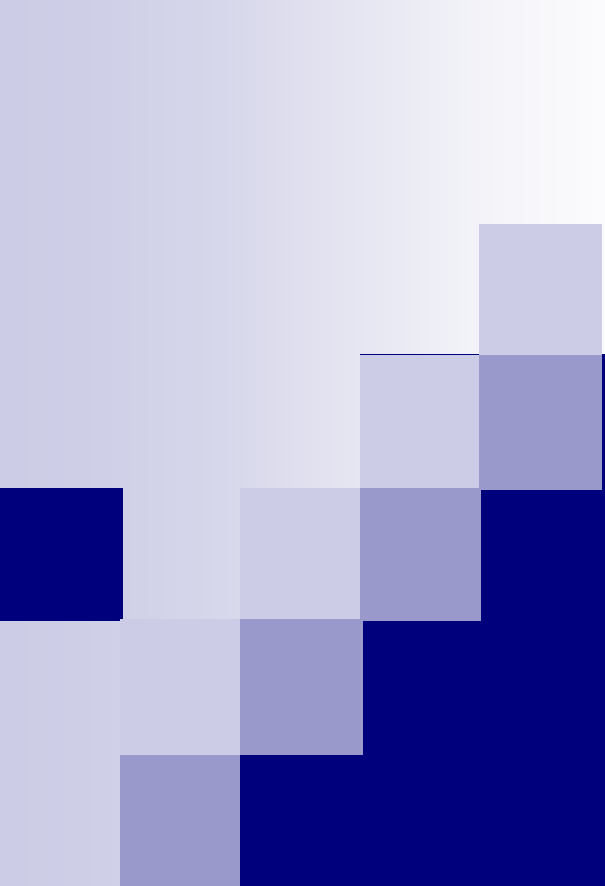




Web Services en Java

Taller de Programación

Instituto de Computación – Facultad de Ingeniería
Universidad de la República



Contenido

- Motivación y Conceptos
- Funcionamiento
- Annotations
- Desarrollando una aplicación con Web Services
- Caso de Estudio
- Demo



Contenido

- **Motivación y Conceptos**
- Funcionamiento
- Annotations
- Desarrollando una aplicación con Web Services
- Caso de Estudio
- Demo



Motivación (I)

- Necesidad

- ☐ Distribuir aplicaciones

- Técnicas

- ☐ Sockets
 - ☐ Mensajería
 - ☐ RMI
 - ☐ Web Services



Motivación (II)

- Que provéen los Web Services?
 - Desarrollar aplicaciones distribuidas
 - Permite realizar solicitudes de procesamiento remoto mediante un protocolo estandar
 - La utilización de librerías permite ocultar dificultades en el manejo del protocolo
 - Marco para comunicar aplicaciones “heterogeneas”



Conceptos

■ Cliente

- ☐ Es la entidad que desea consumir un servicio

■ Proxy

- ☐ Es quien le brinda al cliente facilidades para acceder a un servicio.
- ☐ Encapsula la implementación de dicho consumo.

■ Servidor

- ☐ Es la entidad que brinda la infraestructura para publicar un servicio y consumirlo

■ Server Stub

- ☐ Es quien implementa la lógica del servicio
- ☐ Existe en el entorno del servidor



Conceptos (II)

■ Registro

- Permite buscar servicios a los clientes que lo necesiten
- El proveedor de un servicio publica allí la descripción del mismo

■ Descripción del servicio

- Define mediante un XML como es la estructura de un determinado servicio (WSDL)
- Declara tipos, mensajes y Ports en los que se basa el servicio que representa

