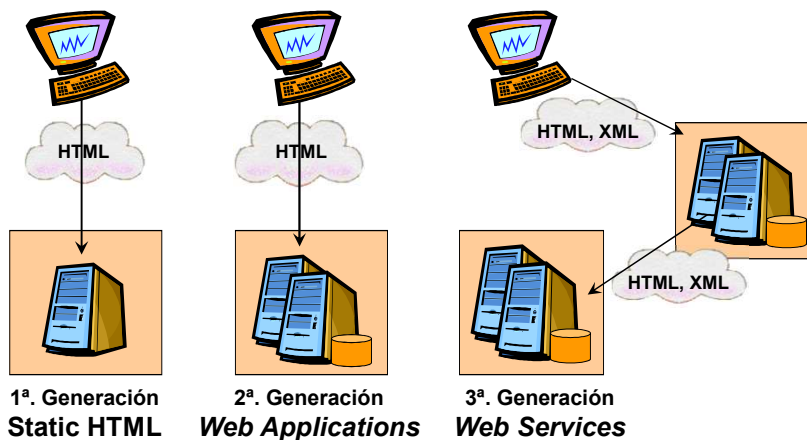


# Servicios Web y Mashups

Dr. José Luis Zechinelli Martini  
[jose Luis.zechinelli@udlap.mx](mailto:jose Luis.zechinelli@udlap.mx)  
LIS – 3061

Gracias a la Dra. Genoveva Vargas Solar, CNRS-LIG, Francia

## Evolución de la Web



## Necesidades de las aplicaciones

- Acceso programable a los servicios:
  - Bolsa de valores
  - Búsquedas en directorios-catálogos
  - Autorización de tarjetas de crédito
  - Autenticación de clientes
  - Registro de nuevos clientes
  - ...
- Interfaz de acceso por funciones:
  - Largos y restrictivos URL
  - Poder descubrir dinámicamente los servicios
  - Parámetros estructurados y diversificados

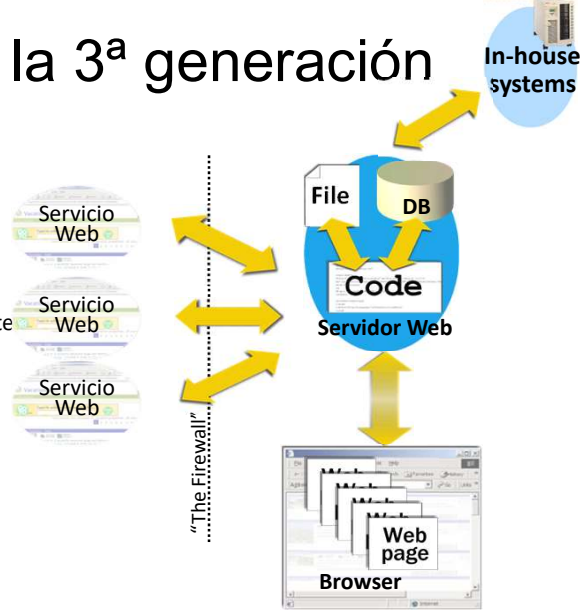
3

## La Web de la 3ª generación

### En el futuro:

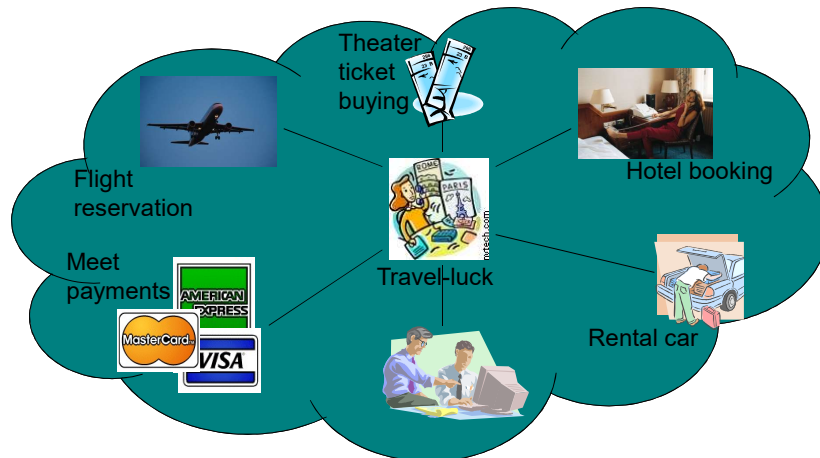
Un sitio Web es un componente que provee servicios en XML

- Estructura / semántica
- Fusión posible



"Páginas dinámicas" 4

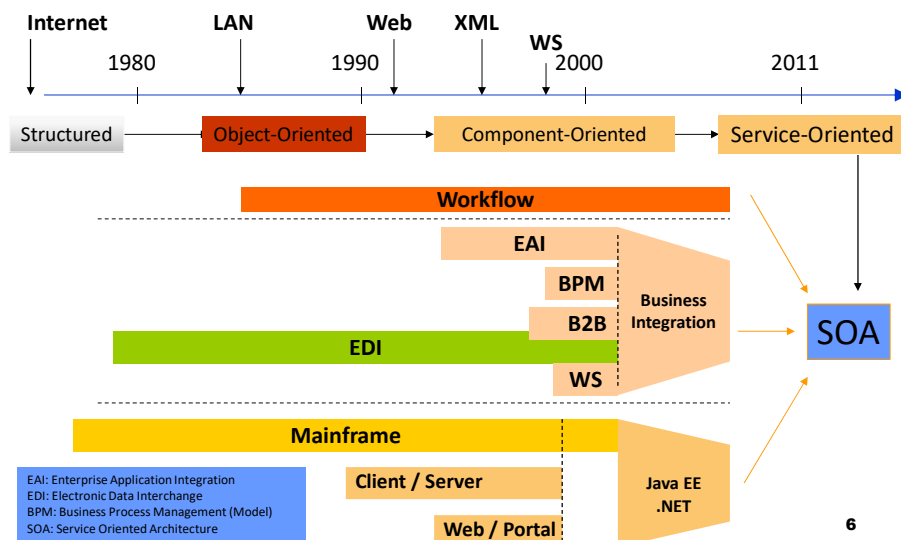
## Sistemas distribuidos



Uso de sistemas existentes como unidades de construcción para componer aplicaciones distribuidas

5

## Servicios Web: Tecnologías



6

# Programación orientada a servicios

- Con una **nueva tecnología** de los objetos distribuidos:
  - Invocación distante de los servicios Web: SOAP (~IIOP)
  - Descripción de los servicios Web: WSDL (~IDL)
  - Registro y descubrimiento de los servicios Web: UDDI (~NameService)
- Basados sobre los **estándares XML**:
  - Estándares del W3C: XML, SOAP, WSDL
  - Estándares industriales: UDDI, ebXML
  - Propietarios: DISCO, WSDD, WSFL, ASMX, ...
- Implementaciones **actuales**:
  - Microsoft .Net
  - Sun JavaONE: Java EE + Web Services (WSDP = JAXP, JAX-RPC, JAXM...)
  - Apache SOAP / Axis, IBM WSTK
  - Oracle, Bea, Iona, Enhydra, ...

7

# Plan

- ✓ **Programación orientada a servicios**
- Servicio:
  - Definición
  - Llamados SOAP y REST
  - Construcción
- Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

8

## Servicio

- Acción que se desarrolla en favor de un tercero
- Un servicio ofrece una función que se exporta a través de una interfaz
- Un servicio es un **sistema de software** diseñado para **soportar la interacción interoperable computadora a computadora** a través de la red (e.g., Internet)
  - Reutilizable
  - Independiente de
    - La plata-forma (UNIX, Windows, ...)
    - La implementación (VB, C#, Java, ...)
    - La arquitectura sub-yacente (.NET, Java EE, SOAP / Axis, ...)

9

## Problemas básicos a resolver

- **Cómo hacer que la invocación del servicio sea parte del lenguaje** (de una manera más o menos transparente):
  - No hay que olvidar un aspecto importante: Todo lo que usted diseñe, otros tendrán que programarlo y utilizarlo
- **Cómo intercambiar datos entre máquinas** utilizando representaciones diferentes:
  - tipo de formatos de datos (por ejemplo, *little endian* vs. *big endian*)
  - estructuras de datos (necesidad de ser aplanada y la reconstrucción)
- **Cómo encontrar el servicio realmente requerido** entre un conjunto potencialmente grande de servicios y servidores:
  - El objetivo es que el cliente no tiene necesariamente que saber dónde reside el servicio o el servidor que proporciona el servicio
- **Cómo lidiar con los errores** de la invocación a servicios (de una manera más o menos elegante): El servidor está caído, la comunicación está caída, el servidor ocupado, las solicitudes están duplicadas ...

10

# Plan

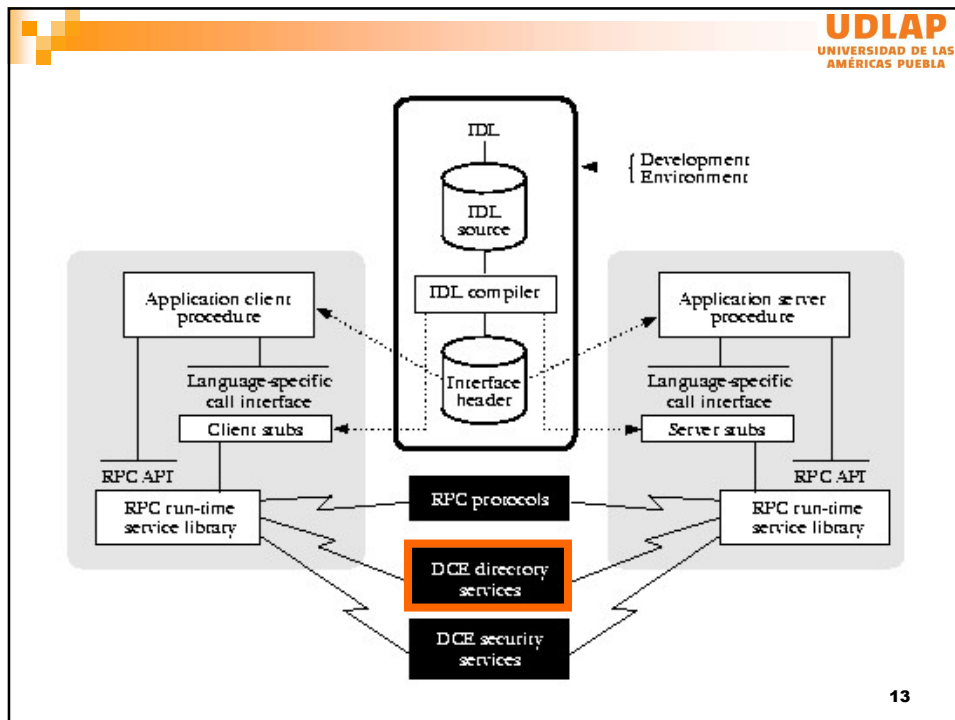
- ✓ Programación orientada a servicios
- Servicio:
  - ✓ Definición
  - Llamados SOAP y REST
  - Construcción
- Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

11

## RPC-Binding

- Un servicio es proporcionado por un servidor ubicado en una determinada dirección IP e instalado en un determinado puerto:
  - El *binding* es el **proceso de asignar un nombre a una dirección y a un puerto** que pueden ser utilizados para fines de comunicación
  - El *binding* puede hacerse:
    - **A nivel local**: el cliente debe conocer el nombre (dirección) del host del servidor
    - **Distribuido**: hay un servicio independiente a cargo de la asignación de nombres y direcciones (servidor de nombres, directorio)
      - Los servicios deben ser accesibles por todos los participantes
- En general con un servidor de nombres, varias operaciones son posibles:
  - **REGISTER** (suscribir interfaz): Un servidor puede registrar los nombres de los servicios y el puerto correspondiente
  - **WITHDRAW**: Un servidor puede retirar un servicio
  - **LOOKUP** (importar interfaz): Un cliente puede solicitar la dirección y el puerto de un servicio determinado
- También debe haber una manera de localizar el servidor de nombres (sitio predefinido, variables de entorno, *broadcast* a todos los nodos)

