Servicios Web Seguros

Sesión 1: Introducción a los Servicios Web seguros

Puntos a tratar

- Arquitectura de los Servicios Web
- Tecnologías básicas
- Tecnologías Java EE para Servicios Web
- Invocación de servicios
- Implementación de servicios
- Servicios web avanzados
  - Optimización
  - Seguridad

Componentes software

- El diseño del software tiende a ser cada vez más modular
  - Aplicaciones compuestas por componentes reutilizables

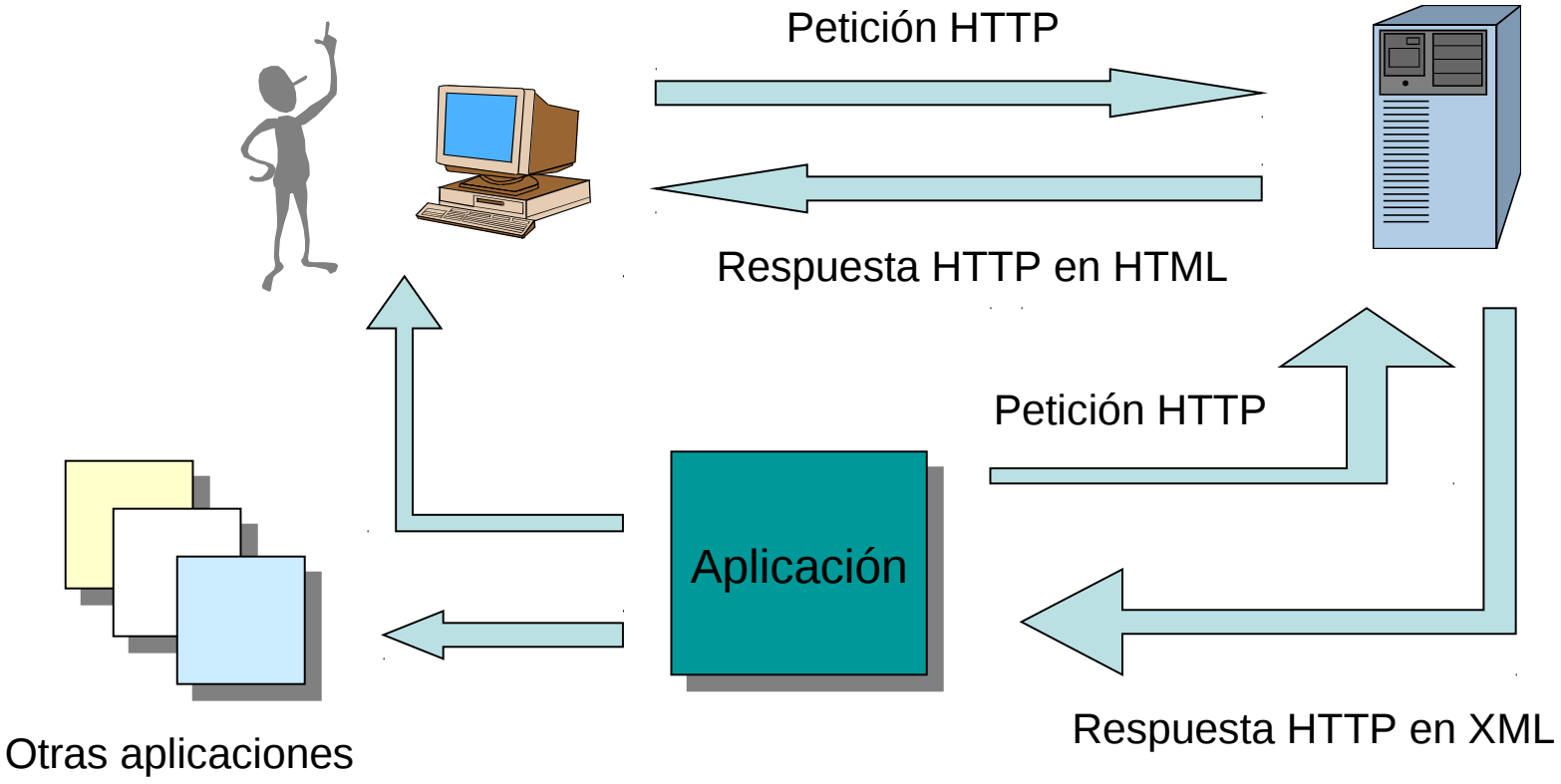
    P.ej. Objetos CORBA o EJBs
  - Estos componentes pueden encontrarse distribuidos

