Taller de Sistemas Empresariales

Capa de Servicios en Sistemas Empresariales

Web Services

Instituto de Computación

Facultad de Ingeniería

Universidad de la República de Uruguay

Agenda

- Introducción
- SOAP Web Services
- SOAP Web Services en Java EE
- RESTful Web Services
- RESTful Web Services en Java EE

Introducción

Web Services

- El término Web Service nace aproximadamente en el año 2000
- Surgen como una necesidad de la industria en las áreas:
  - Business to Business (B2B)
  - Enterprise Application Integration (EAI)

Introducción

Web Services: ¿Por qué surgen?

- Business to Business integration (B2B)
  - Acuerdos comerciales entre organizaciones
  - Ej: Posibilidad de pagar facturas a través de redes de cobranza
- Enterprise Application Integration (EAI)
  - Sistemas desarrollados por diferentes empresas
    - Diferentes tecnologías
  - Silos de información
    - Contabilidad, inventario, logística
  - Fusión de empresas

Introducción

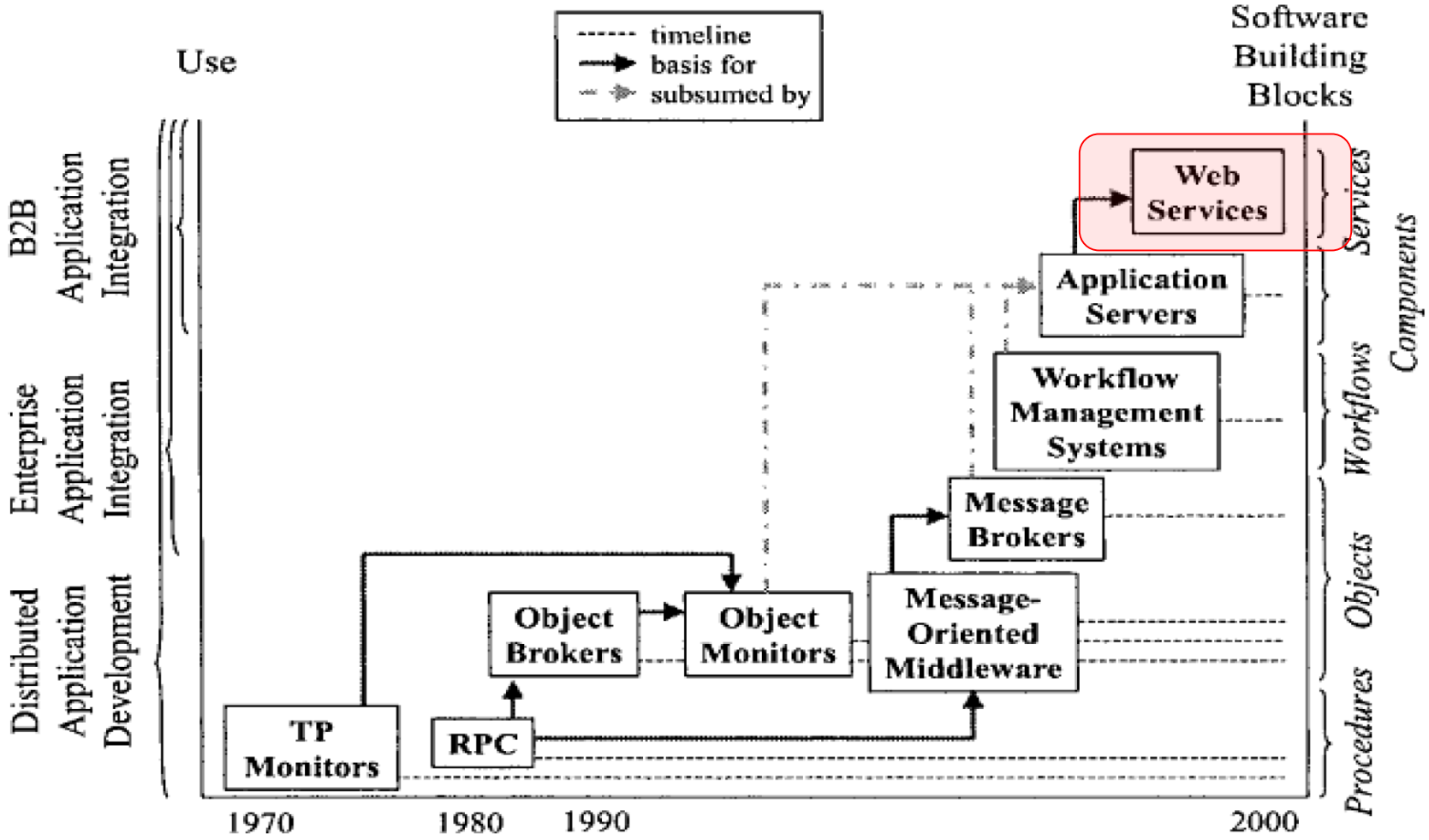
Web Services (según W3C)

- Un Web Service es una aplicación de software identificada por una URI, cuyas interfaces y formas de acceso pueden ser definidas, descriptas y descubiertas como artefactos XML, y soporta la interacción directa con otros componentes de software utilizando mensajes basados en XML, intercambiados a través de protocolos basados en internet

  http://www.w3.org/TR/ws-desc-reqs/#definitions

Introducción

Web Services: Perspectiva Histórica



Semantic Management of Middleware. Ramesh Jain. Amit Sheth. Springer 2006.

Introducción

Web Services

- Mecanismo para que aplicaciones cliente y servidor se comuniquen a través de los protocolos de la web (HTTP/HTTPS)
- Permite que diversas aplicaciones en múltiples tipos de plataformas y frameworks puedan interoperar
- Tipos de Web Services:
  - "Big" Web Services
  - "RESTful" Web Services

(Jendrock et all, 2014)

Introducción

Web Services: Big Web Services

- Utilizan mensajes XML que siguen el protocolo SOAP (Simple Object Access Protocol)
- Generalmente proveen un documento que permite describir las funciones provistas por el web service (WSDL)
- Pueden soportar aspectos no funcionales, como ser transaccionalidad, seguridad, mensajería confiable
- Pueden soportar invocaciones y procesamiento asíncrono

(Jendrock et all, 2014)

Introducción

Web Services: RESTful Web Services

- REST es el estilo arquitectónico sobre el que está basado la web
- No requieren el uso de WSDL, SOAP o XML
- Están muy integrados con el protocolo HTTP
- Soportan el uso de cache, provisto por las tecnologías de la web
- Ambas partes deben acordar el contexto y contenido del material intercambiado

(Jendrock et all, 2014)

Introducción

SOAP Web Services vs RESTful Web Services

- 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
  - Escalar en clientes

Introducción

Plataformas de Web Services

- Las plataformas de Web Services son un conjunto de herramientas para un lenguaje de programación específico

- Permiten:
  - invocar Web Services
  - realizar el *deploy* de Web Services
- Las plataformas tienen en general componentes en el servidor y en el cliente

