Web Services

Estilos de arquitectura e implementación en JEE

Agenda

- **Introducción**
- "Big" Web Services
  - SOAP
  - Web Services Description Language (WSDL)
  - Descubrimiento de servicios
- RESTful WebServices
- API JAX-WS (Big WS)
- API JAX-RS (RESTful WS)

¿Qué es un WS?

- Una aplicación distribuida diseñada para interoperar con otras máquinas sobre la red. La interacción esta basada en interfaces fijas y protocolos definidos

Tipos de WS

- SOAP - BASED
- RESTful

SOAP WS

- Utiliza
  - UUID: Publicar buscar servicios
  - WSDL: Descripción formal de servicios
  - SOAP: Protocolo de invocación de servicios
  - XML: Formato de datos universal
  - HTTP: Uno de los protocolos de transporte

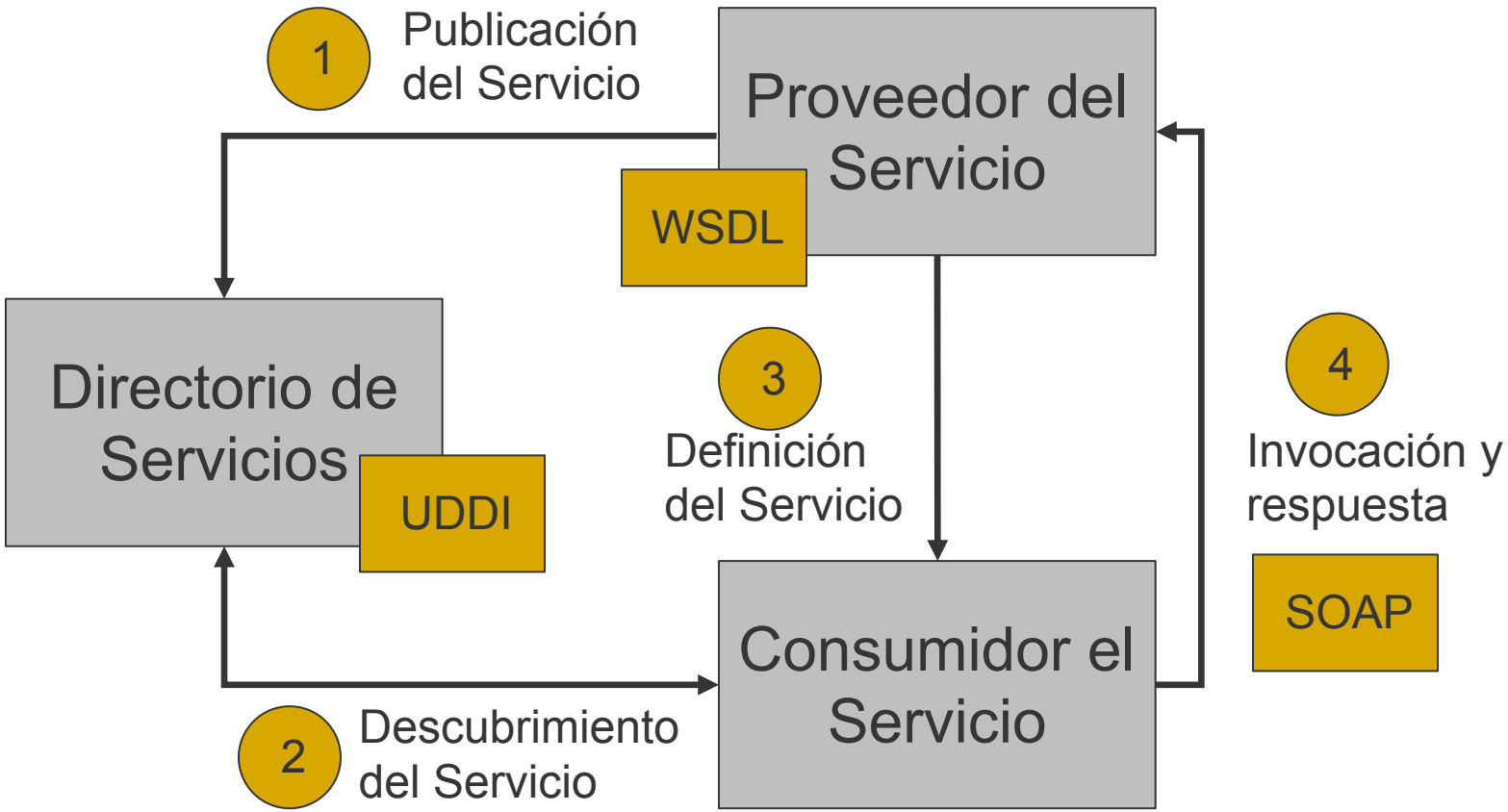
REST WS

- WS diseñado para manipulación de recursos
- Protocolo simple y estándar:
  - GET: Obtener recursos
  - POST: Crea un nuevo recurso
  - PUT: Actualiza un recurso
  - DELETE: Borra un recurso
- Ejemplos: Blogs, TWITER

Agenda

- Introducción
- **“Big” Web Services**
  - SOAP
  - Web Services Description Language (WSDL)
  - Descubrimiento de servicios
- RESTful WebServices
- API JAX-WS (Big WS)
- API JAX-RS (RESTful WS)