12



UDLAP  
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

## Llamado a distancia

- Solicitar la ejecución de un método que exporta un servicio:
  - Vía mensajes SOAP
  - De acuerdo a una especificación WSDL
  - Codificación más o menos importante

14

# Protocolo de comunicación

- Un nuevo protocolo para la Web:
  - Multi-lenguajes, multi-plataformas
  - Respetando los **formatos** de intercambio de la Web
    - Respuestas y consultas en XML
  - Fácil de implementar sobre diferentes **protocolos** de transporte
    - RPC, HTTP, SMTP entre otras MOM (*Message Oriented Middleware*)
  - Permitiendo franquear los « **firewalls** »
    - ATENCIÓN: se pierde el control del acceso a débil granularidad
  - Con una **especificación no-propietaria**, garantizada por un organismo independiente
    - W3C
- SOAP (*Simple Object Access Protocol*)

15

# Llamado SOAP

16



## Simple Access Protocol (SOAP)

- SOAP es un **protocolo mínimo** para llamar a los métodos de los servidores, servicios, componentes, objetos:
  - No imponer un API o ambiente de ejecución
  - No imponer el uso de un ORB (*Object Request Broker* como CORBA, DCOM, ...) o de un servidor Web particular (Apache, IIS, ...)
  - No imponer un modelo de programación: Se pueden usar varios modelos conjuntamente
  - **No reinventar una nueva tecnología**: Uso conjunto de XML y HTTP o SMTP
- Construido para poder ser transportado cómodamente sobre todas las plataformas y las tecnologías

17

## ¿Qué es SOAP?

- Un **formato de mensaje** de comunicación unidireccional que describe cómo un mensaje puede ser empaquetado en un documento XML
- Una **descripción de cómo** un mensaje SOAP (o el documento XML que constituye un mensaje SOAP) deben **transportarse utilizando HTTP** (para la interacción basada en la Web) o **SMTP** (para correo electrónico basado en la interacción)
- Un **conjunto de reglas** que deben seguirse cuando se procesa un mensaje SOAP y una **clasificación simple de las entidades** involucradas en el procesamiento de un mensaje SOAP
  - Asimismo, se especifica qué partes de los mensajes debe ser leídas, por quién y cómo reaccionar en caso de que el contenido no se entienda
- Un **conjunto de convenciones** sobre cómo **convertir una llamada RPC en un mensaje SOAP** y cómo aplicar el estilo RPC de interacción:
  - Esto es cómo el cliente realiza una llamada RPC, la cual se traduce en un mensaje SOAP que transmitida se convirtió en un llamada RPC hacia el servidor
  - La respuesta del servidor se convierte en un mensaje SOAP que se envía al cliente, la cual se transmite al cliente como la devolución de la llamada RPC

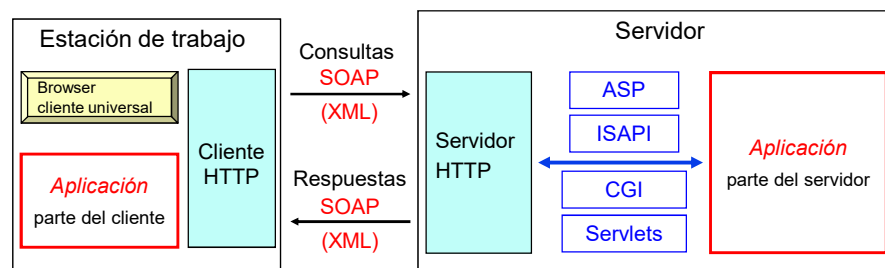
18

## Antecedentes de SOAP

- SOAP fue originalmente concebido para ser la **infraestructura mínima necesaria** para llevar a cabo una llamada RPC a través de Internet:
  - Usar XML como representación intermedia entre sistemas
  - Tener estructuras muy simple de mensajes
  - Utilizar HTTP para hacer túneles a través de los *fire-wall* y usar la infraestructura de la Web
- La idea era **evitar los problemas** asociados con CORBA, IIOP's / GIOP (que cumplen un papel similar, pero que no tienen una representación intermedia estándar y que de todas maneras tienen que ser canalizados a través de HTTP)
- El objetivo era tener una **extensión** que pudiera ser fácilmente **conectada a través de los middlewares** permitiéndoles interactuar a través de Internet en lugar de una LAN como es el caso típico
- SOAP comenzó a ser presentado como un **vehículo genérico para el intercambio de mensajes** a través de la Internet permitiendo las interacciones distintas a RPC y el uso de protocolos diferentes a HTTP

19

## En resumen: SOAP = HTTP+XML



20

## ¿Por qué HTTP? (1)

- HTTP está **disponible sobre todas las plataformas** – de forma rápida:
  - HTTP (*HyperText Transfer Protocol*) se ha transformado (**de facto**) en **EL protocolo** de comunicación de Internet
  - HTTP es un protocolo simple, que requiere **poco soporte** para funcionar correctamente
- HTTP:
  - Es un **protocolo sin conexión**
  - Se requiere de **pocos paquetes** para intercambiar la información
  - Ofrece un nivel de **seguridad** simple y efectivo
  - Es el único protocolo **utilizable a través de un "fire-wall"**

21

## ¿Por qué HTTP? (2)

- Protocolo de comunicación "solicitud-respuesta" para el modelo cliente-servidor
- HTTP define:
  - Las operaciones CRUD para la manipulación de los recursos
  - Los códigos de estado de los resultados del procesamiento de una solicitud

22

# HTTP: Métodos

- Los métodos (llamados **verbos**) se especifican en HTTP 1.0

Method	Description	Safe	Idempotent
GET	Requests a specific representation of a resource	Yes	Yes
PUT	Create or update a resource with the supplied representation	No	Yes
DELETE	Deletes the specified resource	No	Yes
POST	Submits data to be processed by the identified resource	No	No
HEAD	Similar to GET but only retrieves headers and not the body	Yes	Yes
OPTIONS	Returns the methods supported by the identified resource	Yes	Yes

- Nota:** La semántica actual de POST está definida por el servidor

# HTTP: Códigos de estado

Status Range	Description	Examples
100	Informational	100 Continue
200	Successful	200 OK
201	Created	
202	Accepted	
300	Redirection	301 Moved Permanently
304	Not Modified	
400	Client error	401 Unauthorized
402	Payment Required	
404	Not Found	
405	Method Not Allowed	
500	Server error	500 Internal Server Error
501	Not Implemented	

# HTTP: Mensajes

- Una **solicitud** contiene:
  - El tipo de representación esperado
  - Utiliza la negociación de contenidos de HTTP
- La **respuesta** contiene:
  - El flujo de bytes
  - El tipo del contenido (por ejemplo, XML, JPEG, HTML)
  - El código de estado

# HTTP: Ejemplo de consulta

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```



Solicitud de  
una cotización

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV =
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## HTTP: Ejemplo de respuesta

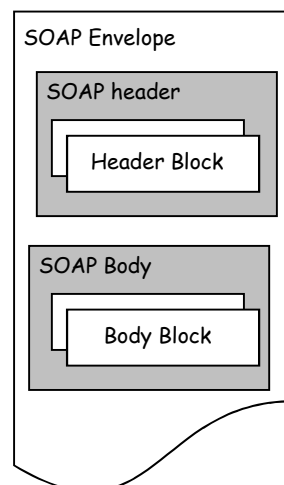
HTTP/1.1 200 OK  
Content-Type: text/xml; charset = "utf-8"  
Content-Length: nnnn

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV =
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle =
    "http://schemas.xmlsoap.org/soap/encoding/" />
<SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m = "Some-URI">
    <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

27

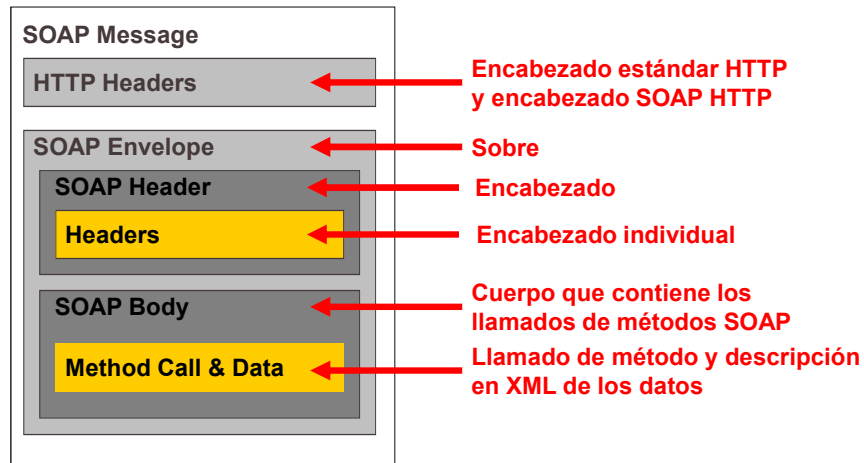
## Mensajes SOAP

- SOAP se basa en el **intercambio de mensajes**
- Los mensajes son **vistos como sobres** en los que las aplicaciones embarcan sus datos para ser enviados
- Un mensaje tiene **dos partes principales**:
  - Encabezado (dividido en bloques)
  - Cuerpo (dividido en bloques)
- SOAP no dice qué hacer con el encabezado y el cuerpo, sólo indica que el **encabezado es opcional** y el **cuerpo obligatorio**
- Sin embargo especifica el uso implícito del encabezado y del cuerpo:
  - El cuerpo es para los **datos de la aplicación**
  - La encabezado es para los **datos de la infraestructura**



28

## Estructura de mensaje SOAP



29

## Mensaje SOAP

- Identificador para la serialización SOAP
- Identificador para el sobre SOAP

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

30

## Estructura de un mensaje

- Sobre / *Envelope*:
  - Elemento raíz
  - Namespace:
 

```
SOAP-ENV http://schemas.xmlsoap.org/soap/envelope/
```
- Encabezado / *Header*:
  - Elemento opcional
  - Contiene las **entradas (bloques) no aplicativas**: Transacciones, sesiones, ...
- Cuerpo / *Body*:
  - Contiene las **entradas (bloques) del mensaje**: Nombre de un procedimiento, valores de los parámetros, valores de regreso (*return*)
  - Puede contener los elementos « *fault* » (errores)

31

## Encabezado SOAP

- No es necesariamente dependiente de la aplicación (la aplicación puede no estar consciente de que un encabezado fue adjuntado al mensaje)
  - Usos típicos: **información de coordinación**, **identificadores** (e.g., para transacciones), **información de seguridad** (e.g., certificados)
- SOAP proporciona mecanismos (atributos) para especificar **quién** debe hacerse cargo de los encabezados y **qué** hacer con ellos:
  - El atributo **actor**: indica quién debe procesar una entrada (bloque) específica del encabezado; el actor puede ser:
    - **none**: es utilizado para propagar la información que no necesita ser procesada
    - **next**: indica que el nodo que recibe el mensaje puede procesar el bloque
    - **ultimateReceiver**: indica que el encabezado será procesado por el destinatario final del mensaje
  - El atributo **mustUnderstand**: determina si es obligatorio (**mustUnderstand=1**) o no (**mustUnderstand=0**) que un nodo (ver atributo actor) procese el mensaje
  - SOAP 1.2 añade el atributo **relay**: reenvía el encabezado si no fue procesado

