Aplicacions Distribuïdes

Silvia Llorente
Jaime Delgado

Distributed Multimedia Applications Group
Departament d'Arquitectura de Computadors

(*) Algunas de las transparencias son de otras fuentes

FACULTAT D'INFORMÀTICA DE BARCELONA

Tema 2. Desarrollo de aplicaciones y servicios basados en HTTP

- Introducción
- Aplicaciones web
  - Desarrollo, herramientas y tipos de aplicaciones: jsp, servlet, php, estáticas, HTML5, etc.

Tema 2. Desarrollo de aplicaciones y servicios basados en HTTP

- Herramientas de programación
  - IDE: Netbeans, Eclipse, Intelli J, otros
  - Servidores de aplicaciones

Tema 2. Desarrollo de aplicaciones y servicios basados en HTTP

- Servicios web
  - SOAP: WSDL, formato mensajes SOAP, invocación de operaciones, recogida de resultados, definición de operaciones, etc.
  - REST: Otra manera de implementar servicios web

Aplicaciones web

- Permiten implementar una aplicación que será accedida a través de Internet utilizando un navegador

- Las que desarrollaremos en clase
  - Están basadas en la especificación de Servlets y Java Server Pages definidas originalmente por Sun, ahora propiedad de Oracle

  - Otras: asp Microsoft, scripts cgi, php, Python, perl, etc.

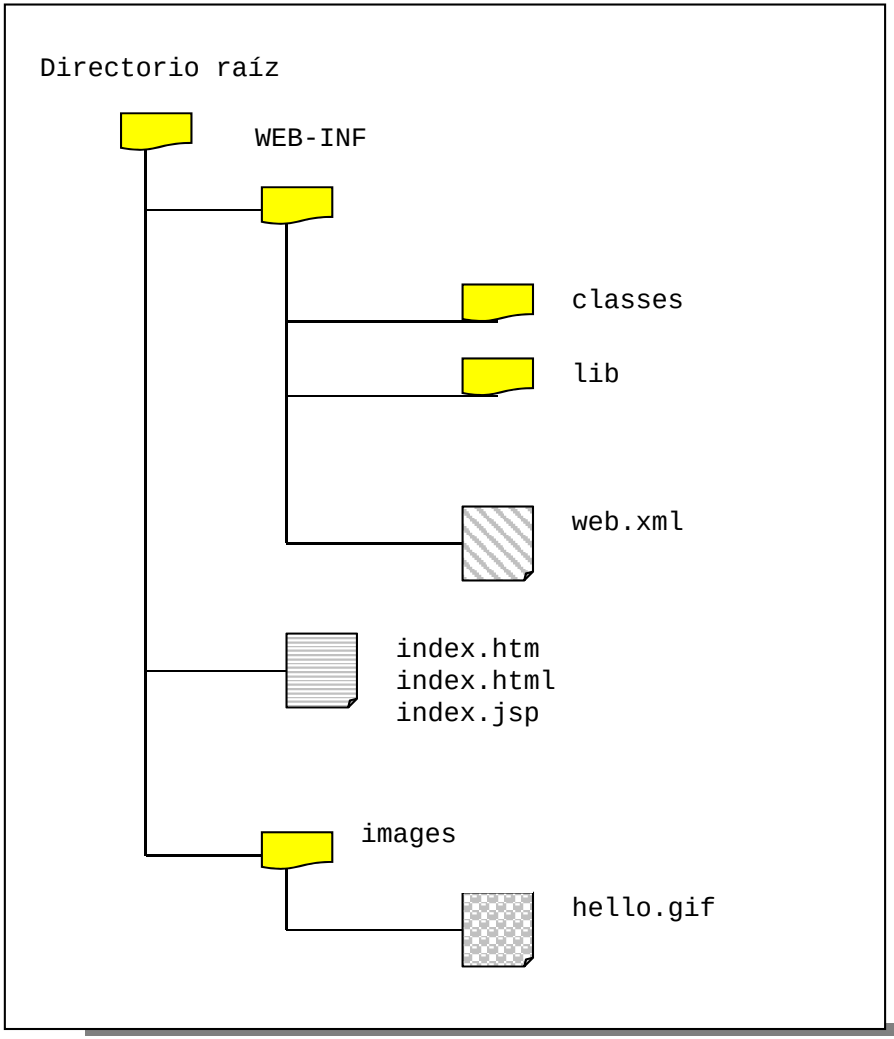
Aplicaciones web estáticas

- Páginas html
- Imágenes
- Recursos estáticos
  - Ficheros de texto
  - Ficheros pdf
  - Otros ...
- Los elementos no varían en el tiempo
  - Aptos para hacer caché (proxy, navegador, etc.)

Aplicaciones web dinámicas

- Utilizamos algún lenguaje de programación que nos genera el código html que llega al navegador
  - Oracle: Java Server Pages, Servlets
  - Microsoft: ASP
  - PHP
  - Ejecutables (.exe, scripts, etc.)
  - Perl
- Podemos utilizar bases de datos, ficheros externos, etc.
- Los elementos se pueden generar de forma dinámica
  - Permiten también elementos estáticos

Aplicaciones web J2EE: Estructura



- Directorio raíz
  - Páginas HTML estáticas
  - Java Server Pages
  - Imágenes u otros ficheros que se quieran hacer públicos (pdf,doc's, etc.)
  - Puede tener subdirectorios

Estructura (continuación)

- Directorio WEB-INF
  - Directorio especial de la aplicación web, que contiene información de configuración
  - Classes
    - Contiene las clases java (.class) que utiliza la aplicación
  - Lib
    - Contiene las librerías java (.jar) que utiliza la aplicación
  - web.xml
    - Fichero de configuración de la aplicación web
    - En las últimas versiones de J2EE se utilizan Java Annotations
    - Ref:
    - https://docs.oracle.com/javase/tutorial/java/annotations/basics.html

