Web Services

Taller de Sistemas de Información 2

LINS
(Laboratorio de Integración de Sistemas)

Ing. Fernando Rodríguez

InCo – UdelaR

Problemática hoy …

- Auge de “component-based programming”
- Integración de componentes de distintos proveedores.
- Respuestas de la industria: CORBA-EJB y DCOM.
- Problemas: uso de protocolos propietarios (conflicto con los mecanismo de seguridad de hoy) y orientados a conexión (complica el manejo de interrupciones en la red, es decir complica el balanceo de carga).

Cuales son los requerimientos ?

- Interoperabilidad
- Solución “Internet-conciente”.
- Interfaces fuertemente tipadas.
- Capacidad de mejorar los estándares de Internet existentes.
- Soporte de múltiples lenguajes.
- Soporte de cualquier infraestructura de distribución de componentes (CORBA o DCOM).

Alternativas iniciales

- CIS (COM Internet Services): permite conexiones DCOM entre cliente y servidor por el puerto 80. No tuvo difusión.
- CORBA via HTTP.

Web Services: definición

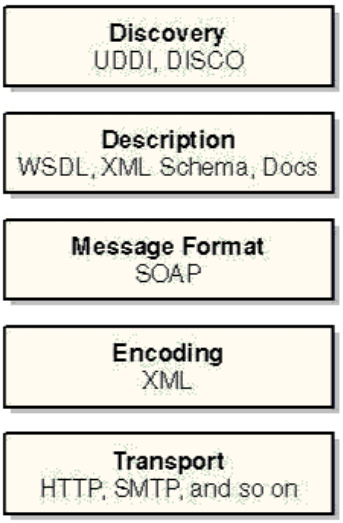
Es la nueva generación en Computación Distribuida.

- Es una colección de funciones que están empaquetadas como una unidad y es publicada en la Red por otros programas.
- Los Web Services son building blocks (bloques constructivos) que permiten crear otros sistemas distribuidos abiertos, y permiten, rápidamente y en forma barata, a compañías e individuos ofrecer sus aportes al resto del mundo.

Características deseadas en un Web Service

- Confiabilidad.
- Alta disponibilidad.
- Tolerancia a fallas.
- Escalable.
- Buena performance.

Building-Blocks necesarios para facilitar la comunicación remota entre el Web Service y sus clientes



Transporte

- HTTP es un protocolo ideal para configuraciones de alta disponibilidad, ya que es intrínsicamente sin estados.
- SMTP es un protocolo ideal para comunicaciones asíncronas. Si se cae el servicio el sistema de mensajería hace los reintentos necesarios en forma automática. Incluso SMTP sirve para no “atorar” a quien atiende solicitudes gracias al encolamiento.
- MSQS es una alternativa de “encolamiento inteligente”, porque los mensajes pueden ser colocados y quitados de la cola dentro del alcance de una transacción. Si una transacción aborta mientras esta sacando el mensaje del MSQS, automáticamente se devuleve el mismo a la cola.

Codificación

- Hay 2 grandes orientaciones en este sentido: text-based y binary-based.
- XML es la solución natural entre los protocolos disponibles a nivel de Internet, porque posee soporte multiplataforma, un sistema de tipos común y soporte de set de caracteres estándares.
- Algunos transporte como SMTP, solo pueden contener texto, por lo que XML es adecuado para éstas situaciones.

Formateo

- En general además del cuerpo del mensaje, en una comunicación, se envía información de metadata. XML no provee ningún mecanismo que diferencie el header del cuerpo del mensaje – esto lo hace SOAP.
- SOAP provee un mecanismo independiente del protocolo, que use para comunicarme, para asociar el header al cuerpo del mensaje: esto es el “envelope”.
- SOAP no impone ninguna restricción acerca de cómo debe formatearse el cuerpo del mensaje.
- Existen formas estándares para formatear un mensaje si se utiliza para invocaciones estilo RPC.

Descripción

- SOAP no provee la información adicional para que el cliente serialice la solicitud e interprete la respuesta. Esto lo hace los XML Schema.
- Un Schema provee un conjunto de definiciones de tipos y elementos predefinidos (también puedo crear otros). Esto permite, además de comunicar el tipo de los datos que se espera vengan dentro del mensaje, validar la información que se recibe o envía.
- No obstante, el Schema no incluye información de que transporte se va a usar, o cual es la interfase para interactuar con el Web Service. Para ello tenemos WSDL (Web Services Description Language).