Ciclo de componentes e interacciones



Tecnologías y protocolos

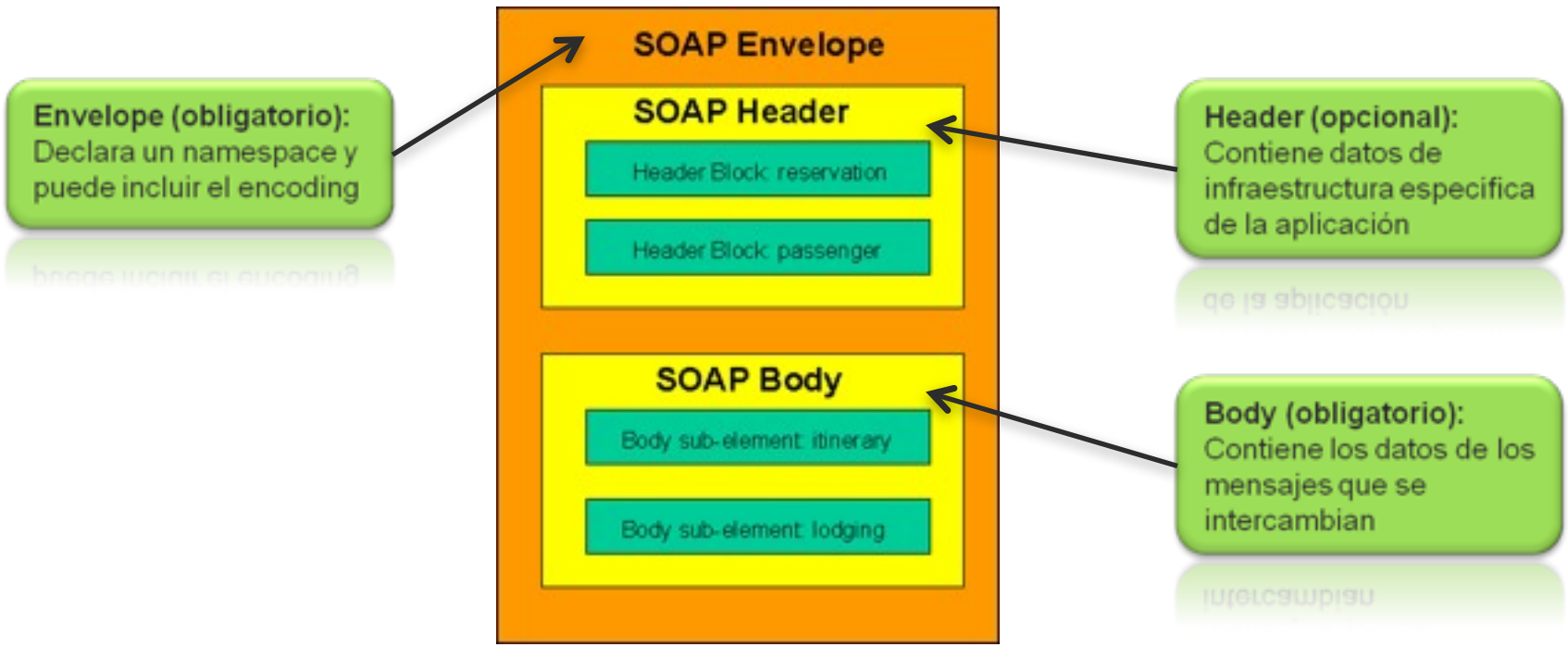
- Utiliza
  - UUID: Publicar buscar servicios
  - WSDL: Descripción formal de servicios
  - SOAP: Protocolo de invocación de servicios
  - XML: Formato de datos universal
  - HTTP: Uno de los protocolos de transporte

SOAP

- Simple Object Access Protocol (SOAP)
  - Es un protocolo distribuido similar a CORBA y Java RMI
  - Permite que las aplicaciones intercambien mensajes sobre un protocolo de red, comúnmente HTTP
  - SOAP utiliza XML, estos documentos contienen varios elementos:
    - Sobre
    - Cabecera
    - Cuerpo

SOAP

- Estructura de mensaje SOAP



Tomado de [5]

