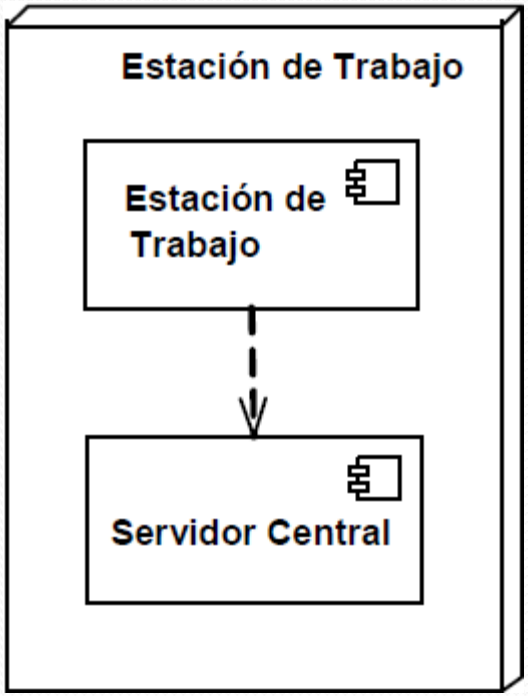
Web Services en Java

Contenido

- Motivación
- Cuestiones técnicas
  - Definiciones previas - Contexto
  - Funcionamiento
- Cuestiones prácticas
  - Annotations
  - Tipos válidos
  - Escenarios de importancia