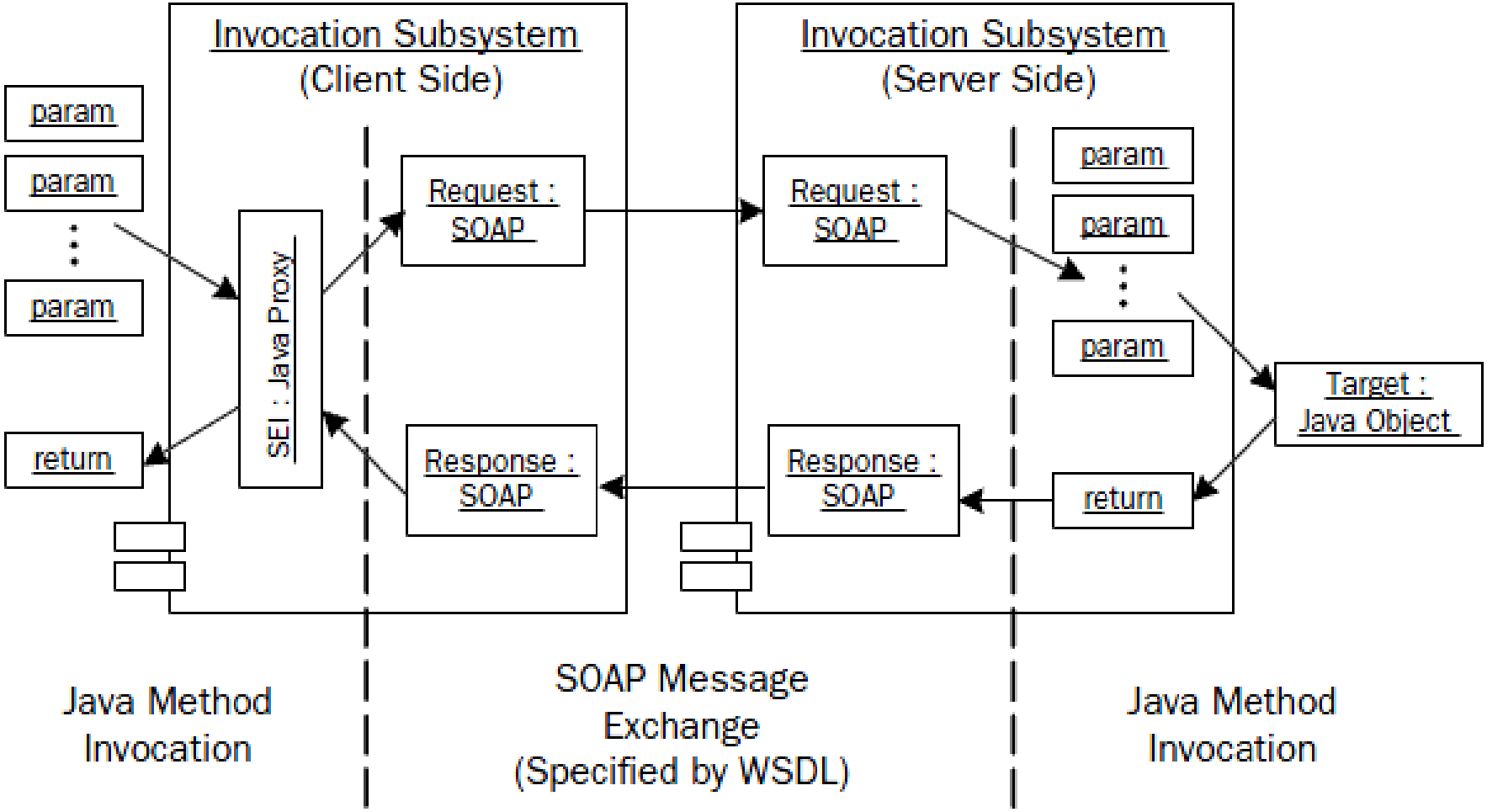
Introducción

Plataformas de Web Services

- Los componentes en el servidor se alojan usualmente en algún tipo de contenedor
  - servidor Java EE,
  - contenedor de servlets, etc
- Las plataformas de Web Services proveen en general tres subsistemas:
  - Invocación
  - Serialización
  - Deployment

Introducción

Subsistema Invocación

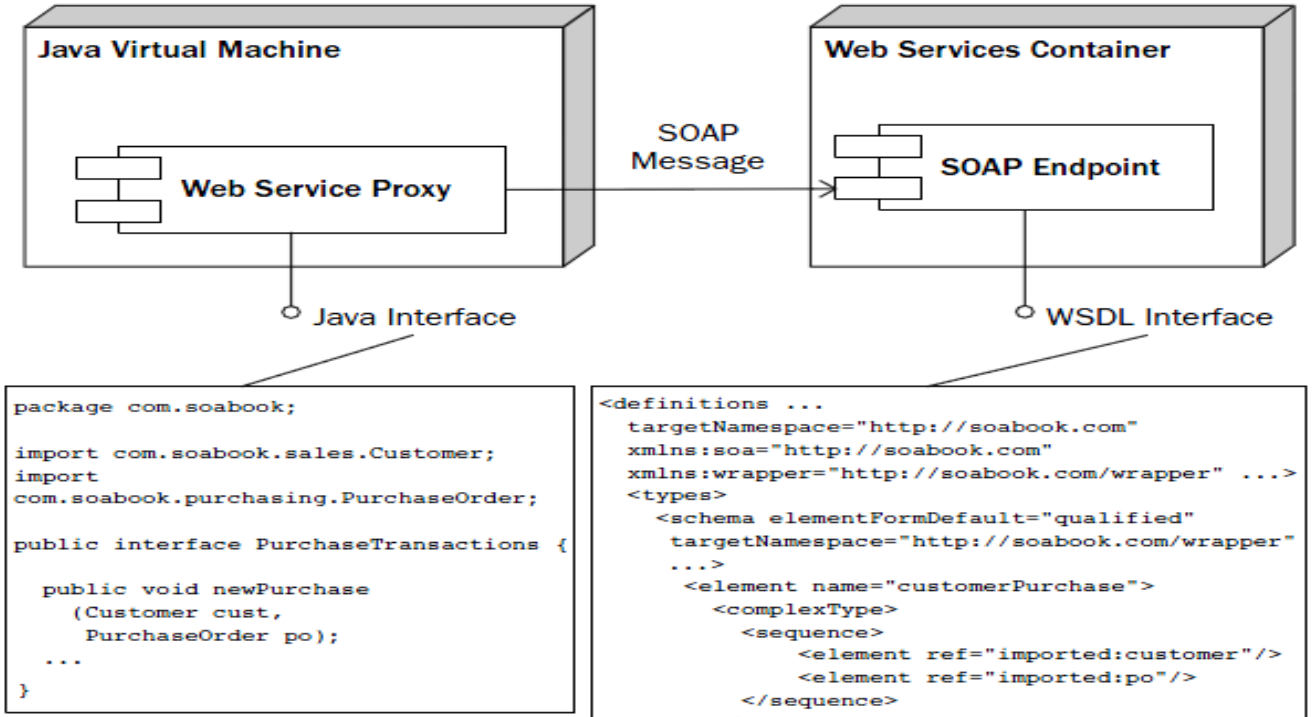


SOA using Java Web Services. Mark D. Hansen. Prentice Hall 2007.

Introducción

Subsistema Serialización

- Transformar instancias de clases Java en elementos XML y viceversa.



SOA using Java Web Services. Mark D. Hansen. Prentice Hall 2007.

Introducción

Subsistema Deployment

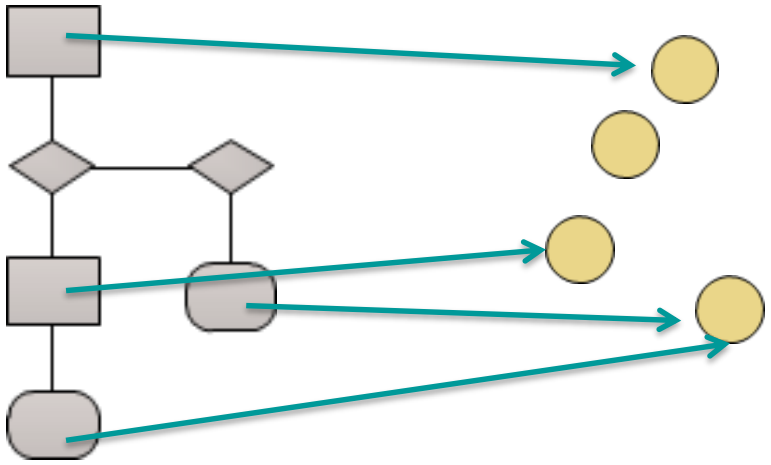
- Este subsistema provee las herramientas para configurar un destino Java para que pueda ser invocado como un Web Service vía mensajes SOAP

Introducción

Service Oriented Computing (SOC)

- La Computación Orientada a Servicios (SOC) es un paradigma de computación que utiliza servicios como elementos fundamentales para dar soporte al desarrollo rápido, y de bajo costo, de aplicaciones distribuidas en ambientes heterogéneos.