SOAP

- Ejemplo de mensaje SOAP


```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <addBid xmlns="http://actionbazaar.com/Bidding">
            <user-id>viper</user-id>
            <item-id>100</user-id>
            <bid-price>2000.24</bid-price>
        </addBid>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

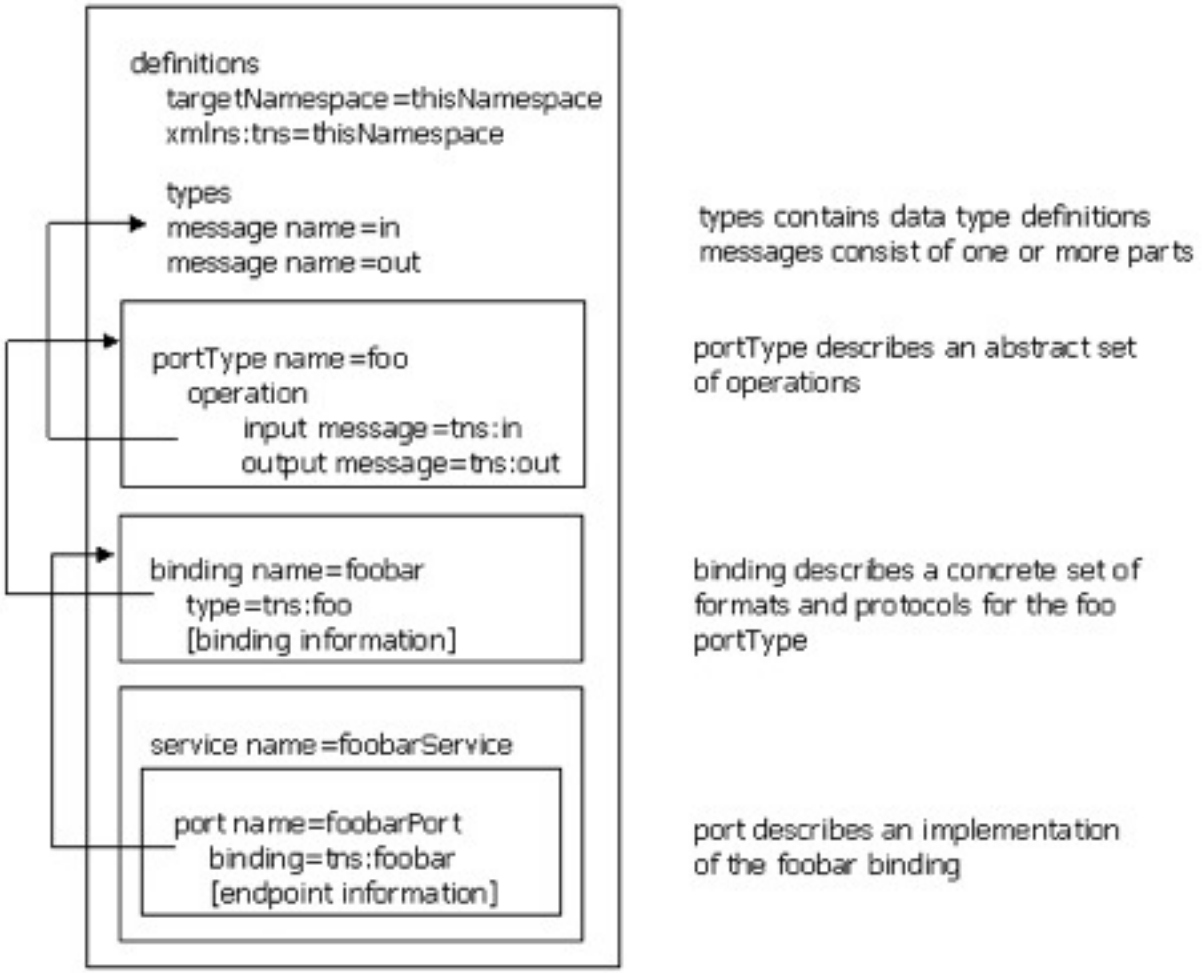
Tomado de [1]

Web Services Description Language (WSDL)

- Es un documento XML que provee toda la información requerida para localizar y acceder a un servicio web
- El proveedor del servicio es el responsable de definir el WSDL
- Especifica el tipo de mensaje, puertos, operaciones soportadas, tipos de datos
- No hay un mapeo directo en JAVA y WSDL. Es soportado por la especificación *Java Architecture for XML Binding (JAXB) 2.0*
  http://download-west.oracle.com/docs/cd/B25221_04/web.1013/b25603/apptypemapping.htm#BABCCAHA

Web Services Description Language (WSDL)

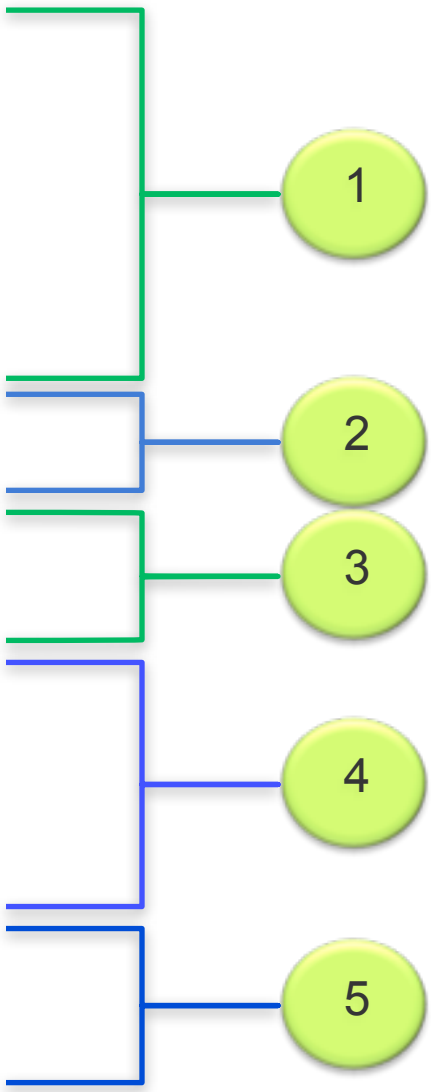
- Modelo de datos de un WSDL



Tomado de [8]

Web Services Description Language (WSDL)