Ejemplo fichero web.xml (J2EE 6)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0“ metadata-complete="true">
   <servlet>
        <servlet-name>ChatServlet</servlet-name>
        <servlet-class>chat.ChatServlet</servlet-class>
   </servlet>
   <servlet-mapping>
        <servlet-name>ChatServlet</servlet-name>
        <url-pattern>/jsp/chat/chat</url-pattern>
   </servlet-mapping>
</web-app>
```

XML Schema

Servlets

- Servlet
  - Clase que se carga dinámicamente en el servidor para añadirle funcionalidad
  - Ejecución en una máquina virtual de java
  - Independiente de la plataforma
  - Generan código html de respuesta al cliente o redirigen a otra página (jsp, html, otro servlet, etc.)

Ciclo de vida de un servlet

1) Crear e inicializar el servlet

2) Procesar peticiones de clientes web

3) Destruir el servlet, liberando memoria y recursos (*garbage collection*)

API Servlets

- 2 paquetes java
  - javax.servlet
  - javax.servlet.http

- Servlets genéricos (javax.servlet)
  - No tienen main()
  - Invocación del método service() cuando se recibe una petición

Servlets HTTP

- Servlets HTTP
  - Específicos para HTTP
  - Método *service* ya implementado
    - Invoca al método HTTP correspondiente
    - doXxx (donde Xxx puede ser Get, Post, otros)

Java Server Pages (JSP)

- Mezcla de código html y código java
- Se utilizan sobre todo para construir interfaces (no procesado de comandos)
- Etiquetas especiales para integrar el código html con el código java
- Referencia
  - http://www.exforsys.com/tutorials/jsp/jsp-introduction.html

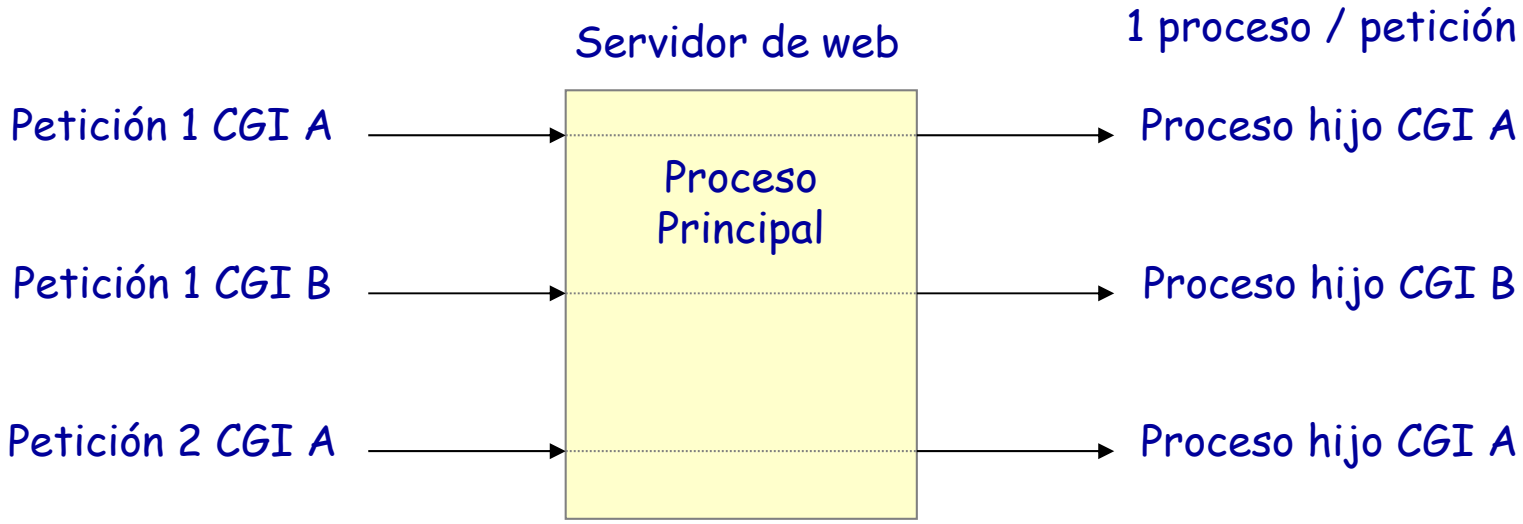
Servlets vs Java Server Pages

- Misma funcionalidad
- Jsp permite instalación y desarrollo más sencillos
- El servidor convierte internamente la JSP en un servlet