32



## Cuerpo SOAP

- Destinado a los **datos específicos de la aplicación** contenidos en el mensaje
- Una entrada (bloque) en el cuerpo es sintácticamente equivalente a una entrada en el encabezado con los atributos: **actor** = **ultimateReceiver**, **mustUnderstand** = 1
- A diferencia de los encabezados, SOAP no especifica el contenido de todas las entradas del cuerpo:
  - Especifica la transformación de RPC a una colección de entradas de cuerpo SOAP
  - Especifica la entrada *Fault* (para informar de errores en el procesamiento de un mensaje SOAP)
- La entrada *Fault* tiene cuatro elementos (versión 1.1):
  - **Fault code**: indica la clase de error (e.g., versión, **mustUnderstand**, cliente, servidor)
  - **Fault string**: explicación legible de la falla (no destinadas a un tratamiento automatizado)
  - **Fault actor**: indica quién originó la falla
  - **Detail**: información específica de la aplicación sobre la naturaleza de la falla

33

## Elemento de falla SOAP

- En la versión 1.2, el elemento *Fault* se especifica con **más detalle**, debe contener dos sub-elementos obligatorios:
  - **Code**: contiene un valor (el código de la falla) y posiblemente un sub-código (para la información específica de la aplicación)
  - **Reason**: igual que **Fault string** en la versión 1.1
- Puede contener **elementos adicionales**:
  - **Detail**: como en la versión 1.1
  - **Node**: el identificador del nodo que produjo la falla (en ausencia, por defecto es el destinatario del mensaje)
  - **Role**: la función desempeñada por el nodo que generó el error
- Los errores en la comprensión de un encabezado obligatorio son respondidos con un elemento *Fault*, el cual incluye un encabezado especial indicando cual de los encabezados originales no se entendió

34

# Reglas de codificación

- Para los **tipos de base**:
  - Se tienen reglas por defecto
  - La codificación es de tipo: **element**
  - Un elemento por campo, el nombre del elemento es el mismo del campo
- Definen un **sistema de tipos (types)**:
  - Compatible con *XML Schema Definition (XSD)*
  - Los tipos SOAP pueden ser descritos usando XSD
  - SOAP utiliza las convenciones XSD para asociar las instancias a los tipos:
 

```
<foo xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      xsi:type="timeInstant">1999-11-12T09:43</foo>
```
  - Las tablas y las referencias son tipificadas de manera específica utilizando XSD

35

# Reglas de codificación

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>■ Tipos primitivos:               <pre>&lt;element name="price" type="float"/&gt; &lt;element name="greeting" type="xsd:string"/&gt;</pre> </li> <li>■ Estructuras:               <pre>&lt;element name="Book"&gt;&lt;complexType&gt;   &lt;element name="author" type="xsd:string"/&gt;   &lt;element name="title" type="xsd:string"/&gt; &lt;/complexType&gt;&lt;/element&gt;</pre> </li> <li>■ Enumeración:               <pre>&lt;element name="color"&gt;   &lt;simpleType base="xsd:string"&gt;     &lt;enumeration value="Green"/&gt;     &lt;enumeration value="Blue"/&gt;   &lt;/simpleType&gt; &lt;/element&gt;</pre> </li> </ul> | <ul style="list-style-type: none"> <li>■ Ejemplos:               <pre>&lt;price&gt;15.57&lt;/price&gt; &lt;greeting id="id1"&gt;Hello&lt;/greeting&gt;</pre> </li> <li>■ Ejemplo:               <pre>&lt;e:Book&gt;   &lt;author&gt;J.R.R Tolkien&lt;/author&gt;   &lt;title&gt;A hobbit story&lt;/title&gt; &lt;/e:Book&gt;</pre> </li> <li>■ Ejemplo:               <pre>&lt;color&gt;Blue&lt;/color&gt;</pre> </li> </ul> |
|--|--|

36

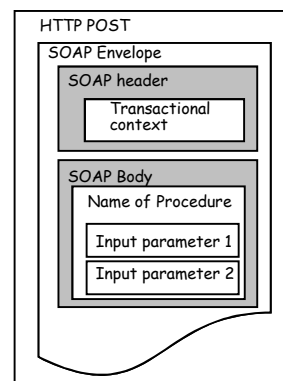
## Procesamiento de mensajes

- SOAP especifica en detalle cómo los mensajes deben ser procesados (en particular, indica cómo debe ser procesado el encabezado)
  - Cada nodo definido en el encabezado tiene una **función** asociada: Hay tres funciones estándares: **none**, **next**, **ultimateReceiver**
  - La función determina **quién** es el responsable (actor) de cada bloque
  - Las aplicaciones pueden definir sus propias funciones y utilizarlas en los mensajes
  - Si un bloque no tiene una función asociada a él, **por defecto** es **ultimateReceiver**
  - Si se incluye la bandera **mustUnderstand=1**, un nodo que coincide con la función indicada debe procesar esa parte del mensaje, de lo contrario, debe generar un error y no seguir con el procesamiento del mensaje
- SOAP 1.2 incluye el atributo **relay**:
  - Si está definido el atributo, un nodo que no procesa esa parte del mensaje, debe transmitirlo (es decir, no debe quitarla)
  - El uso del atributo **relay**, combinado con la función **next**, permite indicar información útil para establecer la persistencia a lo largo de la ruta del mensaje (como información de la sesión)

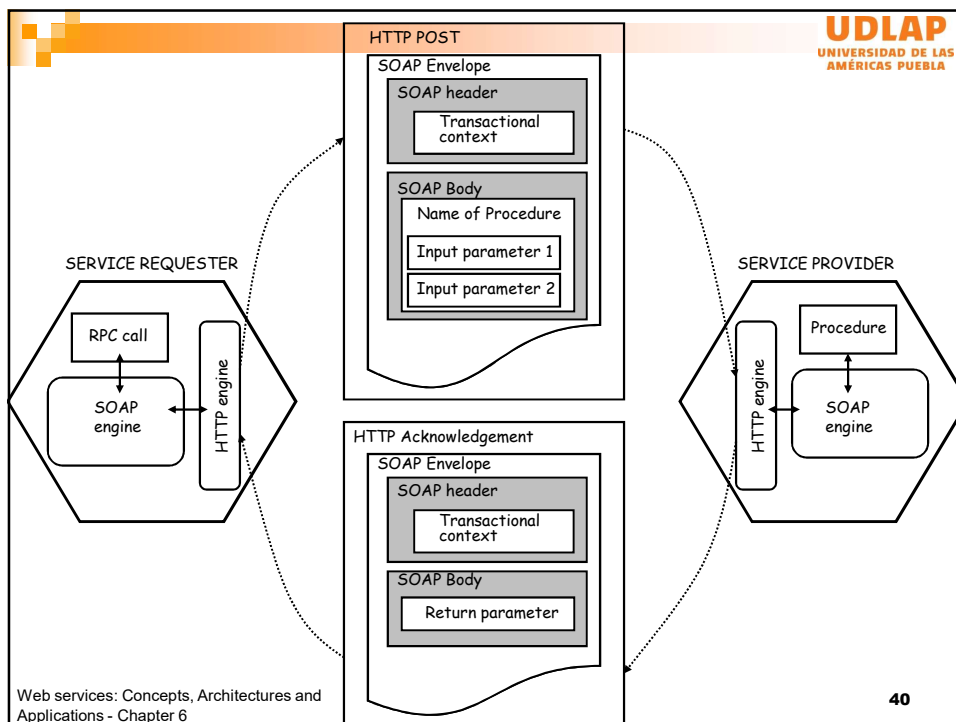
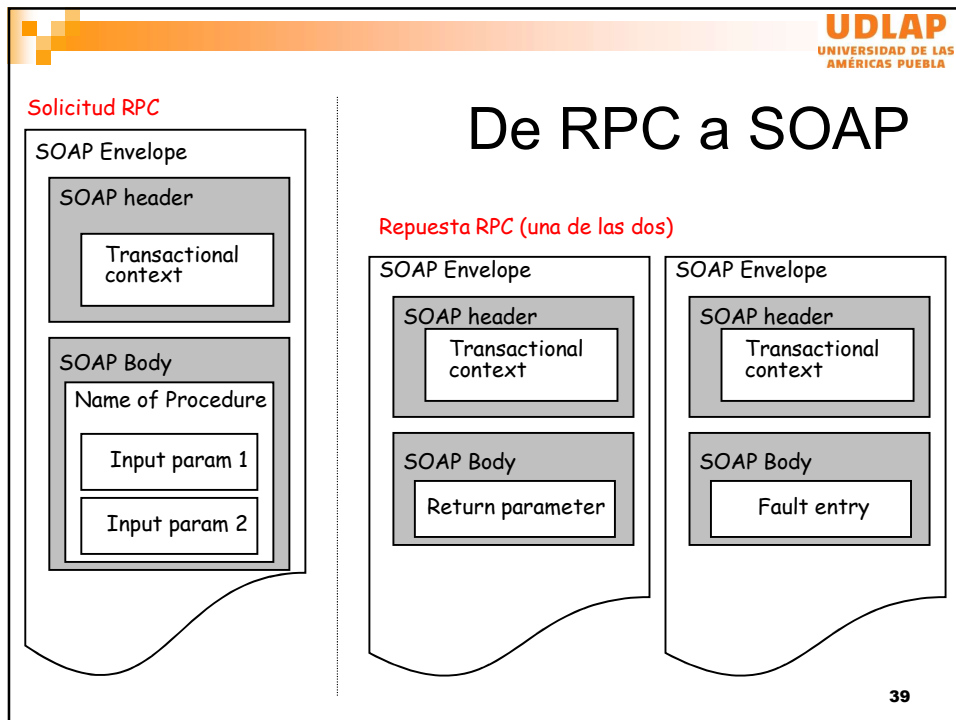
37

## SOAP y HTTP

- Un *binding* (de SOAP a un protocolo de transporte) es una descripción de cómo un mensaje SOAP es enviado utilizando dicho protocolo
- El *binding* típico para SOAP es HTTP
- SOAP puede utilizar GET o POST
  - Con **GET**, la petición no es un mensaje SOAP, pero la respuesta sí
  - Con **POST**, la petición y la respuesta son mensajes SOAP
- SOAP utiliza los mismos códigos de error y estado utilizados en HTTP así que las respuestas HTTP puede ser interpretadas directamente por un módulo SOAP



38



## Consulta en XML

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

41

## Respuesta en XML

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

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

42

## Resumen de SOAP

- SOAP, en su forma actual, constituye un **mecanismo básico** para:
  - Encapsular mensajes en un documento XML
  - Transformar el documento XML con el mensaje SOAP en una petición HTTP
  - Transformar las llamadas RPC a mensajes SOAP
  - Aplicar las reglas sobre cómo procesar un mensaje SOAP (las reglas son más precisas y entendibles en la v1.2 de la especificación)
- SOAP se **aprovecha de la estandarización de XML** para resolver los problemas de representación de datos y serialización (utiliza el esquema XML para representar los datos y las estructuras y se basa también en XML para serializar los datos para la transmisión)
  - Entre más poderoso sea XML y más estándares aparezcan en torno a XML, SOAP pueden tomar ventaja de ellos, simplemente indicando qué esquema y codificación fue utilizada para definir el mensaje SOAP
  - El esquema y la codificación actuales son genéricos, pero pronto habrá estándares verticales implementando esquemas y codificaciones adaptados a un área de aplicación particular (e.g., los esfuerzos en torno a EDI)
- SOAP es un **protocolo muy sencillo**:
  - Destinado a la transferencia de datos de una plataforma *middleware* a otra
  - A pesar de sus pretensiones de ser abierto (que son verdaderas), las especificaciones actuales están muy ligados a RPC y HTTP