Aplicaciones Basadas en Servicios



Servicios

Introducción

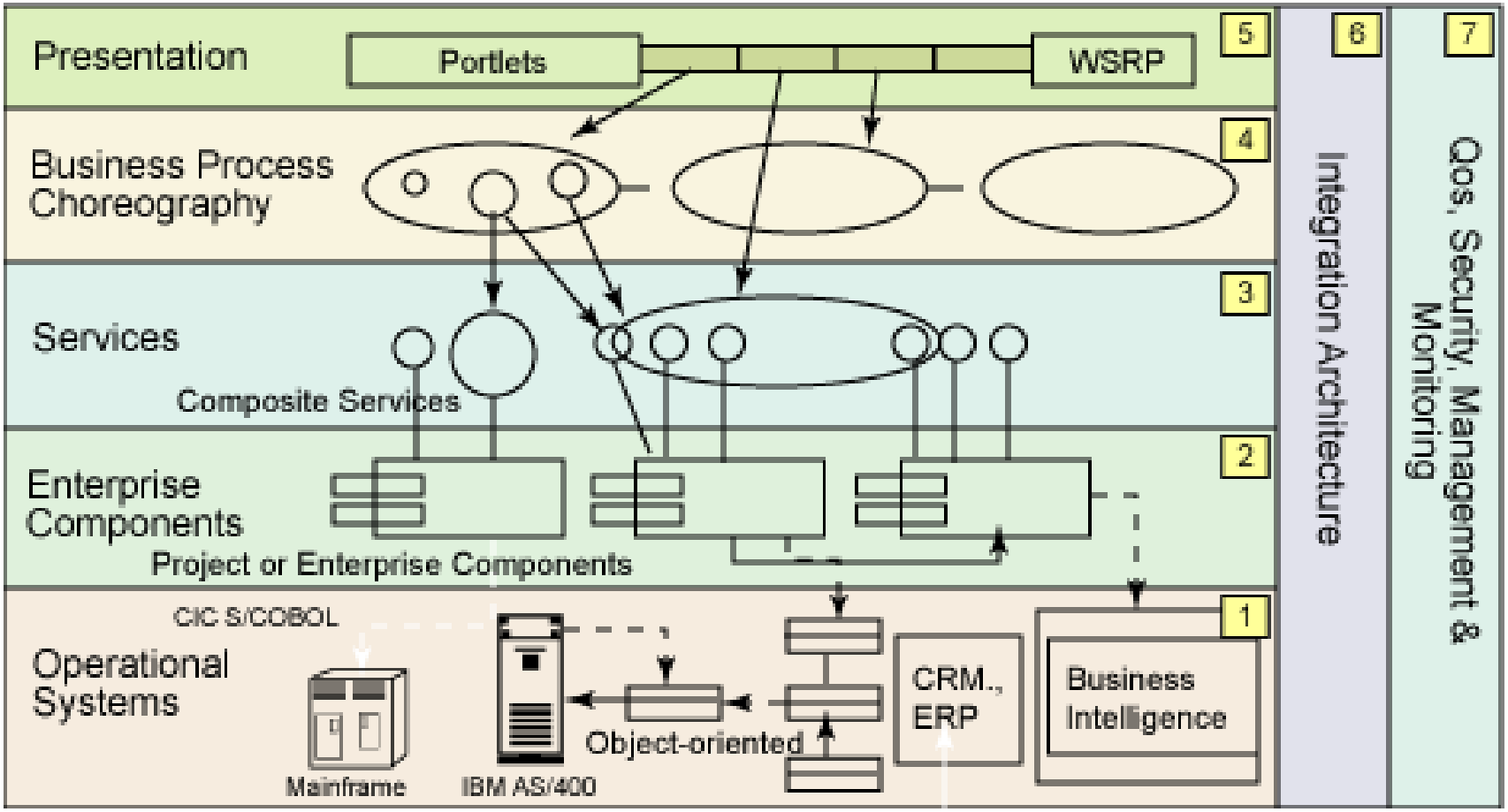
Service Oriented Architecture (SOA)

- La puesta en práctica del paradigma SOC requiere la implementación de una Arquitectura Orientada a Servicios (SOA)

- Una SOA es una forma lógica de diseñar un sistema de software para proveer servicios, a usuarios finales, aplicaciones u otros servicios, a través de interfaces públicas que pueden ser descubiertas.

Introducción

Capas de una SOA

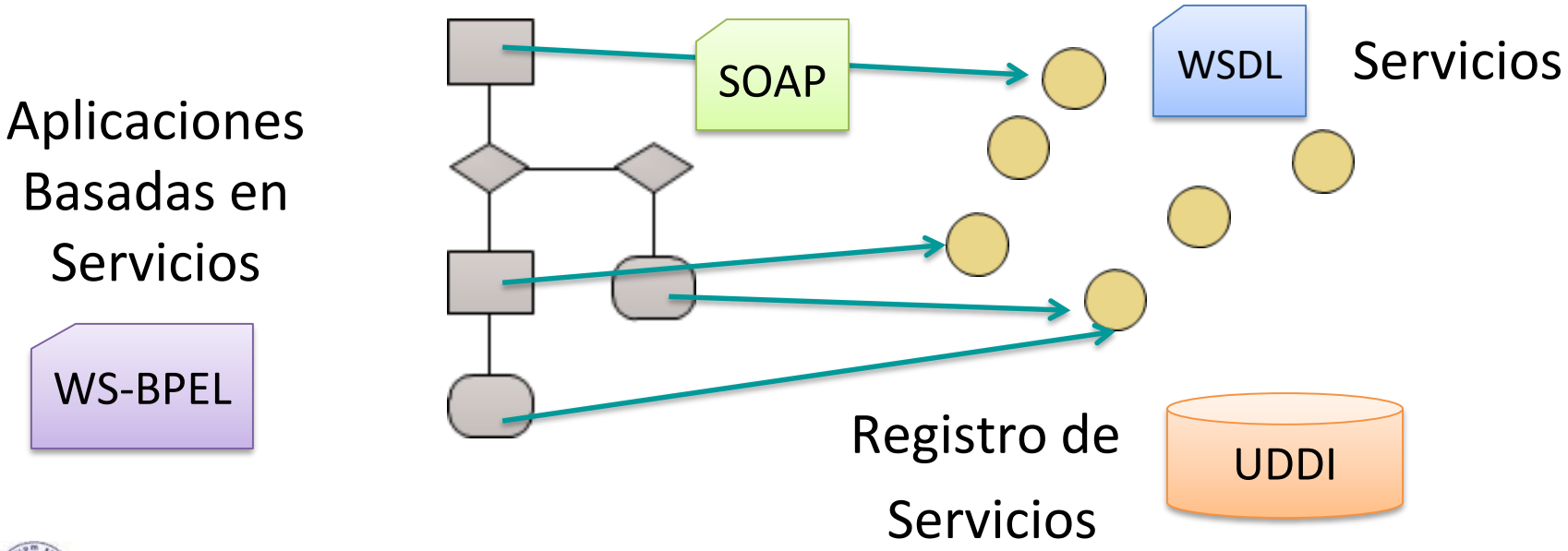


http://www.ibm.com/developerworks/library/ws-soa-design1/

Introducción

Web Services y SOA

- Si bien los principios de SOC no dependen de una tecnología en particular, dadas sus características, los Web Services se han convertido la tecnología preferida para implementar una SOA



Introducción

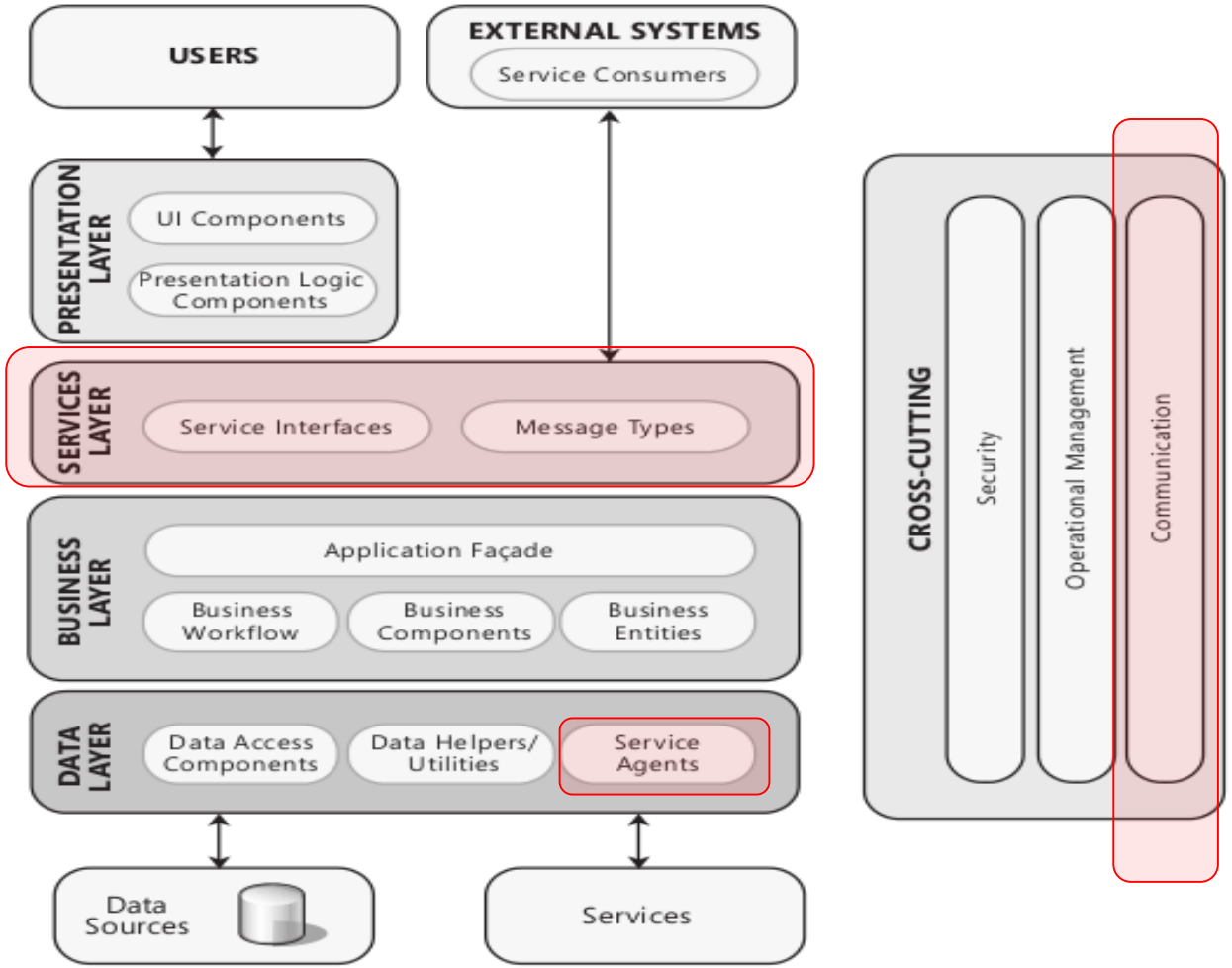
Web Services y SOA

- La tecnología de Web Services representa el uso de estándares y tecnologías para la invocación e interoperabilidad
- Los Servicios SOA son servicios que realizan una actividad clave de un proceso de negocios y se describen como servicios de negocio
- Estos servicios de negocio pueden ser expuestos como Web Services pero
  - Servicio SOA != Web Service

