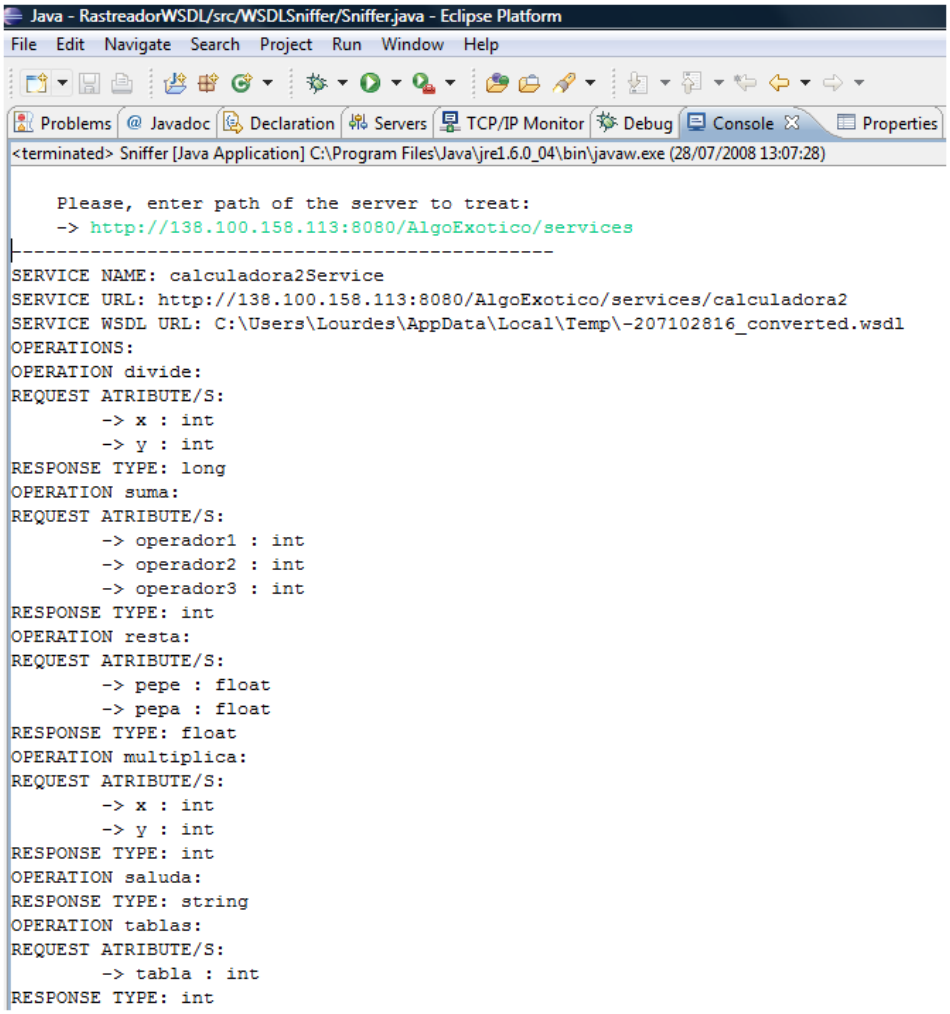

Figura 17 WSDLSniffer ejecución

La información de la figura 17 estará estructurada de la siguiente manera, para cada uno de los servicios del documento tratado:

1. Nombre del servicio
2. URL del servicio
3. URL del documento de descripción del servicio

Para cada servicio:

4. Operaciones del servicio
    a. Nombre de la operación
    b. Atributo/s si lo hubiera
        i. Nombre y tipo del atributo
    c. Tipo de respuesta si lo hubiera

Se al tener en cuenta que se puede utilizar las funcionalidades de la API como operaciones desligadas unas de otras. Utilizando en cada caso la operación desarrollada dentro de cada clase por sí sola, importando, claro, la API WSDLSniffer.

4. c. Descripción de métodos definidos

WSDLSniffer está dividida en varios packages[19], que dan soporte a las necesidades creadas de la lectura de un esquema XML en un documento WSDL. WSDLSniffer, tiene como primer paquete de soporte, al API Woden, descrita en un bloque anterior. Si se desea consultar más información sobre esta API, se puede encontrar en la web oficial, http://ws.apache.org/woden/.

Esta API, proporciona la lectura y tratamiento de un documento WSDL, como se ha descrito anteriormente, pero no obstante, no son suficientes los métodos descritos en ésta, por ello ha sido necesaria la realización de paquetes complementarios a la API Woden. Por ejemplo, una de los métodos de lectura que proporcionaba Woden, ha sido modificado para el correcto funcionamiento de WSDLSniffer.