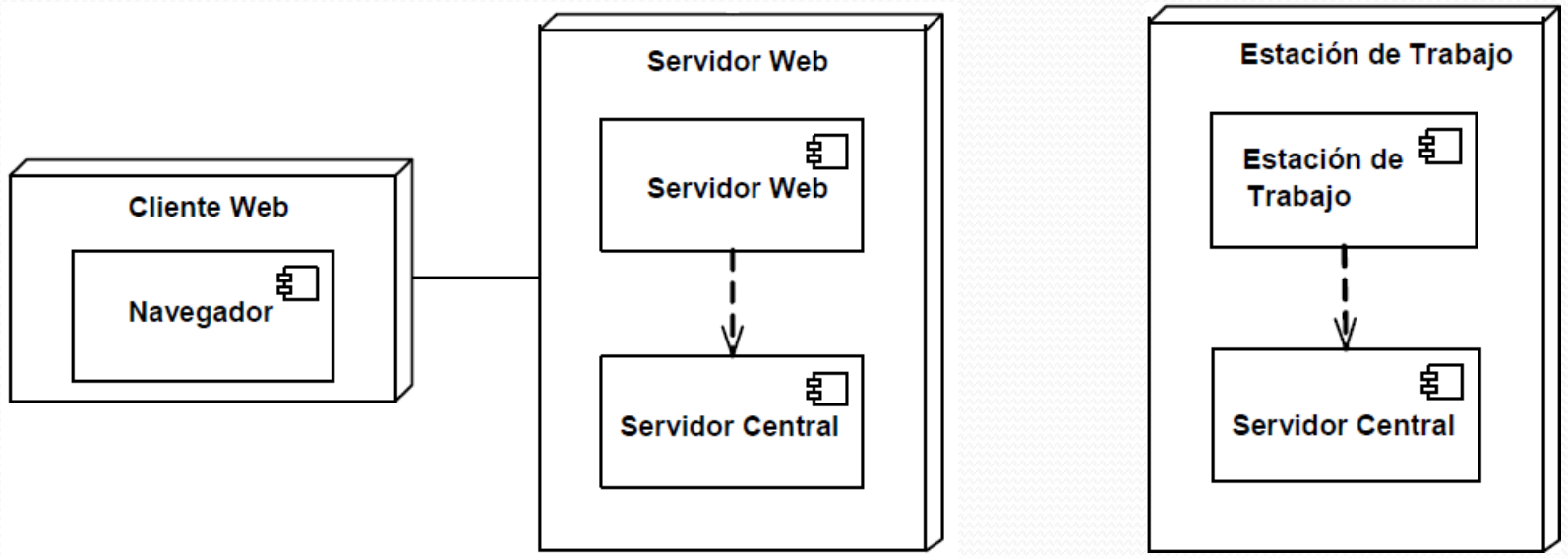
Motivación

- Diagrama de distribución – Iteración 1



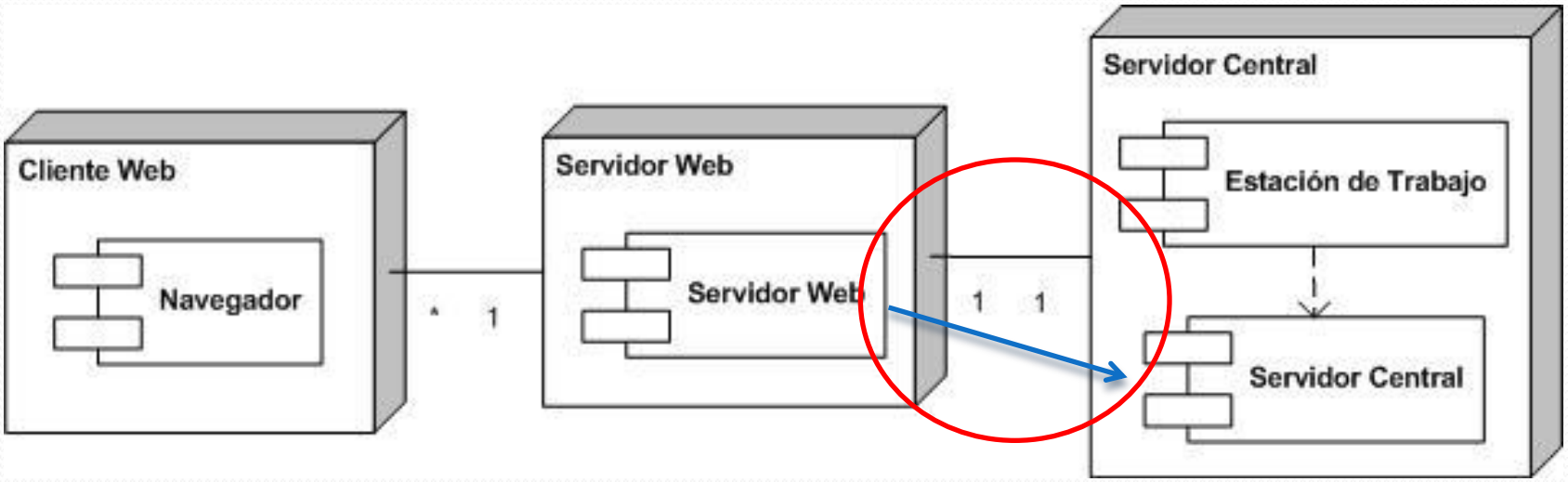
Motivación

- Diagrama de distribución – Iteración 2



Motivación

- Diagrama de distribución – Iteración 3



Middleware

Cuestiones técnicas - Contexto

- ¿Qué es el Middleware?
  - Es el "pegamento" (glue) que ayuda a la conexión entre programas (o bases de datos).
  - Más formalmente:
    - Es el soft-sistema que permite las interacciones a nivel de aplicación entre programas en un ambiente **distribuido**.
    - Por soft-sistema (system software) se entiende el software posicionado entre una aplicación y un sistema de menor nivel (S.Op, DBMS, Servicio Red).
    - Un ambiente computacional se dice distribuido cuando sus programas o BDs están ubicados en dos o más computadores.

¿Para qué usar Middleware?

- Dadas dos aplicaciones que se quieren conectar, se usa para resolver la comunicación entre los procesos.
  - Si no hay middleware se complica el desarrollo de aplicaciones:
  - Se deberían programar módulos de bajo nivel.

- El soft de middleware permite realizar esta conexión a través de **interfaces de alto nivel**, que permiten, por ejemplo, ver un procedimiento remoto como si fuera local.

Comunicación remota

- Comunican 2 sistemas:
  - Sockets
  - Mensajería
    - Mecanismo Asincrónicos
  - Remote Procedure Call (RPC, RMI, etc.).
    - Invocación a procedimientos remotos como si fueran locales al programa.
  - Web Services (SOAP, REST).
    - Invocación a procedimientos a través de HTTP.