http://j2eearchitec.blogspot.com/2013/01/SOA-Basics-III.html

Introducción

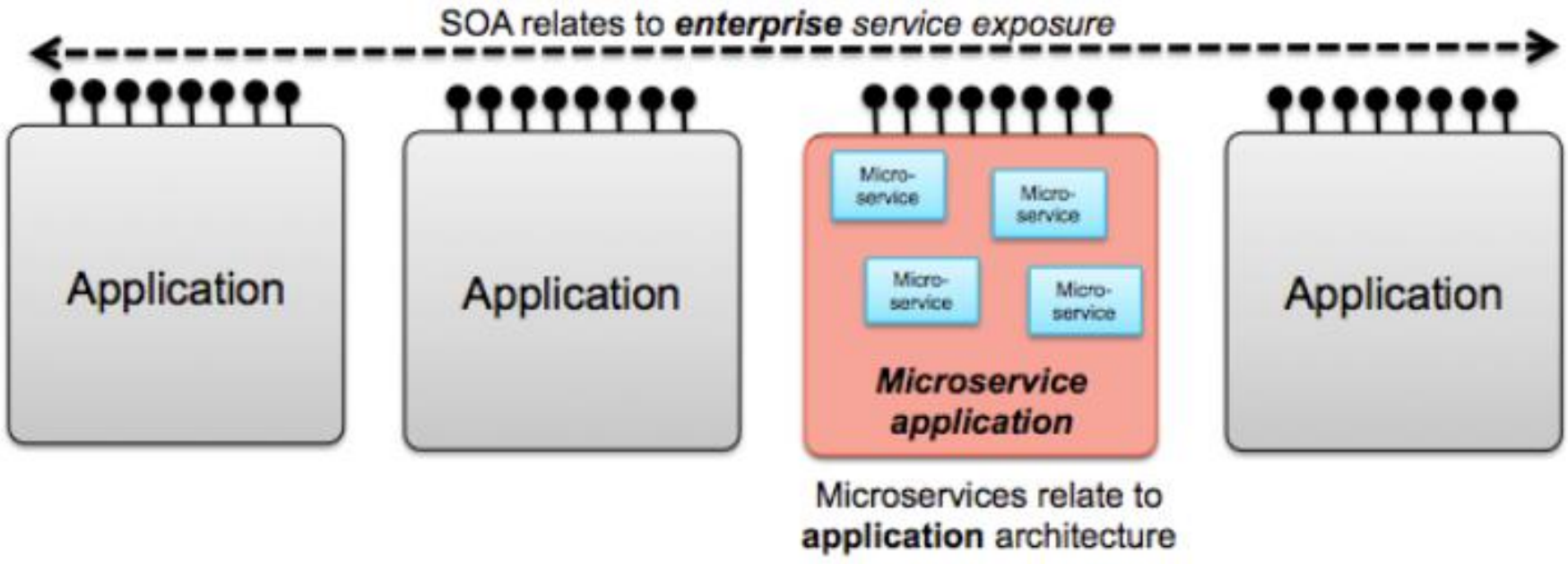
Web Services en Sistemas Empresariales



Microsoft Patterns & Practices. Microsoft Application Architecture Guide v2.0

Introducción

Y Microservicios?

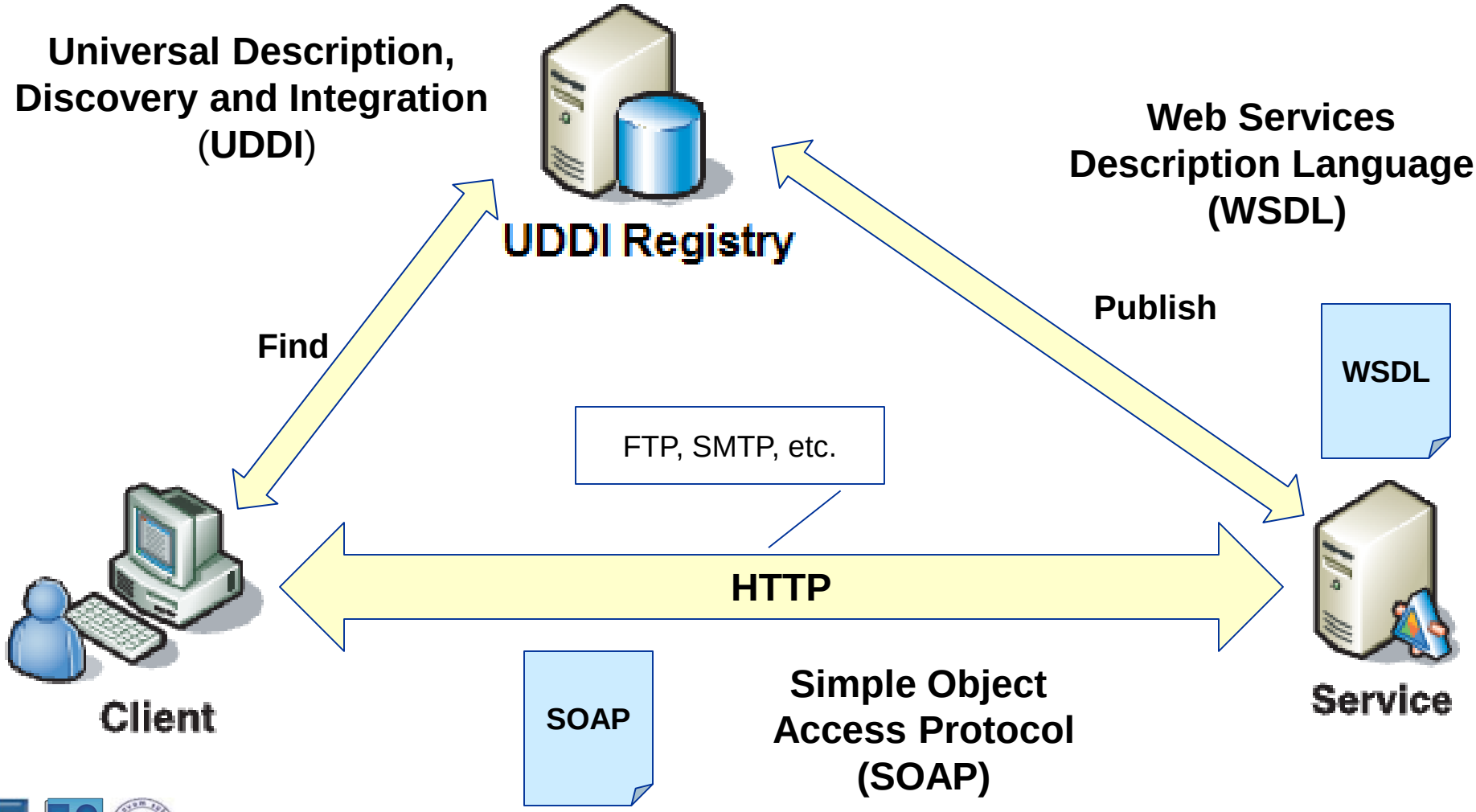


https://developer.ibm.com/tutorials/1601_clark-trs/

SOAP Web Services

SOAP Web Services

Primera Generación de Estándares



SOAP Web Services

Primera Generación de Estándares

- La tecnología de Web Services está construida sobre tres especificaciones básicas, basadas fuertemente en XML:
  - Web Services Description Language (WSDL)
  - Simple Object Access Protocol (SOAP)
  - Universal Description, Discovery and Integration (UDDI)
- Se los conoce como “primera generación de estándares de WS”