Directorio

- UDDI: directorio centralizado para publicación y búsqueda de Web Services.
- DISCO: mecanismo propietario para publicar los Web Services publicado en un Web Site. El principal consumidor de DISCO es VisualStudio.NET.

Qué es lo que falta en Web Services ?

- Elementos no definidos en Web Services, que si lo están en otras infraestructuras de componentes distribuidos:
  - API específica de Web Services para inicializar runtime, instanciar componentes y reflejar la metadata que describe los componentes
  - Manejo de tiempo de vida de los objetos, pooling de objetos y soporte para transacciones distribuidas.

Ciclo de vida de un Web Service y alternativas al desarrollar

Escenarios más comunes al desarrollar un Web Service:

• *"Green field":* desarrollador comienza de 0, implementando la aplicación y luego el Web Service asociado.

• *"Botton-up":* es el caso cuando tengo un aplicativo legado que quiero adaptar a la arquitectura de Web Services.

• *"Top-down":* tengo la definición de un Web Service (en WSDL), y tengo que implementar la aplicación que resuelve la funcionalidad.

• *"Meet-in-the-middle":* dispongo de el aplicativo y la definición del Web Service (en WSDL), y hay que vincularlos. En general se utilizan bridges que adaptan la interface de uno al otro.

SOAP

SOAP

- SOAP es una aplicación de la especificación XML. SOAP = XML messaging.
- Fue aceptado por todos los pesos pesados de la industria (Microsoft, SUN, IBM, BEA, SAP, etc)
- Si bien existen alternativas a SOAP, como SUN RPC, Microsoft DCE, Java RMI y CORBA ORPC, todas ellas son propietarias.

http://www.w3.org/TR/SOAP

Ventajas de SOAP

- No esta ligado a ningún lenguaje.
- No esta ligado a ningún protocolo.
- No esta ligado a ninguna infraestructura de distribución de objetos.
- Re-usa estándares de la industria (“no reinventa la reuda”).
- Habilita la interoperabilidad entre distintos entornos.

Mensaje SOAP

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Optional header information goes here. -->
    <To>Scott</To>
    <From>Suzanne</From>
  </soap:Header>
  <soap:Body>
    <!--Message goes here. -->
    Please pick up some milk on your way home from work.
  </soap:Body>
</soap:Envelope>
```

- El SOAP Header es opcional.
- El SOAP Body contiene el mensaje en cuestión a ser procesado.

Mensaje SOAP (2)

- Un mensaje SOAP esta compuesto por un envelope. El mismo a su vez se divide en un header y un body (c/u como elementos de un documento XML, cuyo elemento Root es Envelope justamente).
- Header y body a su vez pueden tener elementos hijos.
- Cada elemento SOAP, tiene el prefijo soap porque todos están en el mismo namespace. Este referencia al SOAP schema donde se define el Envelope.

Elemento Header

- Puede tener información de:
  - Encoding / Compression algorithms
  - Autenticación
  - Firmas digitales
  - Información de ruteo.
  - Transacciones.
  - Información de tarifación.
- Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Digest>B839D234A3F87</Digest>
  </soap:Header>
  <soap:Body>
    <StockReport>
      <Symbol>MSFT</Symbol>
      <Price>74.56</Price>
    </StockReport>
  </soap:Body>
```

Elemento Header (2)

- Como el header es opcional, el que lo recibe puede ignorarlo. Para obligar que sea procesado un determinado elemento, debo especificar el atributo mustUnderstand en 1 para ese elemento.
- Es una forma de identificar entre datos del header que son informativos y los que son críticos.
- Con el atributo actor de los elementos, puedo especificar un URI que actúen de intermediarios entre el cliente y el “default actor” (servicio que responde la solicitud del cliente) que recibe la operación.

Elemento Body

- No hay restricciones de cómo se puede codificar la información de éste elemento. Puede ser un simple texto, un encoded byte array o un XML.
- La especificación SOAP incluye un método de codificación que puede ser utilizado para serializar los datos el body. Justamente es recomendable utilizar un set de reglas de serialización estándares.
- El body se estructura distinto, si lo utilizamos para hacer invocaciones vía RPC o si lo usamos para intercambio de datos/documentos.