Web Services

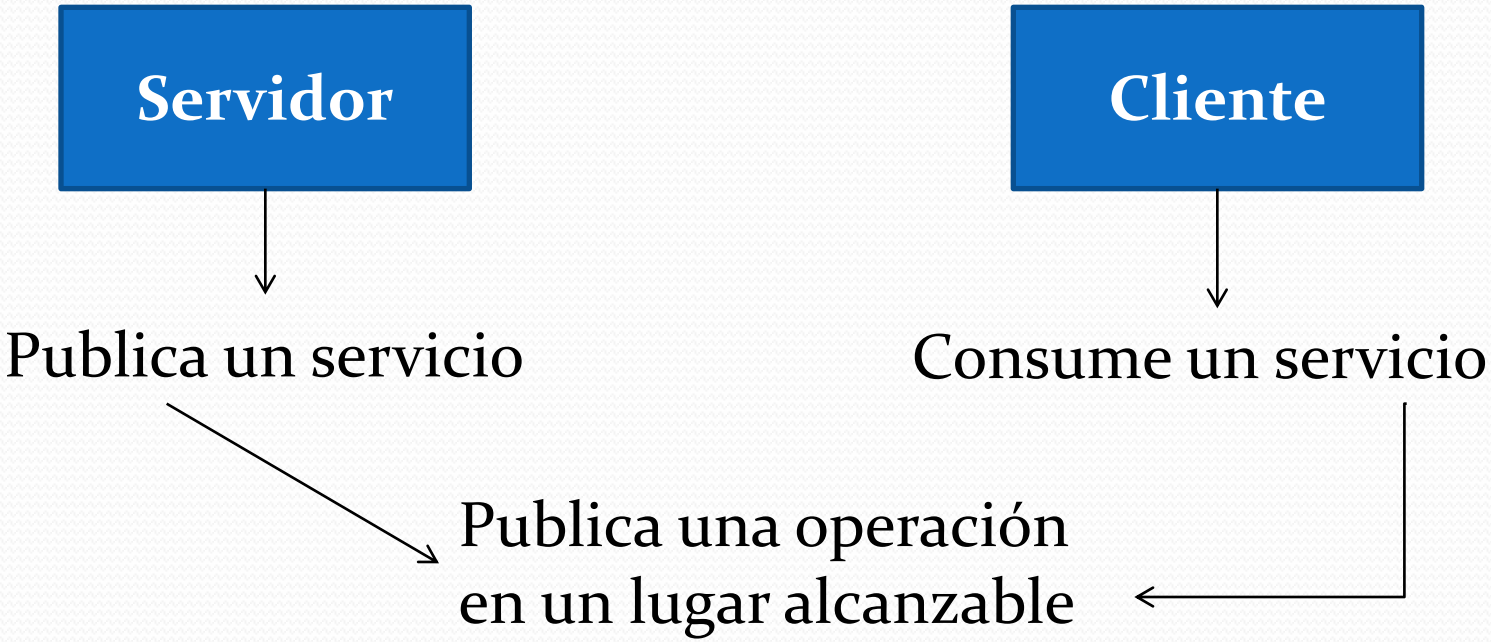
- El objetivo principal de los Web Services es proveer una forma sencilla de exponer funcionalidades de una forma controlada y estándar, para que otra aplicación pueda consumirlas.
- Proveen:
  - Desarrollar aplicaciones distribuidas
  - Permite realizar solicitudes de procesamiento remoto mediante un protocolo estándar.
  - Más…

Web Services – 2 opciones

- REST
  - Orientado a recursos
  - Enfoque en escalabilidad y *performance*
- SOAP
  - Orientado a operaciones
  - Enfoque en interoperabilidad.
- VS.
  - Ninguno es mejor que el otro a priori
  - SOAP más apropiado para integración de sistemas heterogéneos con requerimientos empresariales
  - REST está orientado a aplicaciones Web con gran cantidad de clientes y desconocidos