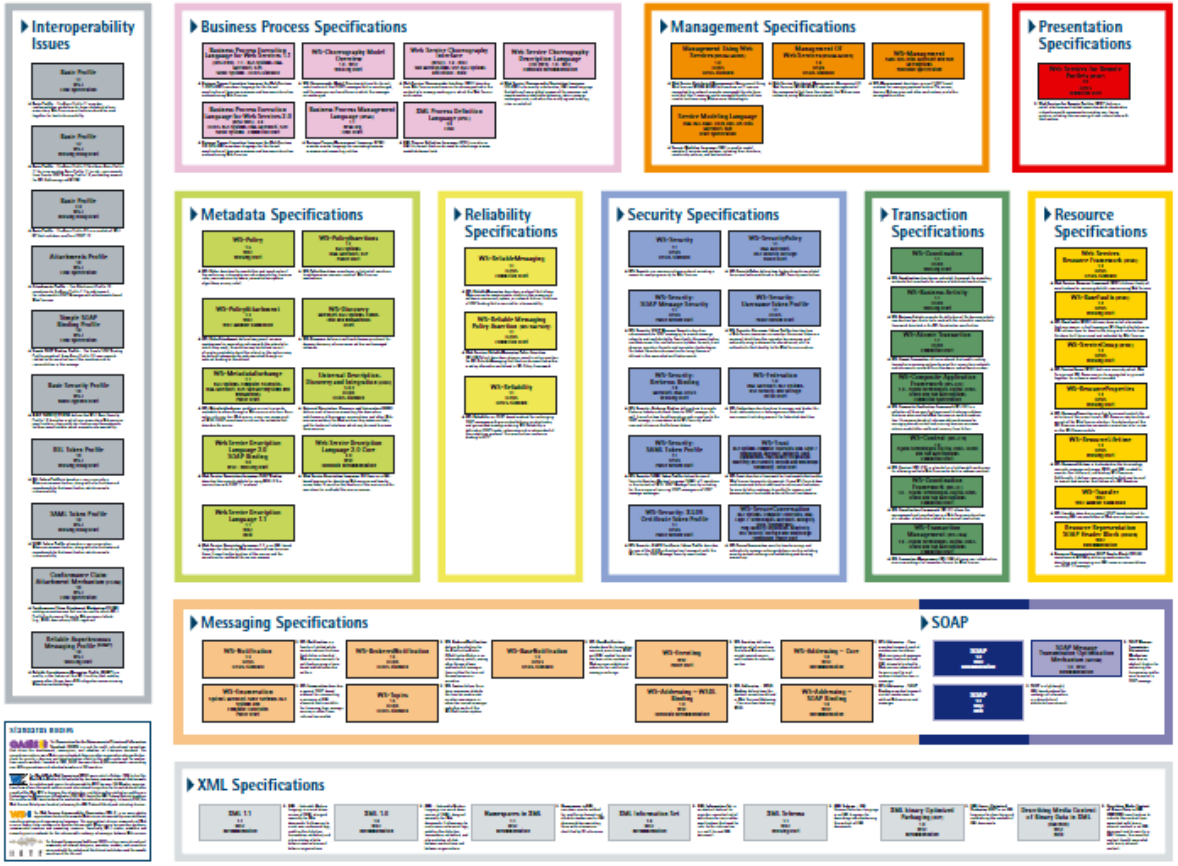
SOAP Web Services

Segunda Generación de Estándares

- Los estándares básicos no abordan problemáticas comunes en contextos empresariales
  - Confiabilidad, seguridad, transacciones, etc.
- Surgen entonces un conjunto de nuevas especificaciones (conocidas como WS-* o “segunda generación de estándares”)
- Cada una aborda una problemática específica y están orientadas a bloques y a su composición

SOAP Web Services

Segunda Generación de Estándares



http://www.innoq.com/soa/ws-standards/poster/

SOAP Web Services

Estándares

- Actualmente entonces, la tecnología de Web Services está basada en un gran número de especificaciones que:
  - en general, son propuestas por la industria
    - Microsoft, IBM, Oracle, etc.
  - son estandarizadas por distintas organizaciones
    - W3C, OASIS, etc.
  - son implementadas por distintos proveedores
    - Apache, JBoss, Sun, Microsoft, IBM, Oracle, etc.

SOAP Web Services

SOAP

- Simple Object Access Protocol

- Especificación que describe mecanismos y un formato de mensaje (basado en XML) para intercambiar información entre aplicaciones, en un ambiente distribuido y descentralizado

SOAP Web Services

SOAP: Mensaje

- La estructura básica de un mensaje SOAP consiste de un elemento "**Envelope**" el cual contiene
  - un elemento opcional "**Header**"
  - un elemento requerido "**Body**"
    - que puede incluir un elemento "**Fault**"

- El "Envelope" es el elemento raíz de todo mensaje SOAP e identifica un documento XML como un mensaje SOAP

SOAP Web Services

SOAP: Mensaje

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

SOAP Web Services

SOAP: Header del Mensaje

- El “Header”, de estar presente, debe ser el primer elemento del “Envelope”
- Provee un mecanismo de extensión que permite incluir información extra en mensajes SOAP (seguridad, transacciones, etc)
- Puede contener varios “header blocks” que son una forma de agrupar lógicamente la información

SOAP Web Services

SOAP: Body del Mensaje

- Contiene la información a ser intercambiada entre el cliente y servicio

- En el “Body” típicamente se especifica:
  - una solicitud para efectuar cierta operación
  - la respuesta a cierta solicitud que puede ser:
    - un resultado o
    - un error (fault)

SOAP Web Services

SOAP: Body del Mensaje - Fault

- El elemento “Fault”, contenido en el “Body”, indica una condición de error en el procesamiento del mensaje SOAP
- Tiene 5 sub-elementos:
  - Code
  - Reason
  - Detail
  - Node (opcional)
  - Role (opcional)

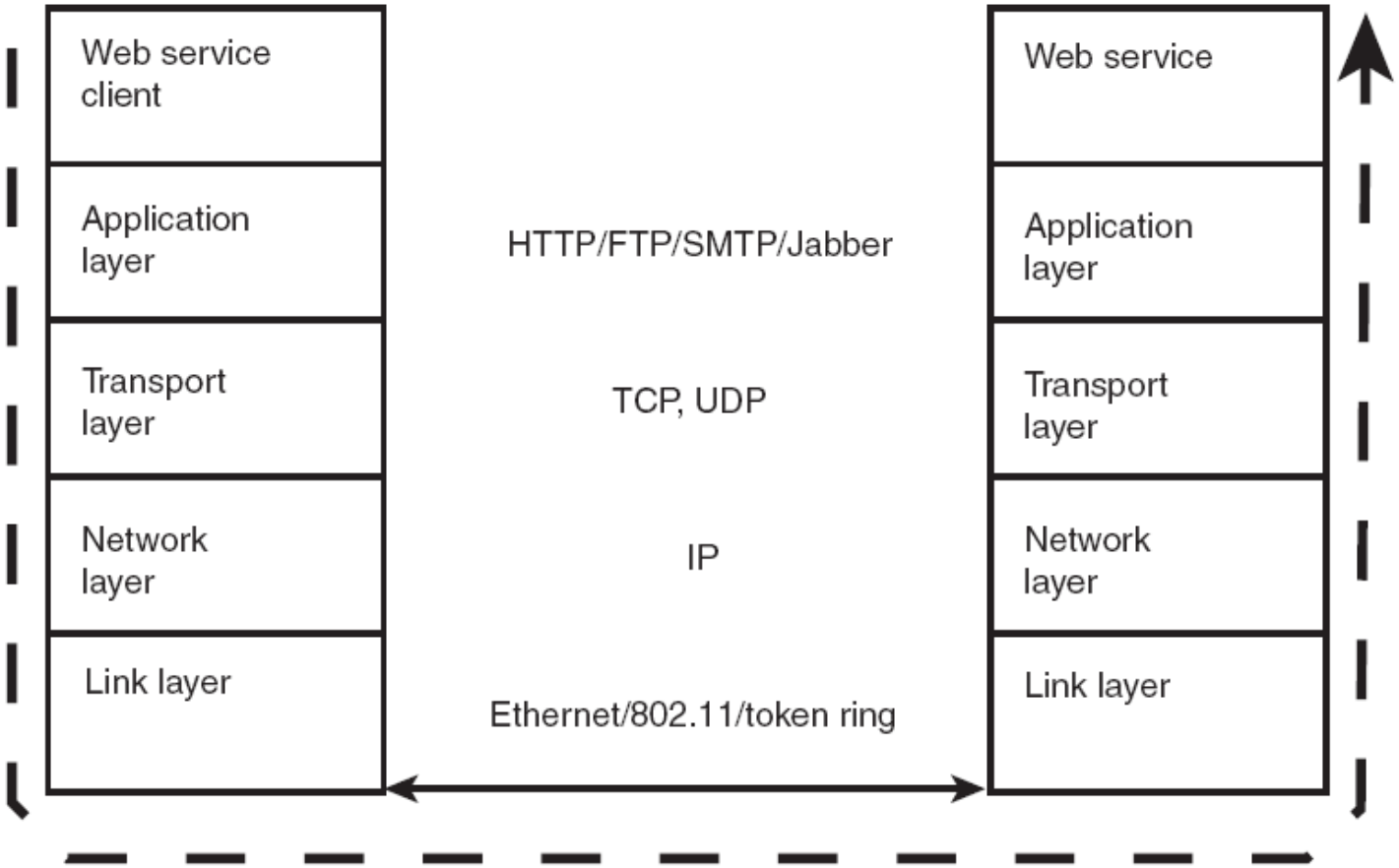
SOAP Web Services

SOAP: Transporte de Mensajes

- SOAP no impone el uso de un determinado protocolo para el intercambio de mensajes
- El concepto de "*binding*" SOAP permite especificar:
  - cómo los mensajes SOAP se encapsulan en un protocolo de transporte
  - cómo los mensajes SOAP deben ser tratados con las primitivas del protocolo
- SOAP incluye:
  - un "*binding*" para el protocolo HTTP
  - un conjunto de reglas para definir nuevos "*bindings*"