```
<?xml version = '1.0' encoding = 'UTF-8'?> <definitions xmlns=http://schemas.xmlsoap.org/wsdl/"  xmlns:soap=http://ejb3inaction.example.buslogic/ >
<types> <schema xmlns="http://www.w3.org/2001/XMLSchema"  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
        <element name="addBid" type="tns:addBid"/>
        <complexType name="addBid">
           <sequence>
              <element name="userId" type="string" nillable="true"/>
              <element name="itemId" type="long" nillable="true"/>
              <element name="bidPrice" type="double" nillable="true"/>
           </sequence>
        </complexType>
        <element name="addBidResponse" type="tns:addBidResponse"/>
        <complexType name="addBidResponse">
           <sequence>  <element name="return" type="long" nillable="true"/>  </sequence>
        </complexType>
</schema> </types>
<message name="PlaceBidBeanPortType_addBid">
        <part name="parameters" element="tns:addBid"/>
</message>
<portType name="PlaceBidBean">
      <operation name="addBid">
           <input message="tns:PlaceBidBeanPortType_addBid"/>
           <output message="tns:PlaceBidBeanPortType_addBidResponse"/>
      </operation>
</portType>
<binding name="PlaceBidBeanSoapHttp" type="tns:PlaceBidBean">
     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
     <operation name="addBid">
          <soap:operation soapAction=""/>
          <input>  <soap:body use="literal"/>  </input>
          <output> <soap:body use="literal"/> </output>
     </operation>
</binding>
<service name="PlaceBidBeanService">
     <port name="PlaceBid" binding="tns:PlaceBidBeanSoapHttp">
          <soap:address location="${oracle.scheme.host.port.and.context}/PlaceBid"/>
     </port>
</service>
</definitions>
```



Tomado de [1]

Web Services Description Language (WSDL)

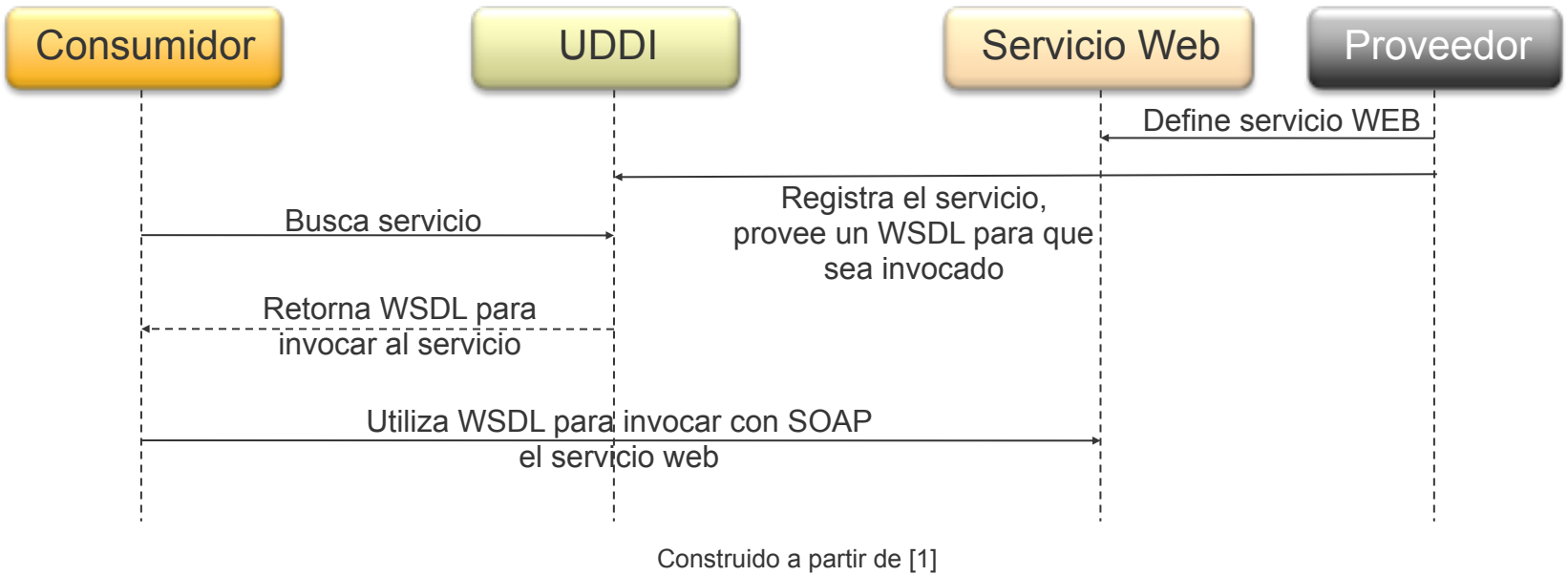
1. **ComplexType.** Define los tipos de datos que se utilizan cuando el servicio web es invocado, estos tipos de datos son utilizados por los métodos
2. **Message.** Define los mensajes que soporta el servicio. El *message part* es una parte del mensaje, se pueden definir parámetros de entrada y datos de salida
   - En el ejemplo, el mensaje *PlaceBidBeanPortType_addBid*, recibe como parámetro un valor de tipo *addBid*, definido en el *complexType*
3. **Port Type.** Define las operaciones que pueden ser ejecutadas y los mensajes relacionados. Para cada operación define las entradas y salidas (*input*/*output*)
4. **Binding.** Los detalles de un servicio: mensajes, operaciones y protocolos son definidos a través del binding
5. En el ejemplo se utiliza el binding *PlaceBidBeanSoapHttp*

Descubrimiento de servicios

- Universal Description, Discovery, and Integration (UDDI)
  - Permite publicar y descubrir un servicio web
  - Utiliza el protocolo SOAP para el descubrimiento de un servicio
  - Se utiliza de forma opcional, si es para SOA se utilizan otros mecanismos de descubrimiento
  - Maneja los siguientes estructuras de datos
    - **BusinessEntity:** Contiene información acerca de la empresa que publica el servicio
    - **BusinessService:** Contiene la descripción del servicio
    - **BindingTemplate:** Información técnica que permite determinar los puntos de entrada y las especificaciones para invocar al servicio

Descubrimiento de servicios

- Proceso de descubrimiento



Construido a partir de [1]

1. El proveedor define el servicio y lo publica en el UDDI, incluyendo el WSDL para que un cliente se comunique con el servicio

Transporte de mensajes

- Los mensajes son transmitidos por el protocolo de red HTTP
  - Utilizado para comunicación en la WEB
- Se pueden utilizar otros protocolos como:
  - SMTP
  - FTP
  - JMS

Agenda

- Introducción
- “Big” Web Services
  - SOAP
  - Web Services Description Language (WSDL)
  - Descubrimiento de servicios
- **RESTful WebServices**
- API JAX-WS (Big WS)
- API JAX-RS (RESTful WS)

RESTful WS

- SOAP-WS
  - Usados para mensajería
  - Usados para RPC
- RESTful WS
  - Mas livianos
  - Impulsados popr la Web 2.0 y nusvos frameworks, e.g., RAILS
  - Amazon, Google, Yahoo! deprecaron SOAP-WS para utilizar RESTful-WS

Representational State Transfer (REST)

- Estilo arquitectural
- La Web como modelo exitoso de arquitectura
- Abstrae los mecanismos de la web para aplicarlos en otras tecnologías / metodologías

Conceptos RESTful

- Recursos: Cualquier cosa a referenciar
- URI (URL, URN)
- Representaciones (HTML, CSV)
- WADL (el WSDL de RESTful-WS)
- HTTP

El protocolo HTTP para RESTful-WS

HTTP Methods

- GET: Obtener recursos
- POST: Crea un nuevo recurso
- PUT: Actualiza un recurso
- DELETE: Borra un recurso
- Otros: HEAD, TRACE, OPTIONS, CONNECT

Negociación de contenido

- 2 métodos para seleccionar la mejor representación
  - Sobrecarga de URLs (/xxx.csv o /xxx.html)
  - Utilizar el encabezado HTTP:
    - Accept, Accept-Charset, Accept-Encoding
- Tipos de Contenidos
  - MIME types: text/plain, text/xml, json, etc.

Códigos de respuestas

- 1xx: Información
- 2xx: Éxito
- 3xx: Redirección
- 4xx: Error del Cliente
- 5xx: Error del servidor

Caché y condicionales

- Utilizar hash para representar recursos
- Solicitar recursos solo si el hash ha cambiado
- Ejemplo:
  - Req.: GET /book/12345
  - Ans.: 200 ok <book> ...</book>, ETag= 32876
  - Req.: GET /book/12345, if-Non-Match: 32876
  - Ans.:304 Not Modified

Consideraciones de diseño

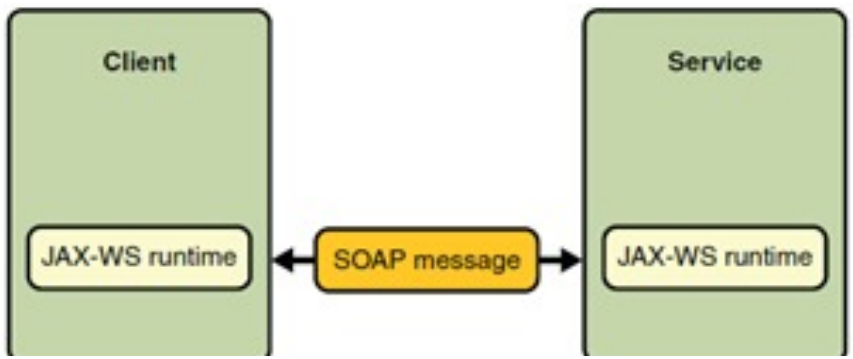
- Interface uniforme
- Addressability: Recursos fácilmente accesibles
- Conectividad (todos los recursos conectados)
- No manejo de estado en el servidor

Agenda

- Introducción
- “Big” Web Services
  - SOAP
  - Web Services Description Language (WSDL)
  - Descubrimiento de servicios
- RESTful WebServices
- **API JAX-WS (Big WS)**
- API JAX-RS (RESTful WS)

API JAX-WS

- El API de Java para servicios web basados en XML es **JAX-WS 2.0** → Especificación que define el estándar para servicios web en JEE, es una extensión del API para XML-RPC (JAX-RPC) 1.0.
- Es una tecnología para construir servicios web y clientes que se comunican utilizando XML
- Permite a los desarrolladores escribir orientado a mensajes Ej: Servicios web orientado a RPC
- La invocación a un servicio web es representada con protocolos basados en XML como SOAP, sobre el protocolo de red HTTP



Tomado de [2]

API JAX-WS

- Beneficios JAX-WS
  - Simplifica el desarrollo de aplicaciones que exponen servicios WEB
  - Plataforma independiente de JAVA
  - Permite utilizar servicios de distintas tecnologías, no todos deben ejecutarse en plataforma JAVA
  - Usa tecnologías definidas por el consorcio W3C
  - Utiliza HTTP, SOAP y WSDL para describir el servicio

API JAX-WS

- Permite exponer como servicios web a clases java, incluyendo los beans de sesión en EJB 3
  - Servicio web java
  - Servicio web EJB
- Ventajas de Servicio web EJB
  - Incluye transacción declarativa y seguridad
  - Se puede interceptar el servicio si se requiere
  - Exponer aplicaciones de negocio
  - Uso de protocolos adicionales como RMI

API JAX-WS

- Comparación entre un servicio web java y un servicio web EJB

Característica	Servicio web Java	Servicio web EJB
POJO	Si	Si
Dependencia de inyección para recursos, unidades de persistencia entre otros	Si	Si
Métodos de ciclo de vida	Si	Si
Transacción Declarativa	No	Si
Seguridad Declarativa	No	Si
Requiere una herramienta externa para procesar las anotaciones	Si	El contenedor no lo requiere

API JAX-WS

- Proceso para generar un servicio web

Proceso convencional	Con JAX-WS
Generar el WSDL para el servicio web	Generado automáticamente por el contenedor durante el despliegue del proyecto
Definir la interfaz de punto final del servicio	
Identificar el endpoint en el archivo *ejb-jar.xml*	
Empaquetar el proyecto con el archivo *webservices.xml*	

Agenda

- Introducción
  - Ciclo de Vida de un SW
  - Beneficios SW
  - Propiedades SW
  - Componentes SW
- SOAP
- Web Services Description Language (WSDL)
- Descubrimiento de servicios
- API JAX-WS
  - **Anotaciones API JAX-WS**
- Invocar un servicio WEB desde un EJB
- SOA

Anotaciones API JAX-WS

- Utiliza anotaciones para definir un servicio WEB
  - `@WebService`
  - `@SOAPBinding`
  - `@WebMethod`
  - `@WebParam`
  - `@WebResult`
  - `@OneWay`
  - `@HandlerChain`

Anotaciones API JAX-WS

- Ejemplo JAX-WS