■ Protocolo de comunicación

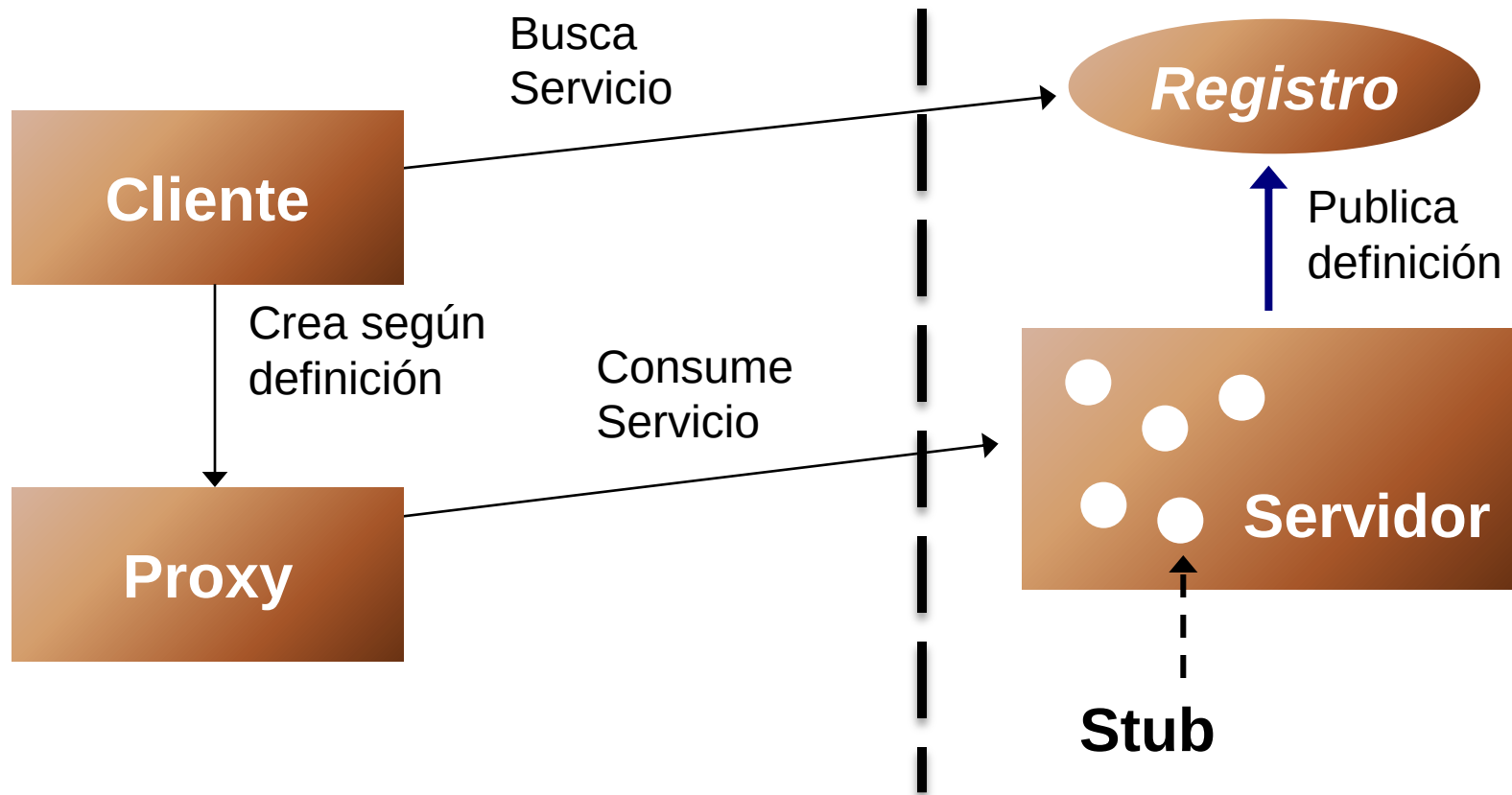
- SOAP
- Independiente de la plataforma y del lenguaje
- Soporta varios transportes (HTTP, SMTP, JMS y más)



Contenido

- Motivación y Conceptos
- **Funcionamiento**
- Annotations
- Desarrollando una aplicación con Web Services
- Caso de Estudio
- Demo