Elemento Fault

- Es el mecanismo para retornar errores a los clientes.
- Sin importar el encoding, la especificación SOAP indica el formato para reportar errores. Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>Client.Security</soap:faultcode>
      <soap:faultstring>Access denied.</soap:faultstring>
      <soap:faultactor>http://abc.com</soap:faultactor>
      <soap:detail>
        <MyError>
          <Originator>File System</Originator>
          <Resource>MySecureFile.txt</Resource>
        </MyError>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
```

Tipos de Errores (elemento faultcode)

- **VersionMismatch**

El nombre namespace provisto es inválido.

- **MustUnderstand**

Uno de los elemento del header que tenia el atributo mustUnderstand en 1 no fue entendido o no fue obedecido por el servidor.

- **Client**

El contenido del mensaje fue la causa del error. Posiblemente porque se incluyo un mensaje mal formado o con información incompleta en el mismo.

- **Server**

La causa del problema no tiene que ver con el contenido del mensaje. Ejemplos de esto, es que el servidor no haya podido lograr una conexión con un servidor de base de datos para procesar el mensaje.

RPC-Style SOAP

- Solicitud:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add>
      <x>1</x>
      <y>2</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

- Respuesta:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResult>
      <result>1</result>
    </AddResult>
  </soap:Body>
</soap:Envelope>
```

SOAP Encoding

- Tipos simples: strings, integers, date/time, booleans, etc.

```
<Age>31</Age>
```

- Tipos compuestos: struct

```
public struct RectSolid
{  public  int length;
   public int width;
   public int height;
}
public int CalcVolume(RectSolid r)
{   return (r.length * r.width * r.height);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CalcVolume>
      <r>
        <length>2</length>
        <width>3</width>
        <height>1</height>
      </r>
    </CalcVolume>
  </soap:Body>
</soap:Envelope>
```

SOAP Encoding (2)

- Tipos compuestos: array de 1 dimensión y del mismo tipo

```
int[] a = {1, 2, 3};
int total;
total = AddArray(a);
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <AddArray>
      <a soap-enc:arrayType="xsi:int[3]">
        <int>1</int>
        <int>2</int>
        <int>3</int>
      </a>
    </AddArray>
  </soap:Body>
</soap:Envelope>
```

SOAP Encoding (3)

- Tipos compuestos: array de 1 dimensión y de distinto tipos !

```
object[] stuff = new object[3];
stuff[0] = (int)100;
stuff[1] = (float)2.456;
stuff[2] = (string)"Kitchen Sink";
CollectThings(stuff);
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <CollectThings>
      <things soap-enc:arrayType="xsi:ur-type[3]">
        <object>100</object>
        <object>2.456</object>
        <object>Kitchen Sink</object>
      </things>
    </CollectThings>
  </soap:Body>
</soap:Envelope>
```

SOAP Encoding (4)

- Tipos compuestos: array multidimensional

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <PrintSeatLabels soap-enc:arrayType="xsi:string[3,4]">
   <seats>
    <string>row 1, seat 1</string>
    <string>row 1, seat 2</string>
    <string>row 1, seat 3</string>
    <string>row 1, seat 4</string>
    <string>row 2, seat 1</string>
    <string>row 2, seat 2</string>
    <string>row 2, seat 3</string>
    <string>row 2, seat 4</string>
    <string>row 3, seat 1</string>
    <string>row 3, seat 2</string>
    <string>row 3, seat 3</string>
    <string>row 3, seat 4</string>
   </seats>
  </PrintSeatLabels>
 </soap:Body>
```

Tipos de datos

- SOAP define 3 formas distintas de expresar los tipos de datos de un tag:
  - Utilizar el atributo xsi:type en cada tag, explicitamente referenciando el tipo de datos de acuerdo con la especificación del XML Schema.

```
<person>
    <name xsi:type=“”xsd:string”>John Doe</name>
</person>
```

  - Referenciar un XML Schema que define particularmente ese tipo de datos exacto.

```
<person xmlns=“personschema.xsd”>
    <name>John Doe</name>
</person>
<!-- en personschema.xsd se define el elemento como
    type=xsd:string -->
```

  - Referenciar otro tipo de documento schema que defina el tipo de datos de un tipo de elemento dentro del cual se declara.