A continuación se pasan a describir los paquetes y métodos que componen el API WSDLSniffer 1.0

[19] Packages: Paquetes

4.c.i. WSDLSniffer.Snnifer.java

WSDLSniffer.Sniffer.java es la parte principal de la aplicación, ya que es aquí donde se encuentra el main de la API. *Sniffer.java,* contiene el método público *Sniffer(String URL Server),* que desencadena todo el proceso de tratamiento del documento WSDL.

Para el ejemplo anterior, el constructor recibe la URL del servidor y mediante el método getLinks() de la clase WSDLSniffer.utils.links.java, recupera las direcciones de los documentos .wsdl de cada servicio.

Para cada uno de estos servicios se creará (mediante el método getWSDLService())un objeto Service, de la susodicha clase.

4.c.ii. WSDLSniffer.utils

Dentro de este paquete, se encuentran todos los métodos necesarios para tratamientos secundarios, como pueden ser, el obtener un link de un documento HTML, o convertir un documento WSDL 1.1 en otro documento WSDL 2.0. A continuación en se pasa a describir las tres clases que constituyen este paquete.

- **Convert**: esta clase, es relativamente similar a una clase convert, del API Woden. Ha sido modificada, ya que no cumplía todos los requisitos específicos, para convertir documentos WSDL 1.1 en WSDL 2.0, que es con el formato con el que trabaja el API Woden.
- **Links**: clase que devuelve los hiperenlaces de un documento HTML, necesarios para la lectura de los documentos WSDL que se encuentran en un servidor.
- **Utils**: Crea un directorio para la utilización en local del WSDL, eliminando problemas y posibles retrasos en el análisis de los servicios.

4.c.iii. WSDLSniffer.wsdl

En la figura 19 se puede ver un esquema descriptivo de esta parte. Es en este paquete donde se han desarrollado las clases necesarias para el correcto funcionamiento de la API WSDLSniffer. Se ha de tener en cuenta, que para entender la creación de este paquete, un servicio cualquiera estará formado por distintas operaciones (en el caso tuviera), y esas operaciones, siempre tendrán unos atributos que la distingan.

A continuación se pasa a describir las diferentes clases que forman este paquete:

- **Attribute**: en esta clase, se define el elemento más pequeño de un servicio. Crear un objeto de la clase Attribute, será necesario si el servicio tratado tiene operaciones, en caso contrario, no se necesita la esta clase para ningún motivo. Esta clase define un objeto Attribute, con sus métodos get y set, a sus atributos. También posee un método para la conversión a cadena de caracteres.

- **Operation**: en esta clase, se definen los objetos a operaciones, es decir, cuando un servicio posee unas determinadas operaciones, que no quiere decir que en todos los casos un servicio poseerá operaciones, se crea un objeto a esta clase. Esta clase posee diversos constructores que le permiten crear diferentes operaciones, dependiendo de las necesidades que se creen en el momento de la lectura de los servicios. Estos cuatro constructores distintos, se definen dependiendo de las operaciones que se lean en el servicio.

  Aparte de tener, la clase Operation, los get y set correspondientes, tiene un método *addAttribute(Attribute attribute)*, que añaden un atributo a la lista de objetos Attribute.

  Esta clase también posee un método para la conversión a cadena de caracteres.

- **Service**: esta clase, se podría decir que es la principal de todos, ya que siempre un documento WSDL, poseerá unos determinados servicios, ya que éste es el objetivo de la aplicación WSDLSniffer. Esta clase únicamente tiene un constructor, ya que lo que diferencia un servicio u otro es las operaciones que cada uno posee. A destacar, aparte de los set y get correspondientes, posee una serie de métodos que permiten la manipulación controlada de objetos de la clase Operation.

  Otros métodos a destacar dentro de esta misma clase, son los que permiten coger o poner una URL para el documento WSDL.

4.c.iv. Esquema de clases de la API WSDLSniffer 1.0

La figura 18 muestra una visión general de los paquetes, clases y métodos que forman el API WSDLSniffer. Para poder ver en detalle los paquetes y clases que maneja WSDLSniffer 1.0, se han creado las figuras 19 y 20.