  - Genera el código java completo, integrando el código html con el código java (toda la página html se genera con java)
  - Compila el código
  - Invoca ese código cuando se llama a la jsp

Servlets vs Java Server Pages (II)

- La clase compilada se guarda en directorio interno del servidor → Difícil de depurar (aunque los IDE han mejorado)
  - \NetBeansProjects\WebApplication1\build\generated\src\org\apache\jsp

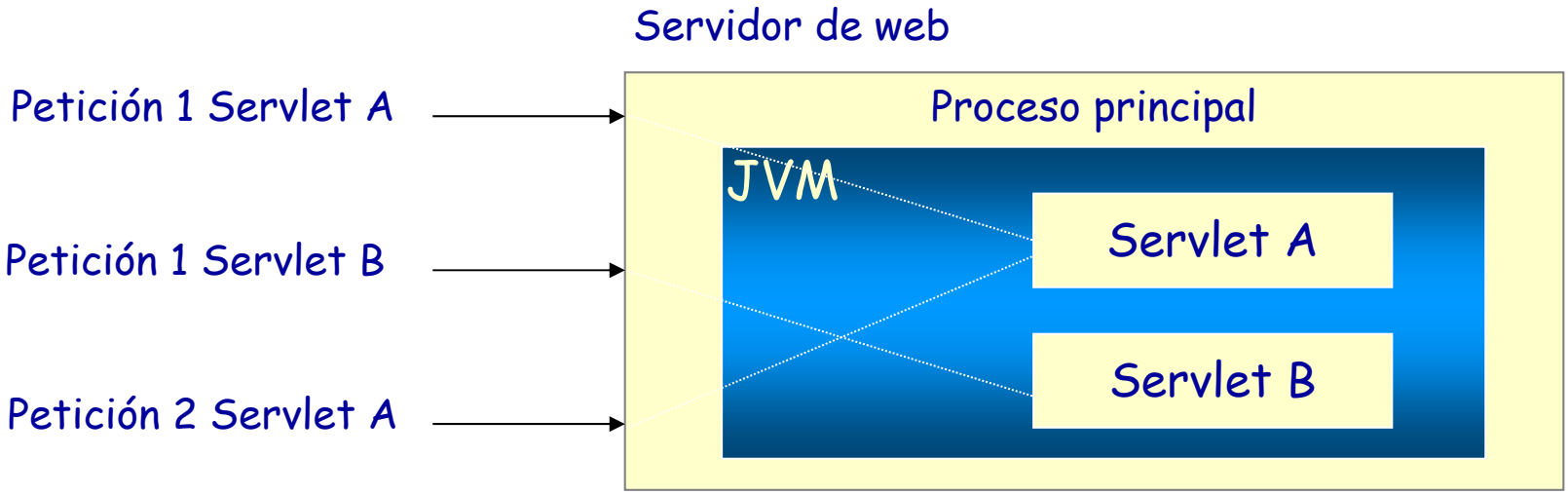
Entorno ejecución: Ejemplo CGI



CGI: Common Gateway Interface

Referencia: http://www.oreilly.com/openbook/cgi/ch01_01.html

Entorno ejecución: Ejemplo servlets



Ciclo de vida de los servlets
Referencia: http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Servidores de Aplicaciones

- Los que veremos en la asignatura están basados en el modelo J2EE definido por Sun (Ahora Oracle)

- Permiten trabajar tanto con aplicaciones web como con servicios web y componentes de negocio (Java Beans, no los veremos en clase)

Servidores de Aplicaciones

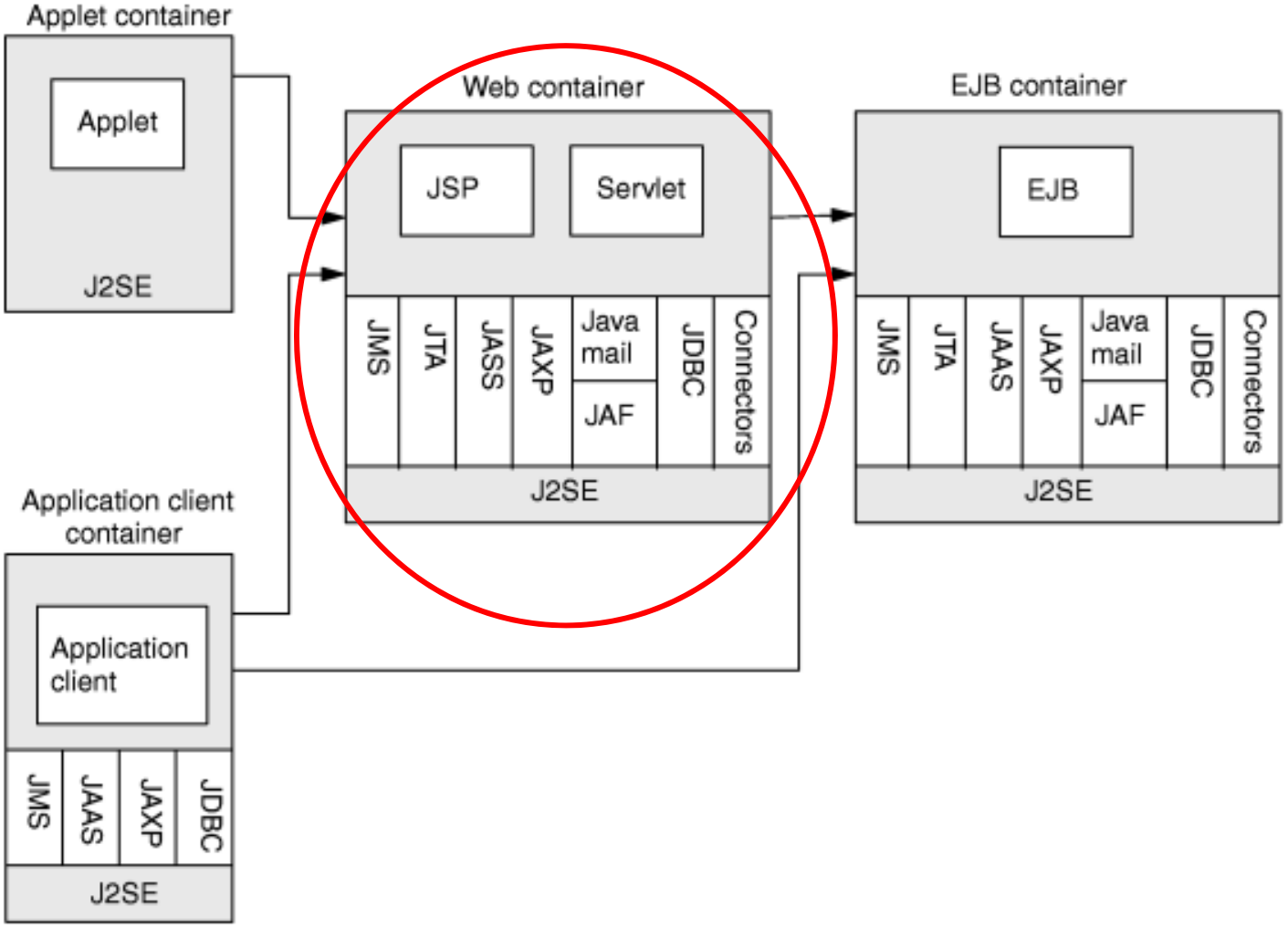
- Ofrecen una serie de funcionalidades que dan soporte a la implementación de servicios y aplicaciones web complejas (utilizados también en la empresa)

- La aplicaciones J2EE son portables entre distintos servidores de aplicaciones, puesto que están desarrollados con lenguaje Java y tienen una estructura completamente definida
  - Ficheros .war, Web Application Archive

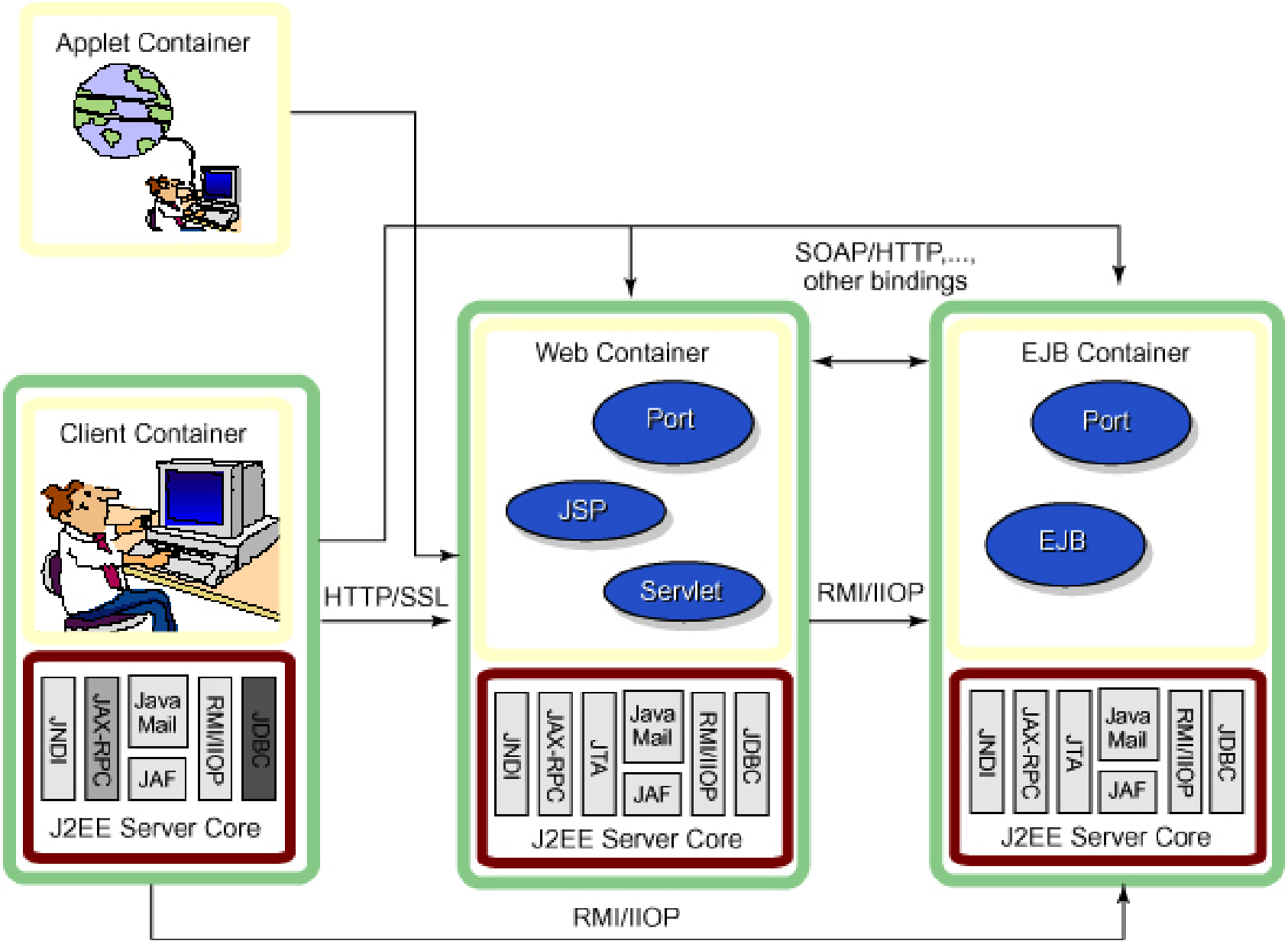
Referencia: Implementaciones compatibles J2EE

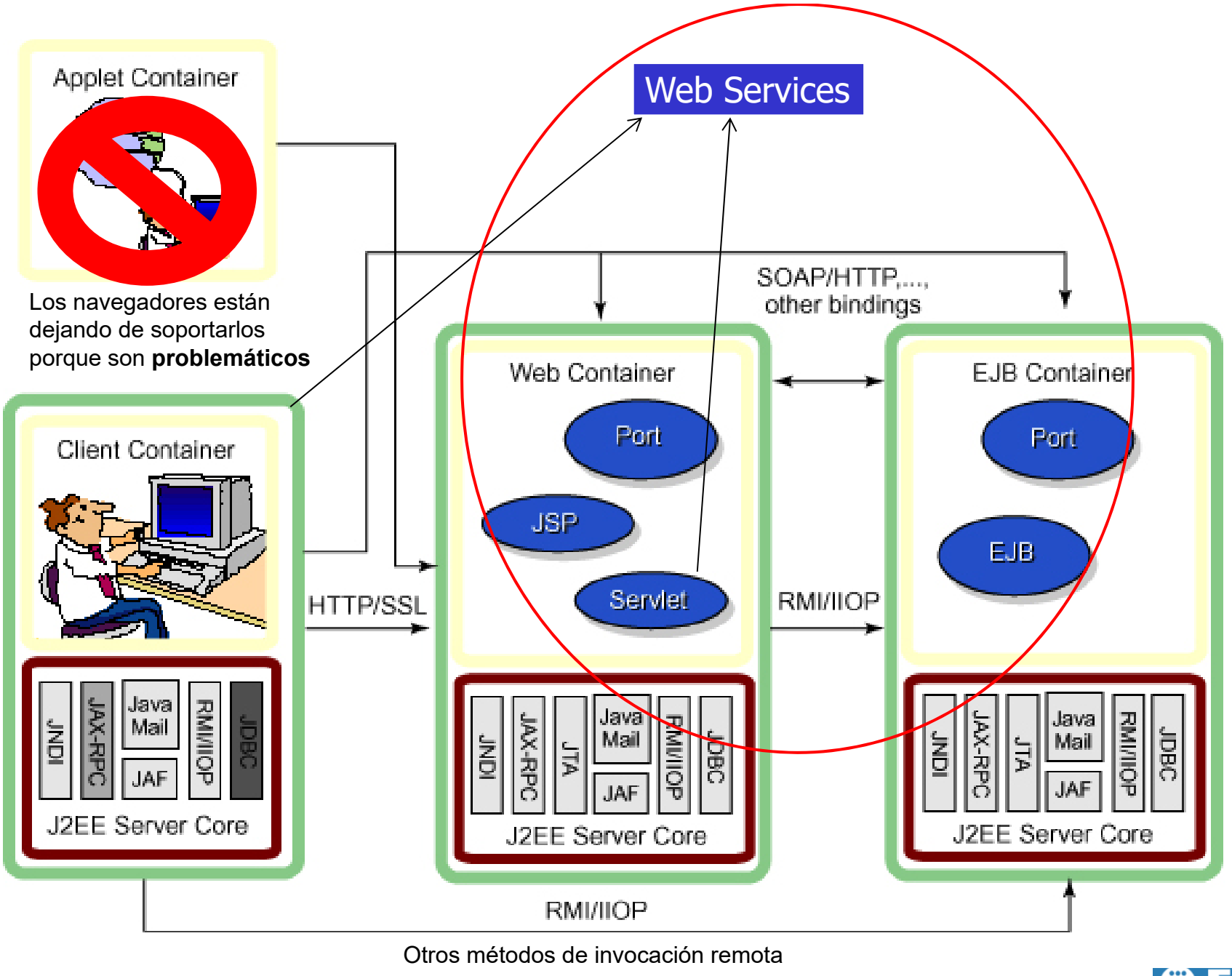
http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html

Arquitectura de un servidor de Aplicaciones J2EE



J2SE = Java 2 Platform, Standard Edition
JAAS = Java Authentication and Authorization Service
JAF = JavaBeans Activation Framework
JAXP = Java API for XML Parsing
JDBC = Java Database Connectivity
JMS = Java Message Service