```
<person xmlns=“urn:some_namespace”>
    <name>John Doe</name>
</person>
<!-- urn:namespace indica en el cual los valores de los elementos
    son strings -->
```

WSDL (Web Services Description Language)

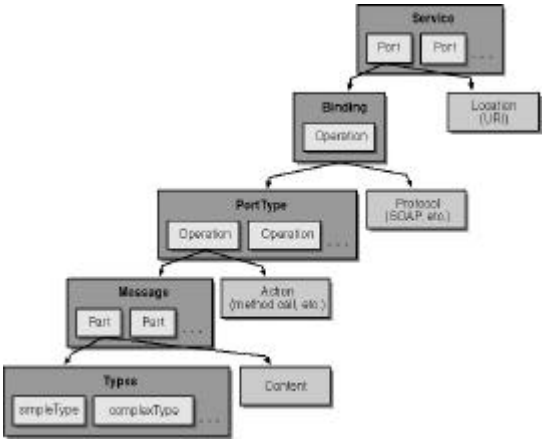
WSDL: definición.

- Es una gramática XML, orientada a describir en forma estructurada, la funcionalidad de un Web Service y la forma en que esa funcionalidad se hace disponible.
- Describe un servicio, como una colección de “comunication endpoints” (puertos) capaces de intercambiar mensajes.
- Cada port tiene un definición abstracta (port type) y una definición concreta (binding).
- Permite describir en forma abstracta operaciones y mensajes, prescindiendo de las especificaciones de protocolo y tipos de datos.
- Vincula las descripciones abstractas a una implementación concreta de protocolos y tipos de datos, permitiendo el reuso de las definiciones abstractas.
- Es extensible tanto en lo que respecta a tipos de datos (XSD) como a protocolos y formatos de mensajes.
- Provee documentación sobre el servicio que describe

Elementos WSDL que describen un Web Service

- Service = Conjunto de “ports” relacionados que implementan el Web Service.
- Port = “Port type” + “Binding”
  Es un descripción abstracta de una acción soportada por el Web Service. Cada operación se corresponde a un mensaje de input o de output
- Port type = Colección de “operations” o “signatures” de los métodos que definen el intercambio ordenado de los mensajes.
- Bindings = especifica los protocolos usa cada “port” y el “encoding”.
- Message = descripción de los datos que van a ser transmitidos. Son una colección de “data values” de un tipo particular (utilizando XML Schema como mecanismo de tipación).

Composición de un Web Service



Elemento definitions

- Es el elemento Root de un documento WSDL.
- Como el XML Schema, puede definir su propio namespace. La única restricción es que no incluya una URI relativa.

Ejemplo:

```
<?xmlversion="1.0"encoding="utf-8"?>
<definitions
   targetNamespace="http://somedomain/Calculator/wsdl"
xmlns:tns="http://somedomain/Calculator/wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<!--Definitionswillgohere.-->
</definitions>
```

Elemento definitions (2)
Definición de tipos y namespaces

- Uso de “import” para importar una referencia de un tipo de definición:

```
<?xml version=“1.0” encoding=“UTF-8”?>
<wsdl:definitions name=“HelloWorldDescription”
   targetNamespace=“urn:HelloWorld”
   xmlns:tns=“urn:HelloWorld”
   xmlns:types=“urn:MyDataTypes”
   xmlns:soap=“http://schemas.xmlsoap.org/wsdl/soap
   /”>
   xmlns:wsdl=“http://schemas.xmlsoap.org/wsdl/”>
   <wsdl:import namespace=“urn:MyDataTypes”
                location=“telephonenumber.xsd”/>
</wsdl:definitions>
```

Elemento Type

```
<types>
  <schema attributeFormDefault="qualified"
  elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"

   targetNamespace="http://somedomain/Calculator/schema
   ">
  <element name="Add">
    <complexType>
     <all>
       <element name="x" type="int"/>
       <element name="y" type="int"/>
     </all>
    </complexType>
   </element>
   <element name="AddResult">
    <complexType>
     <all>
       <element name="result" type="int"/>
     </all>
    </complexType>
   </element>
```

Elemento Message

- 1era alternativa para escribir un mensaje:

```
<message name="AddMsgIn">
  <part name="parameters"
  element="s:Add"/>
</message>
```

- 2da alternativa para escribir un mensaje:

```
<message name="AddMsgIn">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:int"/>
</message>
```

- Hay un elemento message por cada

Elemento portType

- Es una definición de una interfase en la cual cada método es una operación. Una operación esta compuesta por 1 o más mensajes dependiendo el tipo.