- Servicio
  - Unidad funcional de software
  - Ofrece una determinada interfaz y cumple ciertos requisitos
  - Deberá poder ser integrado en la aplicación y combinado con otros servicios de forma independiente.

Descripción de Servicio Web

- Un Servicio Web es un servicio al que se podrá acceder mediante protocolos Web estándar
  - Los mensajes para invocar el servicio se codifican en XML
  - Estos mensajes se pueden transportar utilizando HTTP
- Normalmente constará de una interfaz (conjunto de métodos) que podremos invocar de forma remota desde cualquier lugar de la red
  - Nos permiten crear aplicaciones distribuidas en Internet
- Son independientes de la plataforma y del lenguaje de programación en el que estén implementados

Web “para humanos” vs. “para máquinas”



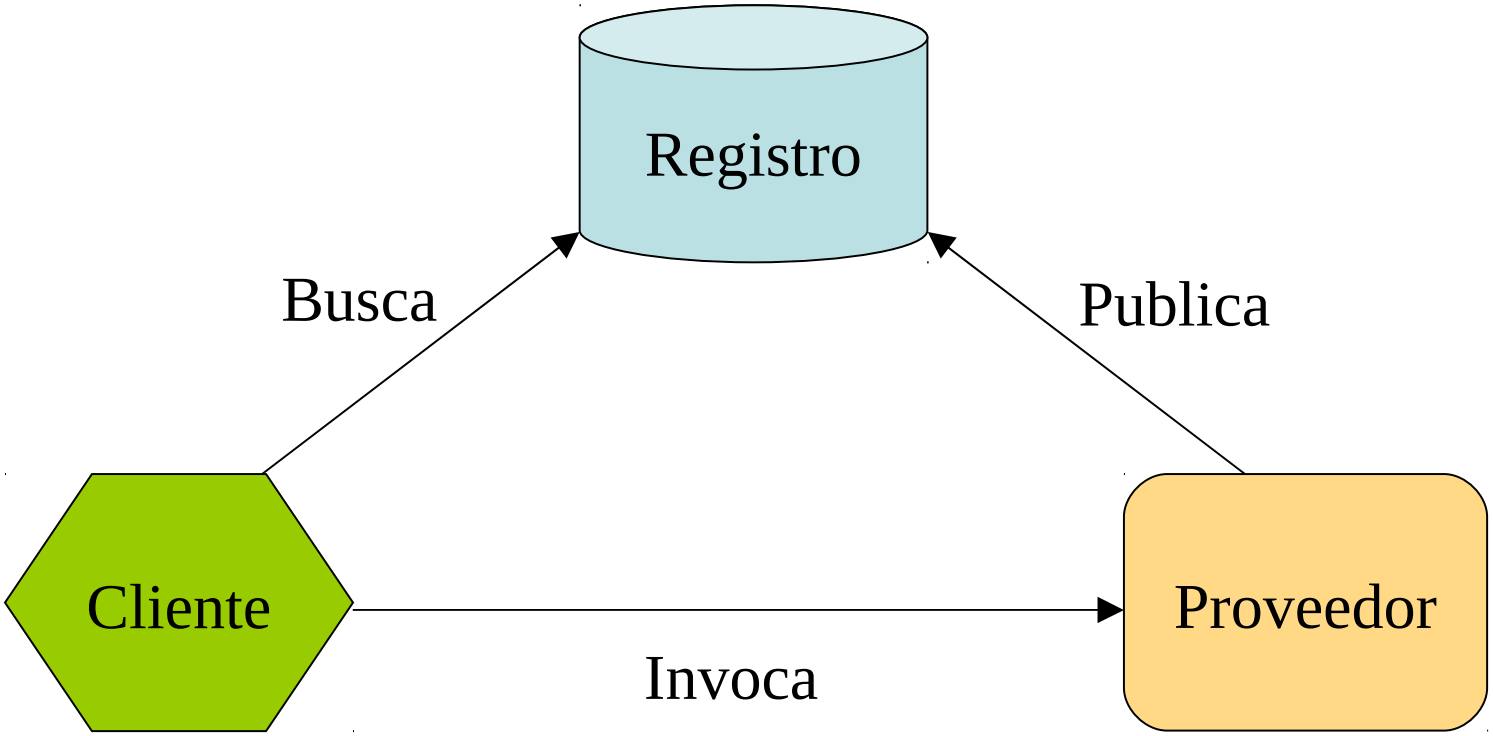
Características de los servicios

- Deben ser accesibles a través de la Web
  - Debe utilizar protocolos de transporte estándares como HTTP y codificar los mensajes en un lenguaje estándar (XML).
- Deben describirse a si mismos
  - De esta forma una aplicación podrá conocer cuál es la interfaz del servicio, y podrá integrarlo y utilizarlo de forma automática.
- Deben ser localizables
  - Debe existir algún mecanismo de localizar un servicio que realice una determinada función, sin tenerlo que conocer previamente el usuario.

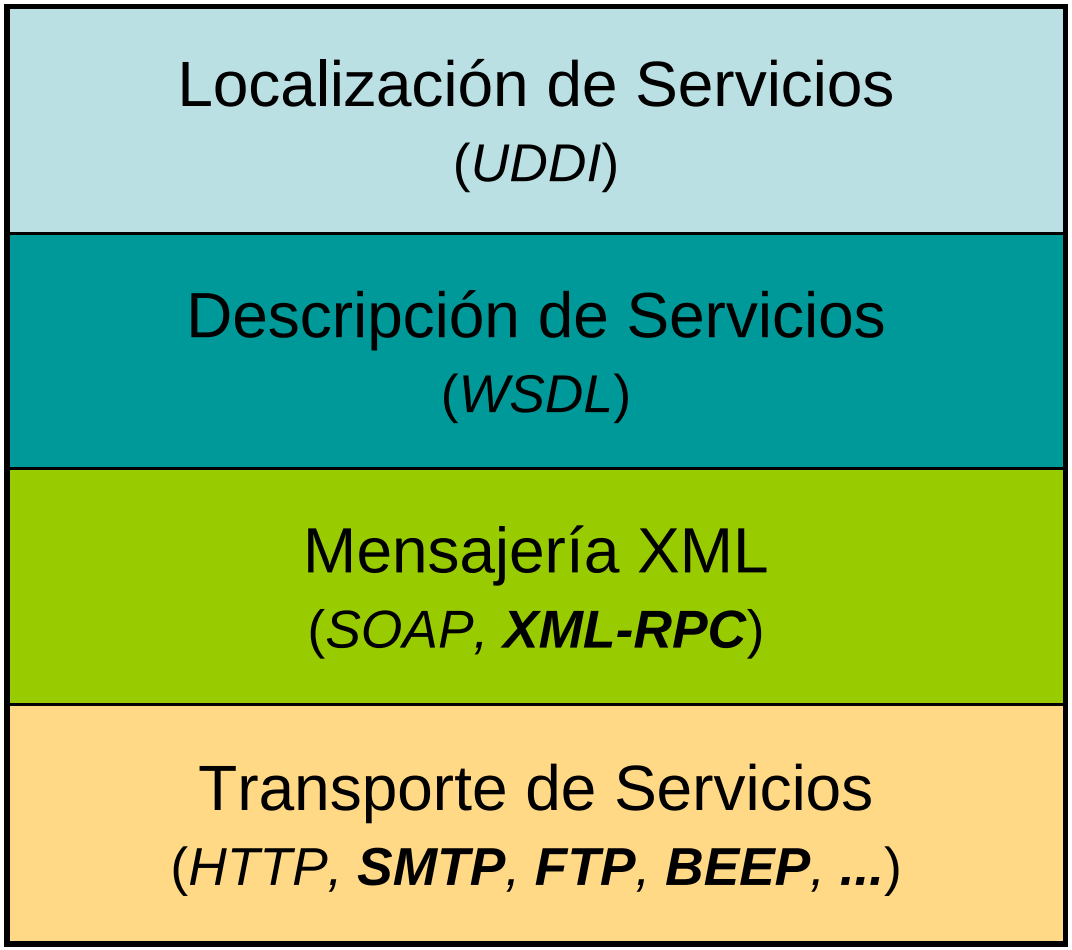
Arquitecturas Orientada a Servicios (SOA)