```
@WebService(targetNamespace= "urn:ActionBazaarPlaceBidService")
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT)
@Stateless(name = "PlaceBid")
public class PlaceBidBean implements PlaceBid {

    @PersistenceContext private
    EntityManager em;

    public PlaceBidBean() { }

    @WebMethod
    @WebResult(name = "bidNumber")
    public Long addBid(
        @WebParam(name = "User")    String userId,
        @WebParam(name = "Item")    Long itemId,
        @WebParam(name = "Price")    Double bidPrice) {
                return persistBid(userId, itemId, bidPrice);
        }

    private Long persistBid(String userId, Long itemId, Double
    bidPrice)
    { … }
}
```

@WebService: Expone el EJB como un servicio web

@WebMethod: Expone el método en el servicio web

Controlan parámetros en el WSDL

Tomado de [1]

Anotaciones API JAX-WS

- **`@WebService`**
  - Indica que una clase Java está implementando un servicio Web o indica que una SEI (*Service Endpoint Interface*) está implementando una interfaz de servicio Web
  - Utilizado en un bean o en una interfaz
  - Cuando se utiliza sobre un bean, el contenedor genera la respectiva interfaz
  - Cuando se utiliza en la interfaz, los métodos públicos serán expuestos en el servicio web

Anotaciones API JAX-WS

- Elementos de la anotación **`@WebService`**

```
@WebService
public interface PlaceBidWS {
    public Long addBid(String bidderId, Long itemId, Double bidPrice);
}

@Stateless(name = "PlaceBid")
public class PlaceBidBean implements PlaceBidWS, PlaceBid { ... }
```