```
<portType name="CalculatorPortType">
  <operation name="Add">
    <input message="tns:AddMsgIn"/>
    <output message="tns:AddMsgOut"/>
    <fault message="tns:CalculateFaultMsg"
   name="CalculateFault"/>
    </operation>
</portType>
```

- Los tipos de operaciones son: request-response, one-way, solicit-response y notification. Los 2 primeros son derivados de mensajes del cliente al servidor, y los otros 2 son originados por mensajes del servidor al cliente.

Resumen: describiendo la interfase del Web Service

```
<definitions …>
   <wsdl:message name="sayHello_IN">
         <part name="name" type="xsd:string" />
   </wsdl:message>
   <wsdl:message name="SayHello_Out">
         <part name="greeting" type="xsd:string" />
   </wsdl:message
   <wsdl:portType name="HelloWorldInterface">
         <wsdl:operation name="SayHello">
               <wsdl input message="tns:sayHello_IN" />
               <wsdl:output message="tns:sayHello_Out" />
         </wsdl:operation>
   </wsdl:portType>
   …
</definitions>
```

- **Hay 4 tipos de operaciones posibles: input, output, input output, output input**

Elemento binding

```
<wsdl:binding name="HelloWorldBinding"
              type="tns:HelloWorldInterface">
  <!-- define el protocolo de transporte y el estilo del mensaje SOAP-->
  <soap:binding style="rpc"        puede ser "rpc" o "document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- define el valor del header soapAction cuando HTTP es usado-->
  <wsdl:operation name="sayHello">
        <soap:operation soapAction="urn:Hello" />
  <!-- especifica si las partes del mensaje serán condif o literales-->
        <wsdl:input>
              <soap:body use="encoded" namespace="..."
                    encodingStyle="..." />
        </wsdl:input>
        <wsdl:output>
              <soap:body use="encoded" namespace="..."
                    encodingStyle="..." />
```

Elemento binding (2)
utilizando HTTP-GET