JSP = Java Server Page
JTA = Java Transaction API




Applet Container
Web Services
Los navegadores están dejando de soportarlos porque son **problemáticos**
SOAP/HTTP,..., other bindings
Web Container
EJB Container
Port
Port
JSP
EJB
Servlet
Client Container
HTTP/SSL
RMI/IIOP
JNDI
JAX-RPC
Java Mail
JAF
RMI/IIOP
JDBC
JTA
J2EE Server Core
RMI/IIOP
Otros métodos de invocación remota

Tomcat

- Servidor de aplicaciones que implementa J2EE
- Sigue la estructura de aplicaciones web definida por Sun
- Referencias
  - Página oficial de Apache Tomcat
    - https://tomcat.apache.org/

Glassfish

- Servidor de aplicaciones J2EE de Sun
- Sigue la estructura de aplicaciones web definida por Sun (Ahora Oracle)
- Referencias
  - Página oficial de Glassfish
    - https://glassfish.java.net/

Netbeans

- Entorno de programación integrado que nos permite desarrollar aplicaciones en distintos lenguajes
- Ofrece plugins de soporte a desarrollo de aplicaciones web, servicios web, conexión con bases de datos, etc.
- Proporcionado por Oracle de forma gratuita
- Página oficial:
  - https://netbeans.org/

Eclipse

- Entorno de programación integrado que nos permite desarrollar aplicaciones en distintos lenguajes

- Ofrece distintas versiones, en función del lenguaje y plugins de soporte a desarrollo de aplicaciones web, servicios web, conexión con bases de datos, etc.

- Gratuito, originalmente de IBM
- Portable, no necesita instalación, solo copiarlo
- Página descarga:
  - https://eclipse.org/downloads/

Intelli J IDEA

- Entorno de programación integrado que nos permite desarrollar aplicaciones en distintos lenguajes
- Apropiado para desarrollo de aplicaciones móviles
- Página descarga:
  - https://www.jetbrains.com/idea/download/

Servicios Web

Existen 2 Enfoques:

- SOAP: Simple Object Access Protocol