Web Services – Funcionamiento

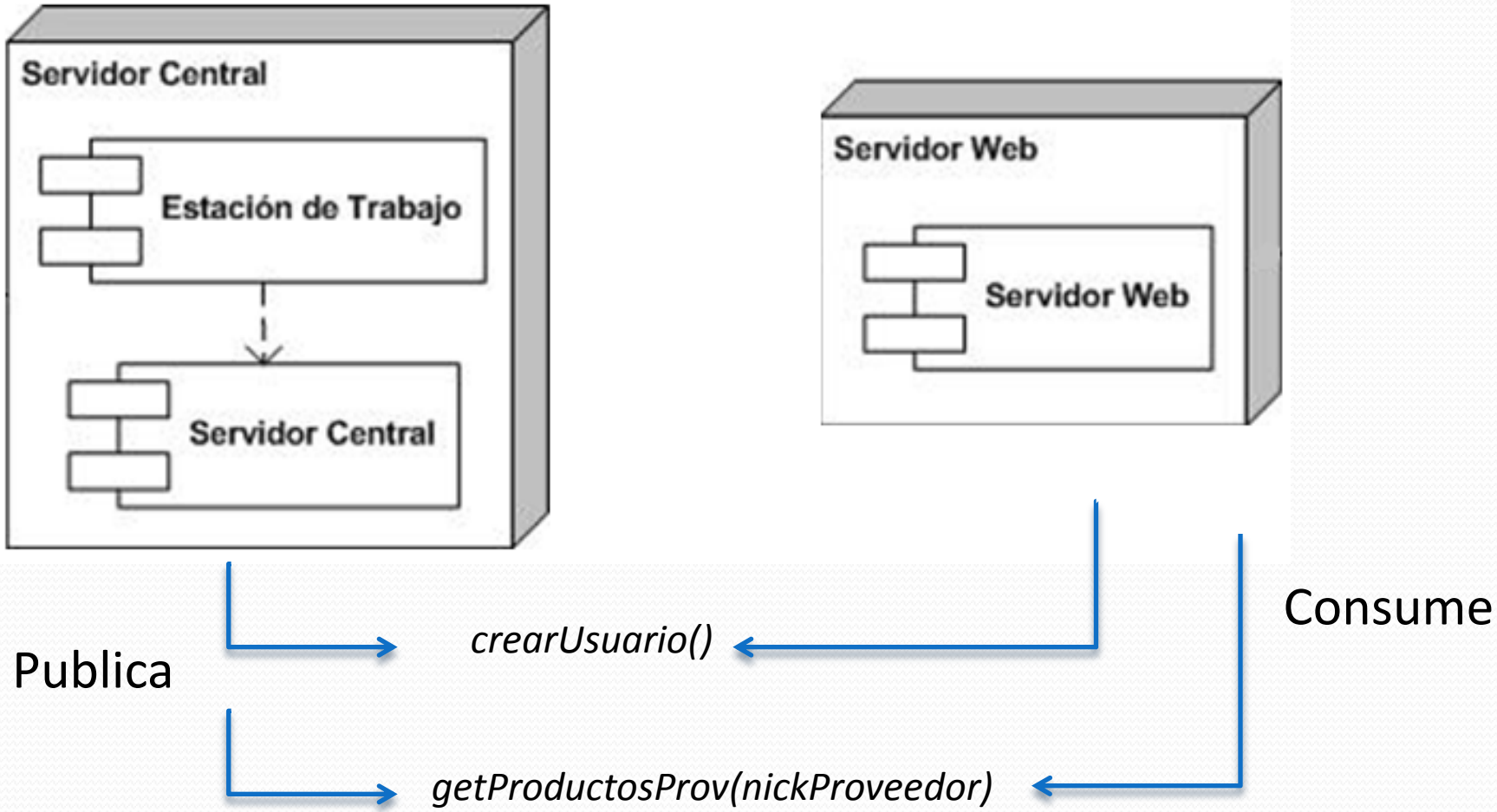
- Comencemos a crear un Web Service…
  - ¿Qué necesitamos?



Paréntesis

- Hagamos un mapeo a nuestra realidad…
  - Tenemos un componente **Servidor WEB** que quiere consumir una **Lógica** (Servidor Central) que ya no es alcanzable de manera local como antes.
  - Se propuso WS para lograr comunicarlos de manera remota.
  - Entonces, la idea sería que nuestro Servidor Central publique sus operaciones, y que el Servidor Web las consuma.
  - Nuestro Servidor Web sería el Cliente.

Paréntesis (2)