SOAP Web Services

SOAP: Transporte de Mensajes



Sams Teach Yourself Web Services in 24 Hours. Stephen Potts, Mike Kopack. Sams Publishing 2003.

SOAP Web Services

SOAP: Transporte de Mensajes

- Dado que en los mensajes SOAP no se incluye la dirección del WS destino, los *binding* SOAP tienen otra función implícita:
  - direccionamiento de los mensajes

- La forma de especificar el WS destino depende entonces del protocolo de transporte utilizado, por ejemplo:
  - HTTP: URL del recurso destino
  - SMTP: la dirección “to” en el header del e-mail

SOAP Web Services

WSDL

- Web Services Description Language

- Lenguaje basado en XML que permite describir la interfaz y otras características de un Web Service

- Un documento WSDL consta de dos partes:
  - descripción abstracta
  - descripción concreta

SOAP Web Services

WSDL – Descripción Abstracta

- La descripción abstracta describe de forma general la estructura de la interfaz del Web Service, que incluye operaciones, parámetros y tipos de datos abstractos
- Los cuatro elementos XML que componen la descripción abstracta son:
  - <wsdl:types>
  - <wsdl:message>
  - <wsdl:portType>
    - <wsdl:operation>

SOAP Web Services

WSDL – types

- El elemento "*types*" encapsula todas las definiciones abstractas de tipos de datos

```
<wsdl:types>
   <xsd:schema
   targetNamespace="http://..../weatherService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="City"
                type="xsd:string"/>
        <xsd:element name="Country"
                type="xsd:string"/>
   </schema>
<wsdl:types>
```

SOAP Web Services

WSDL – messages

- El elemento "*messages*" representa de forma abstracta los parámetros de entrada y salida para una operación

```
<message name="getWeatherIn">
   <part name="CityIn" element="tns:City"/>
   <part name="CountryIn" element="tns:Country"/>
</message>
```

SOAP Web Services

WSDL – portType

- El elemento “portType” es el contenedor de todas las operaciones abstractas y describe una interfaz específica del servicio

```
<portType name="getWeatherPortType">
   <operation name="getWeather">
        <input message="getWeatherIn"/>
        <output message="getWeatherOut"/>
   </operation>
</portType>
```

SOAP Web Services

WSDL – Descripción Concreta

- La descripción concreta asocia a una descripción abstracta una dirección de red concreta, un protocolo de comunicación y estructuras de datos concretas
- Los tres elementos XML que componen la descripción concreta son:
  - <wsdl:binding>
  - <wsdl:service>
    - <wsdl:port>

SOAP Web Services

WSDL – binding

- Este elemento asocia un “portType”, y sus mensajes y operaciones, a un protocolo de transporte y un formato de mensaje

```
<binding name="weatherServiceSoapBinding"
        type="getWeatherPortType" >
        <soap:binding style="rpc" transport="http"/>
        <operation name="getWeather">
             <input soap:body use="encoded"/>
             <output soap:body use="encoded"/>
        </operation>
</binding>
```

https://www.ibm.com/developerworks/library/ws-whichwsdl/

SOAP Web Services

WSDL – service y port

- El elemento "*port*" especifica la dirección de red para un determinado "*binding*"
- El elemento "*service*" es un contenedor de elementos "*port*"

```
<service name="WeatherService">
   <port name="weatherServicePort"
        type="tns:weatherServiceSoapBinding">
        <soap:address
              location="http://.../weather/"/>
   </port>
</binding>
```

SOAP Web Services en Java EE

SOAP Web Services en Java EE

Implementación de SOAP Web Services: Ejemplo

```
import javax.jws.WebService;

@WebService
public class CitizenBasicInformation {

    public Person getCitizenInformation(Document doc) {
```

```
import java.util.Date;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Person {
    private String name;
    private String lastname;
    private Date birthdate;
```

```
@XmlRootElement
public class Document {
    private String type;
    private String number;
```

SOAP Web Services en Java EE

Implementación de SOAP Web Services: Ejemplo

```
<wsdl:types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://tse.i
    <xs:element name="document" type="tns:document"/>
    <xs:element name="getCitizenInformation" type="tns:getCitizenInformation"/>
    <xs:element name="getCitizenInformationResponse" type="tns:getCitizenInforma
    <xs:element name="person" type="tns:person"/>
    <xs:complexType name="getCitizenInformation">
      <xs:sequence>
        <xs:element minOccurs="0" name="arg0" type="tns:document"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="document">
      <xs:sequence>
        <xs:element minOccurs="0" name="number" type="xs:string"/>
        <xs:element minOccurs="0" name="type" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
```

SOAP Web Services en Java EE

Implementación de SOAP Web Services: Ejemplo

```
<wsdl:portType name="CitizenBasicInformation">
  <wsdl:operation name="getCitizenInformation">
    <wsdl:input message="tns:getCitizenInformation" name="getCitizenInform
    <wsdl:output message="tns:getCitizenInformationResponse" name="getCiti
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="CitizenBasicInformationServiceSoapBinding" type="tns:CitizenBasi
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:service name="CitizenBasicInformationService">
  <wsdl:port binding="tns:CitizenBasicInformationServiceSoapBinding" name="CitizenBasicIn
    <soap:address location="http://localhost:8080/CitizenInfoWS/CitizenBasicInformation"/
  </wsdl:port>
```

SOAP Web Services en Java EE

Implementación de SOAP Web Services: Ejemplo

```
<soapenv:Envelope xmlns:soapenv="http:/
   <soapenv:Header/>
   <soapenv:Body>
      <tse:getCitizenInformation>
         <arg0>
            <number>12345678</number>
            <type>CI</type>
         </arg0>
      </tse:getCitizenInformation>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/en
   <soap:Body>
      <ns2:getCitizenInformationResponse xmlns:ns2="http://ts
         <return>
            <birthdate>2018-04-17T13:51:40.937-03:00</birthda
            <lastname>Smith</lastname>
            <name>John</name>
         </return>
      </ns2:getCitizenInformationResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP Web Services en Java EE

Implementación de SOAP Web Services

- la clase de implementación debe ser anotada con @javax.jws.WebService
- la clase puede implementar cero o más interfaces que tienen que anotarse con @WebService.
- la clase debe ser pública y no debe ser "final" o "abstract"
- la clase debe tener un constructor por defecto público
- la clase no debe definir el método “finalize”

SOAP Web Services en Java EE

@WebService

- la annotation @javax.jws.WebService define una clase o interfaz Java como un Web Service
- Ejemplo usando interfaces:

```
@WebService
public interface CitizenBasicInformationInterface {

    public Person getCitizenInformation(Document doc);
}
```

```
@WebService(endpointInterface="uy.edu.fing.inco.tse.CitizenBasicInformationInterface")
public class CitizenBasicInformation {

    public Person getCitizenInformation(Document doc) {
```

SOAP Web Services en Java EE

@WebService

- la annotation @WebService tiene un conjunto de atributos que permiten configurar varios aspectos del Web Service
  - nombre, namespace, ubicación del WSDL, etc

```
@WebService(serviceName = "InfoPersonas")
public class CitizenBasicInformation {

    public Person getCitizenInformation(Document doc) {
```

SOAP Web Services en Java EE

@WebMethod

- Por defecto, todos los métodos públicos de la clase de implementación del Web Service (o de su interfaz):
  - se exponen como operaciones del Web Service
  - utilizan reglas de *mapping* por defecto
- La anotación @WebMethod permite especificar algunos aspectos de este *mapping*

SOAP Web Services en Java EE

@WebMethod

- Por defecto, todos los métodos públicos de la clase de implementación del Web Service (o de su interfaz):
  - se exponen como operaciones del Web Service
  - utilizan reglas de *mapping* por defecto
- La anotación @WebMethod permite especificar algunos aspectos de este *mapping*

SOAP Web Services en Java EE

@WebMethod