```
<wsdl:binding name="HelloWorldBinding"
              type="HelloWorldInterface">
  <http:binding verb="GET"/>
  <wsdl:operation name="sayHello">
        <http:operation location="sayHello" />
        <wsdl:input>
              <http:urlEncoded />
        </wsdl:input>
        <wsdl:ouput>
              <mime:content type="text/plain" />
        <wsdl:output>
  <wsdl:operation>
</wsdl:binding>
```

- ***La invocación podría ser*** ***http://localhost/sayHello?name=John***
- ***La respuesta es un string de datos en formato MIME.***

Herramientas

- Existen varias herramientas disponibles para generar automáticamente el documento WSDL correspondiente a un Web Services:
  - VisualStudio.Net (WSDL.EXE)
  - IBM Web Services Toolkit
  - Apache Web Services Toolkit
  - Microsoft Soap Toolkit V2 Rc 0.

Observaciones

- WSDL es sin duda el estándar para describir servicios de web.
- Existen otros lenguajes que apuntan a resolver el mismo problema en ambientes más especializados.
- Uno de ellos es ebXML (orientado a transacciones comerciales) www.oasis.org

Desventajas:

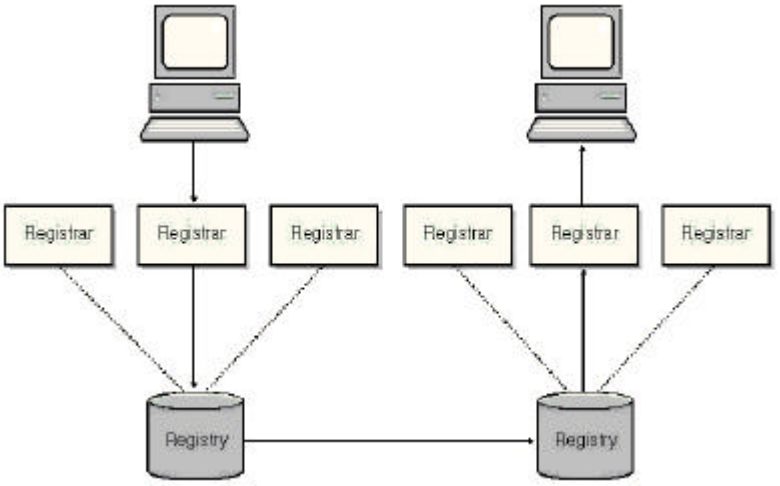
- WSDL no proveen versionado.
- WSDL carece de la posibilidad de especificar una secuencia de las operaciones necesarias en un intercambio de mensajes (por ejemplo, un login previo).

UDDI (Universal Description, Discovery and Integration) y DISCO

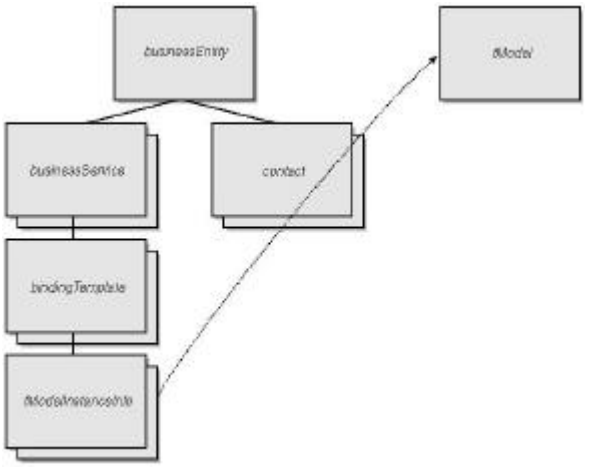
UDDI

- Provee un repositorio centralizado para publicar información técnica acerca de Web Services.
- Esta compuesto por un conjunto de repositorios y de registradores.
  - Los repositorios siguen el modelo de replicación single-master, y propaga los cambios a los demás repositorios. Estos repositorios son en si también Web Services.
  - Un registrador es una empresa que provee servicio de registración para sus clientes – por ejemplo Microsoft tiene un aplicativo que via una interfase gráfica escrita en HTML, me permite dar de alta en el respositorio.
- Solamente Microsoft e IBM están haciendo hosting de “registries”. HP y SAP se comprometieron en hacerlo tambien.

Arquitectura UDDI



Esquema elementos en UDDI



Business Entity

- Provee información de quien desarrollo el Web Service: la compañía, información de contacto en la misma, categorías de la industria, identificador de negocios y la lista de los servicios provistos
- A continuación presentamos un ejemplo ...

```
<businessEntity businessKey="uuid:11111111-2222-3333-4444-555555555555"
      operator="http://www.ibm.com"
      authorizedName="John Doe">
  <name>Acme Company</name>
  <description>
   We create cool Web Services
  </description>
  <contacts>
   <contact useType="general info">
     <description>General Information</description>
     <personName>John Doe</personName>
     <phone>(123) 123-1234</phone>
     <email>jdoe@acme.com</email>
   </contact>
  </contacts>
  <businessServices>
   ...
  </businessServices>
  <identifierBag >
   <keyedReference ModelKey="UUID:11111111-2222-3333-4444-555555555556"
                name="D-U-N-S"
                value="123456789" />
  </identifierBag>
  <categoryBag>
   <keyedReference ModelKey="UUID:11111111-2222-3333-4444-555555555557"
                name="NAICS"
                value="111336" />
  </categoryBag>
<businessEntity>
```

Business Services

- Representa un único Web Service provisto por la “Business Entity”.
- La descripción incluye: tipo de Web Service y a que categorías pertenece.
- La forma de identificar todas las “business entities” y los “business services” en UDDI es a través de el uuid (“universally unique identifiers”).

Ejemplo de Business Service

```
<businessService serviceKey=“uuid:11111111-2222-4444-5555-666666666666”
                 businessKey=“uuid:11111111-2222-4444-5555-666666666667”>
  <name>Hello World Web Services</name>
  <description>A friendly Web Service</description>
  <bindingTemplates>
    ….
  </bindingTemplates>
  <categoryBag />
</businessService>
```

Binding Templates

- Son la descripción técnica de los Web Services representados por la estructura “business service”.
- Representan la implementación del Web Service.
- Básicamente equivalen a el elemento “service” descripto en WSDL.
- Como un mismo servicio puede estar implementado de diferentes formas y puede ser asociado a múltiples protocolos o diferentes direcciones, puede haber varios binding templates para un mismo Web Service
- Veamos un ejemplo a continuación …