Funcionamiento General





Enfoque Top-Down

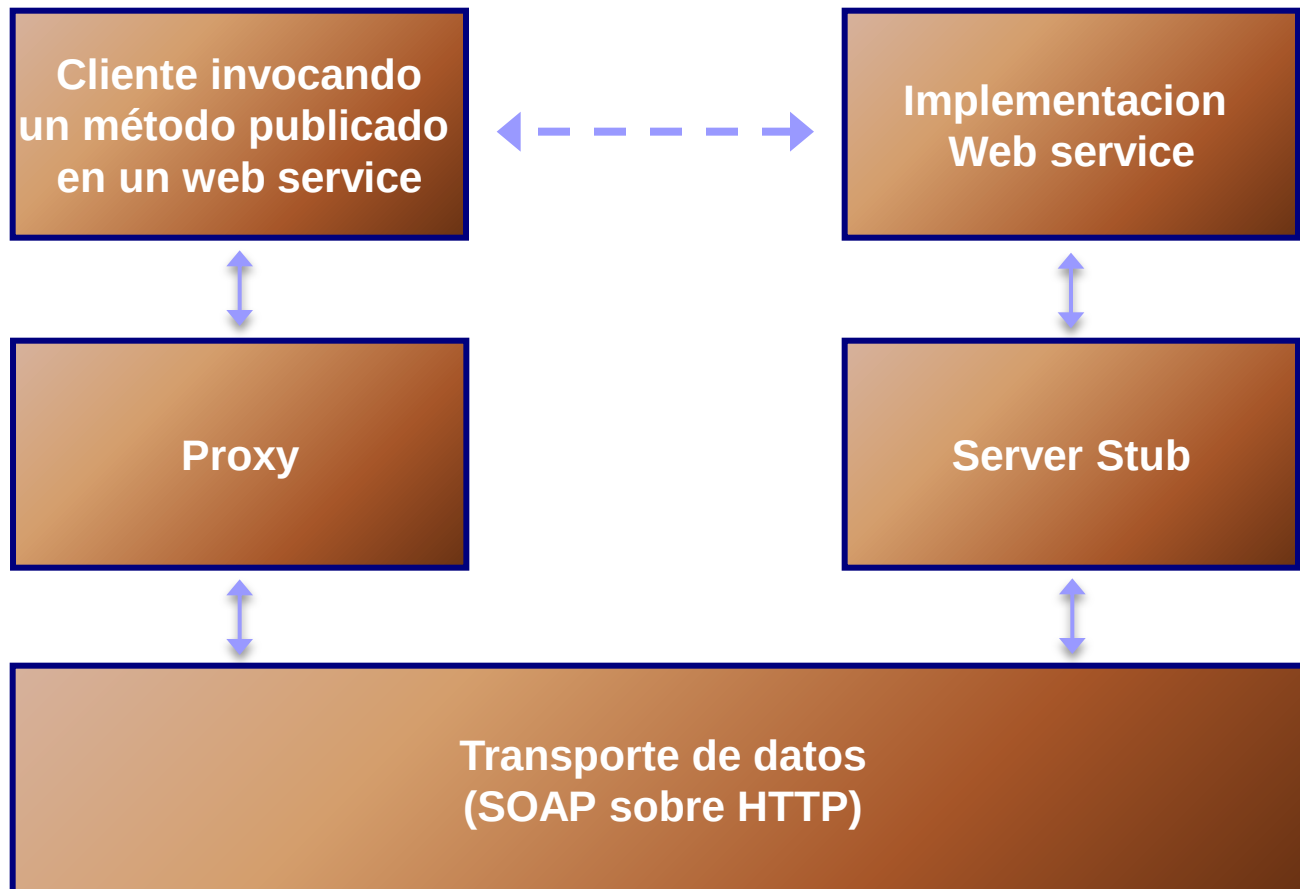
- Se define como debe ser la estructura del servicio
 - Declarar el WSDL del servicio
- Se generan los stubs y el esqueleto de las clases que implementan el servicio
 - `Wsd12Java tool`
 - Se escribe el código que implementa la lógica del servicio deseado