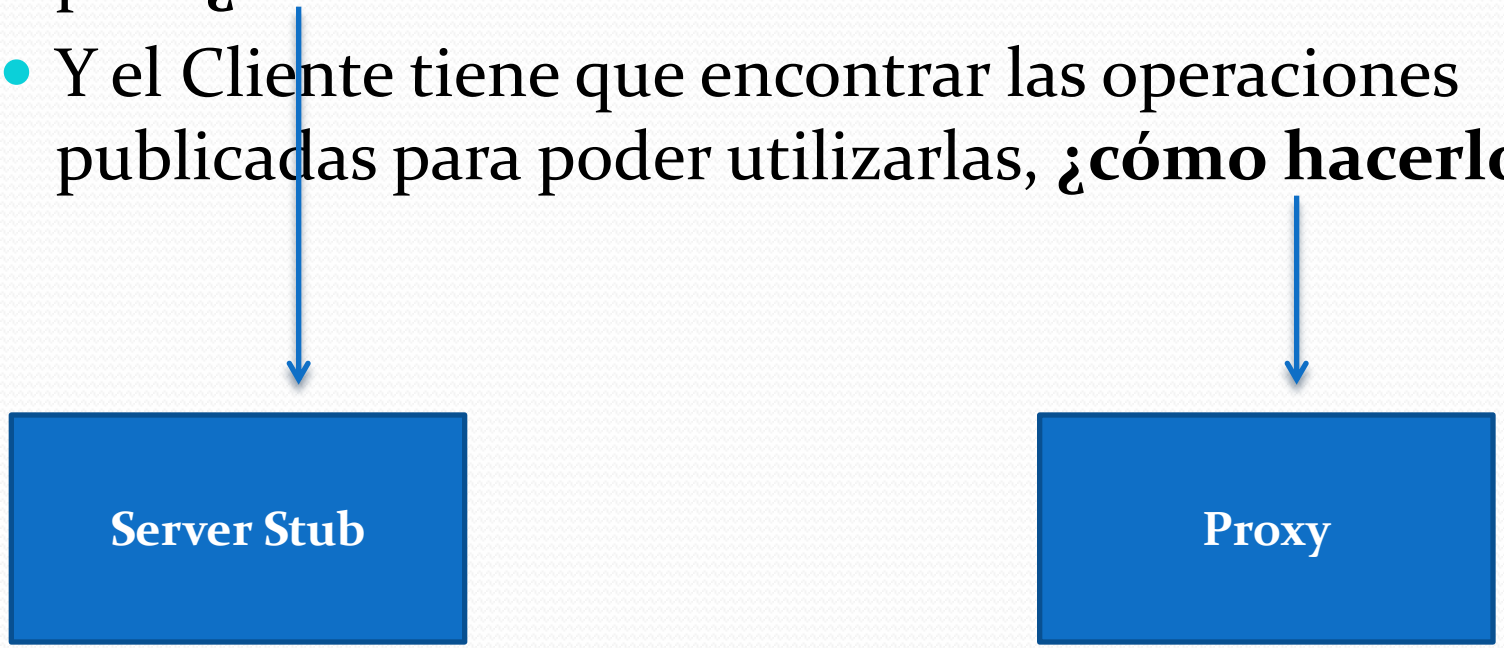
WS - Formalmente

- Cliente
  - Es la entidad que desea consumir un servicio
- Servidor
  - Es la entidad que brinda la infraestructura para publicar un servicio y consumirlo

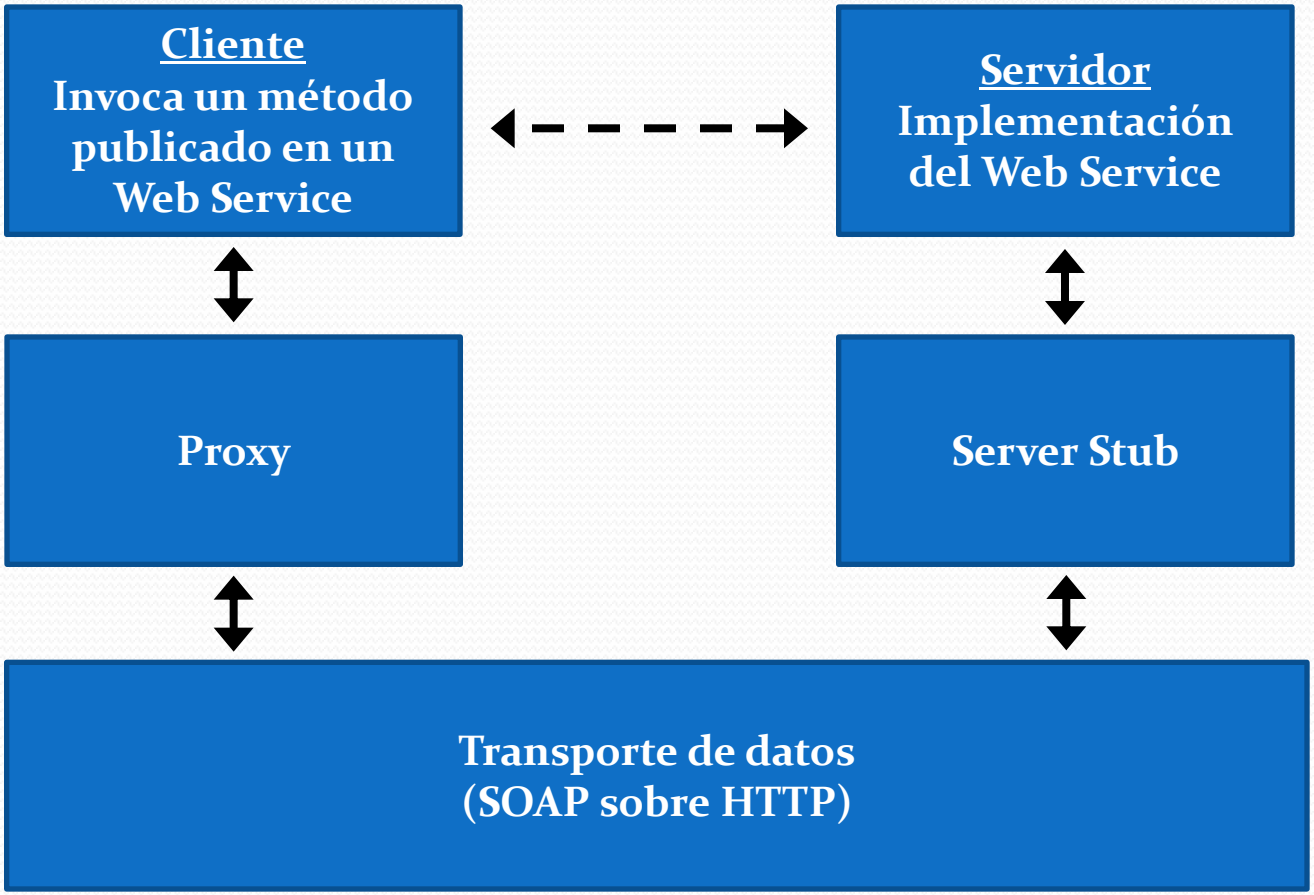
- A tener en cuenta…
  - El servidor tiene que publicar sus operaciones en un lugar alcanzable, y el Cliente tiene que saber el lugar exacto para encontrar las operaciones.