- REST: REpresentational State Transfer

Características Web Services

- Independientes de
  - Plataforma
  - Lenguaje de programación
- Funcionan sobre HTTP
- Se puede utilizar sin problema para atravesar firewalls (utiliza el puerto 80 http, 443 https)

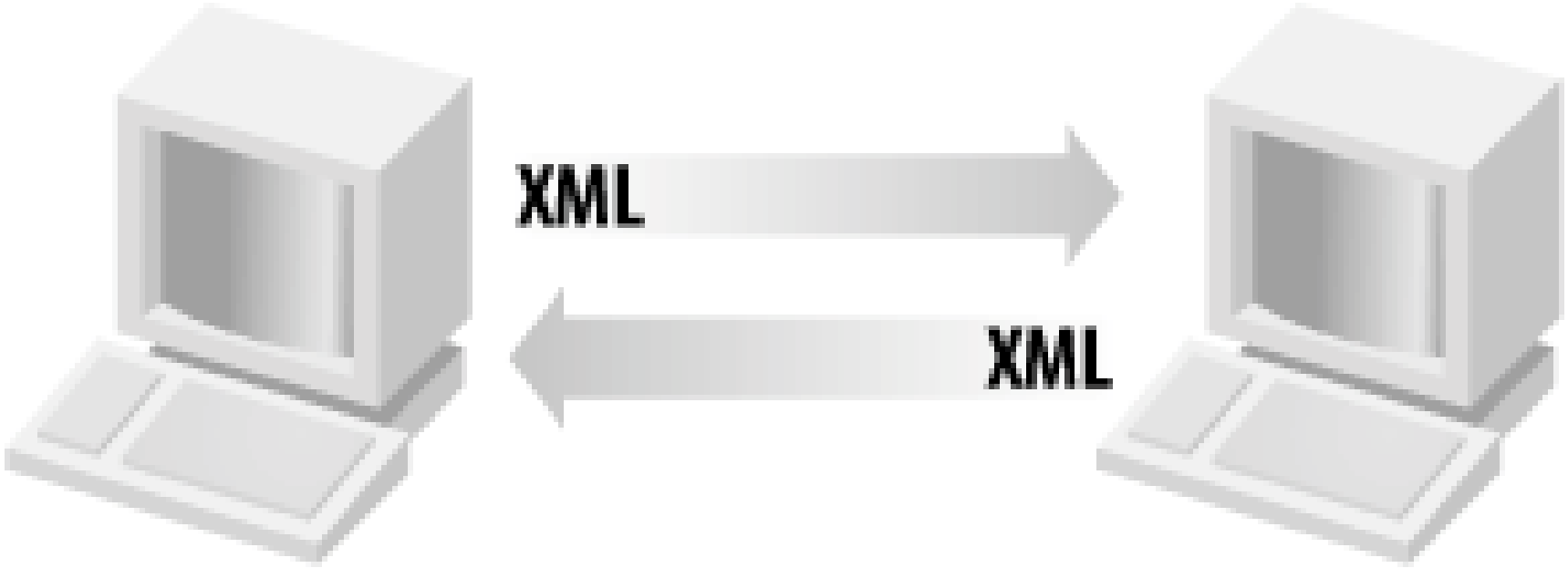
Características Web Services (II)

- No ofrecen características de seguridad: cifrado, gestión de la sesión, calidad de servicio
  - Se pueden añadir utilizando tokens de autenticación de usuarios, Web Service Security, HTTPS (sockets seguros)

Servicios web

- Derivado de RPC (Remote Procedure Call)
- Llamada a métodos remotos vía web utilizando mensajes XML como lenguaje de intercambio de datos
- Objetivo:
  - Llamadas remotas independientes del lenguaje de programación y del sistema operativo donde esté instalado el servicio

Servicio web básico



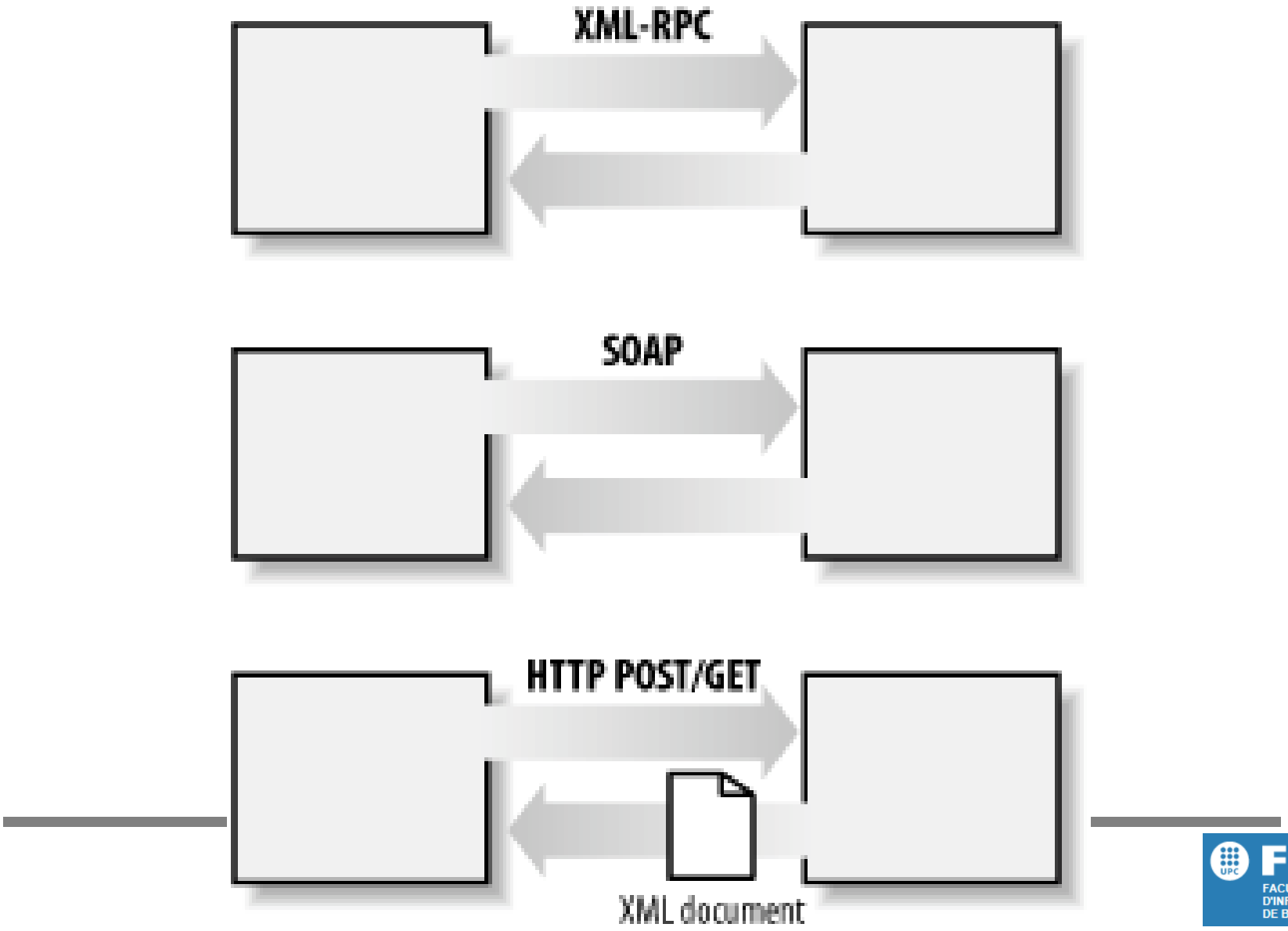
Ordenador A

Lenguaje: *C#*
Sistema Operativo: *Windows 10*

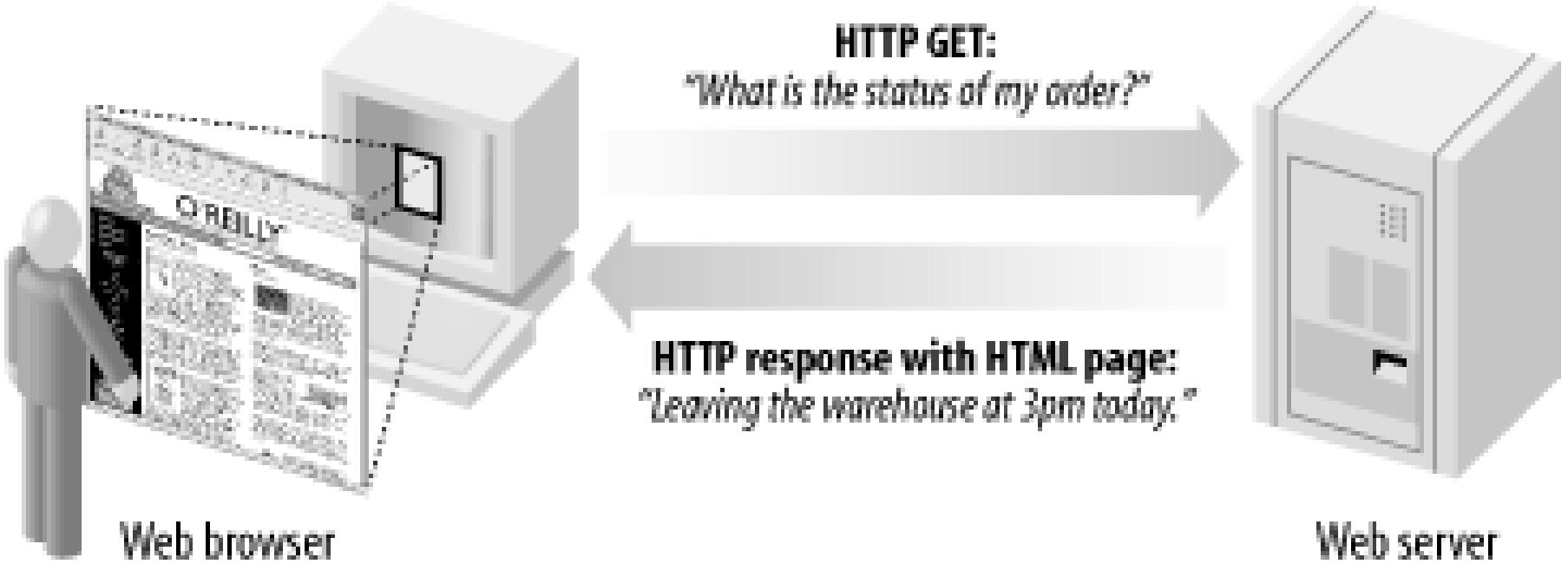
Ordenador B

Lenguaje: *Java*
Sistema Operativo: *Linux*

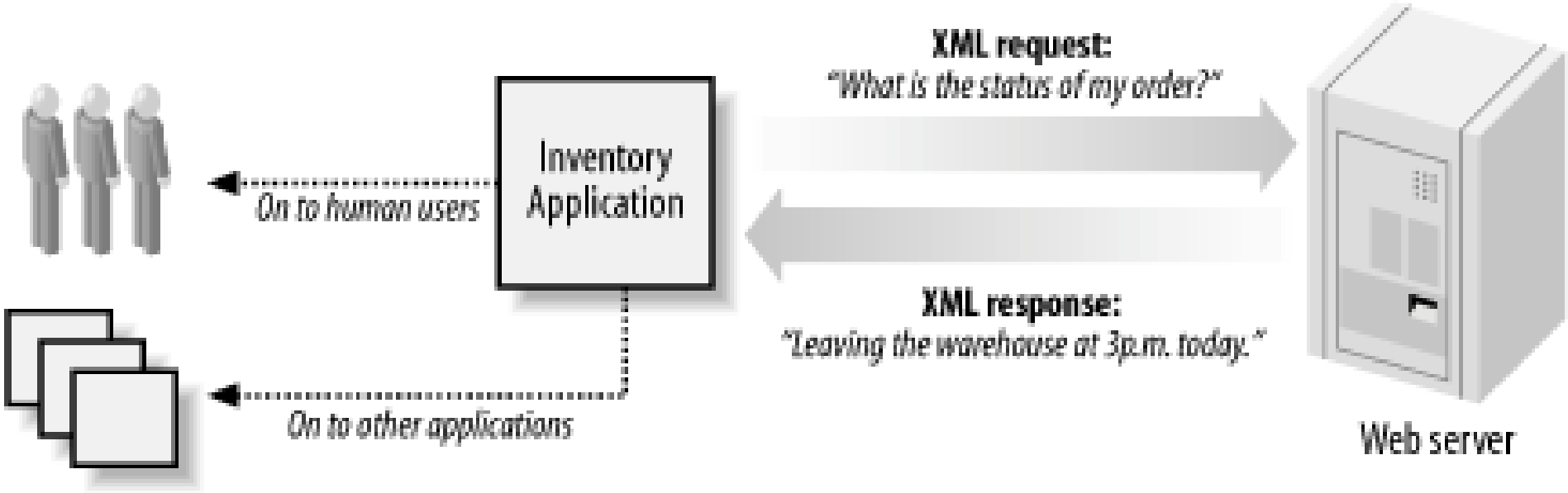
Servicio web con mensajes



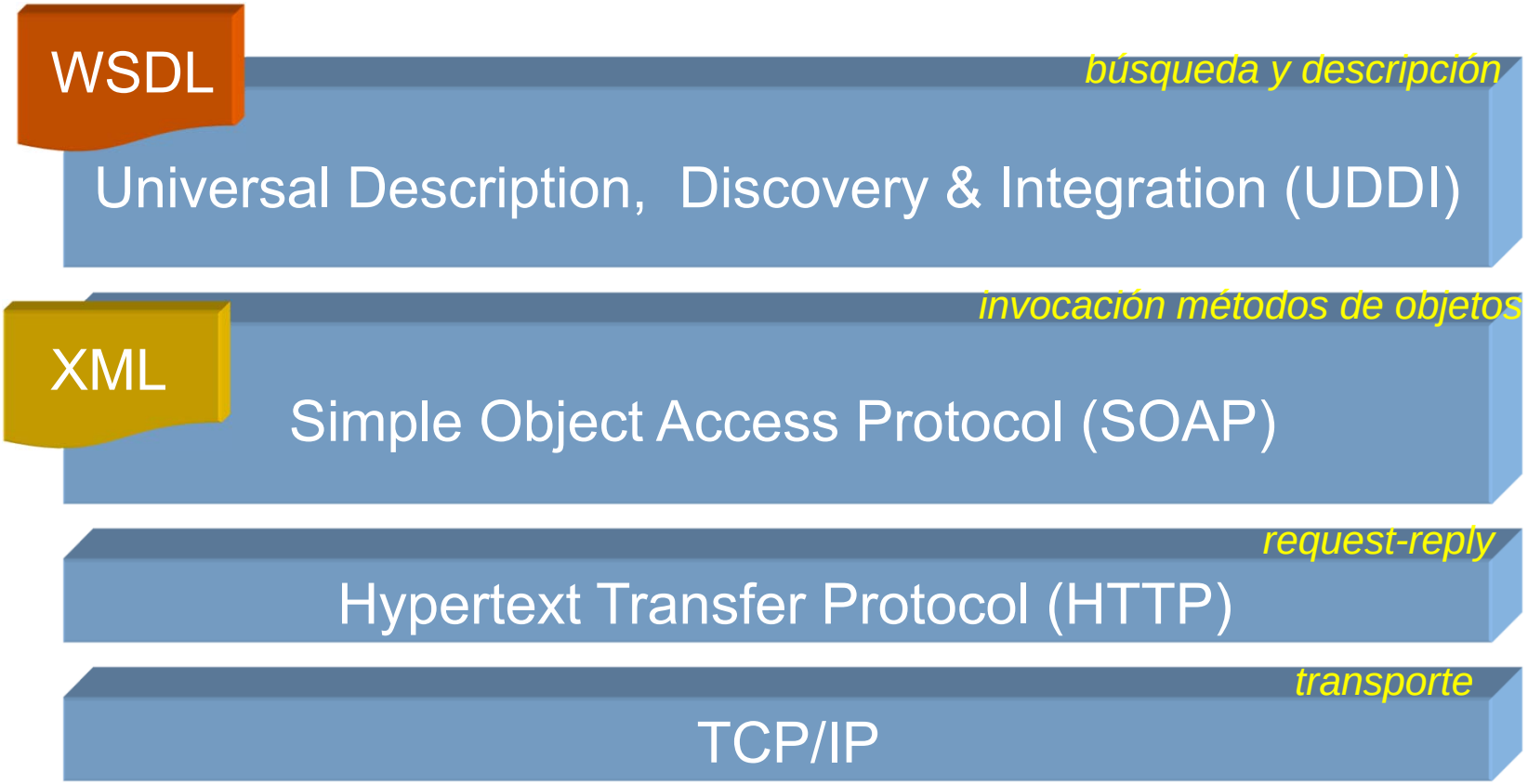
Web para el usuario



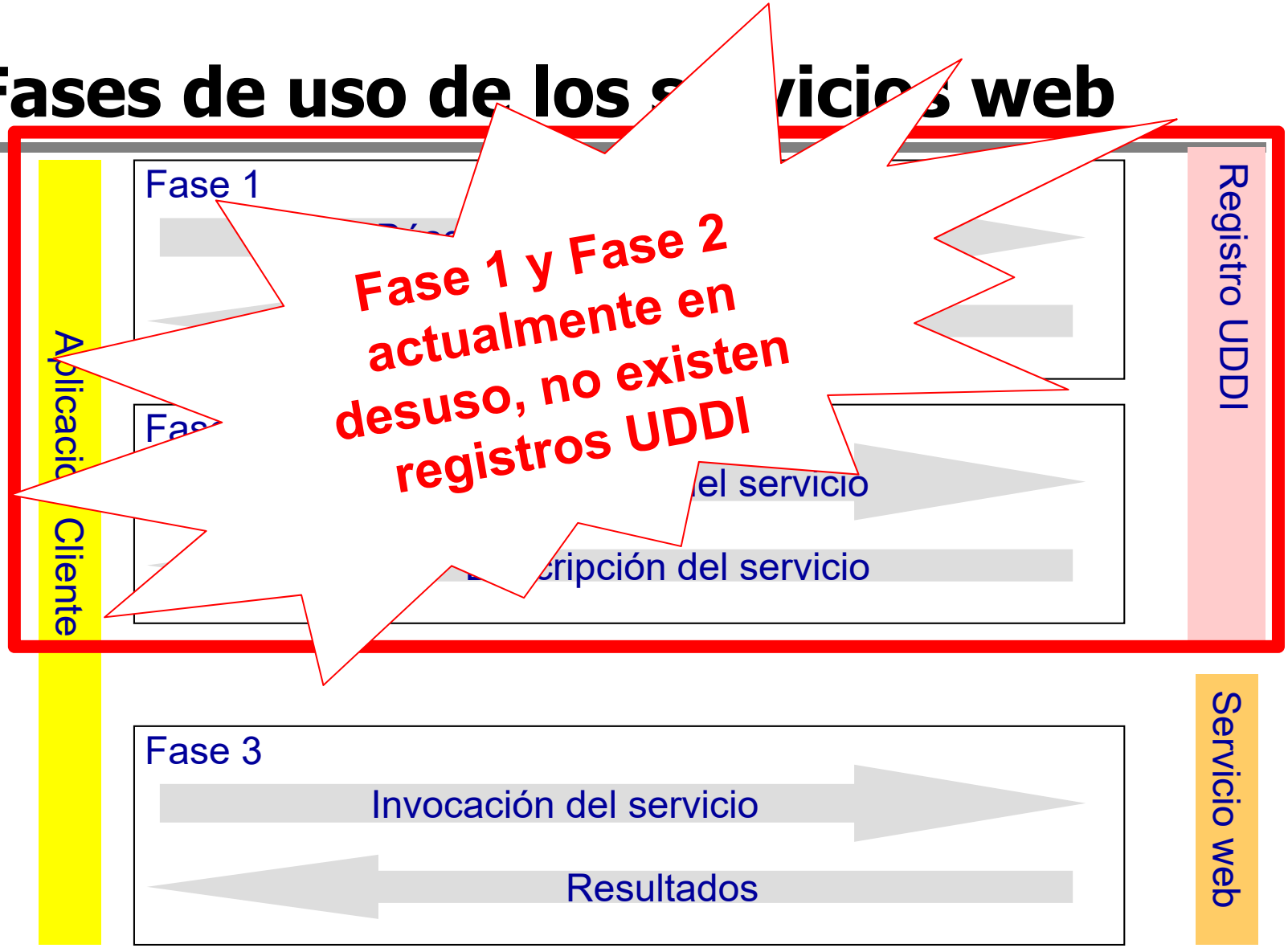
Web automatizada



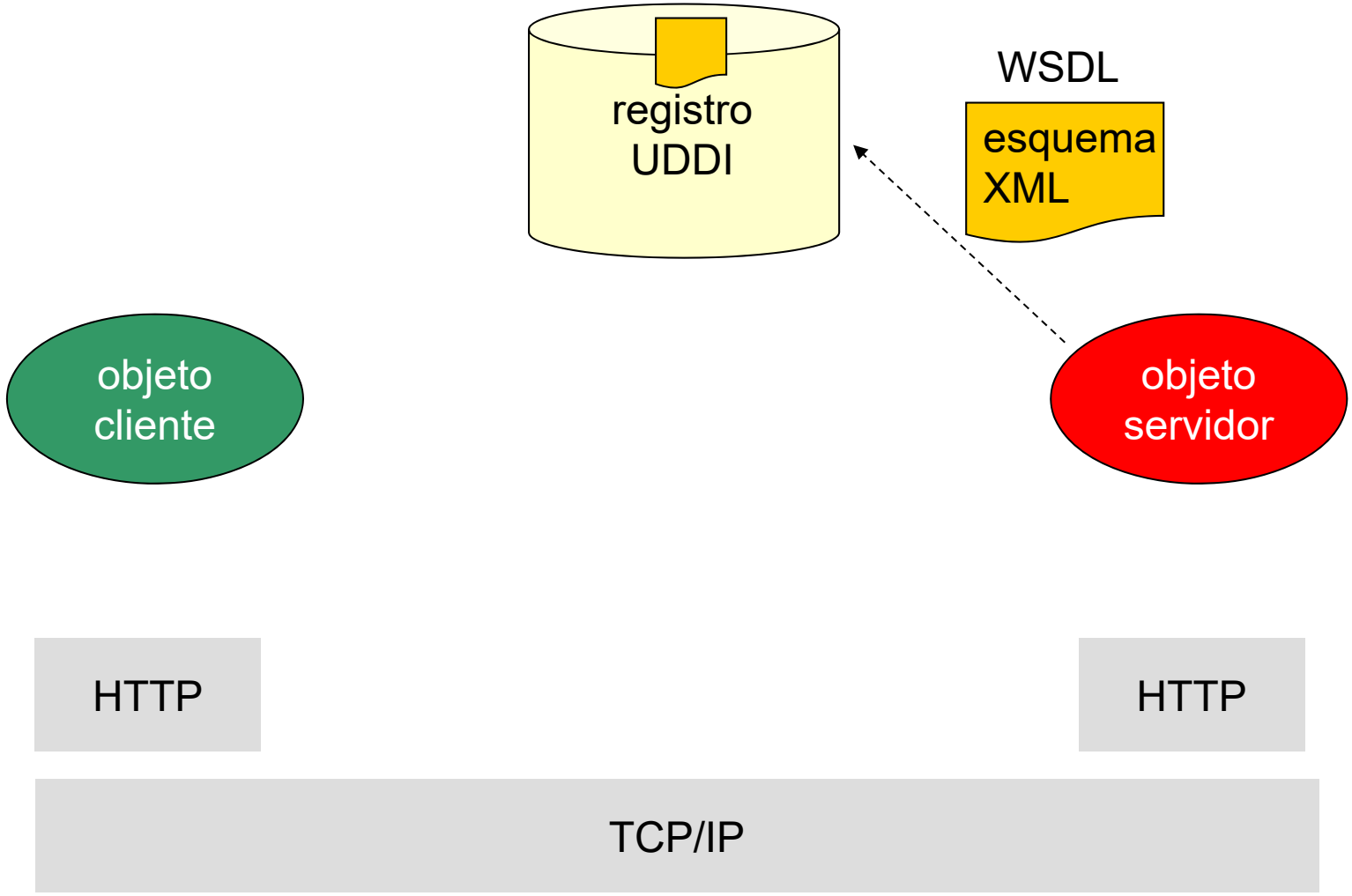
Protocolos y formatos



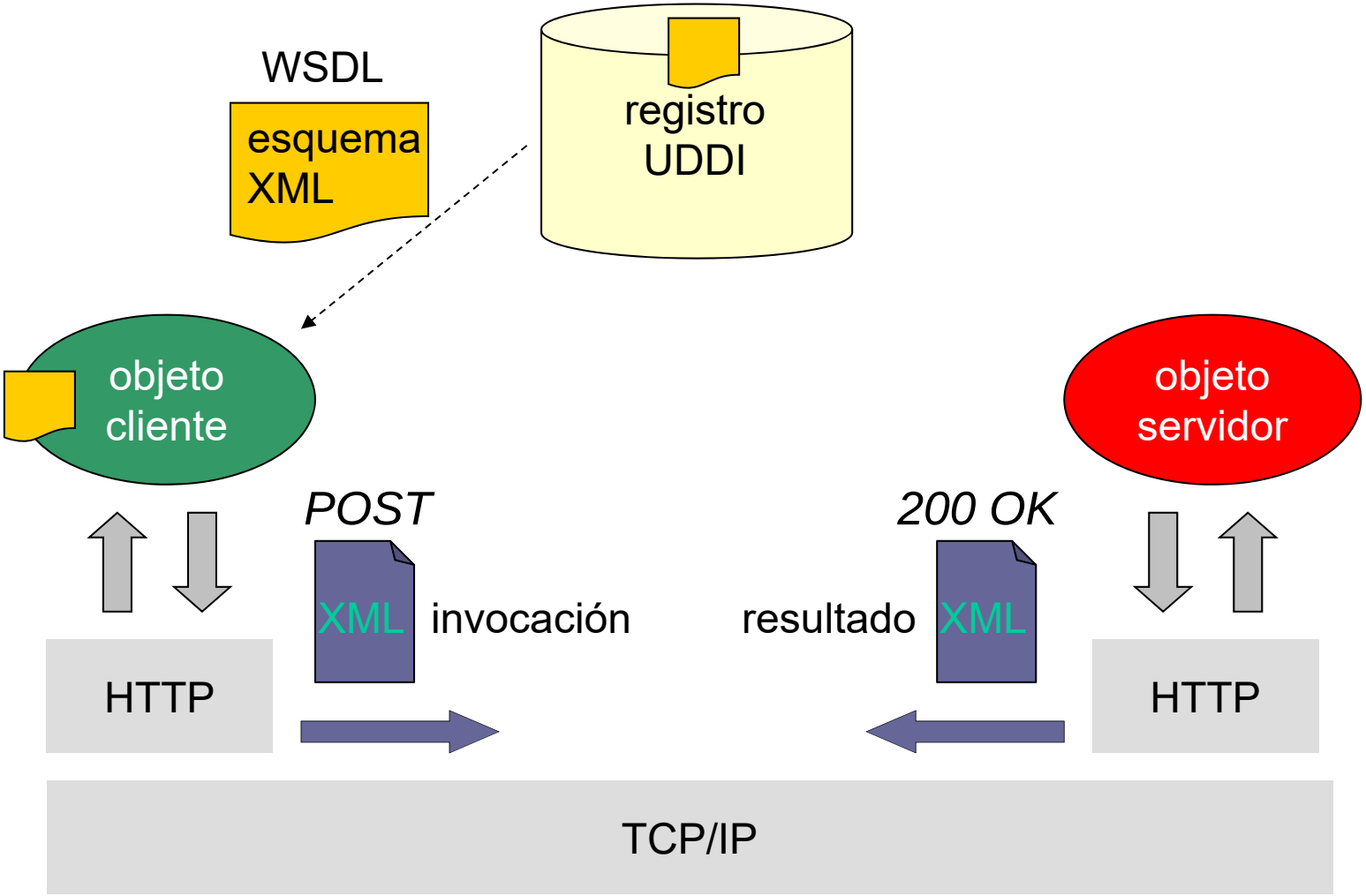
Fases de uso de los servicios web



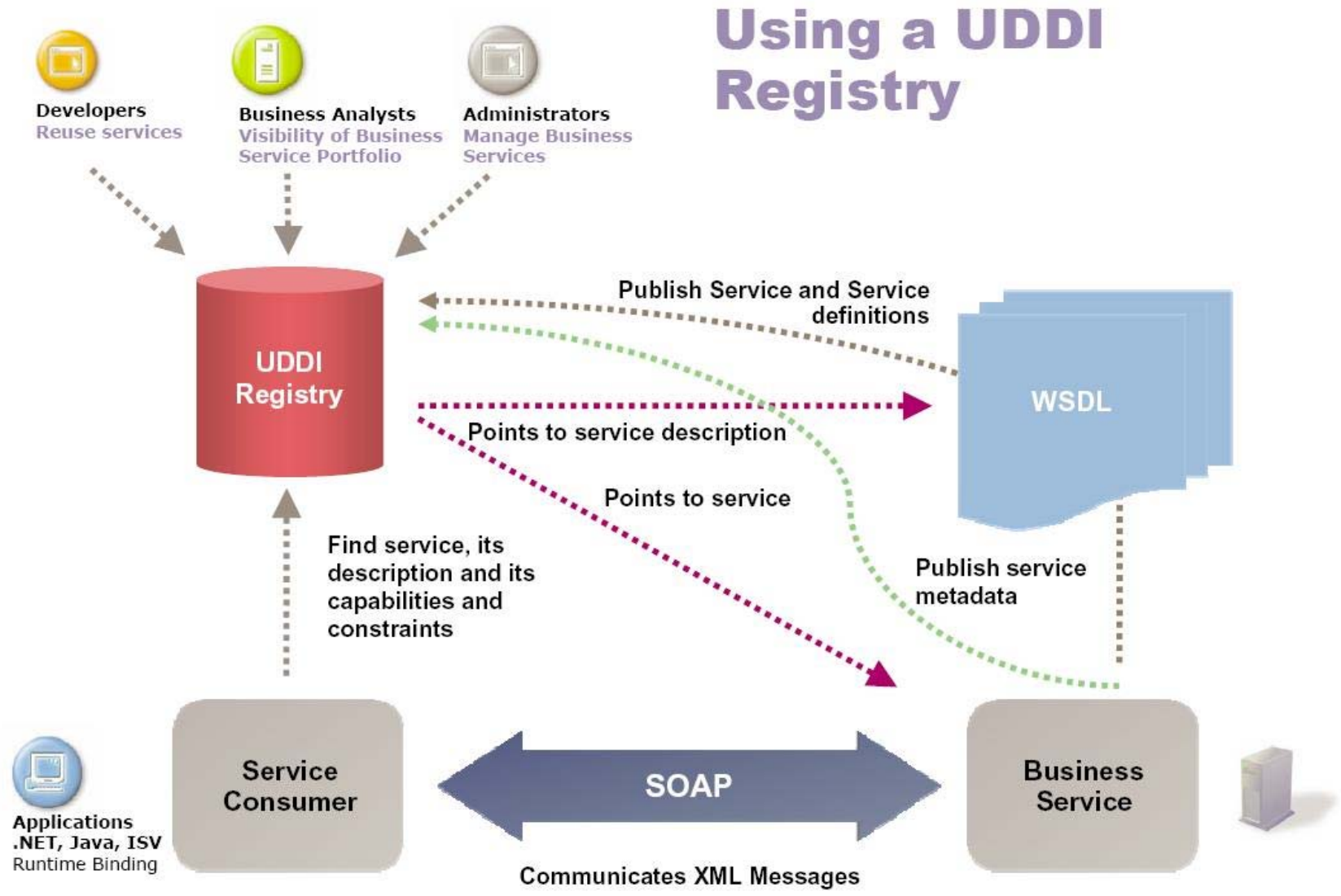
Invocación de métodos remotos con SOAP



Invocación de métodos remotos con SOAP



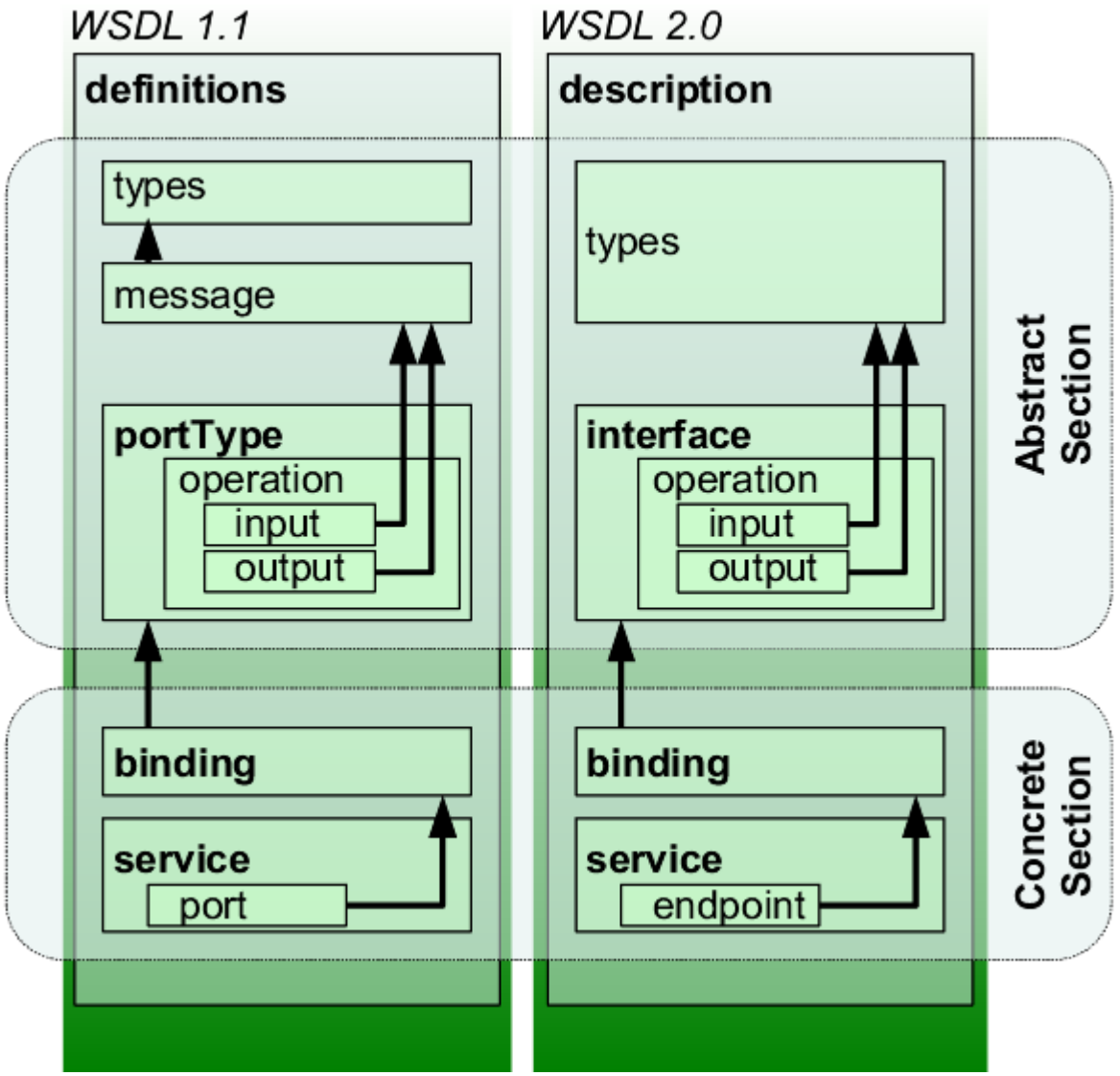
Arquitectura WS SOAP, invocación automática, en desuso



WSDL

- Web Services Description Language (2.0) – SOAP y REST

- Web Services Definition Language (1.1) - SOAP



Ejemplo WSDL 1.1 – SOAP mensajes

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>
```