```
@Target({TYPE})
public @interface WebService {
    String name() default "";
    String targetNamespace() default "";
    String serviceName() default "";
    String wsdlLocation() default "";
    String endpointInterface() default "";
    String portName() default "";
};
```

Tomado de [1]

Anotaciones API JAX-WS

- **`@SOAPBinding`**
  - Especifica el mapeo del servicio Web con el protocolo de mensajes SOAP
  - **`@SOAPBinding.Style`** → Indica el estilo de codificación para enviar mensajes hacia y desde el servicio Web
    - Modifica el atributo style del elemento *soap:binding* del WSDL
  - **`@SOAPBinding.Use`** → Indica que el formato de los mensajes enviados desde y hacia el servicio Web
    - Afecta el atributo use del elemento *soap:body* del WSDL

Anotaciones API JAX-WS

- Descripción interfaz de `@SOAPBinding`

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target({TYPE})
public @interface SOAPBinding {
    public enum Style {
        DOCUMENT,
        RPC
    };
    Style style() default Style.DOCUMENT;

    public enum Use {
        LITERAL,
        ENCODED
    };
    Use use() default Use.LITERAL;

    public enum ParameterStyle {
        BARE,
        WRAPPED
    };
    ParameterStyle parameterStyle() default ParameterStyle.WRAPPED;
}
```

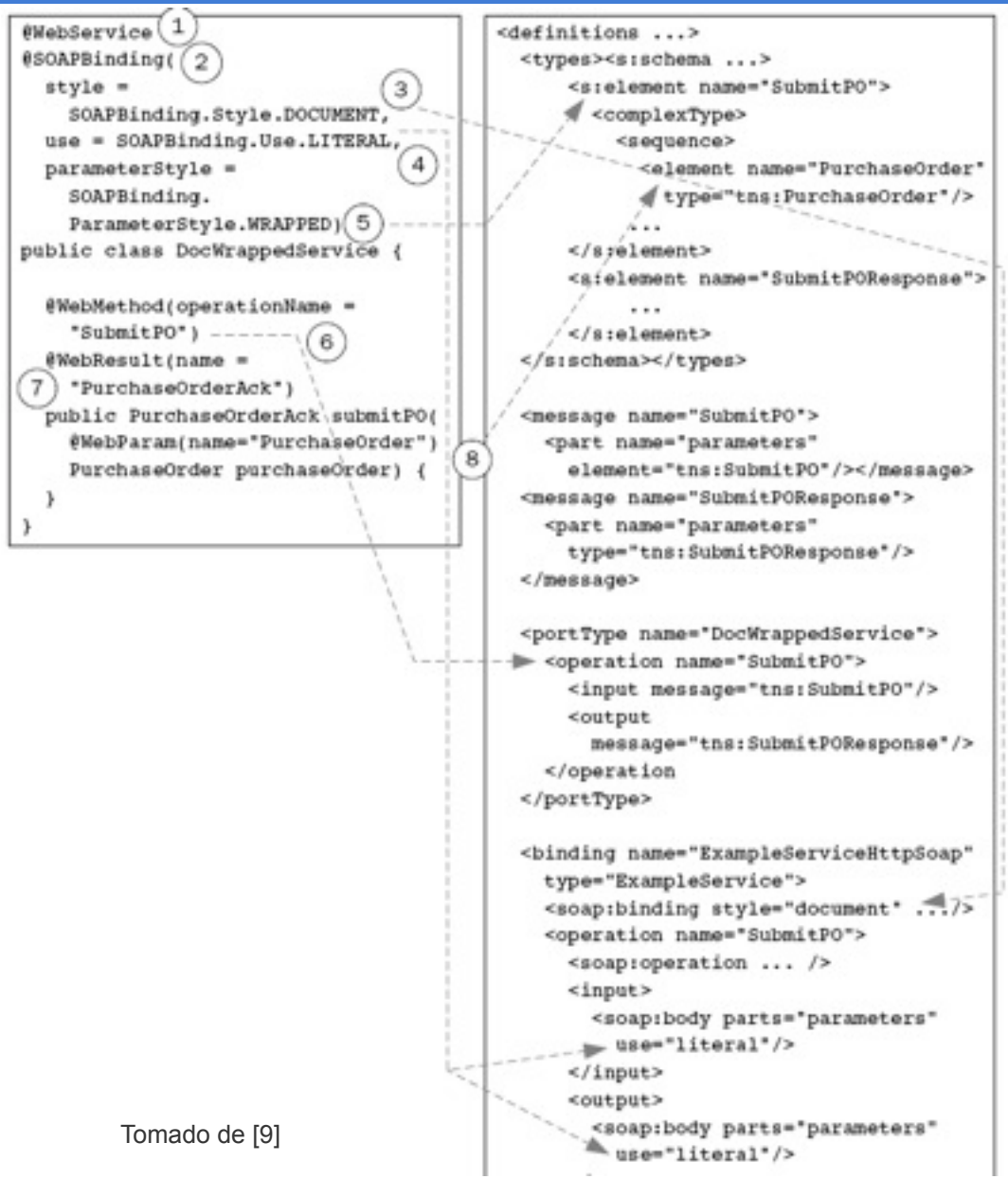
Estilo del servicio WEB

Estilo del mensaje. Utilizar LITERAL cuando se trabaja con clientes que no son desarrollados, el ENCODED puede presentar problemas

Estilo de los parámetros

Tomado de [1]

Anotaciones API JAX-WS



```
@WebService (1)
@SOAPBinding( (2)
  style =
    SOAPBinding.Style.DOCUMENT, (3)
  use = SOAPBinding.Use.LITERAL, (4)
  parameterStyle =
    SOAPBinding.
    ParameterStyle.WRAPPED) (5)
public class DocWrappedService {

  @WebMethod(operationName =
    "SubmitPO") (6)
  @WebResult(name =
(7) "PurchaseOrderAck")
  public PurchaseOrderAck submitPO(
    @WebParam(name="PurchaseOrder")
    PurchaseOrder purchaseOrder) {
  }
}
```

```
<definitions ...>
  <types><s:schema ...>
      <s:element name="SubmitPO">
        <complexType>
          <sequence>
            <element name="PurchaseOrder"
              type="tns:PurchaseOrder"/>
            ...
      </s:element>
      <s:element name="SubmitPOResponse">
            ...
      </s:element>
  </s:schema></types>

  <message name="SubmitPO">
    <part name="parameters"
      element="tns:SubmitPO"/></message>
  <message name="SubmitPOResponse">
    <part name="parameters"
      type="tns:SubmitPOResponse"/>
  </message>

  <portType name="DocWrappedService">
    <operation name="SubmitPO">
      <input message="tns:SubmitPO"/>
      <output
        message="tns:SubmitPOResponse"/>
    </operation
  </portType>

  <binding name="ExampleServiceHttpSoap"
    type="ExampleService">
    <soap:binding style="document" .../>
    <operation name="SubmitPO">
      <soap:operation ... />
      <input>
        <soap:body parts="parameters"
          use="literal"/>
      </input>
      <output>
        <soap:body parts="parameters"
          use="literal"/>
```