```
<bindingTemplate serviceKey="uuid:11111111-2222-4444-5555-666666666666"
        businessKey="uuid:11111111-2222-4444-5555-666666666667">
 <description>Hello World SOAP Binding</description>
 <accessPoint URLType="http">
  http://localhost:8080
 <accessPoint>
 <TModelInstanceDetails>
  <TModelInstanceInfo TModelKey="uuid:11111111-2222-4444-5555-
   666666666668">
   <instanceDetails>
    <overviewDoc>
     <description>
      references the description of the WSDL service definition
     </description>
     <overviewURL>
      http://localhost/helloworld.wsdl
     </overviewURL>
    </overviewDoc>
   </instanceDetails>
  </TModelInstanceInfo>
 </TModelInstanceDetails>
</bindingTemplate>
```

DISCO

- A diferencia de UDDI que fue pensado como mecanismo centralizado para que una empresa publique los Web Services que dispone, DISCO publica solamente los Web Services de 1 determinado servidor.
- Se puede publicar un índice de primer nivel referenciado a los Web Services o a otros archivos DISCO.

DISCO (2)

- El “Add Web Reference Wizard” de Visual Studio .NET utiliza DISCO para hallar los Web Services.
- Ejemplo de archivo cfg de DISCO

```
<?xml version="1.0" ?>
<dynamicDiscovery xmlns="urn:schemas-
    dynamicdiscovery:disco.2000-03-17">
<exclude path="_vti_cnf" />
<exclude path="_vti_pvt" />
<exclude path="_vti_log" />
<exclude path="_vti_script" />
<exclude path="_vti_txt" />
</dynamicDiscovery>
```

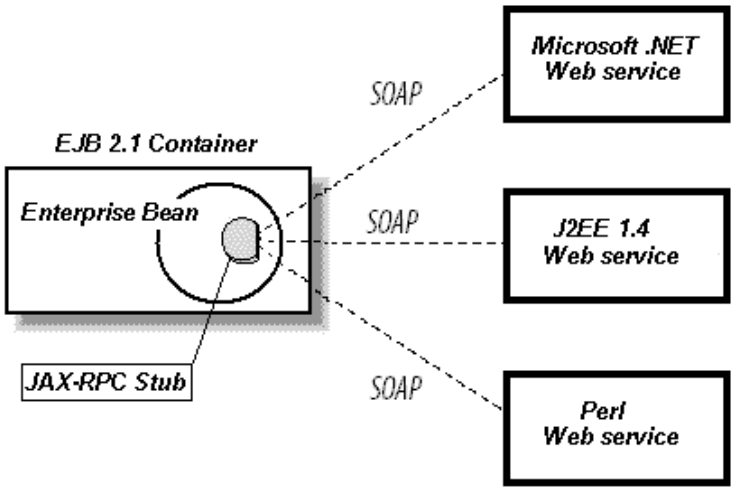
- El archivo default.vsdisco es puesto en el directorio raiz del servidor Web y luego es analizado via ASP.NET

JAX (Java API for XML-RPC)

JAX-RPC

- Es Java RMI sobre SOAP.
- Similar a Java RMI (Java RMI-JRMP) "nativo" y Java RMI-IIOP, pero usa SOAP como protocolo.
- Las implementaciones de terceros de JAX-RPC deben incluir, por lo menos, soporte de RPC encoding de SOAP sobre HTTP, pero cada empresa implementadora puede soportar otros encoding styles, messaging styles y protocolos de Internet.
- JAX-RPC puede ser usado por los session, entity y message-driven beans para invocar operaciones de Web Services. Por ejemplo un stateless session bean puede usar JAX-RPC para hacer una llamada a un Web Service.NET.

JAX-RPC (2)



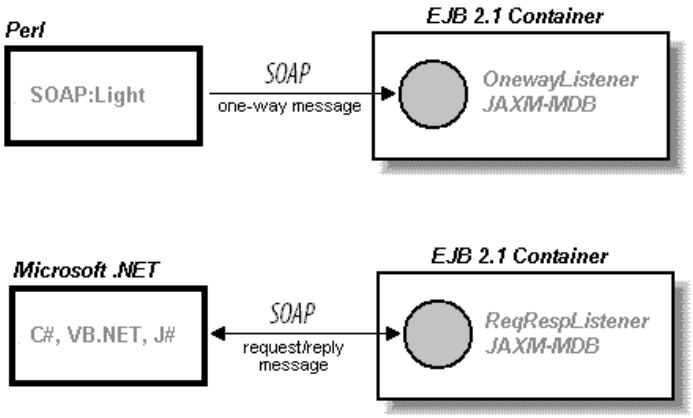
JAXM (Java API for XML Messaging)

- Es una API de mensajería SOAP similar a JMS (Java Message Service).
- Como JMS es una API para enviar y recibir mensajes via MOM (message-oriented middleware), JAXM es una API para enviar y recibir mensajes via Web Services.

JAXM (2)

- JAXM es orientado a documentos; intercambia mensajes SOAP como documentos XML. Los clientes JAXM arman,reciben y manipulan mensajes SOAP utilizando SAAJ (SOAP with Attachments API for Java), quien modela la estructura XML actual de un mensaje SOAP.
- Mecanismo diferente a JAX-RPC, solo se ve una interfase remota con métodos, parámetros y valores de retorno.
- Con JAXM, uno armar sus propios mensajes trabajando directamente con SOAP. Como JAX-RPC, JAXM puede ser usado para intercambiar mensajes SOAP con cualquier Web Service que sea “SOAP-compliant”. Por ejemplo, un “enterprise bean” podría intercambiar mensajes SOAP con un Web Service escrito en Perl.

JAXM (3)



Ejemplo de declaración de métodos accesibles como Web Services

En C#