Web Services

- Por lo tanto…
- El Servidor tiene que publicar sus funcionalidades, pero **¿Cómo lo hace?**
- Y el Cliente tiene que encontrar las operaciones publicadas para poder utilizarlas, **¿cómo hacerlo?**



Web Services

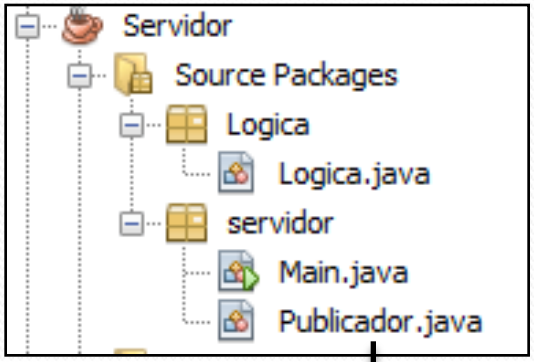


Web Services

- Proxy
  - Es quien le brinda al cliente facilidades para acceder a un servicio.
  - Encapsula la implementación de dicho consumo.
  - Construye objetos Java con el mensaje de respuesta, para poder ser utilizados “cómo si estuvieran locales”
- Server Stub
  - Es quien implementa la lógica del servicio
  - Quien hace posible la publicación
  - Existe en el entorno del servidor
- Vayamos a un ejemplo…

Web Services - Ejemplo

Servidor



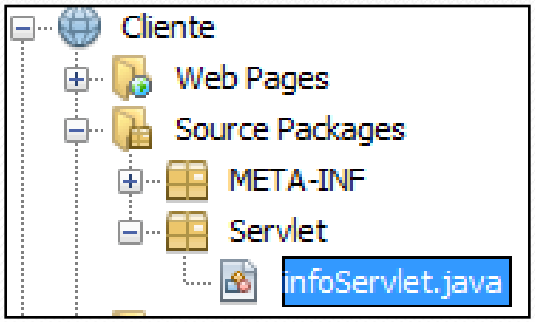
Clase encargada de la publicación.

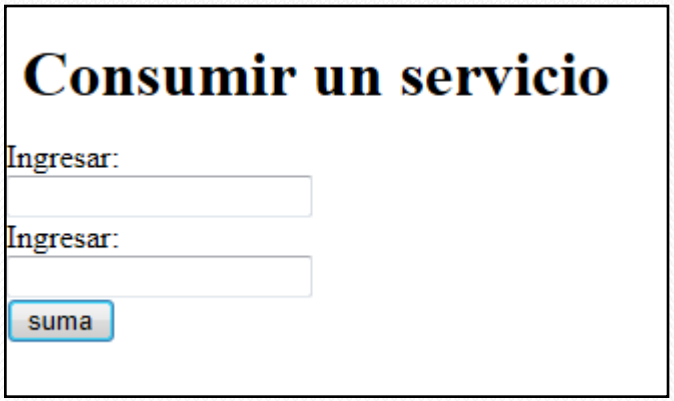
Método que quiero publicar

```
@WebMethod  //Operación a publicar
public int suma(int i, int j){
    Logica log = new Logica();
    int ret = log.sumar(i,j);
    return ret;
}
```