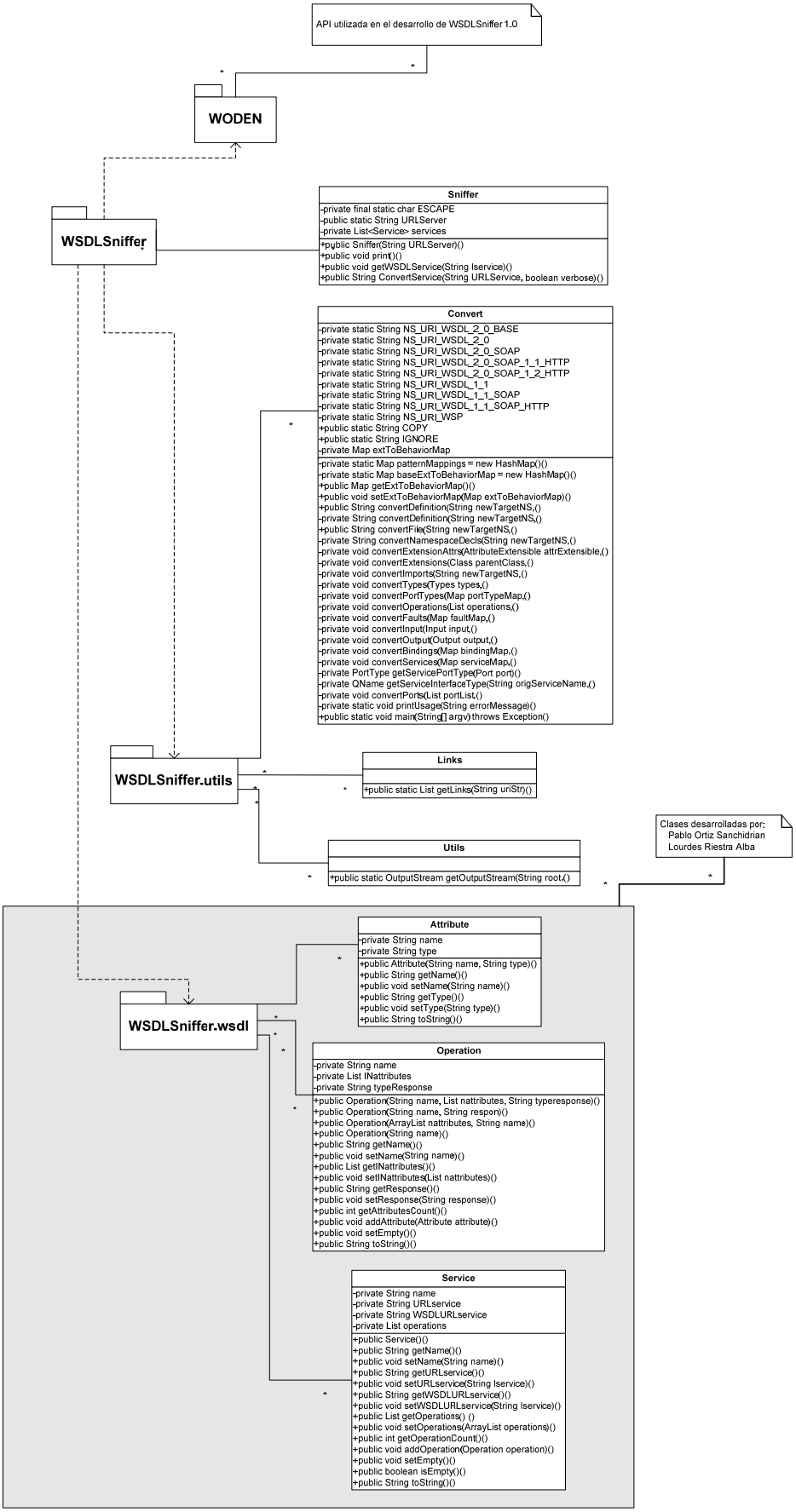

Figura 18 Diagrama general de clases WSDLSniffer 1.0

Primera parte del esquema: API´s utilizadas en el desarrollo de la aplicación WSDLSniffer 1.0.