```
@WebService(serviceName = "InfoPersonas")
public class CitizenBasicInformation {

    @WebMethod (operationName="obtenerDatos")
    public Person getCitizenInformation(Document doc) {


    @WebMethod(exclude=true)
    public String getVersion() {
```

```
<wsdl:portType name="CitizenBasicInformation">
  <wsdl:operation name="obtenerDatos">
    <wsdl:input message="tns:obtenerDatos" name="obtenerDatos"></wsdl:in
    <wsdl:output message="tns:obtenerDatosResponse" name="obtenerDatosRe
  </wsdl:operation>
</wsdl:portType>
```

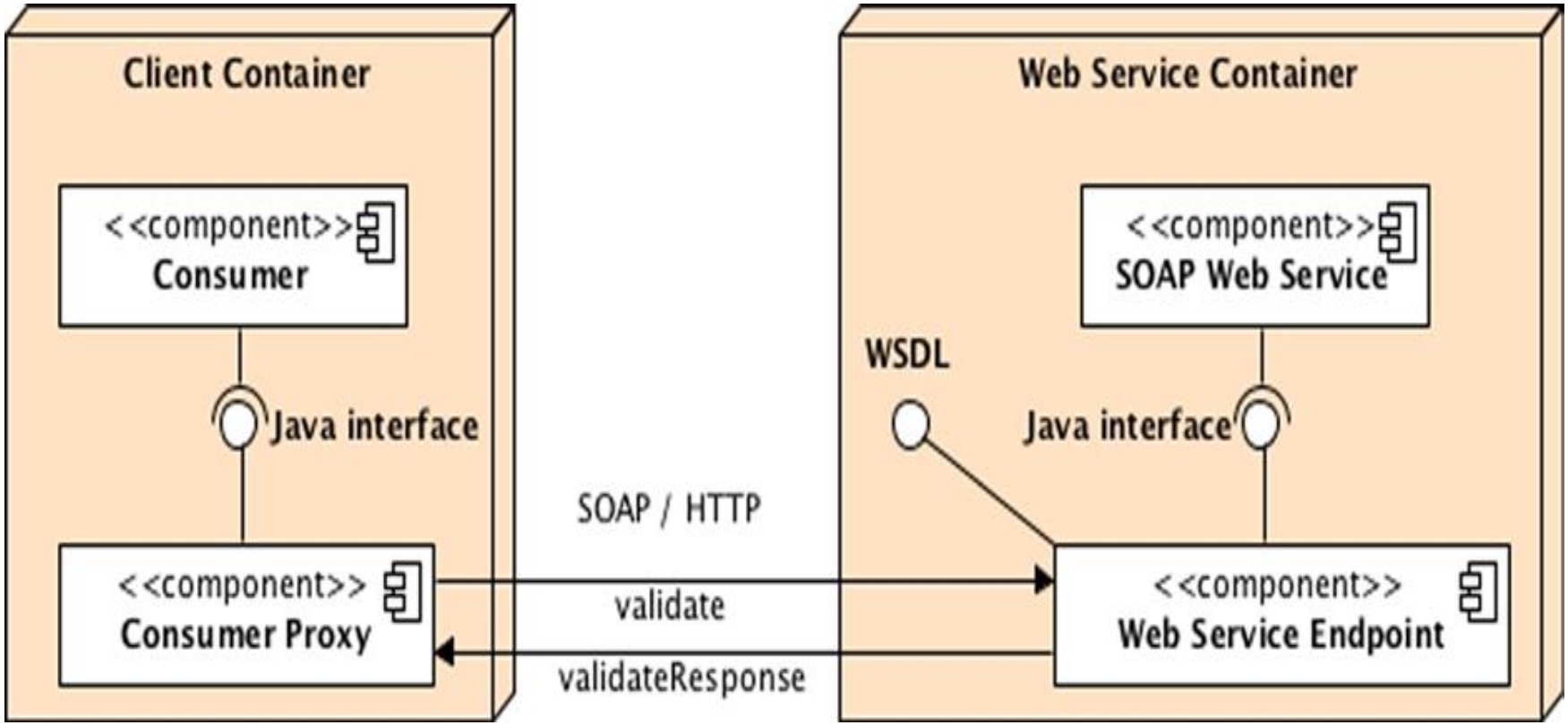
SOAP Web Services en Java EE

Otras anotaciones

- @WebResult
  - Permite customizar el nombre del mensaje devuelto como resultado de la operación
- @WebParam
  - Permite especificar aspectos de los parámetros que son enviados al servicio (p. ej. nombre, tipo)
- @OneWay
  - Permite indicar que un método no retorna valores (p. ej. métodos que retornan void)
  - De esta forma el contenedor puede realizar optimizaciones con los métodos de este tipo

SOAP Web Services en Java EE

Invocando un web service JAX-WS



(Goncalves, 2013)

SOAP Web Services en Java EE

Invocación programática

- Desde fuera del container, se puede invocar programáticamente

```
InfoPersonas info = new InfoPersonas();
CitizenBasicInformation port = info.getCitizenBasicInformationPort();

Document documento = new Document();
documento.setType("CI");
documento.setNumber("12345678");

Person persona = port.obtenerDatos(documento);
System.out.println(persona.getName());
```

SOAP Web Services en Java EE

Invocación programática

- Requiere que se hayan generado los artefactos Java a partir del WSDL del servicio web
- Se puede utilizar
  - la herramienta wsimport provista por Java estándar
    - wsimport <URL al WSDL del servicio>
  - herramientas provistas por el IDE
- Los artefactos Java generados (clases) deben ser incluidas en el classpath del proyecto

SOAP Web Services en Java EE

Invocación con inyección

- Si el cliente está dentro del contenedor se puede utilizar injection para obtener una referencia al SOAP Web Service
  - @WebServiceRef

RESTful Web Services

RESTful Web Services

Motivación

- WS-* es muy completo pero
  - Es complejo
    - Gran cantidad de estándares
    - Existen muchos más a los vistos en el curso
  - Es pesado
    - Recordar serialización SOAP
- Es necesario algo más simple y liviano
  - REST!

RESTful Web Services

REST (REpresentational State Transfer)

- Es un estilo arquitectónico para aplicaciones que utilizan hipermedia interconectada
- Todo es tratado como RECURSOs que se identifican por URIs (Uniform Resource Identifier).

- Es aplicado para la construcción de servicios web, livianos, mantenibles y altamente escalables
- Un Web Service basado en REST se denomina RESTful Web Service

RESTful Web Services

Recursos

- Un recurso es cualquier cosa con la cual un cliente quiera interactuar, y que sea identificable a través de un hipervínculo
  - un libro
  - un perfil
  - un resultado de una búsqueda, etc.
- Puede almacenarse en cualquier lado
  - archivo
  - base de datos, etc.

RESTful Web Services

URIs

- Un recurso se identifica por una URI (Uniform Resource Identifier)
- Una URI es un identificador único para un recurso
- Una URI debe ser lo mas descriptiva posible y apuntar a un único recurso

RESTful Web Services

URIs

- El formato estándar de una URI es el siguiente:
  - http://host:port/path?queryString#fragment

- Un ejemplo:
  - http://www.weather.com:8080/weather/2013/01/01?location=Lisbon,Portugal&time=morning

(Goncalves, 2013)

RESTful Web Services

URIs

- Generalmente las URIs para acceder a recursos tienen el siguiente formato:
  - Protocol://ServiceName/ResourceType/ResourceID

- Algunas recomendaciones
  - Usar sustantivos en plural para nombrar un recurso
  - Evitar los espacios, usar "_" o "–"
  - Las URIs son sensibles al case.

- Evitar los verbos para describir la URI de un recurso
  - http://myservice/FetchPerson/1, http://myservice/DeletePerson/1
  - http://myservice/pagos/confirmar

RESTful Web Services

URIs y query parameters

- Se utilizan query parameters para:
  - Filtros en búsquedas:
    - http://myservice/personas?name=juan
  - Ordenamiento de resultados
    - http://myservice/personas?sort=age
  - Personalizar la respuesta
    - http://myservice/personas/1?format=xml

- NO se deben utilizar para identificar recursos
  - http://myservice/personas?id=1

RESTful Web Services

Representaciones de Recursos

- Cuando un cliente interactúa con un recurso, siempre lo hace a través de representaciones del mismo
- El recurso SIEMPRE existe en el servidor
- Una representación es cualquier información acerca del estado de un recurso

RESTful Web Services

Representaciones de Recursos

- Un recurso puede tener múltiples representaciones

- Existen dos formas de seleccionar la representación que se quiere:
  - A través de una URL especifica
    - http://www.apress.com/java
    - http://www.apress.com/java/csv
    - http://www.apress.com/java/xml
  - A través de negociación de contenido

RESTful Web Services

Representaciones de Recursos

- Recurso: Persona
- Representación:
  - JSON: Deseable para que una página web realice invocaciones AJAX.
  - XML: Deseable para algunas comunicaciones B2B