Web Services - Ejemplo

Cliente





Llama

Lugar donde se consumirá el servicio

```
<body>
    <h1>Consumir un servicio</h1>
    <form style="margin: -8.02px; width: 10px; height: auto;
                        "action="infoServlet" ACTION="POST">
        <label>Ingresar:</label>
        <input id="nroi" type="text" name="nroi"/>
        <label>Ingresar:</label>
        <input id="nroj" type="text" name="nroj"/>
        <%session.setAttribute("procedencia","index");%>
        <input type="submit" value="suma" name="operador" />
    </form>
</body>
```

Web Services - Ejemplo

- Generando el Server Stub...

```
@WebService //Indico que en esta clase hay operaciones a publicar
@SOAPBinding(style = Style.RPC, parameterStyle = ParameterStyle.WRAPPED)
public class Public
    private Endpoin
    //Constructor
    public Publicad
    //Operaciones l
    @WebMethod(excl
    public void pub
        endpoint =
    }
    @WebMethod(excl
    public Endpoint
        return
    }
    @WebMethod  //Operación a publicar que consumirá en el Cliente
    public int suma(int i, int j){
        Logica log = new Logica();
        int ret = log.sumar(i,j);
        return ret;
    }
}
```

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Publicador p = new Publicador();
        p.publicar();
    }

}
```

ar en donde
ico el servicio

tub
cador", this);

Generación del Server Stub

Web Services - Ejemplo

- Ejecuto la aplicación Servidor…
  - Esto generará el server stub, registrando mi óperación publicada en el lugar especificado.
  - Para verificar que se publicó correctamente, se puede ingresar en el browser: *url-publicacion?wsdl*
  - **WSDL (*web service description language)*** formato XML que describe el funcionamiento de un WS.
  - Al ingresar a la URL anterior, se debería visualizar el archivo XML en el Browser. De esta manera me aseguro que la publicación ha sido con éxito
- Ahora es el turno del Cliente…

Web Services - Ejemplo

- Para la aplicación cliente…
  - Se mencionó anteriormente que el cliente debe encontrar el servicio, indicándole el lugar exacto en donde debe consumir las operaciones.
  - Para ello debemos generar el Proxy:
  - Tenemos dos formas posibles:
    - Usando el IDE (Nuevo WS, getURL, listo…)
    - Usando el comando –*wsimport* (Abrimos consola, indicamos URL, listo…)
  - Ambos generan a nivel interno, el código pertinente que permite al Cliente hacer uso del Servicio, y crea una serie de objetos que son los que se utilizan de forma local.
  - Se hace notar, que en ambos casos es imprescindible indicar la URL (lugar exacto) de donde se encuentran los métodos.

Web Services

Cliente – InfoServlet.java

```
HttpSession sesion = request.getSession();
String procedencia = (
String address = "";
if (procedencia.equals
    //jsp destino
    address = "resultado.jsp";
    //obtengo los valores de los input (Cajas de texto)
    String sumandoi = request.getParameter("nroi");
    String sumandoj = request.getParameter("nroj");
    int i = Integer.parseInt(sumandoi);
    int j = Integer.parseInt(sumandoj);
    int resultado = 0;
    //me defino un servicio, para consumir el servicio que me publicaron
    servidor.PublicadorService service = new servidor.PublicadorService();
    servidor.Publicador port = service.getPublicadorPort();
    resultado = port.suma(i,j);
    //Guardo el resultado en sesión
    sesion.setAttribute("res", resultado);
}
```

De esta manera se realiza una llamada al servicio publicado, en este caso a la operación *suma*.

Se utiliza el código generado, por El comando –wsimport o por el IDE

Web Services - Resumen

- Pasos para crearlo
  - Definir que componentes se van a comunicar remotamente
  - Elegir que componente publica los servicios (Servidor)
  - Elegir que componente consume los servicios (Cliente)
  - Definir la clase (o interfaz) anotada que correrá en el servidor
  - Definir la dirección en la que se publicará el servicio
  - Generar Stubs del servidor
  - Escribir un cliente que usa el servidor
  - Generar los proxys necesarios
  - **Ejecutar el servidor y luego el cliente**

Cuestiones prácticas

- Si bien ya se explicó el funcionamiento de WS, aún faltan ciertos aspectos por entender.
- Si bien teóricamente la publicación y consumición de servicios funciona, a veces hay que definir ciertos criterios para alcanzar que dos programas trabajando en computadoras diferentes, logren el total entendimiento….

Annotations

- 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
  - **`@WebParam`**

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

Annotations

- **`@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

Tipos válidos

- Tipos nativos de Java
  - Se mapean casi directamente con los nativos definidos en SOAP
- Datatypes
  - Debe ser un **Java Bean**
    - Clase que cumple con ciertas convenciones
    - Entre ellas: Serializable, Constructor sin argumentos, métodos get y set
- Excepciones
  - Se mapean a los **SOAP Faults**
  - También se serializan en xml

Tipos válidos

- Importante…

- Hay ciertos tipos de Java los cuales no pueden ser publicados.
  - Por ejemplo colecciones o tipos obsoletos (Vector, Date)

Escenarios de importancia

- Debemos tener en cuenta
  - Si bien se definieron los tipos válidos, para su publicación se requieren ciertos requerimientos.
  - A continuación se expone un ejemplo en el que dos aplicaciones de escritorio (Servidor y Cliente) se comunican remotamente vía WS, en el que mediante métodos se envían ciertos types:
    - DataPersona con un pseudoatributo set<DataPersona>
    - Una imagen .png
    - Una excepción definida en el Servidor

Ejemplo - Servidor

- Así definimos el Datatype a publicar del lado del servidor

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

DataPersona