43

## SOAP y modelo cliente-servidor

- La estrecha relación entre SOAP, RPC y HTTP tiene dos razones principales:
  - SOAP ha sido inicialmente diseñado para el tipo de interacción cliente-servidor que normalmente se implementa usando RPC o variaciones del mismo
  - RPC, SOAP y HTTP siguen modelos muy similares de interacción que pueden ser muy fácilmente mapeados entre sí (y esto es lo que ha hecho SOAP)
- Las **ventajas de SOAP** se derivan de su capacidad de proporcionar un **vehículo universal** para la transmisión de información a través de plataformas *middleware* y aplicaciones ambas heterogéneas
  - En este sentido, SOAP desempeñará un papel crucial en los esfuerzos de integración de aplicaciones empresariales del futuro como lo establece la norma que ha estado ausente en todos estos años
- Las **limitaciones de SOAP** se derivan de su adhesión al **modelo cliente-servidor**:
  - el intercambio de datos como parámetros en llamadas a métodos
  - los patrones de interacción rígidos que son altamente síncronos
- Acerca de su sencillez: SOAP no es suficiente en una aplicación real, muchos aspectos están faltando

44

## SOAP como protocolo simple

- SOAP:
  - No incluye nada acerca de fiabilidad, intercambio de mensajes complejos, transacciones, seguridad, ...
  - Por esta razón, SOAP **no es suficiente** para implementar **aplicaciones empresariales de gran talla** que incorporen características típicas de *middleware*, como las transacciones o la entrega fiable de mensajes
- SOAP no impide que tales características sean implementadas, pero necesitan ser estandarizadas para ser útiles en la práctica:
  - WS-Security, WS-Coordination, WS-Transactions, ...
  - Una gran variedad de normas adicionales se están proponiendo para agregar la funcionalidad faltante

45

## Llamado REST

46

## REST (*Representational State Transfer*)

- **Estilo de arquitectura** para el desarrollo de sistemas distribuidos orientados a recursos
- Características:
  - **Identificadores** únicos de recursos
  - **Interfaz** estándar para acceder a los recursos
  - Los recursos pueden tener **múltiples representaciones**
- REST describe la **arquitectura de la Web**

## REST: Servicio RESTful HTTP

- Servicio adoptado por REST para **exponer las estructuras** de los datos y las funcionalidades
- Construido encima de las tecnologías Web, éste utiliza:
  - **URIs** para la identificación de recursos
  - **Operaciones HTTP** para acceder a los recursos
  - **Tipos de datos comunes** (por ejemplo, XML, JSON, HTML, RSS, Atom) para la representación de los recursos



## URI (*Uniform Resource Identifier*)

- URI es un esquema **uniforme** y extensible
- Un **recurso** es una entidad que puede ser llamada, direccionada y administrada en la red
- Un **identificador** es una secuencia de caracteres que apunta a un recurso

## URI: Tipos

- *Uniform Resource Locator* (URL):
  - Identifica y proporciona los medios para la localización de un recurso
- *Uniform Resource Name* (URN):
  - Persistente, incluso si el recurso deja de funcionar o de estar disponible

## URI: Esquema

- Define las reglas para la **asignación de identificadores** usando diferentes protocolos
- Ejemplos:

### HTTP

http://www.udlap.mx

### MAIL

mailto:joseluis.zechinelli@udlap.mx

### GEO

geo: 48.890172, 2.249922

### Spotify

spotify:user:javieraespinosa

### Skype

skype:javiera.espinosa

### LastFM

lastfm://user/javieraespinosa

## URI: Sintaxis

- BNF:
  - protocolo :// host [ : port ] path [ ? query ] [ # fragment ]
- Ejemplo:
  - http :// www.facebook.com : 80 / joseluis.zechinelli ? sk=info

### Caracteres permitidos

0 ... 9 Digit

A ... Z Alphabet

- ... ~ ASCII symbols

### Caracteres reservados

/ ? # [ ] @ :

! \$ & ' ( ) \* + , ; =

## URI: Codificación

- Principio:

- Codifica **símbolos no ASCII** y **caracteres reservados**, con el triple "% HEXADIG HEXADIG"

- Ejemplo:

- Hi Zoé! → Hi%20Zo%C3%A9

## JSON (*JavaScript Object Notation*)

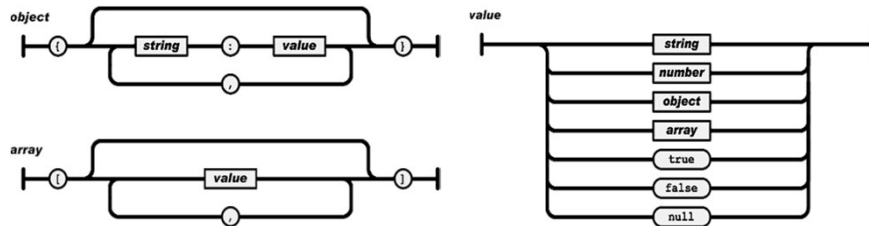
- **Formato de texto** ligero para el intercambio de datos estructurados:

- Basado en un modelo de datos complejos
- Al igual que XML

- **Conceptos fundamentales:**

- **Objeto:** Conjunto desordenado de pares "*nombre: valor*"
- **Array:** Conjunto ordenado de valores
- **Valor:** Miembro de la cadena, booleano, objeto o conjunto de matriz

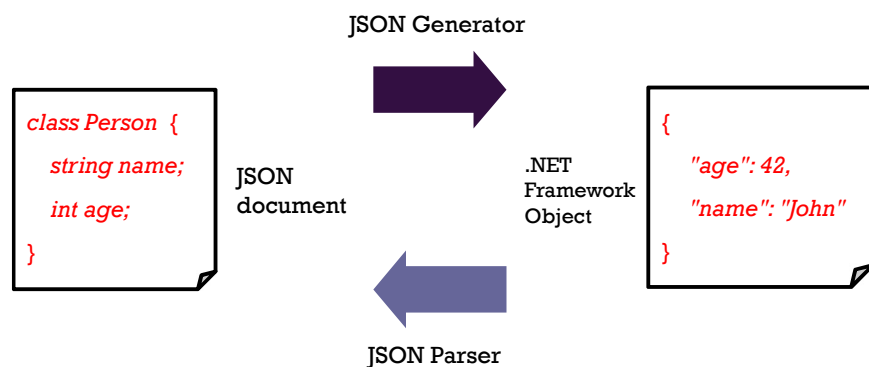
## JSON: Sintaxis



Fuente: <http://www.json.org/>

55

## JSON: Ejemplo de serialización



56

# Plan

- ✓ Programación orientada a servicios
- Servicio:
  - ✓ Definición
  - ✓ Llamados SOAP y REST
  - Construcción
- Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

57

# Desarrollo automático

- El objetivo final de WSDL es proporcionar apoyo para la **automatización del proceso de desarrollo de servicios Web**:
  - Dada una descripción WSDL, un **compilador genera los stubs y skeletons** necesarios para desarrollar los clientes que pueden interactuar con el servicio
  - Es por esto que WSDL **se basa en un protocolo estándar (SOAP)** para que los **stubs** genéricos puedan ser creados
  - WSDL es un **punto** entre los **servicios internos y externos (Web)**
- Del mismo modo, el objetivo final de ebXML es **automatizar el proceso de elaboración de un convenio de colaboración**, implementándolo y aplicando sus normas:
  - Dado un acuerdo de colaboración, uno debe ser capaz de generar **automáticamente el stub o el skeleton** de los procesos de negocios individuales
  - Los clientes **sólo tienen que extender el proceso del stub** con su propia lógica interna
  - Es por esto que ebXML **necesita más que SOAP** para controlar y dirigir el flujo de mensajes entre los clientes

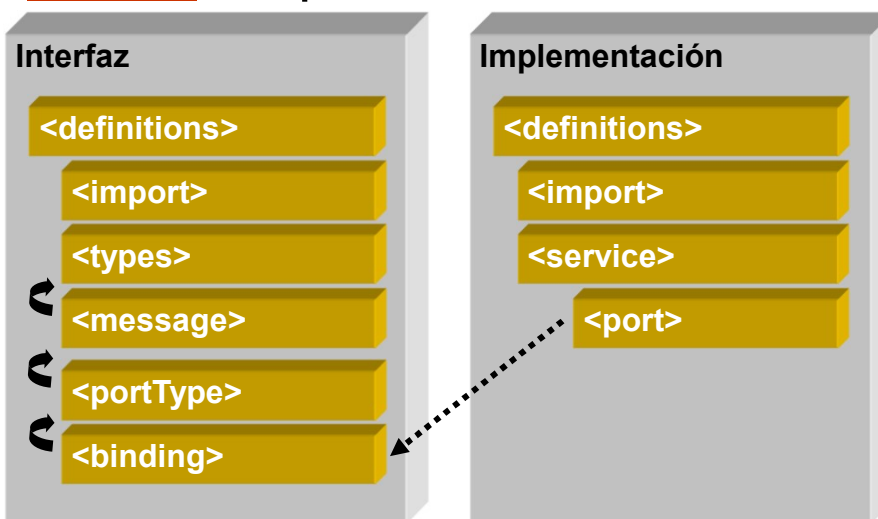
58

## Conversations


- WSDL, en su versión actual, es una **extensión de las IDL** que apoya la interacción a través del Internet:
  - XML como sintaxis y sistema de tipos
  - Posibilidad de agrupar las operaciones en un servicio
  - Diferentes opciones para acceder al servicio (direcciones y protocolos)
- Esta es su gran **ventaja** ...
  - Es sencillo adaptar los *middleware* existentes para usar o soportar WSDL
  - Es trivial traducir las IDL existentes a WSDL (traducción automática)
- ... pero también su **desventaja** dado que:
  - El comercio electrónico y las interacciones B2B no son llamadas simples a servicios
  - WSDL **no refleja la estructura de los procedimientos a seguir** para interactuar correctamente con un servicio (*conversations*)
    - El protocolo de negocios = conjunto válido de las conversaciones
- Sin un protocolo de negocios, la mayoría de los desarrollos se siguen haciendo de forma manual

59

## WSDL: Especificación



60



UDLAP  
UNIVERSIDAD DE LAS  
AMÉRICAS PUEBLA

Copyright Springer Verlag Berlin Heidelberg 2004

```

<?xml version="1.0"?>
<definitions name="Procurement"
  targetNamespace="http://example.com/procurement/definitions"
  xmlns:tns="http://example.com/procurement/definitions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/" >

  <message name="OrderMsg">
    <part name="productName" type="xs:string"/>
    <part name="quantity" type="xs:integer"/>
  </message>

  <portType name="procurementPortType">
    <operation name="orderGoods">
      <input message="OrderMsg"/>
    </operation>
  </portType>

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

  <service name="ProcurementService">
    <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
      <soap:address location="http://example.com/procurement"/>
    </port>
  </service>

</definitions>

```

```

<message name="OrderMsg">
  <part name="productName" type="xs:string"/>
  <part name="quantity" type="xs:integer"/>
</message>

```

Parte abstracta

mensaje

```

<portType name="procurementPortType">
  <operation name="orderGoods">
    <input message="OrderMsg"/>
  </operation>
</portType>

```

operación y  
tipo del puerto

```

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