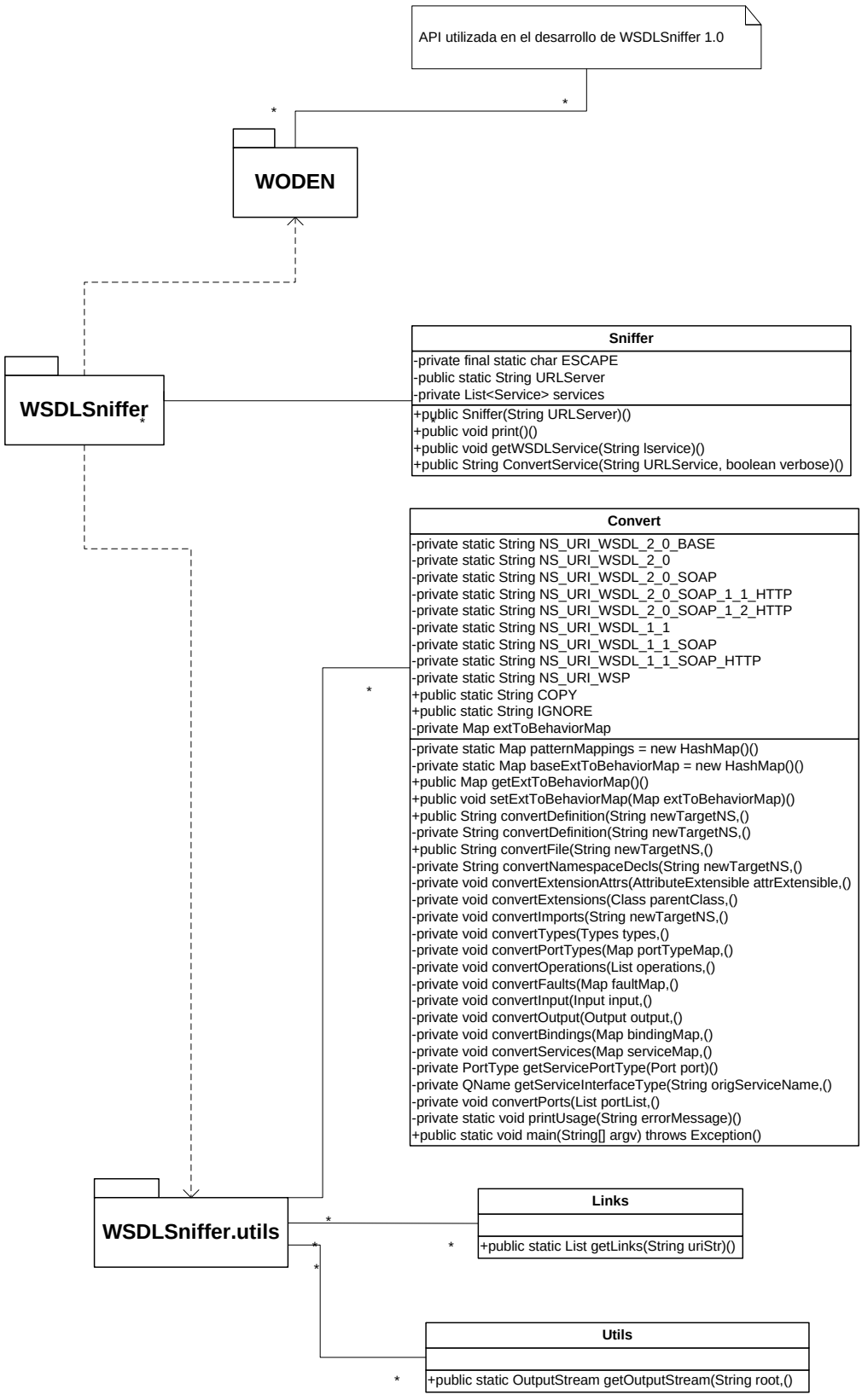

Figura 19 Clases utilizadas por WSDLSniffer

Segunda parte del esquema: API´s desarrolladas para la aplicación WSDLSniffer 1.0.