```
@XmlAccessorType(XmlAccessType.FIELD)
public class DataPersona {

    private String nombre;
    private String apellido;
    private ArrayList<DataPersona> amigos = new ArrayList<DataPersona>();
```

Ejemplo - Servidor

- Al igual que en el ejemplo anterior, poseemos una clase Lógica
- La misma posee los métodos que queremos que el Cliente consuma

```
public class Logica {

public String obtenerApellido(DataPersona dp){
    return dp.getApellido();
}

public DataPersona obtenerAmigos(String apellido){
    DataPersona dp = new DataPersona("Sofia", apellido);
    DataPersona dp1 = new DataPersona("Ana Ines", "Lopez");
    DataPersona dp2 = new DataPersona("Leticia", "Pastorini");
    dp.addAmigo(dp1);
    dp.addAmigo(dp2);
    return dp;
}
```

Ejemplo - Servidor

Idem ejemplo anterior

- También se posee una clase Publicador.java
- La misma se encarga de la publicación de métodos

```
@WebMethod
public String obtenerApellido(DataPersona dp){
    Logica l = new Logica();
    return l.obtenerApellido(dp);
}
@WebMethod
public DataPersona obtenerAmigos(String apellido){
    Logica l = new Logica();
    return l.obtenerAmigos(apellido);
}
@WebMethod
public byte[] getImage(@WebParam(name = "fileName") String name)
                throws Exception {
    byte[] byteArray = null;
    try {
            File f = new File(name);
            FileInputStream streamer = new FileInputStream(f);
            byteArray = new byte[streamer.available()];
            streamer.read(byteArray);
    } catch (Exception e) {
            throw e;
    }
    return byteArray;
}
```

Tenemos en cuenta la excepción en la Publicación.

Envíamos un array De Bytes

Ejemplo - Cliente

- Consumimos el servicio que nos provee obtener los amigos de una persona

Definimos el servicio para consumirlo.

```
private void buttonAmigosActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    complejoservidor.publicar.PublicadorService service =
                    new complejoservidor.publicar.PublicadorService();
    complejoservidor.publicar.Publicador port = service.getPublicadorPort();
    complejoservidor.publicar.DataPersona dp1= port.obtenerAmigos("Suarez");
    //listAmigos.set;
    ArrayList <String> lista = new ArrayList<String>();
    lista.add(dp1.getAmigos().get(0).getNombre()+ " "
                    + dp1.getAmigos().get(0).getApellido());
    lista.add(dp1.getAmigos().get(1).getNombre()+ " "
                    + dp1.getAmigos().get(1).getApellido());
    Vector v = new Vector(lista);
    listAmigos.setListData(v);

}
```

Obtenemos el DataPersona publicado para luego obtener sus amigos

Ejemplo - Cliente

- Consumimos el servicio que nos provee obtener la imagen
- Obtenida desde el Servidor

```
private void buttonImagenActionPerformed(java.awt.event.ActionEvent evt) {
byte[] img = null;
complejoservidor.publicar.PublicadorService service =
        new complejoservidor.publicar.PublicadorService();
complejoservidor.publicar.Publicador port = service.getPublicadorPort();
try {
    labelex.setVisible(false);
    img = port.getImage(txtNombreImagen.getText());
    Image image = Toolkit.getDefaultToolkit().createImage(img);
    Icon warnIcon = new ImageIcon(image);
    labelImagen.setIcon(warnIcon);
    labelImagen.validate();
    } catch (Exception ex) {
        labelex.setVisible(true);
    }
}
```