```

Parte concreta

binding

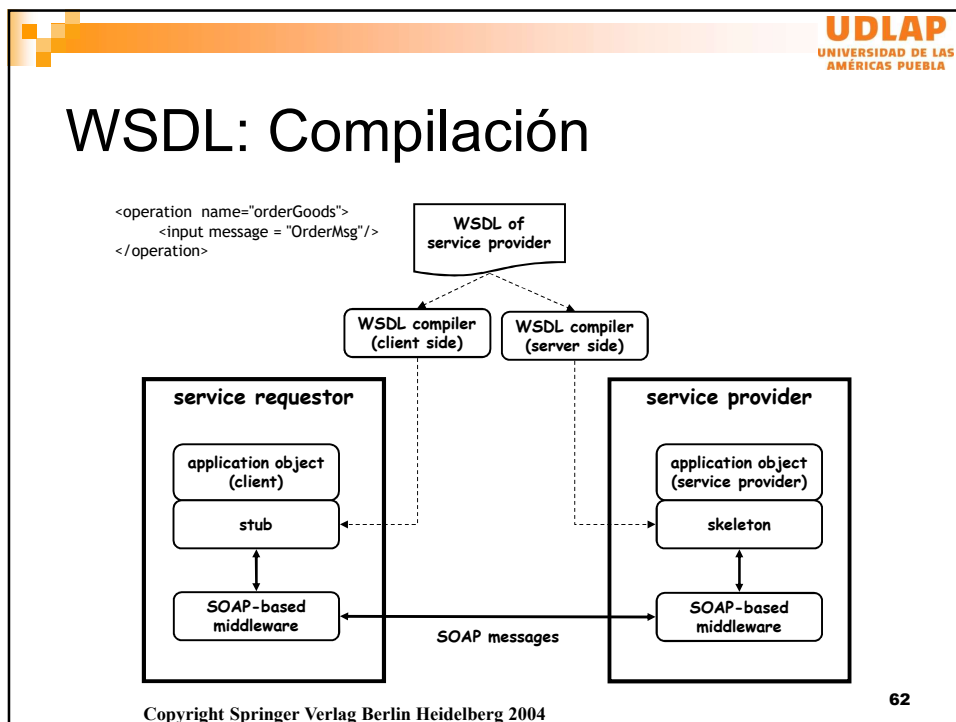
```

<service name="ProcurementService">
  <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
    <soap:address location="http://example.com/procurement"/>
  </port>
</service>

```

Puerto y  
servicio

61



# Plan

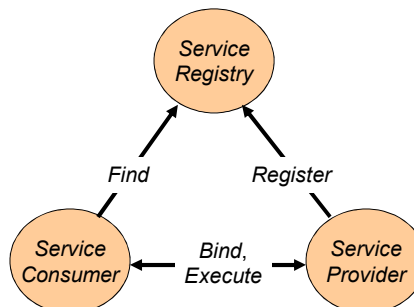
- ✓ Programación orientada a servicios
- ✓ Servicio:
  - Definición
  - Llamados SOAP y REST
  - Construcción
- Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

63

# Arquitectura a servicios

SOA es la **agregación de componentes** que **satisfacen un administrador de empresa** (*business*); SOA conlleva componentes / servicios

- Una aplicación SOA es una composición de servicios
- Un servicio es una unidad atómica SOA
- Un servicio encapsula un proceso de negocios
- La utilización de un servicio implica: encontrar, ligar, ejecutar
- Ejemplo: **Servicios Web**

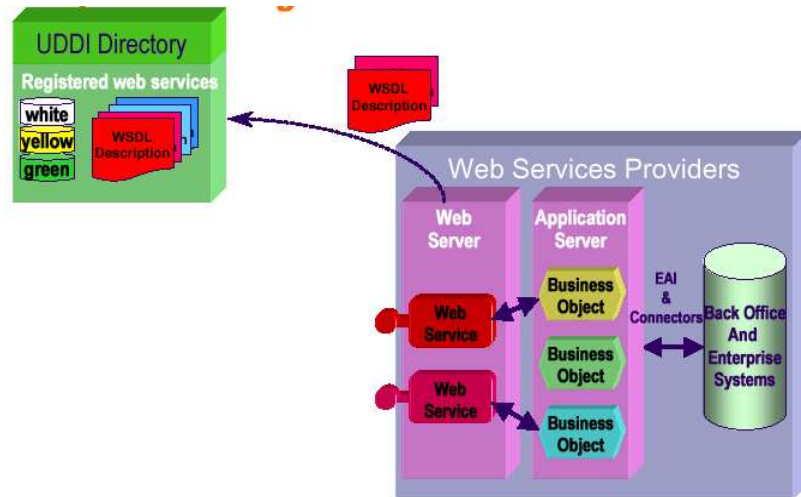


Los actores SOA

64

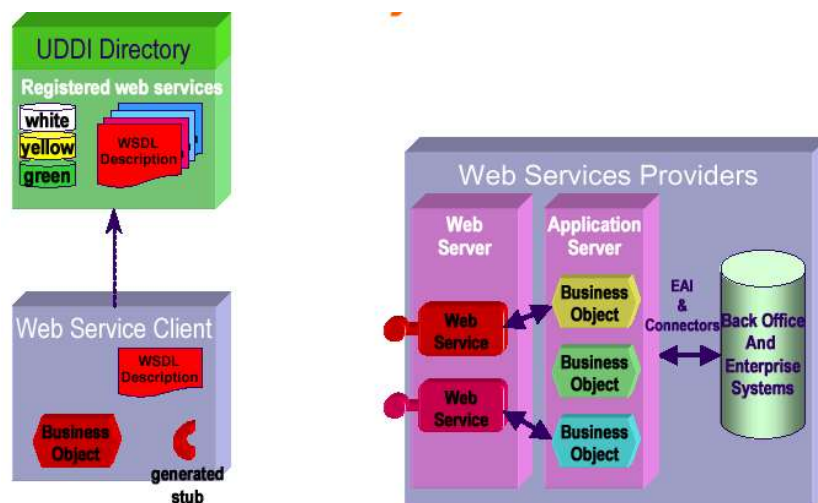


## Registro de un servicio



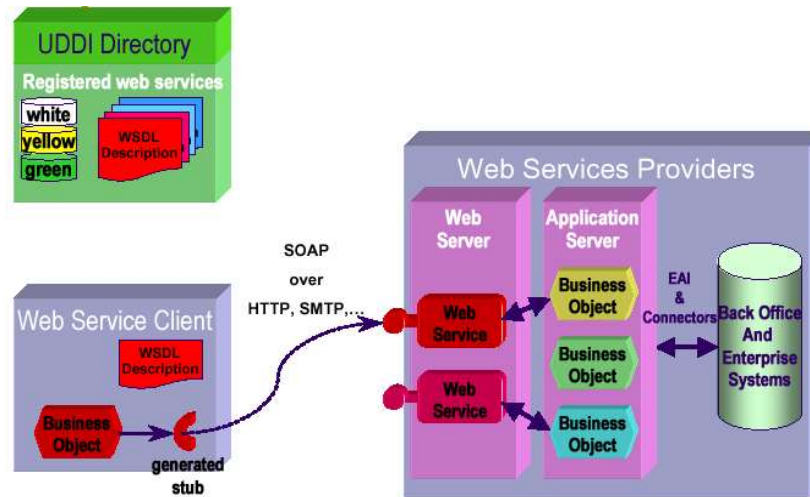
65

## Descubrimiento de un servicio

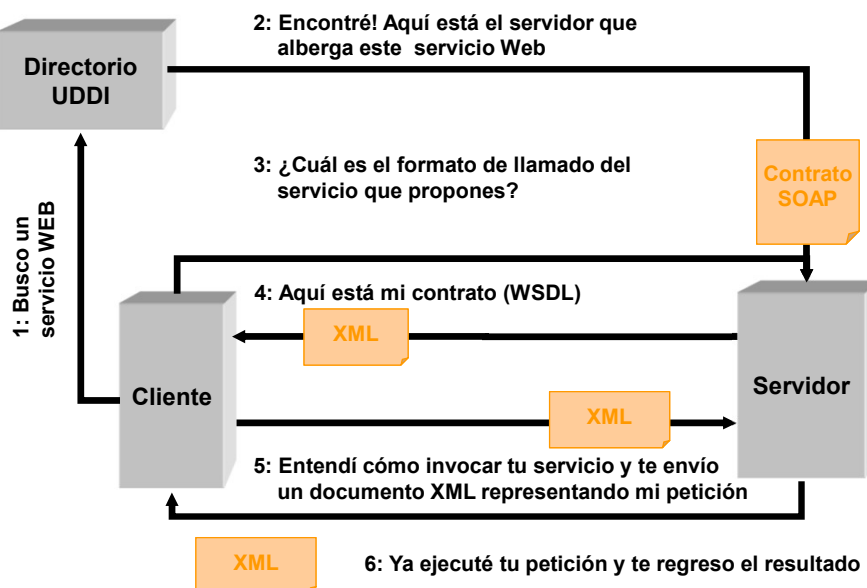


66

## Invocación de un servicio



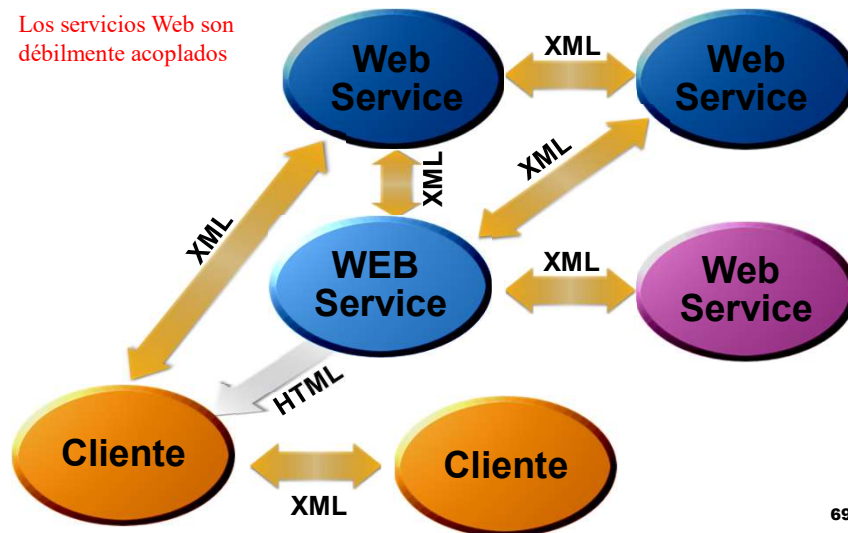
67



68

## Modelo cliente – servidor

Los servicios Web son débilmente acoplados



69

## Plan

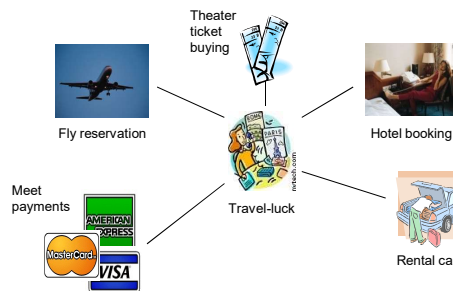
- ✓ Programación orientada a servicios
- ✓ Servicio:
  - Definición
  - Llamados SOAP y REST
  - Construcción
- Construcción de aplicaciones a base de servicios:
  - ✓ Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

70

# Composición

- Construir servicios Web con valor agregado especificando las interacciones entre los servicios:

- Coreografía
- Orquestación

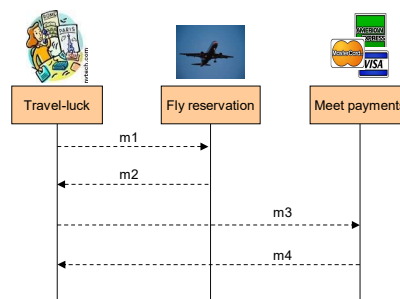


71

# Coreografía

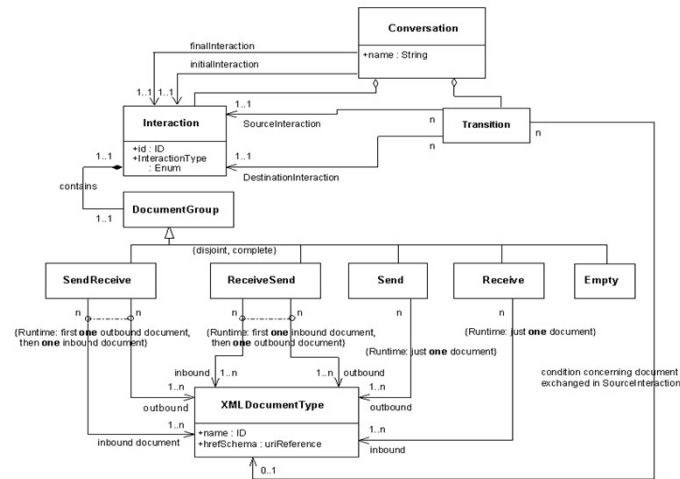
- Captura el **intercambio de mensajes** públicos que se producen entre los múltiples servicios Web