```
[Web
  Service(Namespace="http://microsoft.com/
  Web Services/")] public class MyWeb
  Service { // implementation }
```

En VB.NET

Invocación de Web Service usando SOAP

```
POST /Payment/CreditCard.asmx  HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Validate"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <Validate xmlns="http://tempuri.org/">
   <cardNumber>string</cardNumber>
   <expDate>dateTime</expDate>
  </Validate>
 </soap:Body>
</soap:Envelope>
```

Respuesta de Web Service usando SOAP

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

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <ValidateResponse xmlns="http://tempuri.org/">
   <ValidateResult>boolean</ValidateResult>
  </ValidateResponse>
 </soap:Body>
</soap:Envelope>
```

Invocación y respuesta de Web Service usando HTTP GET

```
GET
    /Payment/CreditCard.asmx/Validate?cardNumber=string&expDate=
    string HTTP/1.1
Host: localhost

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

<?xml version="1.0" encoding="utf-8"?>
<boolean xmlns="http://tempuri.org/">boolean</boolean>
```

Invocación y respuesta de Web Service usando HTTP POST

```
POST /Payment/CreditCard.asmx/Validate HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

cardNumber=string&expDate=string
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<boolean xmlns="http://tempuri.org/">boolean</boolean>
```

Ejemplo de declaracion de un Web Service (.asmx en .NET)

```
public class CreditCard : System.Web.Services.Web Service
{
    [WebMethod]
    public bool Validate(string cardNumber, DateTime expDate)
    {
        if(expDate >= DateTime.Today) {
          int total = 0;
          int temp = 0;
          char [] ccDigits = cardNumber.ToCharArray();
          for(int i = 0; i < cardNumber.Length; i++) { … }
          if((total%10) ==0) {
                    return true;
          }
          else {
                    return false;
          }
        }
        else {
          return false;
        }
    }
….
```

Ejemplo de uso de un Web Service para validar

```
private void placeOrder_Click(object sender, System.EventArgs e)
{
    localhost.CreditCard cc = new localhost.CreditCard();

    if(cc.Validate(this.creditCardNumber.Text,
    this.expirationDate.SelectedDate))
    {
        this.status.Text = "Thank you for your order.";
    }
    else
    {
        this.status.Text = "Credit card invalid.";
    }
}
```

Bibliografía

- Building XML Web Services for the .NET Platform (Microsoft Press 2002 – Scott Short)