(8)

Tomado de [9]

Anotaciones API JAX-WS

- **@WebMethod**
  - Expone el método como parte de un servicio Ww

```
@Target({METHOD})
public @interface WebMethod {
    String operationName() default "";
    String action() default "" ;
    boolean exclude() default false;
};
```

  - Si no se quiere exponer algún método se puede utilizar **@WebMethod(exclude=true)**

Anotaciones API JAX-WS

- Con esta anotación se puede definir el nombre de la operación con el atributo *operationName* y la acción SOAP *action* que se utiliza en WSDL
- El elemento action determina el elemento en el encabezado de petición HTTP y define el destino del mensaje

```
@WebMethod(operationName = "addNewBid",
      action = "http://actionbazaar.com/NewBid")
public Long addBid ( ... ) {
   …
}
```

```
<portType name = "PlaceBidBean">
   <operation name = "addNewBid">
      ...
   </operation>
</portType>
```

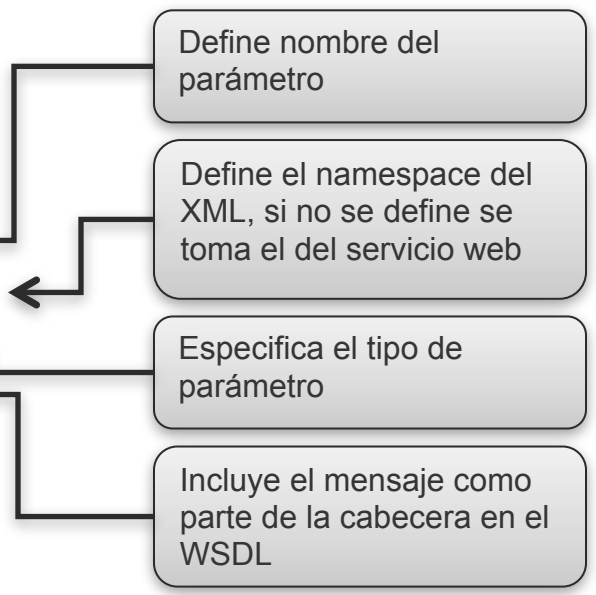
```
<operation name = "addNewBid">
   <soap:operation soapAction =
         "http://actionbazaar.com/NewBid"/>
   ...
</operation>
```

Tomado de [9]

Anotaciones API JAX-WS

- **@WebParam**
  - Se puede utilizar en conjunto con la anotación **@WebMethod** para definir los parámetros de un mensaje generado en el WSDL

```
@Target({PARAMETER})
public @interface WebParam {
    public enum Mode { IN, OUT, INOUT };
    String name() default "";
    String targetNamespace() default "";
    Mode mode() default Mode.IN;
    boolean header() default false;
    String partName() default "";
};
```



Tomado de [1]

Anotaciones API JAX-WS

- Ejemplo **@WebParam**

```
@WebMethod
public Long addBid(
    @WebParam(name = "user",mode = WebParam.Mode.IN)
    String userId, ...)
{
    ...
}
```

```
@WebParam(name = "user", mode = WebParam.Mode.INOUT)
    Holder<String> userId, ...)
{
    ...
}
```

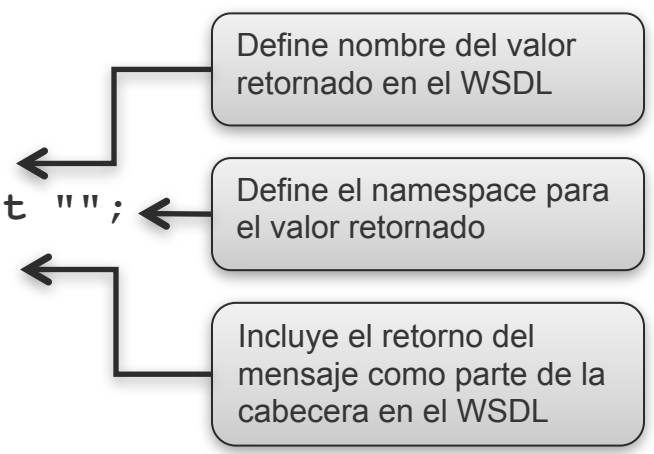
Tomado de [1]

Anotaciones API JAX-WS

- **@WebResult**
  - Opera en conjunto con **@WebMethod**. Se utiliza para especificar el nombre del mensaje de retorno en el WSDL

```
@WebMethod
@WebResult(name = "bidNumber")
public Long addBid(...) { }
```

```
public @interface WebResult {
    String name() default "return";
    String targetNamespace() default "";
    boolean header() default false;
    String partName() default "";
};
```



Tomado de [1]

Invocar un servicio WEB desde un EJB

- **@WebServiceRef**
  - Se puede invocar un servicio Web desde un MDB o un bean de sesión

```
@Stateless
public class TrackOrderBean implements TrackOrder {
    @WebServiceRef(TrackDeliveryService.class)
    private TrackDeliverySEI deliveryService;

    public String checkOrderDeliverStatus(String shipId) {
        ...
        String deliveryStatus =
            deliveryService.checkDeliveryStatus(shipId);
        ...
    }
}
```

Inyecta el servicio Web

Invoca el servicio Web

Tomado de [1]

Invocar un servicio WEB desde un cliente

- Ejemplo **@WebServiceRef**

```
import javax.xml.ws.WebServiceRef ;
import actionbazaarplacebidservice.PlaceBidService;

@WebServiceRef(wsdlLocation="http://localhost:8080/
                                    PlaceBidService/PlaceBidBean?WSDL")
private static PlaceBidService placeBidService;

public static void main(String [] args) {
    try {
       actionbazaarplacebidservice.PlaceBidBean placeBid =
                              placeBidService.getPlaceBidBeanPort();
       System.out.println("Bid Successful, BidId Received is: "
          + placeBid.addBid("dpanda", Long.valueOf(9001), 2000005.50 ));
    } catch (Exception ex) {
       ex.printStackTrace();
    }
}
```