- WS-CDL



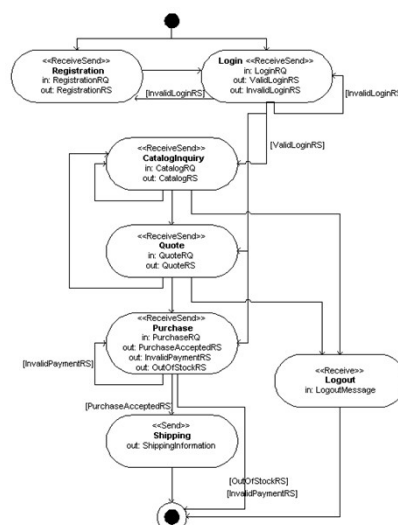
72

# Especificación WS-CDL



73

# Diagrama de conversación



74

## Ejemplo: Interacciones

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
  xmlns="http://www.w3.org/2002/02/wsdl10"
  initialInteraction="Start" finalInteraction="End" >

  <ConversationInteractions>
    <Interaction interactionType="ReceiveSend" id="Login">
      <InboundXMLDocument hrefSchema="http://conv123.org/LoginRQ.xsd"
        id="LoginRQ"/>
      <OutboundXMLDocument hrefSchema="http://conv123.org/ValidLoginRS.xsd"
        id="ValidLoginRS"/>
      <OutboundXMLDocument hrefSchema="http://conv123.org/InvalidLoginRS.xsd"
        id="InvalidLoginRS" />
    </Interaction>
    ...
  </ConversationInteractions>
```

75

## Ejemplo: Transiciones (1)