- Desarrollo de servicios altamente reutilizables
  - Interfaz estándar bien definida
  - Sin estado
  - No deben depender del estado de otros componentes
- Orquestación de servicios
  - Combinar servicios para construir aplicaciones
  - Se pueden formar diferentes flujos para implementar los procesos de negocio
- Los servicios pueden ser de diferentes tipos
  - Servicios Web, JMS, etc.

Componentes de una SOA



Capas de los Servicios Web



SOAP

- Protocolo derivado de XML
- Se usa para intercambiar información
- Dos tipos:
  - Mensajes orientados al documento
    - Cualquier tipo de contenido
  - Mensajes orientados a RPC
    - Tipo más concreto que el anterior
    - Nos permite realizar llamadas a procedimientos remotos
      - La petición contiene el método a llamar y los parámetros
      - La respuesta contiene los resultados devueltos
- Nos centraremos en el segundo tipo

Elementos de SOAP

- Sobre SOAP (*Envelope*). Contiene:
  - Descripción del mensaje (destinatario, forma de procesarlo, definiciones de tipos)
  - Cabecera (opcional) y cuerpo SOAP
- Cabecera SOAP (*Header*). Contiene:
  - Información sobre el mensaje (obligatorio, actores, etc)
- Cuerpo SOAP (*Body*). Contiene:
  - Mensaje (en caso de RPC la forma del mensaje se define por convención)
  - Error (opcional)
- Error SOAP (*Fault*)
  - Indica en la respuesta que ha habido un error en el procesamiento de la petición

Elementos de SwA

- Con SOAP podemos intercambiar cualquier documento XML, pero no otro tipo
  - Por ejemplo, una imagen.
- SwA (SOAP with Attachment) nos permite añadir datos que no sean XML al mensaje
- Parte adjunta (*Attachment*)
  - Contiene los datos no XML

Ejemplo SOAP

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <ns:getTemperatura xmlns:ns="http://j2ee.ua.es/ns">

    <area>Alicante</area>
  </ns:getTemperatura>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

WSDL

- Lenguaje derivado de XML
- Describe la interfaz de los Servicios Web
  - Operaciones disponibles
  - Parámetros de las operaciones
  - Resultados devueltos
  - Tipos de datos de estos parámetros y resultados
- Además contiene la dirección del *endpoint*
  - URL a la que hay que conectarse para acceder al servicio
- Nos permite integrar un servicio automáticamente en nuestra aplicación, o que otros usuarios utilicen los servicios que hayamos desarrollado nosotros

Elementos WSDL

Elementos de WSDL lógicos

- El elemento raíz del documento es `definitions`, contiene:
  - `types`: Tipos de datos que se intercambian
  - `message`: Mensajes que se intercambian durante la invocación de las operaciones. Cada operación tendrá un mensaje de entrada (petición) y uno de salida (respuesta)
  - `portType`: Define las operaciones que ofrece el servicio. Cada una tendrá un mensaje de entrada y salida de los anteriores

Elementos de WSDL físicos

- `binding`: Indica protocolo y formato para los mensajes anteriores. Formatos comunes son:

  `rpc/encoded` (desaprobado)

  `document/literal`

- `service`: Define el servicio mediante una colección de puertos a los que acceder.