Enfoque Bottom-Up

- Se define la interface Java que se expondrá como servicio
 - Se “anota” la clase indicando que la misma se expondrá
 - Se “anotan” los métodos y parámetros de los métodos a exponer
- Se generan los stubs para el servidor y el WSDL correspondiente
 - Dichos procesos se realizan mediante comandos

Funcionamiento



Funcionamiento (II)

■ Proxy

- Se encarga de encapsular el acceso al canal
- Serializa los parametros transformandolos en xml
- Contruye objetos Java con el mensaje de respuesta
- Construye el mensaje SOAP a enviar al servidor
- Se pueden generar con la informacion que publica el WSDL
 - Comando **wsimport**

Funcionamiento (III)

■ Server Stub

- Realiza el “marshalling” parametros de entrada recibidos en el mensaje SOAP y serializan el mensaje de respuesta
- Generan la estructura necesaria para definir “Handler Chains”
- Se generan tambien automaticamente
 - Comando **apt**



Funcionamiento (IV)

■ Tipos válidos en Web Services

□ Tipos nativos de Java

- Se mapean casi directamente con los nativos definidos en SOAP

□ Datatypes

- Se define en el WSDL la estructura del tipo definido
- Debe ser un Java Bean
- Tipo “anotado” con directivas de serializacion

□ Excepciones

- Se mapean a los SOAP Faults
- También se serializan en xml

Funcionamiento (V)

- Comportamiento de los parámetros
 - Se “anotan” en la clase que define el servicio
 - In, out, inout
 - Dependiendo del “estilo” de binding se pueden definir servicios con varios parámetros de salida
 - Colecciones
 - Dependiendo del framework disponible se pueden definir como arrays estáticos o dinámicos



Contenido

- Motivación y Conceptos
- Funcionamiento
- **Annotations**
- Desarrollando una aplicación con Web Services
- Caso de Estudio
- Demo



Annotations

■ Que son?

- Metadata para el compilador, runtime o el programa en sí

■ Que fin tienen?

- Suprimir warnings o indicar polimorfismos
- Generar código o archivos de configuración
- Obtener información reflexiva de una clase
- Ejemplo: `@Override`, `@WebService`, `@ClassPreamble`

Annotations (II)

■ Annotations y Web Services

- Generan código, archivos de configuración, tipos auxiliares y automatizan su serialización
- En la jdk 6 hay que procesarlas a parte (pre compilación)

■ Cuales utilizar?

□ **@WebService**

A nivel de clase, indica que la misma debe ser expuesta como Web Service

□ **@WebMethod**

A nivel de método, indica que el método será incluido en la interfaz del servicio

Annotations (III)

■ Cuales utilizar? (cont)

☐ **@WebParam**

A nivel de parámetro, indica el nombre y tipo que tomará dicho parametro en el servicio.

☐ **@SOAPBinding**

Indica el estilo de codificación SOAP que se usará para el servicio

☐ **@XmlAccessorType**

Define el modo en que se serializan los tipos definidos en XML



Contenido

- Motivación y Conceptos
- Funcionamiento
- Annotations
- **Desarrollando una aplicación con Web Services**
- Caso de Estudio
- Demo



Desarrollando una Aplicación con Web Services

- Definir que componentes se van a comunicar remotamente
- Definir como va a ser dicha comunicación
 - Uni-Direccional, Bi-Direccional
- Definir qué métodos deben accederse remotamente



Comunicaciones Uni-Direccionales

■ Cliente - Servidor

- ☐ Definir la clase (o interfaz) anotada que correra en el servidor
- ☐ Generar Stubs del servidor
- ☐ Definir la direccion en la que se publicara el servicio
- ☐ Escribir un cliente que usa el servidor
- ☐ Generar los proxys necesarios
- ☐ Ejecutar el servidor y el cliente



Comunicaciones Bi-Direccionales (I)

■ Cliente – Servidor

- ☐ Un servidor realiza tareas solicitadas por un cliente y las mismas repercuten directamente en todos los clientes