Parámetro Petición

```
<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
```

Resultado Respuesta

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Definición método remoto

Ejemplo WSDL 1.1 – SOAP binding

```
<message >
</message>

<portType>
</portType>
```

Otras Operaciones

```
<binding type="glossaryTerms" name="binding1">
  <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http”/>
  <operation>
    <soap:operation
          soapAction="http://example.com/getTerm"/>
    <input><soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
 </operation>
</binding>
```

Explicación detallada XML based services:
http://www.w3schools.com/xml/xml_wsdl.asp

WSDL 1.1 – Ejemplo SOAP completo

Elemento raíz del WSDL, definitions

```
<?xml version="1.0"?>

<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

…
```

Espacios de nombres que apuntan a ficheros wsdl y xsd que definen las operaciones y los tipos del servicio web

WSDL 1.1 – Ejemplo SOAP completo

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
   xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

Petición

Definición de tipos. Puede estar en un fichero aparte, de tipo xsd

Respuesta

WSDL 1.1 – Ejemplo SOAP completo

```
<message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
</message>
```

Mensajes petición / respuesta

WSDL 1.1 – Ejemplo SOAP completo

```
<portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
                <input message="tns:GetLastTradePriceInput"/>
                <output message="tns:GetLastTradePriceOutput"/>
        </operation>
</portType>
```

Operación SOAP

WSDL 1.1 – Ejemplo SOAP completo

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
</binding>
```

Binding HTTP
Estilo document/literal

Estilos de WSDL

Definen cómo se envían los datos en la petición y la respuesta SOAP

Ejemplo operación: public void myMethod (int x, float y);

Con RPC/encoded:

```
<soap:envelope>
   <soap:body>
      <myMethod>
         <x xsi:type="xsd:int">5</x>
         <y xsi:type="xsd:float">5.0</y>
      </myMethod>
   </soap:body>
</soap:envelope>
```

No cumple con WS-I (Web Service Interoperability), aunque es un WSDL legal

Con RPC/literal:

```
<soap:envelope>
   <soap:body>
      <myMethod>
         <x>5</x>
         <y>5.0</y>
      </myMethod>
   </soap:body>
</soap:envelope>
```

Cumple con WS-I (Web Service Interoperability)

Referencia WS-I: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-brsp

Estilos de WSDL

Ejemplo operación: public void myMethod (int x, float y);
Document/Encoded existe, pero no se utiliza porque no cumple WS-I

Con Document/literal:

```
<soap:envelope>
   <soap:body>
      <xElement>5</xElement>
      <yElement>5.0</yElement>
   </soap:body>
</soap:envelope>
```

No hay operación ni tipo

Con Document/literal wrapped:

```
<soap:envelope>
   <soap:body>
      <myMethod>
         <x>5</x>
         <y>5.0</y>
      </myMethod>
   </soap:body>
</soap:envelope>
```

Mismo formato RPC/literal, pero el WSDL es distinto. Hay que definir un *wrapper* para los parámetros de la operación

Referencia estilos wsdl:

http://www.ibm.com/developerworks/library/ws-whichwsdl/
http://www.ibm.com/developerworks/library/ws-usagewsdl/

WSDL 1.1 – Ejemplo SOAP completo

```
<service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort“ binding="tns:StockQuoteBinding">
                <soap:address location="http://example.com/stockquote"/>
        </port>
</service>

</definitions>
```

Dirección donde se puede encontrar el servicio

SOAP - Estructura

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

Cabecera SOAP

Cuerpo SOAP

Ejemplo HTTP SOAP Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

Petición http

Petición soap

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

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:StockPrice>
    <m:StockName>IBM</m:StockName>
  </m:StockPrice>
</soap:Body>

</soap:Envelope>
```

Ejemplo HTTP SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

Respuesta http

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

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:StockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:StockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Respuesta soap

Análisis de SOAP

- SOAP permite representar objetos con estructura de grafos, no sólo árboles
- Los mensajes SOAP se pueden enviar a varios destinatarios
- Con SOAP es posible encriptar los mensajes de forma que algunos destinatarios vean la parte cifrada y otros no
- SOAP garantiza la entrega del mensaje: si se pierde la conexión, intentará reenviar el mensaje
- El coste es una **complejidad mayor**

Análisis de SOAP

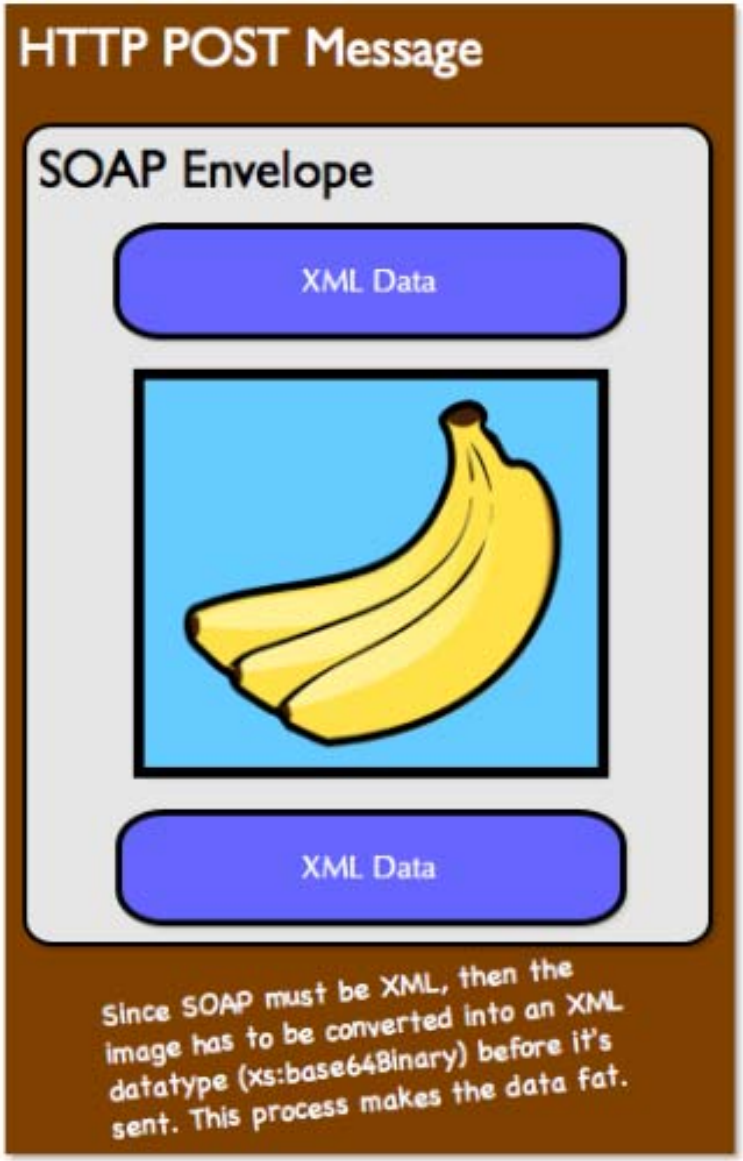
Ventajas

- Permite utilizar distintos *protocolos de transporte*
  Normalmente se utiliza HTTP, pero se pueden utilizar protocolos como SMTP