```
<ConversationTransitions>
  <Transition>
    <SourceInteraction href="Start"/>
    <DestinationInteraction href="Login"/>
  </Transition>
  <Transition>
    <SourceInteraction href="Start"/>
    <DestinationInteraction href="Registration"/>
  </Transition>
  <Transition>
    <SourceInteraction href="Registration"/>
    <DestinationInteraction href="Login"/>
  </Transition>
  ...
```

76

## Ejemplo: Transiciones (2)

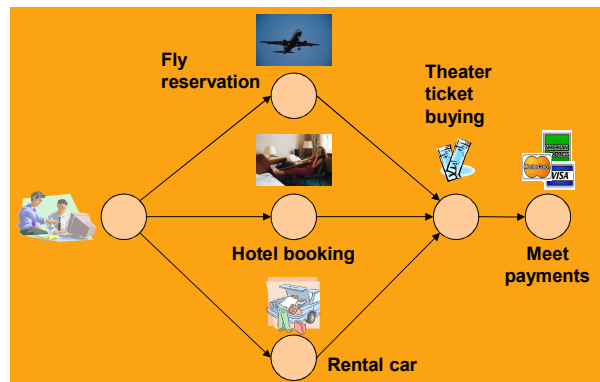
```
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Registration"/>
  <SourceInteractionCondition href="InvalidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Login"/>
  <SourceInteractionCondition href="InvalidLoginRS"/>
</Transition>
...
</ConversationTransitions>
```

77

## Orquestación

- Define la **secuencia** y las **condiciones** para que un servicio pueda invocar otros servicios para realizar alguna función útil

□ WS-BPEL



78

# BPML

- *Business Process Modeling Language:*
  - Desarrollado por BPMI.org (Intalio, Sterling, Sun, CSC)
  - Meta-lenguaje para **describir los procesos de empresa**
  - Lenguaje ejecutado por un sistema BPMS
- Características principales:
  - Actividades de base para enviar, recibir y llamar servicios
  - **Actividades opcionales, secuenciales y paralelas**
  - Composición, **correlación de servicios**
  - **Transacciones largas**
  - Mecanismos de **manipulación de ejecución**

79

# BPEL4WS

- *Business Process Execution Language for Web Services* [IBM, Microsoft, DEA]:
  - **Gramática de XML** describiendo la lógica para la coordinación de servicios, el flujo de control
  - Para ser interpretado y ejecutado por un **motor de orquestación**
- Basado sobre WSDL:
  - Cada **proceso** es presentado **como un servicio Web** utilizando WSDL
  - Los tipos de WSDL son empleados para describir la **información persistente**
  - Las **referencias de WSDL** indican las **llamadas a los servicios** externos
- Las características de orquestación:
  - **Procesos ejecutables** modelan un *workflow* privado y ejecutable
  - **Procesos abstractos** indican un intercambio público de mensajes

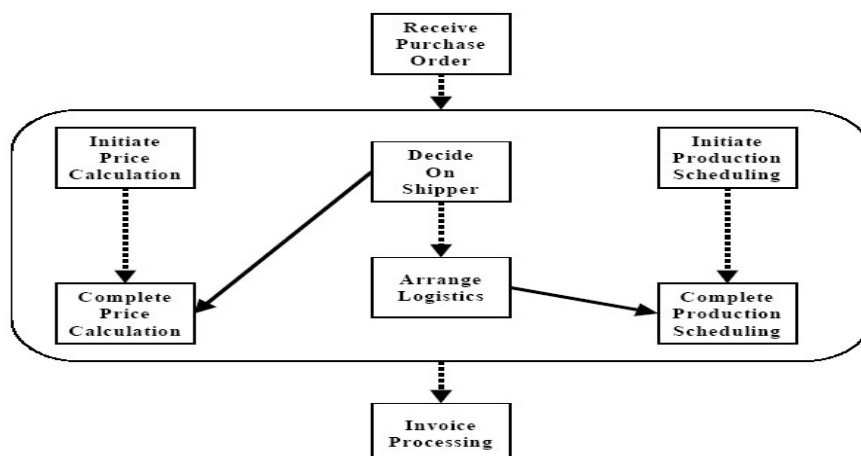
80



## Conceptos principales

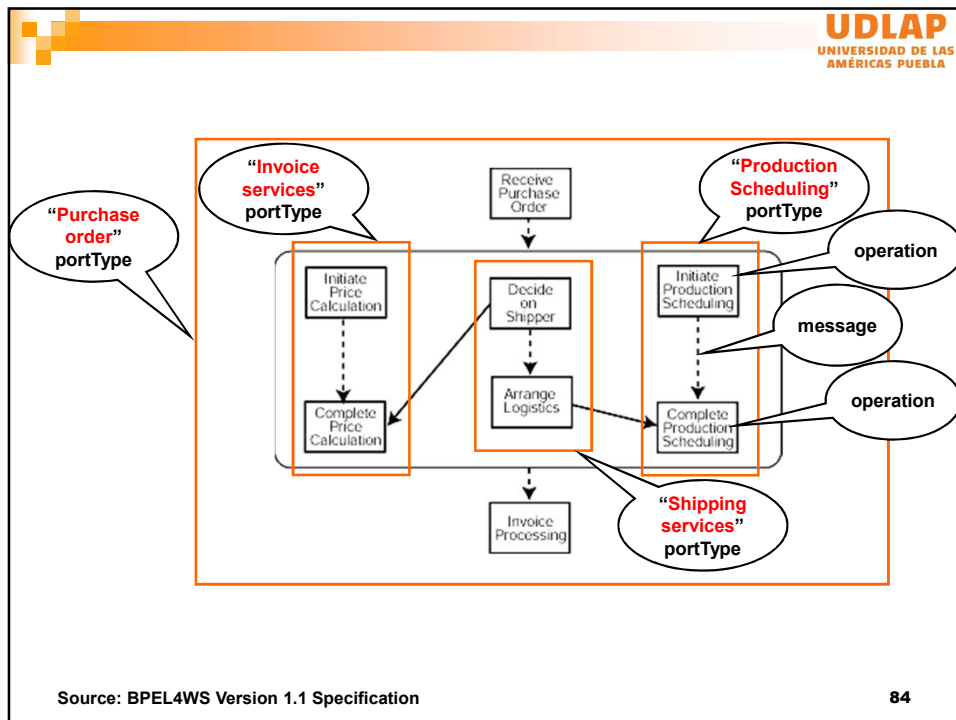
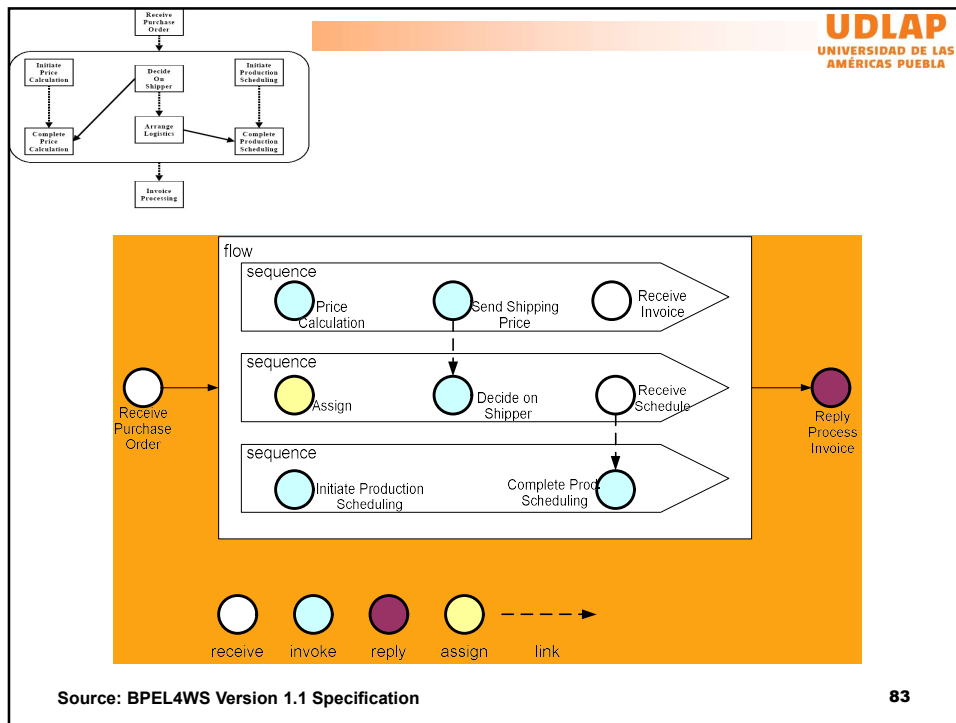
- Interacciones de **larga duración**
- Intercambio de **mensajes síncronos/asíncronos** par a par:
  - Solicitud-respuesta
  - Una sola vía
- **Acceso** a las variables y a los datos
- **Comportamiento** dependiente de los datos (construcción condicional y *timeout*)
- Manipulación y re-estabilización en presencia de **excepciones**
- **Compensación** (inversión de las operaciones)
- **Correlaciones** entre los roles de las contrapartes y de los mensajes producidos

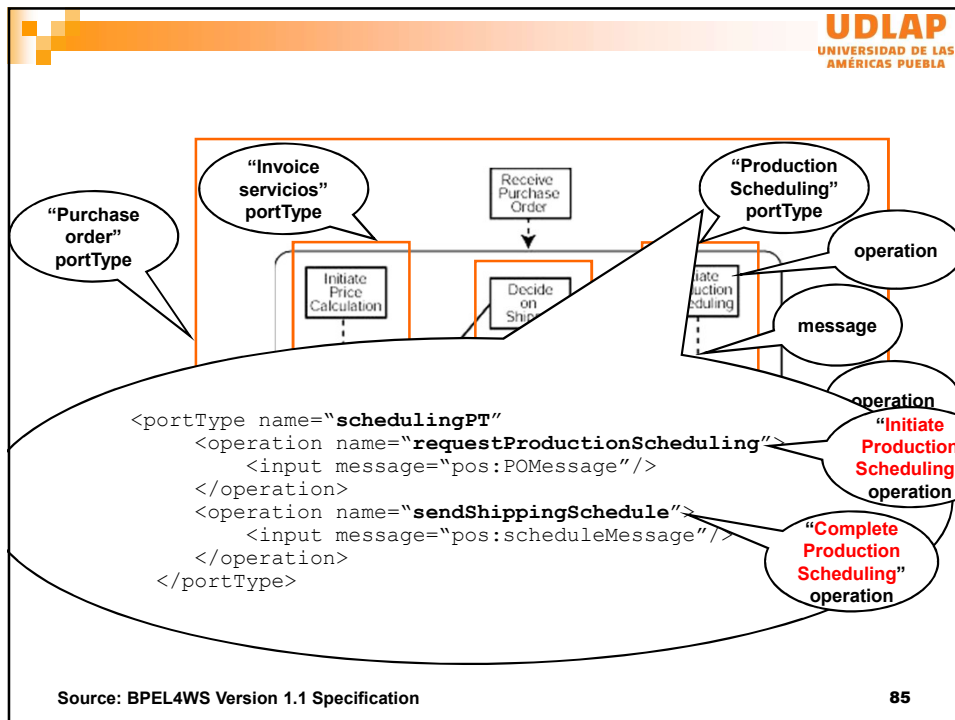
81



Source: BPEL4WS Version 1.1 Specification

82





UDLAP  
UNIVERSIDAD DE LAS AMÉRICAS PUEBLA

## Plan

- ✓ Programación orientada a servicios
- ✓ Servicio:
  - Definición
  - Llamados SOAP y REST
  - Construcción
- ✓ Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- Aspectos no funcionales
- Integración de servicios de datos: Mashups

86

## Aplicaciones basada en servicios

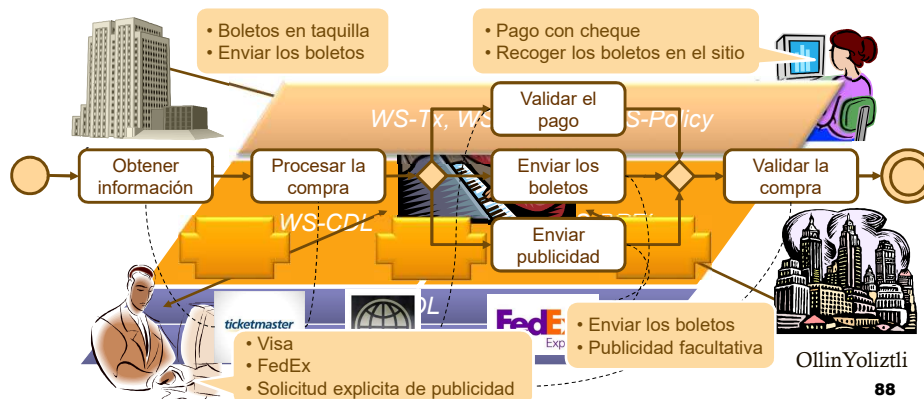
- **Servicio:** Componente de software que exporta un API



87

## Aplicaciones basada en servicios

- **Servicio:** Componente de software que exporta un API
- **Coordinación:** Actividades, flujo, reglas de gestión



88

## Aplicaciones: Resumen



89

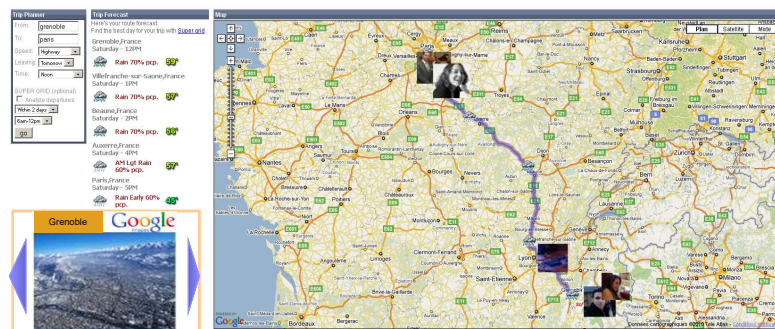
## Plan

- ✓ Programación orientada a servicios
- ✓ Servicio:
  - Definición
  - Llamados SOAP y REST
  - Construcción
- ✓ Construcción de aplicaciones a base de servicios:
  - Arquitectura SOA
  - Composición
- ✓ Aspectos no funcionales
  - Integración de servicios de datos: Mashups

90

## Mashup: Planificador de itinerario

- Dada una ciudad de salida y otra de llegada, dar la **ruta**, el **clima**, los **contactos facebook** y (usando *Google Images Search*) las **imágenes** de las ciudades intermedias



91

## Mashup: Conceptos principales

- **Mashlet:**
  - Es un contenedor reutilizable y atómico que llama a un proveedor de datos
  - Presenta los datos recuperados (por ejemplo, una página Web)
  - Proveedores de datos:
    - *Web scrapping*
    - *Feeds*
    - Servicios Web
- **Mashup**

92

# Mashup: Conceptos principales

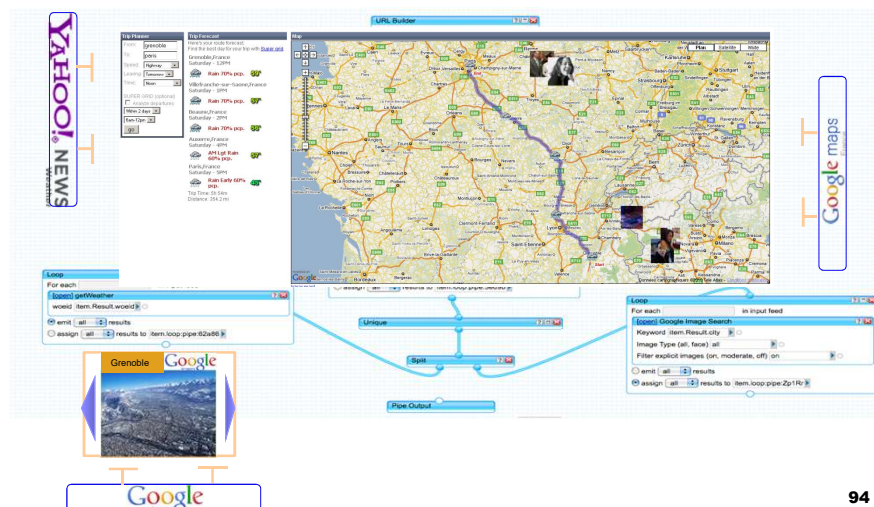
## ■ Mashlet

## ■ Mashup:

- Es una aplicación que agrega, integra, administra y visualiza datos recuperados a partir de varios proveedores
- Ejemplos:
  - Yahoo! Pipes (flujo de datos)
  - MS Montage (organización de datos espaciales)

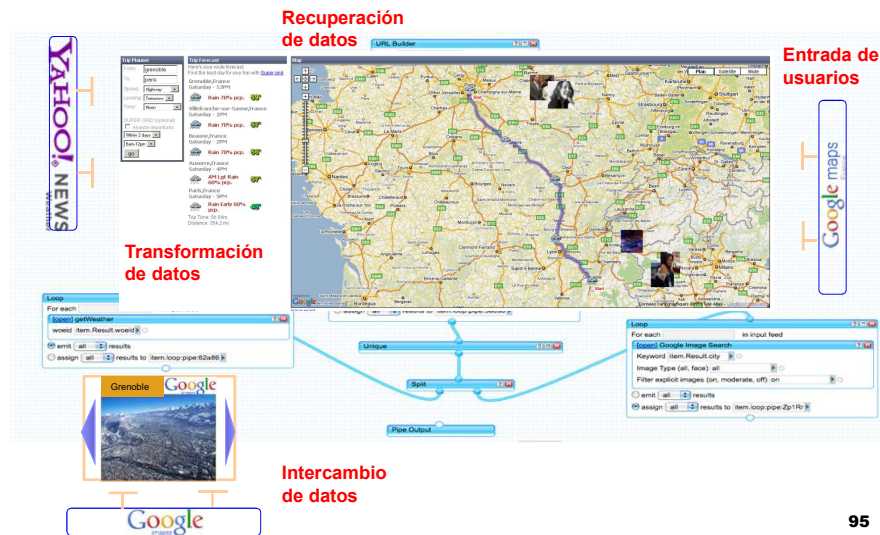
93

# Mashup: Funcionamiento



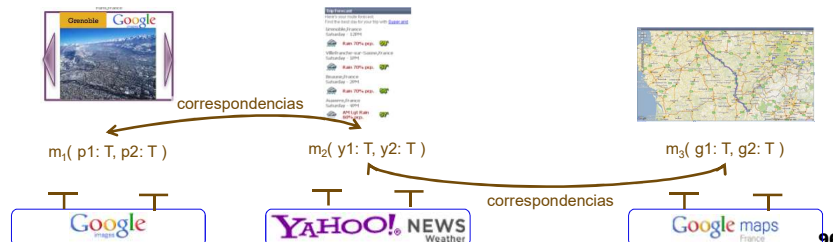
94

## Mashup: Funcionamiento



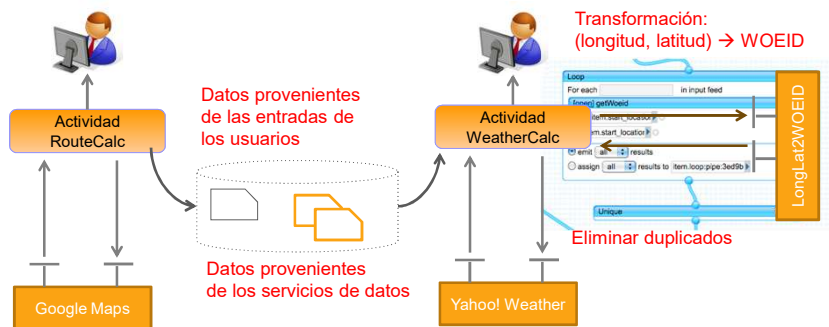
## Mashup: Modelo de datos

- Los datos se puede describir usando **metadatos**: Posición (longitud, latitud) → WOEID (*where on earth identifier*), nombre de la ciudad, etc.