Interfaz de servicio generada

Inyecta una referencia al servicio web

Obtiene un proxy para acceder al servicio Web

Invoca un método del servicio web

Tomado de [1]

Invocar un servicio WEB desde un EJB

- Elementos de **`@WebServiceRef`**

Element	Description
`name`	The JNDI name for the web service. It gets bound to `java:comp/env/<name>` in the ENC.
`wsdlLocation`	The WSDL location for the service. If not specified, then it is derived from the referenced service class.
`type`	The Java type of the resource.
`value`	The service class; always a type extending `javax.xml.ws.Service`.
`mappedName`	Vendor-specific global JNDI name for the service.

Tomado de [1]

Agenda

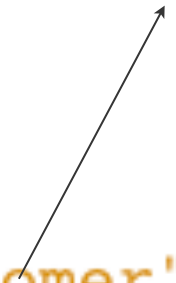
- Introducción
- “Big” Web Services
  - SOAP
  - Web Services Description Language (WSDL)
  - Descubrimiento de servicios
- RESTful WebServices
- API JAX-WS (Big WS)
- **API JAX-RS (RESTful WS)**

JAX-RS

- Api Java para el manejo de RESTful-WS
- Esconde la plomería de bajo nivel
- La vieja y conocida formula:
  - POJO + Anotaciones

Declarando recursos

Anotación @path declara la url del recurso



```
@Stateless
@Path("entities.customer")
public class CustomerFacadeREST
```

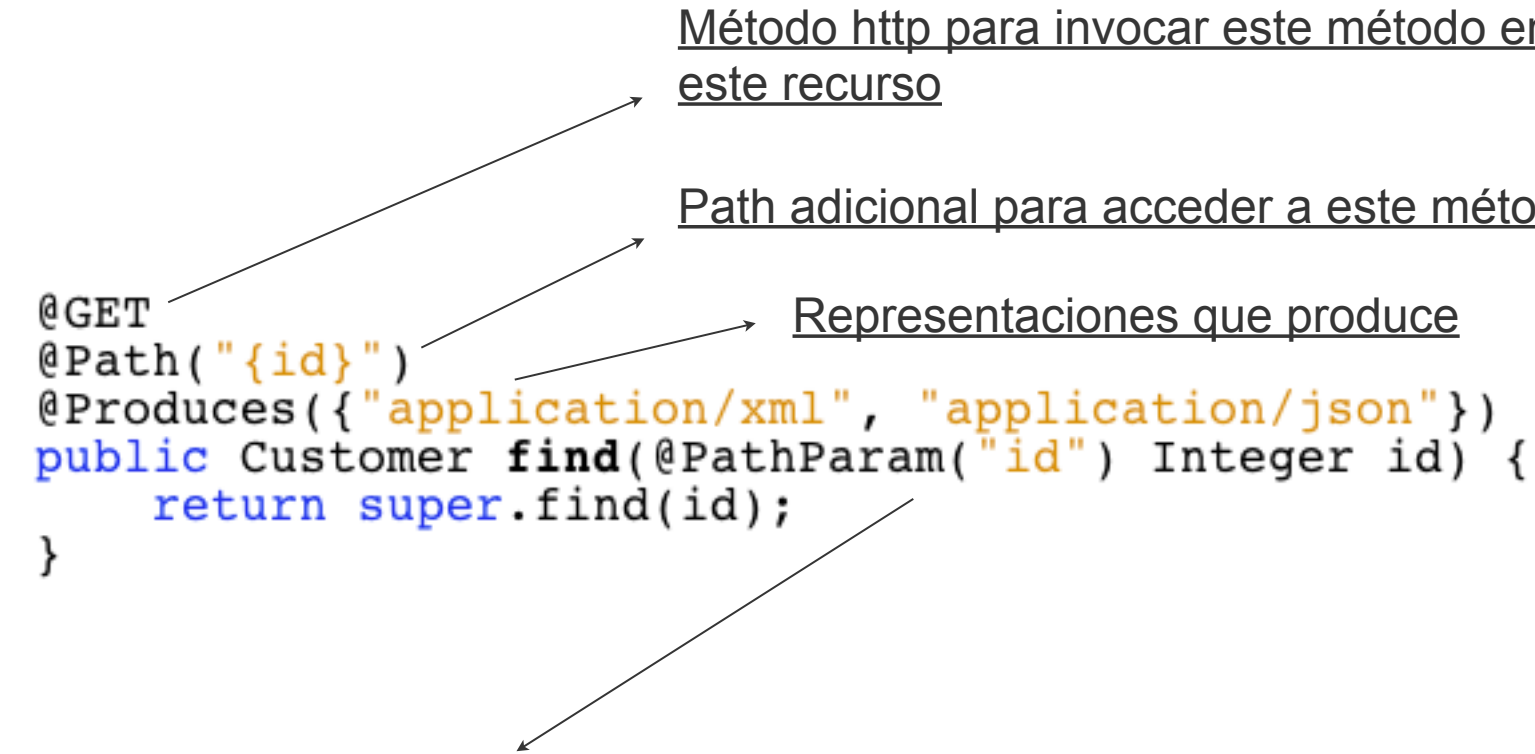
Creando métodos POST

Método http para invocar este método en este recurso

Representaciones que puede consumir

```
@POST
@Override
@Consumes({"application/xml", "application/json"})
public void create(Customer entity) {
    super.create(entity);
}
```

Creando un método GET



Bibliografía

1. Beginning JAVA EE 6 platform With GlassFish 3. Antonio Goncalves
2. The Java™ EE 6 Tutorial. Oracle. 2011.
3. Principles of Computer system Design. Saltzer, Kaashoek.Morgan Kaufmann, 2009.
4. RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. Pautasso, Zimmermann, Leymann. WWW 2008. Beijing, China