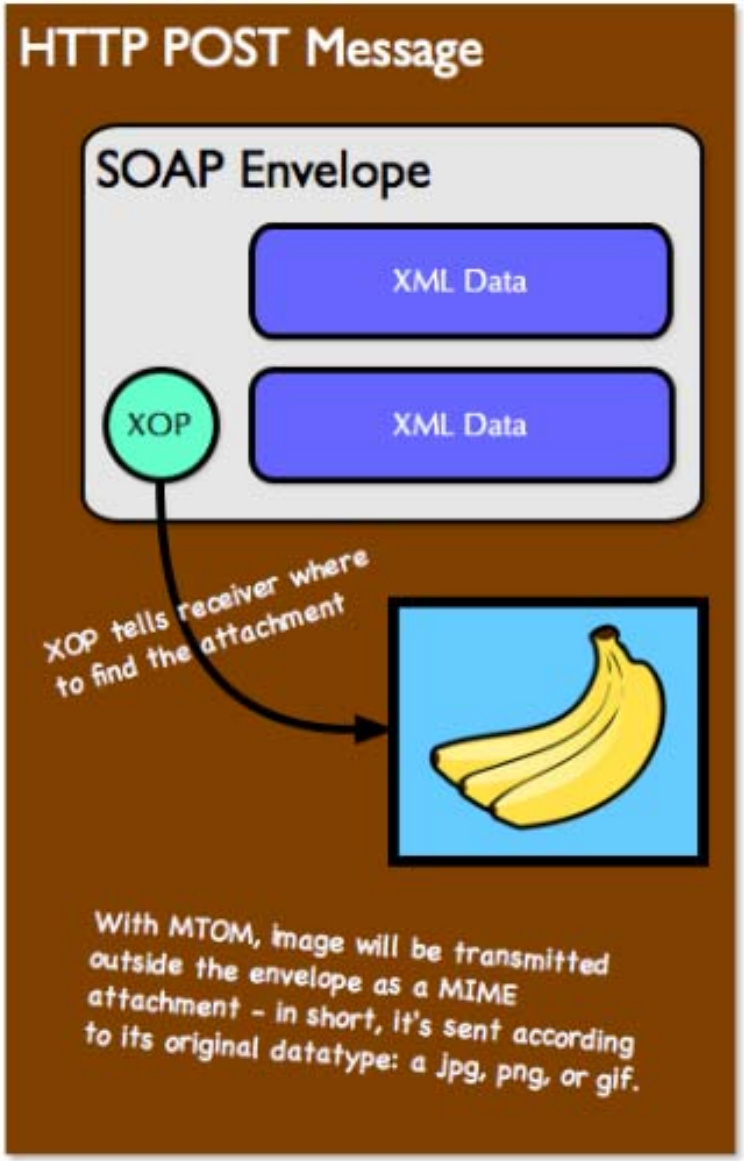
Desventajas

- El *binding* por defecto SOAP/HTTP→ mucha información → lento
- No es problema para mensajes pequeños
- Objetos binarios → *Message Transmission Optimization Mechanism* (MTOM) y *XML-binary Optimized Packaging* (XOP)
  - Ejemplo de funcionamiento

No MTOM



With MTOM-XOP



SOAP + MTOM + XOP

Mensaje SOAP con referencia a la imagen

```
<soap:Body><soap:Envelope>

<tns:data><xop:include href="SomeUniqueID-ThatLeadsToTheImage"/></tns:data>

</soap:Body></soap:Envelope>
```

```
Content-id: "SomeUniqueID"

Content-Type: image/png

image binary data here
```

Envío de la imagen en binario, ocupa mucho menos que en Base64

```
Content-Type: multipart/related; boundary=MIMEBoundary4A7AE55984E7438034;
          type="application/xop+xml"; start="<0.09BC7F4BE2E4D3EF1B@apache.org>";
          start-info="text/xml; charset=utf-8"
```

Cabecera MIME

```
--MIMEBoundary4A7AE55984E7438034
content-type: application/xop+xml; charset=utf-8; type="application/soap+xml;"
content-transfer-encoding: binary
content-id: <0.09BC7F4BE2E4D3EF1B@apache.org>
```

Cabecera SOAP

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="...."....>
   ........
     <xop:Include href="cid:1.A91D6D2E3D7AC4D580@apache.org"
          xmlns:xop="http://www.w3.org/2004/08/xop/include">
     </xop:Include>
   ........
</soapenv:Envelope>
```

SOAP + XOP reference

```
--MIMEBoundary4A7AE55984E7438034
content-type: application/octet-stream
content-transfer-encoding: binary
content-id: <1.A91D6D2E3D7AC4D580@apache.org>
```

Cabecera Contenido binario

```
Binary Data.....
```

Datos binarios

```
--MIMEBoundary4A7AE55984E7438034--
```

Ejemplos WSDL

http://www.w3.org/2001/03/14-annotated-WSDL-examples

Implementaciones WS SOAP

- Java
- PHP
- .NET
- gSOAP
  - C, C++

- Cualquier programa que interprete el XML que recibe como petición y genere el mensaje XML de respuesta, puede ser un servicio web

REST: REpresentational State Transfer

- REST define una arquitectura de cómo implementar aplicaciones distribuidas de forma simple
- Utiliza los métodos HTTP para realizar las operaciones de consulta y modificación
  - No se definen operaciones específicas como en SOAP
- No es un estándar, aunque está basado en estándares

SOAP vs. REST

- SOAP define estructuras complejas para realizar las llamadas, en REST no son necesarias

- Por ejemplo…

Comparando SOAP y REST

- Ejemplo Petición SOAP, enviada con un POST

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
   <soap:body pb="http://www.acme.com/phonebook">
      <pb:GetUserDetails>
         <pb:UserID>12345</pb:UserID>
      </pb:GetUserDetails>
   </soap:Body>
</soap:Envelope>
```

- Ejemplo Petición REST, enviada con GET

```
http://www.acme.com/phonebook/UserDetails/12345
```

REST ejemplo petición

Request:

GET /StockPrice/IBM HTTP/1.1

Host: example.org

Accept: text/xml

Accept-Charset: utf-8

REST ejemplo respuesta

Response:

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

```
<?xml version="1.0"?>
<m:Quote xmlns:m="http://www.example.org/stock">
        <m:StockName>IBM</m:StockName>
        <m:Price>34.5</m:Price>
</m:Quote>
```

Es posible utilizar estructuras complejas en REST, porque no hay limitación en el formato de la respuesta

Peticiones REST complejas

- Petición REST más compleja , enviada también con GET

  `http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe`

- Se utilizan los métodos HTTP para realizar operaciones tipo CRUD (Create/Read/Update/Delete)
  - **Create** – Método HTTP POST
  - **Read** – Método HTTP GET (POST si los parámetros son complejos y tienen que viajar dentro del mensaje de petición / respuesta)

  Además
  - **Update** – Método HTTP POST
  - **Delete** – Método HTTP POST

  - **Otros**: PUT, PATCH, DELETE (para CUD)

WSDL y REST

- WSDL 2.0 permite definir servicios REST
  - ¿Cómo? Veamos un ejemplo

- Vamos definir dos operaciones REST de una librería (http://www.bookstore.com)
  - book list: Devuelve la lista de libros a la venta
    - URL servicio http://www.bookstore.com/books/
  - book details: Devuelve los detalles de un libro concreto
    - URL servicio http://www.bookstore.com/books/ISBN_NUMBER

- Los servicios devuelven información XML

- Referencia

  http://www.ibm.com/developerworks/library/ws-restwsdl/

WSDL 2.0 – Definición servicio REST

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types/>
  <wsdl:interface/>
  <wsdl:binding/>
  <wsdl:service/>
</wsdl:description>
```

Esqueleto WSDL 2.0

description
types
interface
binding
service

```
  <wsdl:service name="BookList" interface="" ??>
    <wsdl:endpoint name="BookListHTTPEndpoint" binding="“ ??
      address="http://www.bookstore.com/books/">
    </wsdl:endpoint>
  </wsdl:service>
</wsdl:description>
```

Definición Service BookList

WSDL 2.0 – Definición servicio REST

description
types
interface
binding
service

```
<wsdl:binding name="BookListHTTPBinding"
   type="http://www.w3.org/ns/wsdl/http"
   interface="">
  <wsdl:operation ref="" whttp:method="GET"/>
</wsdl:binding>
```

Binding

```
<wsdl:service name="BookList" interface="">
  <wsdl:endpoint name="BookListHTTPEndpoint"
     binding="tns:BookListHTTPBinding"
     address="http://www.bookstore.com/books/">
  </wsdl:endpoint>
</wsdl:service>
</wsdl:description>
```

Service

Ya podemos asociar el binding al service

WSDL 2.0 – Definición servicio REST

description
types
interface
binding
service

```
<wsdl:interface name="BookListInterface">
  <wsdl:operation name="getBookList"
     pattern="http://www.w3.org/ns/wsdl/in-out"
     style="http://www.w3.org/ns/wsdl/style/iri"
     wsdlx:safe="true">
    <wsdl:input element="" ??/>
    <wsdl:output element="" ??/>
  </wsdl:operation>
</wsdl:interface>
```

Interface

```
<wsdl:service name="BookList"
interface="tns:BookListInterface">
   <wsdl:endpoint name="BookListHTTPEndpoint"
      binding="tns:BookListHTTPBinding"
      address="http://www.bookstore.com/books/">
   </wsdl:endpoint>
 </wsdl:service>
```

Service

Ya podemos asociar el interface al service

WSDL 2.0 – Definición servicio REST

description
types
interface
binding
service

```
<schema  …>
[…]
<element name="getBookList" type="tns:getBookListType">
            <annotation>
              <documentation>
        The request element for the book list service.
              </documentation>
   </annotation>
  </element>

  <element name="bookList" type="tns:bookListType">
   <annotation>
     <documentation>
       The response element for the book list service.
     </documentation>
   </annotation>
  </element>
```

Petición

Respuesta

Elementos de la petición y la respuesta

WSDL 2.0 – Definición servicio REST

description
types
interface
binding
service

```
<complexType name="getBookListType">
 <sequence>
  <element name="author" type="string" minOccurs="0" maxOccurs="unbounded"/>
  <element name="title" type="string" minOccurs="0" maxOccurs="1"/>
  <element name="publisher" type="string" minOccurs="0" maxOccurs="1"/>
  <element name="subject" type="string" minOccurs="0" maxOccurs="1"/>
  <element name="language" type="string" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

Elementos de la petición, parámetros de búsqueda

```
<complexType name="bookListType">
 <sequence>
  <element name="book" type="tns:bookType" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

Elementos de la respuesta, lista de libros

```
<complexType name="bookType">
 <attribute name="title" type="string"/>
 <attribute name="url" type="anyURI"
    wsdlx:interface="booksvc:BookInterface"
    wsdlx:binding="booksvc:BookHTTPBinding"/>
</complexType>
</schema>
```

Enlace al servicio BookDetails

WSDL 2.0 – Definición servicio REST

description
types
interface
binding
service

```
<wsdl:types>
  <xs:import
namespace="http://www.bookstore.org/booklist/xsd"
      schemaLocation="booklist.xsd"/>
 </wsdl:types>
```

Types