- Los datos son intercambiados por mashlets (**modelo pivote**), los cuales establecen las correspondencias entre los parámetros de los métodos de cada servicio





## Mashups: Gestión de datos (1)



97

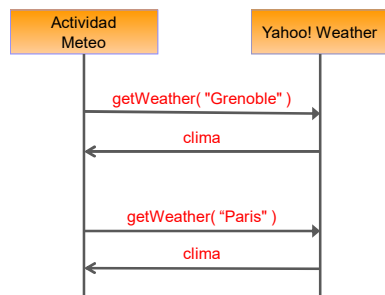
## Mashups: Gestión de datos (2)

- Frecuencia (periódica o no) de **recuperación de datos**
- Los datos recuperados deben ser **almacenados o no**
- Gestión de la **persistencia**
- Gestión de la **seguridad**
- Gestión de **excepciones**
- **QoS**: Frescura, origen, disponibilidad, retraso de servicio
- **Intercambio de datos**: Datos obtenidos de un servicio o de las entradas del usuario
- Coordinación del **acceso** a los recursos
- **Tiempo de vida** de los datos
- ¿Qué datos se **comparten**? Todos o un subconjunto
- ¿Qué actividades puede **leer/escribir** datos?

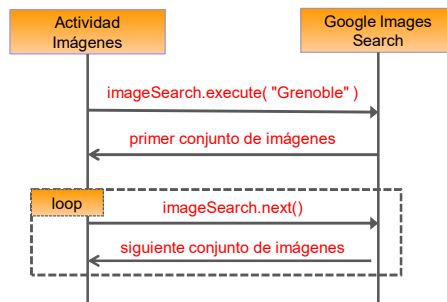
98

## Recuperación de datos

- El resultado es entregado sólo **una vez**:



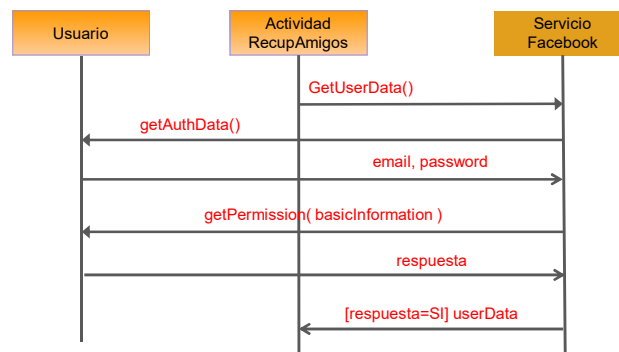
- La entrega se lleva a cabo de forma **iterativa**:



99

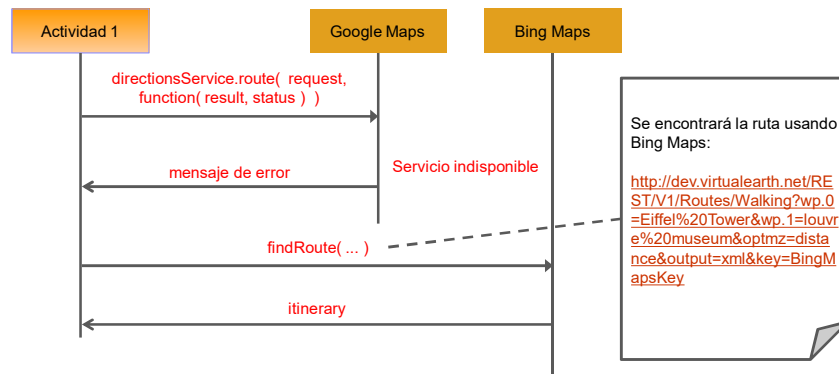
## Gestión de la seguridad

- Protocolo de **autenticación**: Protocolo OAuth 2.0



100

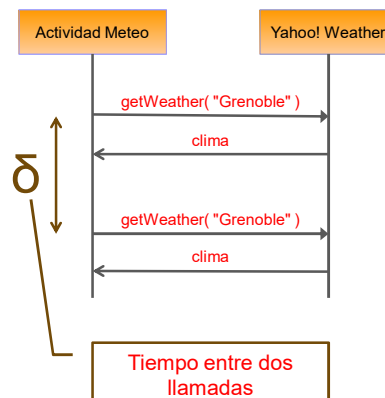
## Gestión de excepciones



101

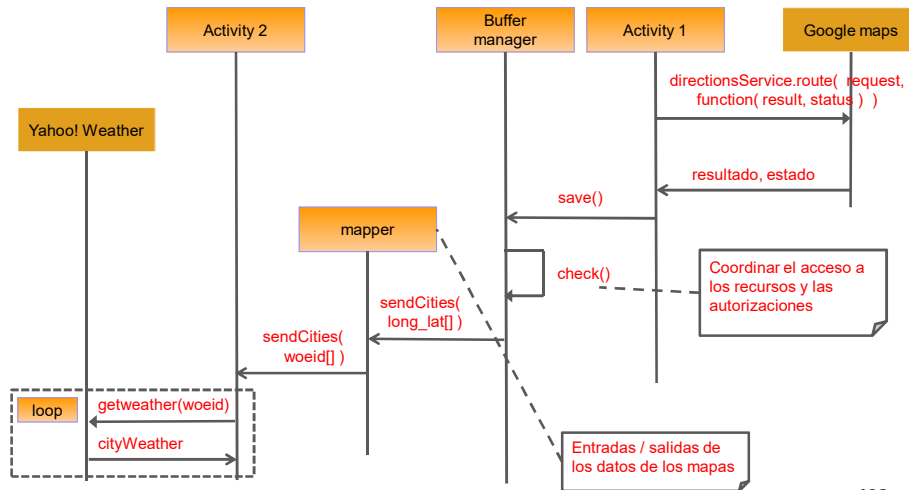
## Frescura de los datos

- $\delta$  es **definida** a través de:
  - Un servicio
  - Un directorio
- $\delta$  es **calculada** por un servicio externo
- $\delta$  **no siempre** está definida



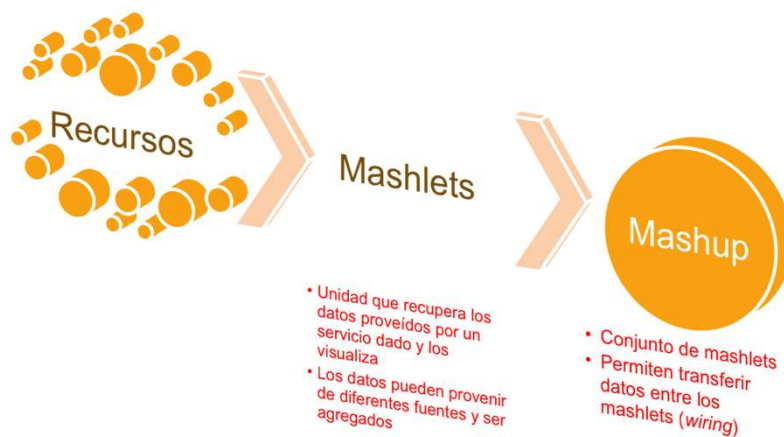
102

## Intercambio de datos



103

## Mashup: Construcción



104

## Mashups: Lógica aplicada (1)

- Un mashup es una aplicación que tiene una lógica:
  - La **lógica de la aplicación** está programada por la coordinación de los servicios de datos
  - La coordinación describe el **orden** en el que se recuperan los datos de los servicios además de otras **operaciones** que se pueden realizar sobre los datos recuperados
- En el ejemplo, se tiene la siguiente coordinación:
  - Primero se recuperan los **datos** del itinerario
  - Luego se **transforman** los datos: (largo, latitud) → WOEID
  - Enseguida se recuperan los **datos** meteorológicos

105

## Mashups: Lógica aplicada (2)

- Una coordinación de servicios está definida por **actividades**, un **flujo de control** y un **flujo de datos**:
  - El control de flujo representa el **orden** en el que las actividades se llevan a cabo (en secuencia, en paralelo)
  - El flujo de datos representa la posibilidad de **compartir o intercambiar** datos entre las actividades

106

## Mashups: Lógica aplicada (3)

- Una actividad es un programa que realiza:
  - La **llamada** a un método (exportado por un servicio) para recuperar datos
    - Por ejemplo, `directionsService.route( request, function(result, status) )`
  - El **tratamiento** de estos datos: Organización, almacenamiento, presentación, ...
    - Agregación, filtrado, combinación, proyección
    - Por ejemplo, la transformación de (largo, latitud) → WOEID
  - La **transmisión** de datos hacia otros servicios
    - Por ejemplo, enviar los datos geográficos a Yahoo! Weather

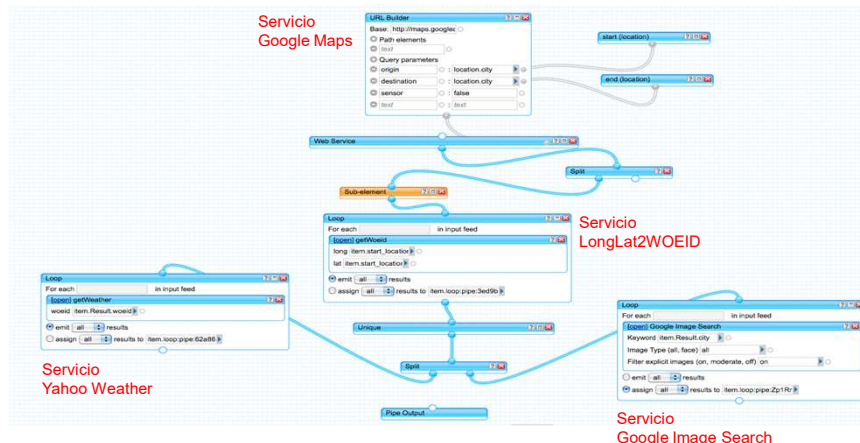
107

## Mashups: Lógica aplicada (4)

- Los ambientes de construcción ofrecen **operadores** para manipular los datos dentro de la lógica de una aplicación
- Por ejemplo, **Yahoo! Pipes** ofrece entre otras operaciones:
  - Count, filter, loop, sort, trunc
  - Expresiones regulares
  - Manipulación de cadenas

108

## Mashups: Lógica aplicada (5)



109



## Yahoo! Pipes (1)

- Permite la creación de mashups:
  - A través de un **editor visual**
  - Usando **diferentes fuentes** tales como *feeds*, páginas Web, servicios
  - Ofreciendo **operadores** para agregar, filtrar, transformar, enriquecer
  - Publicando los **resultados** como RSS *feed* en iGoogle, Yahoo! Dashboard

➤ <http://pipes.yahoo.com/pipes/>

➤ Damia de IBM está construido usando Yahoo! Pipes

110



## Yahoo! Pipes (2)

- **Fuentes de datos:** Servicios REST, módulos específicos (Flicker, base Google, ...)
- Yahoo! Pipes transforma el formato de la fuente (RSS, ATOM, RDF) a su **modelo de datos internos** RSS
- **Modificación de la estructura** de los esquemas de datos: *Regex, rename, sub-element, union*
- **Manipulación de datos:** *Reverse, sort, truncate, tail, count, filter, unique*, operadores de cadenas, operadores matemáticos, operaciones sobre fechas

111

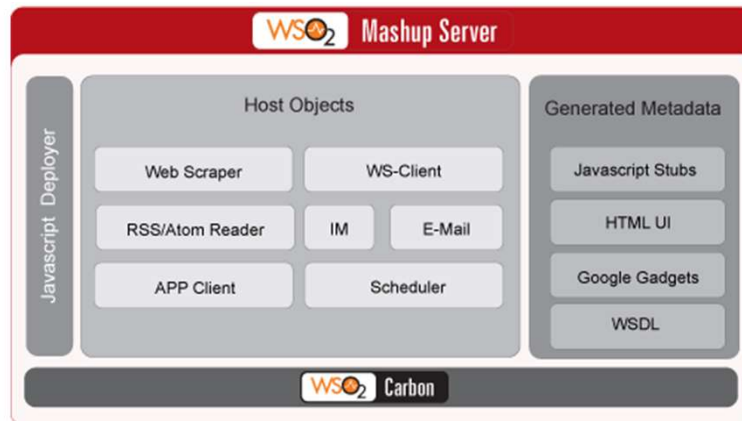
## Servidor de mashup WSO<sub>2</sub> (1)

- El servidor WSO<sub>2</sub> permite crear mashups:
  - Basados en **servicios** Web
  - Los **mashups** son expuestos como servicios Web que pueden ser utilizados por otros mashups o servicios Web
  - Los servicios son generados por el servidor usando el lenguaje **Javascript**
- <http://wso2.com/products/mashup-server/>

112



## Servidor de mashup WSO<sub>2</sub> (2)



113



114