■ Componentes Colaborativos

- ☐ Existen procesos que se desarrollan en varios componentes de una arquitectura distribuida
- ☐ Los procesos definen interacciones entre los componentes
- ☐ Los componentes se “reparten” etapas del proceso
- ☐ Necesitan comunicar resultados de las etapas a los otros componentes



Comunicaciones Bi-Direccionales (II)

■ Alternativas

□ Varios servidores

- Cada componente registra un servidor y define una interfaz de uso
- Cada componente tiene que conocer la dirección de registro de cada otro componente
- Cada componente tiene que conocer el proceso general para decidir como continuarlo

Comunicaciones Bi-Direccionales (III)

■ Alternativas (Cont.)

□ Servidor de Procesos único

- Basado en el patrón Observer
- “Orquesta” la interacción, conoce la estructura del proceso
 - Asocia interacciones a las etapas del proceso general
- Otros componentes se “registran” a etapas del proceso
 - Publican un servicio “cliente” y lo registran en el servidor para una determinada interacción.
 - El servidor construye dinámicamente el proxy necesario
 - Cuando se consume el servicio “cliente”, este inicia un CU en el componente que lo registró
- El acoplamiento entre los componentes controlado
 - No existen dependencias estáticas entre los componentes participantes de la interacción
 - Event-based Architectures and Loose-coupled Integration



Contenido

- Motivación y Conceptos
- Funcionamiento
- Annotations
- Desarrollando una aplicación con Web Services
- **Caso de Estudio**
- Demo

Caso de Estudio

■ Aplicación servidor

- Compuesta por una clase “Main” y expone 6 servicios en 4 Web Services
 - Diferentes tipos en los parámetros y en las respuestas
 - Cada servicio tiene una URL propia
 - No hay necesidad de levantar “hilos”
- Servidor notifica clientes
 - Envía mensajes asincrónicamente
 - Consume un servicio que los clientes publican
 - Previamente deben registrarse



Caso de Estudio

■ Aplicación cliente

- Cliente Swing que consume los servicios

- Publica un servicio de notificación

 - Al iniciar se registra en la aplicación servidor para ser notificada

 - Despliega un mensaje con la notificación

■ Se implementa de este modo una comunicación bi-direccional

Generando el Servicio

- “apt -d pathA pathB/MiClase.java”
 - MiClase debe presentar las annotations que definen el servicio y sus metodos
 - pathA es el camino para ubicar las clases auxiliares para la publicación del servicio
- Luego se compila el código de toda la aplicación servidor

Generando *Proxys*

- “wsimport -p pkg -keep
http://srvName:8080/serviceName?wsdl”
 - Se ejecuta en el cliente
 - Si no cambia la el servicio alcanza hacerlo una única vez
 - La dirección a ingresar es la correspondiente a donde se publicó el servicio
 - pkg es el package en donde quedan generados los proxy types necesarios
 - -keep indica que se quieren mantener los fuentes de los proxy types



Contenido

- Motivación y Conceptos
- Funcionamiento
- Annotations
- Desarrollando una aplicación con Web Services
- Caso de Estudio
- **Demo**

Recomendaciones

- Tener cuidado de NO dejar puertos ocupados al correr las aplicaciones que publican servicios
 - ps
 - kill
 - netstat -el

Referencias

- Web Services Overview

- <http://java.sun.com/webservices/>

- Tutorial de Sun

- <http://java.sun.com/webservices/tutorial.html>

- Links de interés

- http://weblogs.java.net/blog/vivekp/archive/2006/12/webservices_in.html
 - <http://today.java.net/pub/a/today/2007/07/03/jax-ws-web-services-without-ee-containers.html>
 - <http://www.javaworld.com/javaworld/jw-07-2006/jw-0703-mustang.html>

Referencias

■ Links de interés (cont.)

- http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
- http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search_by=Understanding+Web+%20Services+specifications+Part
- <https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html>
- <http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html>

■ Tools

- www.soapui.org/

■ IDE Plugins

- No necesarios en realidad