```
<wsdl:interface name="BookListInterface">
  <wsdl:operation name="getBookList"
     pattern="http://www.w3.org/ns/wsdl/in-out"
     style="http://www.w3.org/ns/wsdl/style/iri"
     wsdlx:safe="true">
    <wsdl:input element="msg:getBookList"/>
    <wsdl:output element="msg:bookList"/>
  </wsdl:operation>
</wsdl:interface>
```

Interface

Ya podemos definir los tipos del mensaje

WSDL 2.0 – Definición servicio REST

description
- types
- interface
- binding
- service

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  targetNamespace="http://www.bookstore.org/booklist/wsdl"
  xmlns:tns="http://www.bookstore.org/booklist/wsdl"
  xmlns:whttp="http://www.w3.org/ns/wsdl/http"
  xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msg="http://www.bookstore.org/booklist/xsd">
```

Description

Esquema externo para los tipos

Ejemplo completo http://www.ibm.com/developerworks/library/ws-restwsdl/

Respuestas REST

- En REST, se puede enviar como respuesta cualquier formato que sea procesable por un programa de forma sencilla (XML, CSV, JSON – JavaScript Object Notation). En SOAP esto es no es posible

- No se recomienda utilizar como formato de respuesta páginas HTML, ya que es difícil de procesar por parte de un programa
  - Se puede hacer si el destinatario de la respuesta es un usuario que va a verla con un navegador

JSON vs. XML

```
{"menu": {
            "id": "file",
            "value": "File",
            "popup": {
                        "menuitem": [
                                    {"value": "New", "onclick": "CreateNewDoc()"},
                                    {"value": "Open", "onclick": "OpenDoc()"},
                                    {"value": "Close", "onclick": "CloseDoc()"}
                        ]
            }
  }
 }
```

```
<menu id="file" value="File">
            <popup>
                        <menuitem value="New" onclick="CreateNewDoc()" />
                        <menuitem value="Open" onclick="OpenDoc()" />
                        <menuitem value="Close" onclick="CloseDoc()" />
            </popup>
</menu>
```

Componentes de REST

- Recursos, identificados por URL's
  - Accesibles de forma universal
  - Los recursos son la clave de un diseño REST correcto
  - SOAP: Operaciones remotas
    - ObtenerDescripcionProducto u ObtenerPrecioProducto
  - REST: Peticiones más simples
    - Producto?campo=Descripcion o Producto?campo=Precio
    - IdProducto/Descripcion o IdProducto/Precio

- Red de recursos
  - Recursos no muy complejos
  - Utilizar enlaces para devolver información adicional, como en las páginas web

Guía de diseño de servicios REST

- Utilizar direcciones lógicas, no físicas
  - Con una dirección lógica podemos devolver igualmente un fichero xml o csv
  - Ejemplo: http://www.servidor.com/restaurantes/1/reserva
    - Indica que queremos hacer una reserva en el restaurante 1. Tras esta URL tiene que haber un fichero físico real, pero que será un servlet, una página php, etc., que permita procesarla

- Una petición no debe devolver mucha información, se puede separar la información por bloques
  - Ejemplo: Mostrar los n primeros productos de un catálogo y utilizar el concepto de paginación

Guía de diseño de servicios REST

- El formato de una respuesta debe estar documentado (aunque no tiene porqué estar estructurado) y no cambiar, ¡los clientes no entenderían la respuesta!

- Las peticiones GET no deberían cambiar el estado del recurso
  - Si hay modificaciones se debería utilizar POST u otras operaciones como DELETE

Componentes de REST

- Sistema cliente – servidor
  - Pero un componente puede actuar de cliente y utilizar otro servicio y de servidor respondiendo a peticiones
- No hay estado, cada petición debe contener los datos necesarios para que se pueda servir de forma independiente
- Se puede hacer cache de los recursos, indicando cuando expiran
  - Se utilizan las cabeceras Cache del protocolo HTTP

Algunos ejemplos reales de API's REST

- Openweathermap.org
  - `http://openweathermap.org/api/`
- Probar / inventar api's nuevas
  - `http://jsonplaceholder.typicode.com/`
- API del Racó de la FIB (hay que autenticarse)
  - `https://api.fib.upc.edu/v2/`

API Open Weather Map

REQUEST:

api.openweathermap.org/data/2.5/weather?lat=35&lon=139

RESPONSE:

{"coord":{"lon":139,"lat":35},
"sys":{"country":"JP","sunrise":1369769524,"sunset":1369821049},
"weather":[{"id":804,"main":"clouds","description":"overcast clouds","icon":"04n"}],
"main":{"temp":289.5,"humidity":89,"pressure":1013,"temp_min":287.04,"temp_max":292.04},
"wind":{"speed":7.31,"deg":187.002},
"rain":{"3h":0},
"clouds":{"all":92},
"dt":1369824698,
"id":1851632,
"name":"Shuzenji",
"cod":200}

API Open Weather Map

- **API call:**

api.openweathermap.org/data/2.5/weather?q={city name}

api.openweathermap.org/data/2.5/weather?q={city name},{country cod

- **Parameters:**

q city name and country code divided by comma,
use ISO 3166 country codes

- **Examples of API calls:**

api.openweathermap.org/data/2.5/weather?q=London

api.openweathermap.org/data/2.5/weather?q=London,uk

API Open Weather Map

- **Request:**

api.openweathermap.org/data/2.5/weather?q=London,uk

http://api.openweathermap.org/data/2.5/weather?q=London,uk
&appid=44db6a862fba0b067b1930da0d769e98

- **Response:**

{"coord":{"lon":-0.13,"lat":51.51},"weather":
[{"id":801,"main":"Clouds","description":"few clouds","icon":"02n"}],
"base":"cmc stations","main":{"temp":285.17,"pressure":1018,
"humidity":66,"temp_min":283.85,"temp_max":286.15},
"wind":{"speed":9.8,"deg":230,"gust":17.5},"clouds":{"all":20},
"dt":1453744044,"sys":{"type":1,"id":5091,"message":0.0102,
"country":"GB","sunrise":1453708113,"sunset":1453739867},
"id":2643743,"name":"London","cod":200}

Algunos ejemplos reales de WS SOAP

- Flightxml.flightaware.com – Información de vuelos
  - `http://flightxml.flightaware.com/soap/FlightXML2/wsdl`

- Flickr, imágenes
  - `https://www.flickr.com/services/api/request.soap.html`
  - `https://www.flickr.com/services/api/response.soap.html`

Web services example - SOAP, WSDL

- **FlightXML 2.0**: SOAP & WSDL.
- FlightXML 2.0 WSDL:
  Uses the "Document/Literal wrapped" method for encoding SOAP messages (instead of the older "RPC/Encoded" in FlightXML 1.0 WSDL).

Request

```
<xs:element type="FlightXML2:AircraftTypeRequest" name="AircraftTypeRequest"/>
<xs:complexType name="AircraftTypeRequest">
   <xs:sequence>
            <xs:element type="xs:string" name="type" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
</xs:complexType>
```

Response

```
<xs:element type="FlightXML2:AircraftTypeResults" name="AircraftTypeResults"/>
<xs:complexType name="AircraftTypeResults">
   <xs:sequence>
            <xs:element type="FlightXML2:AircraftTypeStruct" name="AircraftTypeResult" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
</xs:complexType>
<xs:element type="FlightXML2:AircraftTypeStruct" name="AircraftTypeStruct"/>
<xs:complexType name="AircraftTypeStruct">
            <xs:sequence>
                     <xs:element type="xs:string" name="manufacturer" minOccurs="1" maxOccurs="1"/>
                     <xs:element type="xs:string" name="type" minOccurs="1" maxOccurs="1"/>
                     <xs:element type="xs:string" name="description" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
</xs:complexType>
```

Web services example - SOAP, WSDL

```
...
<wsdl:message name="AircraftTypeIn">
        <wsdl:part name="parameters" element="FlightXML2:AircraftTypeRequest"/>
</wsdl:message>
<wsdl:message name="AircraftTypeOut">
        <wsdl:part name="parameters" element="FlightXML2:AircraftTypeResults"/>
</wsdl:message>
```

Messages
Wrapped style

```
...
<wsdl:portType name="FlightXML2Soap">
        <wsdl:operation name="AircraftType">
                <wsdl:input message="FlightXML2:AircraftTypeIn"/>
                <wsdl:output message="FlightXML2:AircraftTypeOut"/>
        </wsdl:operation>
...
</wsdl:portType>
```

Operation
AircraftType

```
<wsdl:binding type="FlightXML2:FlightXML2Soap" name="FlightXML2Soap">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="AircraftType">
   <soap:operation style="document" soapAction="FlightXML2:AircraftType"/>
        <wsdl:input>
           <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
           <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
```

Binding
AircraftType

Web services example - SOAP, WSDL

REST / JSON:

- FlightXML 2.0 can also be accessed using a light-weight REST inspired protocol.
- Returns its responses encoded in "JavaScript Object Notation" (JSON) format.

Access to methods: perform either a GET or POST request to

http://flightxml.flightaware.com/json/FlightXML2/METHODNAME

using standard CGI-style representation of the arguments.
Requests must supply username and API Key as a "basic" Authorization HTTP header.

Example: Request the current weather at JFK airport in New York:
http://flightxml.flightaware.com/json/FlightXML2/
MetarEx?airport=KJFK&startTime=0&howMany=1&offset=0

Web services example - SOAP

- SOAP Server Endpoint URL: https://api.flickr.com/services/soap/
- To request *flickr.test.echo* service, send a SOAP envelope:

```
<s:Envelope
        xmlns:s="http://www.w3.org/2003/05/soap-envelope"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <s:Body>
                <x:FlickrRequest xmlns:x="urn:flickr">
                        <method>flickr.test.echo</method>
                        <name>value</name>
                </x:FlickrRequest>
        </s:Body>
</s:Envelope>
```

Web services example - SOAP

- SOAP answer to simple echo service:

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope
        xmlns:s="http://www.w3.org/2003/05/soap-envelope"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <s:Body>
                <x:FlickrResponse xmlns:x="urn:flickr">
                        [escaped-xml-payload]
                </x:FlickrResponse>
        </s:Body>
</s:Envelope>
```

Web services example - SOAP

- SOAP answer when error:

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Body>
        <s:Fault>
            <faultcode>flickr.error.[error-code]</faultcode>
            <faultstring>[error-message]</faultstring>
            <faultactor>
             http://www.flickr.com/services/soap/
          </faultactor>
          <details>
              Please see http://www.flickr.com/services/docs/ for more details
          </details>
        </s:Fault>
    </s:Body>
</s:Envelope>
```

Web services example - SOAP

- SOAP alternative answer to simple echo service:

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
        <s:Body>
                <FlickrResponse xmlns="/ns/api#">
                        [xml-payload]
                </FlickrResponse>
        </s:Body>
</s:Envelope>
```

Bibliografía

- Tutoriales diversas tecnologías J2EE (antiguos)
  - http://courses.coreservlets.com/Course-Materials/
- Tutorial servlets
  - http://www.tutorialspoint.com/servlets/servlets_tutorial.pdf
- Tutorial jsp
  - http://www.exforsys.com/tutorials/jsp/jsp-introduction.html
- Ejemplo Servicios Web
  - http://www.w3schools.com/xml/xml_services.asp
- Ejemplo WSDL servicio REST
  - http://www.ibm.com/developerworks/library/ws-restwsdl/
- Referencia SOAP - MTOM
  - http://www.redbooks.ibm.com/redpapers/pdfs/redp4884.pdf

Bibliografía

- Libros on-line sobre Servlets y Java Server Pages

Core Servlets and JavaServer Pages™: Volume 1: Core Technologies, 2nd Edition
Autores: Marty Hall, Larry Brown
http://pdf.coreservlets.com/

More Servlets and Java Server Pages™
Autor: Marty Hall
http://pdf.moreservlets.com/

Aplicacions Distribuïdes

Silvia Llorente
Jaime Delgado

Distributed Multimedia Applications Group
Departament d'Arquitectura de Computadors

(*) Algunas de las transparencias son de otras fuentes

FACULTAT D'INFORMÀTICA DE BARCELONA