  Cada puerto tendrá una URL para acceder al *endpoint*.

  Además contiene documentación en lenguaje natural sobre el servicio.

UDDI

- UDDI nos permite localizar Servicios Web
- Define la especificación para construir un directorio distribuido de Servicios Web
  - Se registran en XML
- Define una API para acceder a este registro
  - Buscar servicios
  - Publicar servicios
- La interfaz de UDDI está basada en SOAP
  - Se utilizan mensajes SOAP para buscar o publicar servicios

Tecnologías de segunda generación

- Una vez establecidas las tecnologías básicas, aparecen extensiones:
  - *WS-Policy* y *WS-PolicyAttachment*

    Describir funcionalidades que no podemos especificar con WSDL.
  - *WS-Security*

    Seguridad a nivel de mensaje.
  - *WS-Addressing* y *WS-ReliableMessaging*

    Servicios Web asíncronos, fiables, con estado.
  - *WS-Coordination* o *BPEL*

    Orquestar servicios web.

JAXP

- Permite procesar documentos XML en Java
- Tiene en cuenta espacios de nombres
- Soporta XSLT
  - Podemos transformar XML a otros formatos
- Librería para tratar XML genérico
  - Otras librerías se apoyan en esta para procesar tipos concretos de lenguajes derivados de XML

    SOAP

    WSDL

    UDDI

JAXM

- Mensajería XML orientada al documento
  - Trabaja con mensajes SOAP y SwA
- Nos permite
  - Extraer el contenido de los mensajes XML recibidos
  - Crear y enviar mensajes XML
    - Síncrona (petición-respuesta)
    - Asíncrona (envío sin esperar respuesta)
- Se divide en dos APIs
  - SAAJ: API independiente y suficiente para:
    - Crear mensajes SOAP y extraer información de ellos
    - Envío síncrono de mensajes
  - JAXM: API dependiente de SAAJ. Incorpora:
    - Envío asíncrono de mensajes

JAX-RPC / JAX-WS

- Infraestructura para hacer RPC mediante XML
  - Utiliza mensajes SOAP
- Depende de SAAJ, pero no de JAXM
  - SAAJ se encarga de
    - Construir y enviar los mensajes
    - Recibir y analizar los mensajes
- Nos permitirá:
  - Invocar Servicios Web de tipo SOAP
  - Crear nuestros propios Servicios Web SOAP
    - A partir de clases Java que implementan su funcionalidad
- A partir de la versión 2.0 pasa a llamarse JAX-WS
  - Utiliza anotaciones para definir los servicios

JAXR

- Permite acceder a registros XML
  - UDDI
  - ebXML
- Utiliza una API estándar Java
  - Se accede de la misma forma a cualquier tipo de registro
- Permite
  - Consultar el registro
  - Publicar servicios en el registro
  - Eliminar o modificar los servicios publicados

JAXB

- Permite asociar esquemas XML a clase Java
- Convierte los tipos de datos utilizados en el servicio:
  - Unmarshalling

    XML → Objeto Java
  - Marshalling

    Objecto Java → XML

- Otras APIs
  - Java API for WSDL (WSDL4J)
  - Web Services Invocation Framework (WSIF)
  - UDDI4J

WS-I Basic Profile (BP)

- Estándar de interoperabilidad para Servicios Web
  - Definido por *Web Services Interoperability Organization*
- Define una serie de reglas para aclarar ambigüedades
  - Las especificaciones de SOAP, WSDL y UDDI no son claras
  - El uso conjunto de estas tecnologías se especifica en BP