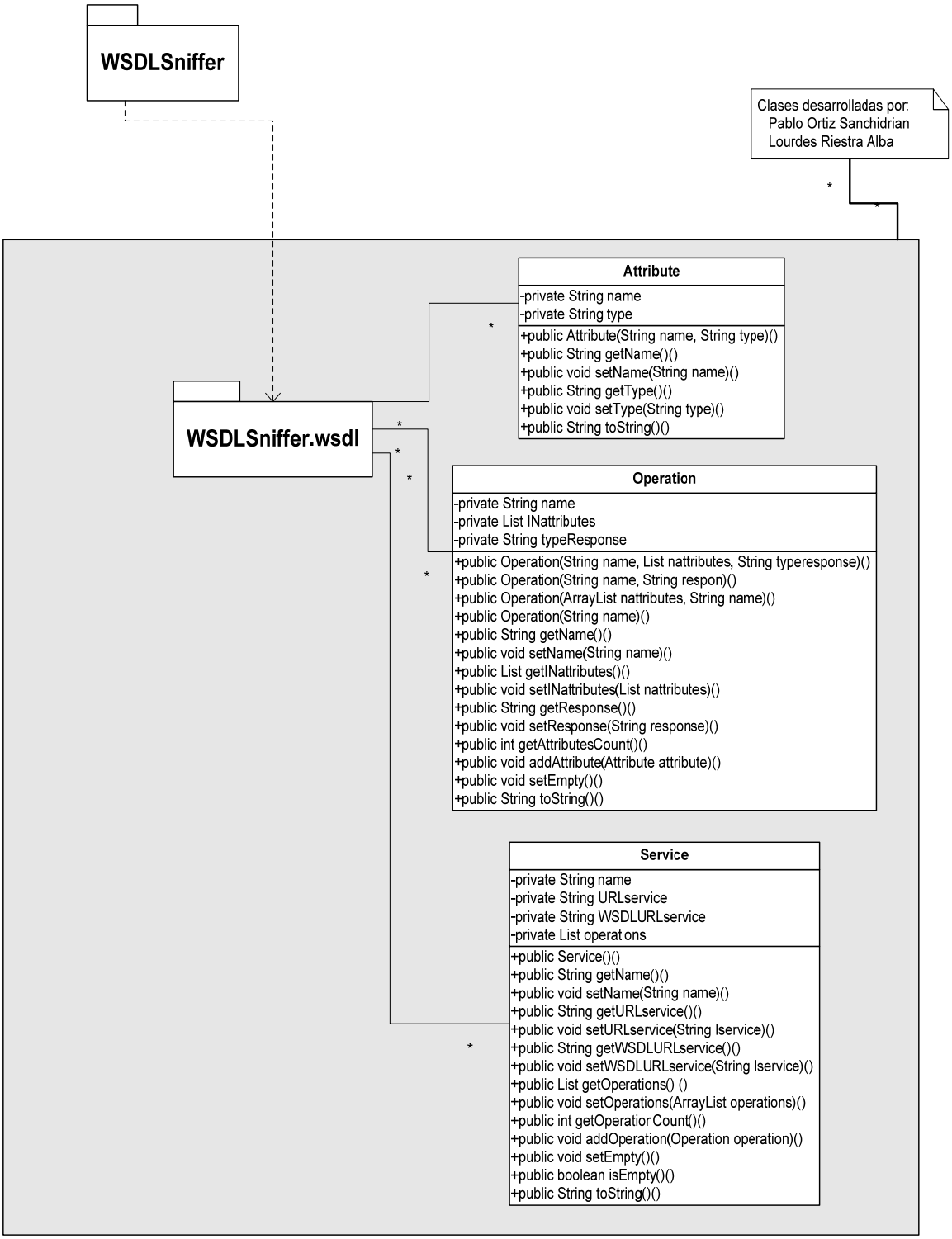

Figura 20 Clases desarrolladas para WSDLSniffer

5. Agradecimientos

Este trabajo ha sido parcialmente subvencionado por el proyecto **CCG07-UPM/TIC-1438 "***Entorno Reconfigurable para la Validación de Sistemas Autónomos*". **RETVAS**: Autonomic Systems Reconfigurable Environment for Validation of Autonomic Systems.

6. Tabla de figuras

7. Bibliografía

1. Bañares, Pedro Alvarez y José Ángel**.** Universidad de Zaragoza. [En línea] http://iaaa.cps.unizar.es/showContent.do?cid=presentacion.ES.

2. Woden Oficial Site. [En línea] http://ws.apache.org/woden/.

3. Web Services. Desarrollo de Web Services con Software Libre. Jennifer Pérez Benedí. [En línea]. http://www.gvpontis.gva.es/fileadmin/conselleria/images/Documentacion/migracionSwAbierto/Anexos/Anexo_C9.pdf

4. Web oficial Axis. [En línea] http://ws.apache.org/axis/java/.

5. W3. [En línea] Junio de 2007. http://www.w3.org/TR/wsdl20-primer/.

8. Anexo 1: Instalación de los plugin necesarios para crear servicios web

Los componentes que se utilizan en el desarrollo de la aplicación de rastreo WSDLSniffer 1.0, se ejecutan bajo el software de libre distribución Eclipse, accesible desde la página Web http://www.eclipse.org/.

Desde http://tomcat.apache.org/, web oficial de apache tomcat[20], se puede descargar la última versión del servicio tomcat. Se instalará automáticamente por defecto en la carpeta C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0. Durante la instalación del servicio se podrá modificar esta ruta e instalarlo donde sea más conveniente en cada caso.

Posteriormente se procederá a instalar Axis, que es el cliente de web service. Desde la página http://ws.apache.org/axis/java/index.html, página oficial de Axis, se puede descargar la última versión del cliente.