```
{
    "Id": "1",
    "Name": "Pablo",
    "Email": "pablo@gmail.com",
    "Country": "Uruguay"
}
```

```
<Persona>
    <Id>1</Id>
    <Name>Pablo</Name>
    <Email>pablo@gmail.com</Email>
    <Country>Uruguay</Country>
</Persona>
```

RESTful Web Services

Conectividad

- Si dos recursos se encuentran fuertemente relacionados (a nivel conceptual), entonces debe existir un enlace (link) entre estos

- Los web services REST deben aprovechar la facilidad de los hyperlinks, para informar a los clientes que hay disponible más información y cómo debe hacerse para acceder a ésta

RESTful Web Services

Conectividad

- Por ejemplo, si se obtiene la información de un CD, se puede acceder a información relacionada:

```
<cd>
  <title>Ella and Louis</title>
  <year ref="http://music.com/year/1956">1956</year>
  <artist ref="http://music.com/artists/123">Ella Fitzgerald</artist>
  <artist ref="http://music.com/artists/456">Louis Armstrong</artist>
  <link rel="self" type="text/json" href="http://music.com/album/789"/>
  <link rel="self" type="text/xml" href="http://music.com/album/789"/>
  <link rel="http://music.com/album/comments" type="text/xml" ➥
      href="http://music.com/album/789/comments"/>
</cd>
```

RESTful Web Services

Negociación de Contenido

- Consiste en elegir la mejor representación cuando para un mismo recurso existen múltiples representaciones disponibles

- Esta basada en los siguientes headers del request:
  - Accept, Accept-Charset
  - Accept-Encoding, Accept-Language
  - User-Agent

RESTful Web Services

Negociación de Contenido

- Por ejemplo, si se utiliza la siguiente URI para acceder a información sobre libros de Java
  - http://www.apress.com/java

- Se puede indicar
  - En Accept el media type text/csv, para indicar que queremos una representación CSV de los datos
  - Podemos usar Accept-Language para indicar “en”, para a su vez pedir el CSV en ingles

RESTful Web Services

- La intención de una llamada a un RESTful Web Service, se obtiene del verbo HTTP
  - GET (recuperar), DELETE (eliminar)…

Verbo HTTP	Significado en términos de CRUD (Create, Read, Update, Delete)
POST	Crear un nuevo recurso a partir de los datos de la solicitud.
GET	Leer un recurso.
PUT	Actualizar un recurso a partir de los datos de la solicitud.
DELETE	Eliminar un recurso.

Java Web Services: Up and Running, 1st Edition. Martin Kalin. O'Reilly. 2009

RESTful Web Services

- De este modo las URIs actúan como identificadores de recursos y los métodos HTTP como verbos que especifican operaciones sobre los mismos

Verbo HTTP / URI	Significado en términos de CRUD
POST emps	Crear un nuevo empleado a partir de los datos de la solicitud.
GET emps	Leer una lista de todos los empleados.
GET emps/27	Leer el empleado 27
PUT emps	Actualizar la lista de empleados con los datos de la solicitud.
DELETE emps	Eliminar la lista de empleados.
DELETE emps/27	Eliminar el empleado 27.

Java Web Services: Up and Running, 1st Edition. Martin Kalin. O'Reilly. 2009

RESTful Web Services en Java EE

Instituto de Computación

Facultad de Ingeniería

Universidad de la República de Uruguay

Restful Web Services en Java EE

Java API for RESTful Web Services – JAX-RS

- Para implementar un Restful Web Service, solo se necesita un cliente y un servidor que soporte HTTP
- Como en el caso de SOAP, a fin de eliminar el trabajo con protocolos de bajo nivel, aparece el API JAX-RS
- La implementación de referencia de esta API, se denomina Jersey (es un proyecto open source)

Restful Web Services en Java EE

Java API for RESTful Web Services – JAX-RS

- JAX-RS es un API que permite especificar un recurso, en base a un POJO
- Para esto, se utiliza la anotación @javax.ws.rs.Path sobre una clase Java que representa un recurso

```
@Path("/book")
public class Libro {

    @GET
    @Produces("text/plain")
    public String getTituloLibro() {
        return "ABCD";
    }

}
```

Restful Web Services en Java EE

Java API for RESTful Web Services – JAX-RS

- En el ejemplo anterior, el recurso Libro está publicado en la URI /book

- El método getTituloLibro() queda asociado al método GET HTTP, produciendo un contenido de tipo “text/plain”

- Para acceder al recurso, solo se debe usar un browser con la dirección http://www.myserver.com/book

Restful Web Services en Java EE

Java API for RESTful Web Services – JAX-RS

- El servicio REST no implementa ninguna interfaz ni extiende ninguna clase
- Se debe usar la anotación @Path
- La clase:
  - debe ser public
  - no debe ser abstract ni final
  - debe contener un constructor por defecto
  - no debe incluir el método finalize()

Restful Web Services en Java EE

Interfaz de acceso al recurso

- JAX-RS define una serie de anotaciones, para indicar qué método HTTP se utiliza para acceder a un recurso
- Estas anotaciones corresponden a los métodos HTTP disponibles
  - @GET, @POST, @PUT, @DELETE, @HEAD, @OPTIONS
- Solo los métodos públicos de una clase pueden exponerse como métodos de recurso

Restful Web Services en Java EE

Interfaz de acceso al recurso

```
@Path("/library")
@Consumes({ "application/json" })
@Produces({ "application/json" })
public class Library {

  @GET
  @Path("/books")
  public Collection<Book> getBooks() {

  @GET
  @Path("/book/{isbn}")
  public Book getBook(@PathParam("isbn") String id) {

  @PUT
  @Path("/book/{isbn}")
  public Book addBook(@PathParam("isbn") String id, @QueryParam("title") String title) {

  @POST
  @Path("/book/{isbn}")
  public Book updateBook(@PathParam("isbn") String id, String title) {

  @DELETE
  @Path("/book/{isbn}")
  public Book removeBook(@PathParam("isbn") String id) {
```

Restful Web Services en Java EE

Consumiendo y Produciendo Tipos de Contenido

- las anotaciones @javax.ws.rs.Consumes and @javax.ws.rs.Produces
  - pueden aplicarse a un recurso con varias representaciones
  - definen los media types de la representación intercambiada entre el cliente y servidor

```
@Path("/library")
@Consumes({ "application/json" })
@Produces({ "application/json" })
public class Library {
```

Restful Web Services en Java EE

Parámetros

- @PathParam
  - Permite extraer “pedazos” de la URL
- @FormParam
  - Permite extraer información de un formulario
- @QueryParam
  - Permite extraer el valor del query parameter con el nombre indicado

Restful Web Services en Java EE

Parámetros

```
@Path("/library")
@Consumes({ "application/json" })
@Produces({ "application/json" })
public class Library {

    @GET
    @Path("/books")
    public Collection<Book> getBooks() {

    @GET
    @Path("/book/{isbn}")
    public Book getBook(@PathParam("isbn") String id) {

    @PUT
    @Path("/book/{isbn}")
    public Book addBook(@PathParam("isbn") String id, @QueryParam("title") String title) {

    @POST
    @Path("/book/{isbn}")
    public Book updateBook(@PathParam("isbn") String id, String title) {

    @DELETE
    @Path("/book/{isbn}")
    public Book removeBook(@PathParam("isbn") String id) {
```

Restful Web Services en Java EE

Client API: Invocando Restful Web Services

- Antes de JAX-RS 2.0 no existía una forma estándar de invocar servicios web REST
- Esta API permite realizar invocaciones a servicios REST a través de HTTP en forma simple
- Las clases necesarias para construir un cliente REST se encuentran en el package javax.ws.rs.client

Restful Web Services en Java EE

Client API: Invocando Restful Web Services

```
Client client = ClientBuilder.newClient();

WebTarget target = client.target("http://localhost:8080/jaxrs-sample/book");

Invocation invocation = target.request().buildGet();

Response response = invocation.invoke();

Book book = response.readEntity(Book.class);
```

Referencias

- Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. Java Platform, Enterprise Edition. The Java EE Tutorial, Release 7. Oracle. 2014. https://docs.oracle.com/javaee/7/tutorial/

- Antonio Goncalves. Beginning Java EE 7. Apress. 2013. https://link.springer.com/book/10.1007/978-1-4302-4627-5

- Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, L. U., Liu, K., … Pasley, J. (2008). Web Service Contract Design and Versioning for SOA (1st ed.). Prentice Hall.