Una vez descargada la utilidad y descomprimida, desde #RUTA_INSTALACIÓN#\axis-1_4\webapps, se encuentra la carpeta axis, la cual se deberá copiar dentro del servicio TOMCAT en la carpeta #RUTA_INSTALACIÓN# \ Apache Software Foundation \Tomcat 6.0\webapps.

Una vez hecho esto, es posible ejecutar eclipse. Se han de seguir los siguientes pasos para incluir dentro de eclipse, un nuevo servidor.

En la pestaña del menú principal, Window, se ha de elegir la opción de Preferences:

---

[20] Tomcat: es un servidor web con soporte aplicaciones que se ejecutan dentro de un servidor, y para aplicaciones que permiten generar contenido dinámico para web. Ha de quedar claro que Tomcat no es un servidor de aplicaciones.

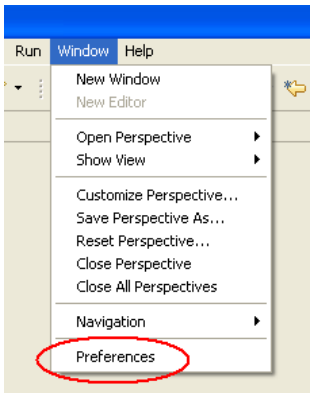

Figura 21 Menú de eclipse Window

La siguiente etapa será instalar el nuevo servidor desde la ventana de acceso siguiente:

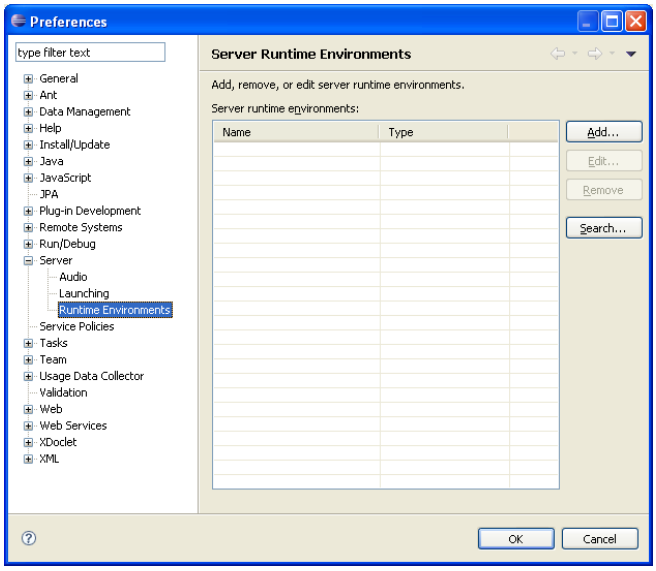

Figura 22 Menú de eclipse Preferences

Se pincha en add y se tendrá la siguiente ventana:

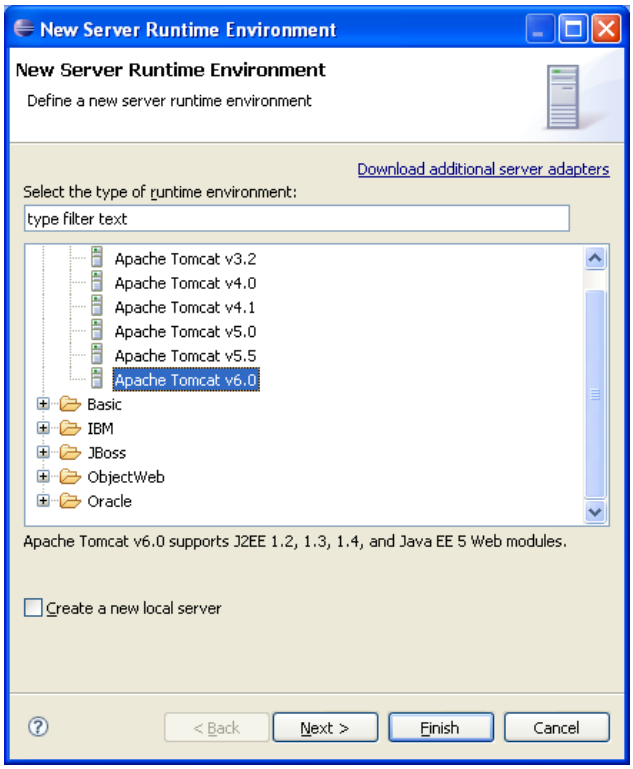

Figura 23 Menú de eclipse New Server Runtime Environment

Para un correcto funcionamiento, se debe elegir la última versión que se tenga instalada del Apache Tomcat y se pinchara en el botón de Finish. En la pantalla descrita anteriormente tendremos la siguiente información:

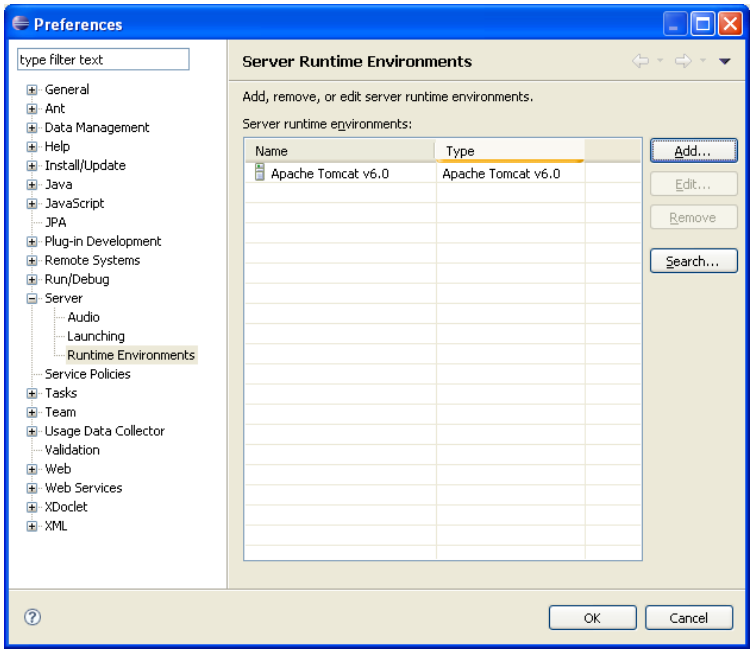

Figura 24 Menú de esclipse preferences

En este momento se finalizará pinchando en el botón de OK.

Ahora se debe añadir el servicio en eclipse. Lo que se ha hecho hasta ahora es el primer paso al ahora de crear un servicio web. Se ha hecho esto ya que se necesitaba, que en eclipse estuviera integrado Tomcat.

Pinchando con el botón derecho, encima de la pestaña Server (debajo de la pantalla) se obtiene el siguiente resultado:

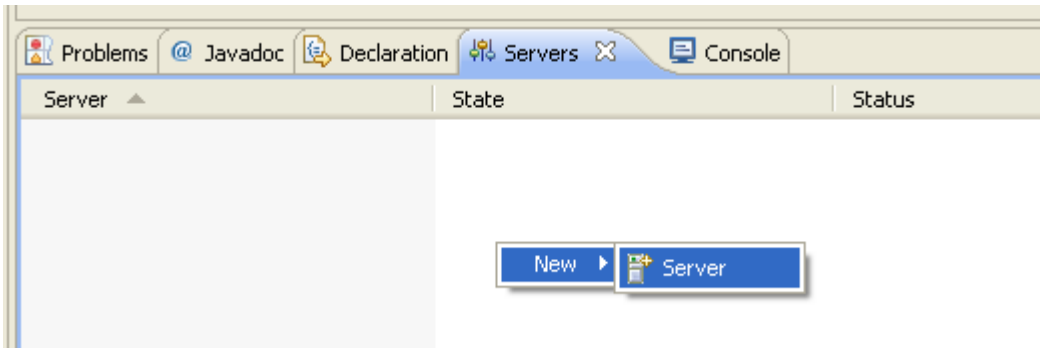

Figura 25 Menú de eclipse Pestaña Servers

Después de realizar esto, aparece la siguiente pantalla:

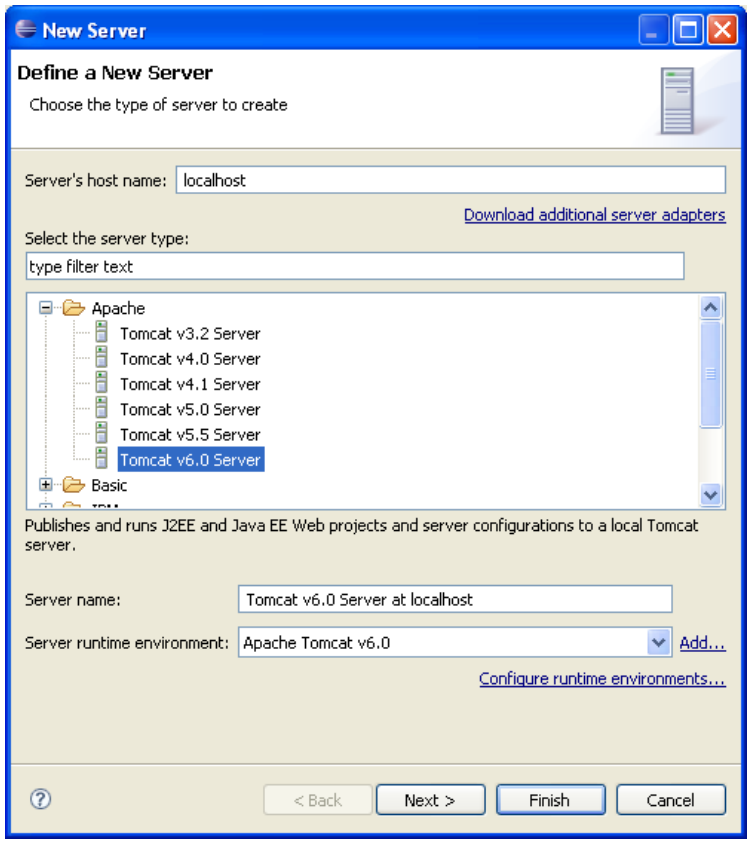

Figura 26 Menú de eclipse New Server

Después de todo este proceso, se pinchará en Finish. Ahora debería aparecer el servidor directamente en el eclipse, de la siguiente manera:

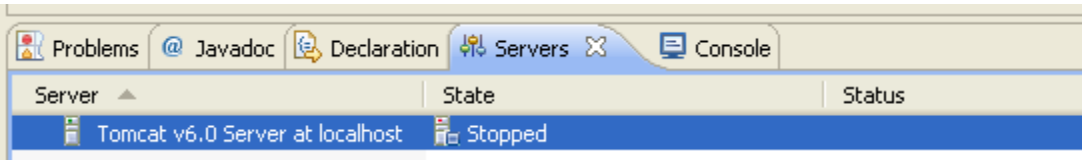

Figura 27 Menú de eclipse Pestaña Servers

Se pinchará con el botón derecho encima del servidor, y arriba de estas pestañas, en la pantalla de edición, se puede ver el siguiente menú:

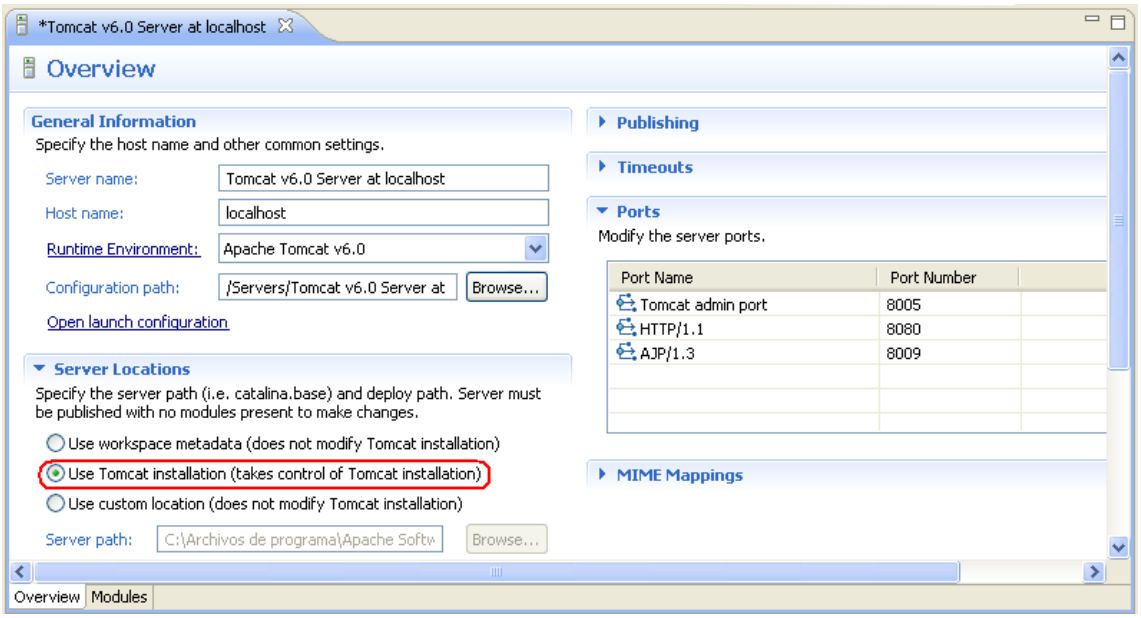

Figura 28 Menú de eclipse Edición Server

Para que funcione correctamente el servicio en la máquina, deberá estar seleccionada la opción señalada arriba en la figura 8.