Interoperabilidad de servicios

- Podremos crear clientes para utilizar cualquier servicio
  - Se invocan mediante protocolos Web estándar
  - Accederemos a cualquier servicio de la misma forma
  - No importa el lenguaje o plataforma del *endpoint*

- Las tecnologías Java de Servicios Web
  - Cumplen *WS-I Basic Profile* (BP)
  - Serán interoperables con cualquier servicio BP

Tipos de acceso

- JAX-RPC/WS nos permite acceder de 2 formas:
  - Creación de un *stub* estático
    - Se genera una capa *stub* en tiempo de compilación
    - Esta capa se genera automáticamente mediante herramientas
    - El cliente accede a través del *stub* como si fuese a un objeto local
  - Interfaz de invocación dinámica (DII)
    - Se hacen llamadas de forma dinámica, sin *stub*
    - Se proporcionan los nombres de las operaciones a ejecutar mediante cadenas de texto a métodos genéricos de JAX-RPC
    - Se pierde transparencia

Librería JAX-RPC / JAX-WS

- A partir de JDK 1.6 se incluye JAX-WS 2.0 en Java SE
  - JAX-WS 2.1 a partir de JDK 1.6.0_04
- JAX-WS no es compatible con servicios `rpc/encoded`
  - Podemos incluir JAX-RPC para este tipo de servicios
  - Ambas librerías no pueden coexistir en una misma aplicación
- En versiones previas de JDK se puede incluir JAX-WS o JAX-RPC
  - Deberemos añadir a nuestro cliente la librería adecuada

Acceso mediante stub estático

- Es la forma más sencilla de acceder
  - Necesitamos una herramienta que genere el *stub* de forma automática

    P.ej. en JDK 1.6 tenemos la tarea `wsimport`
  - El *stub* implementará la misma interfaz que nuestro servicio
  - Utilizaremos el *stub* para acceder al servicio como si fuese un objeto local
  - Es totalmente transparente para nuestro cliente que se esté invocando un Servicio Web
  - No será necesario utilizar código JAX-RPC/WS en nuestro cliente, esta tarea la hace el *stub* generado

Generar el cliente con JAX-WS

- Se utiliza la herramienta `wsimport`

```
wsimport -s src -d bin
    -p es.ua.jtech.servcweb.hola.stub
    http://jtech.ua.es/HolaMundo/wsdl/HolaMundoSW.wsdl
```

- También disponible como tarea de ant

```
<wsimport sourcedestdir="${src.home}"
    destdir="${bin.home}" package="${pkg.name}"
    wsdl="${wsdl.uri}" />
```

- Eclipse y Netbeans permiten generar clientes para servicios de forma automática

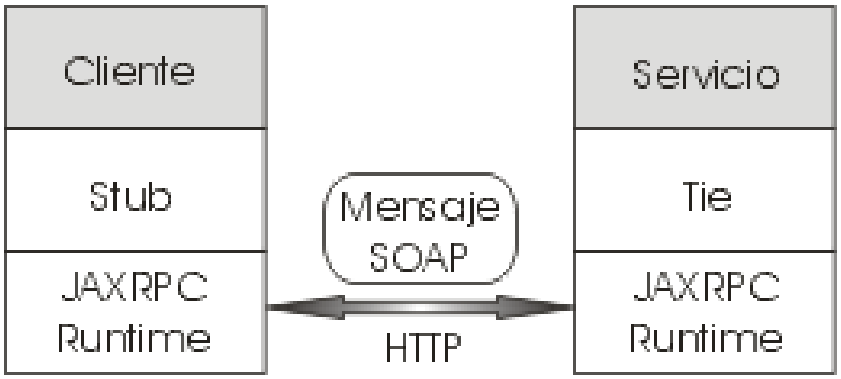
Acceso al servicio

- Una vez obtenido el *stub* (`port`), accedemos al servicio como si fuese un objeto local

```
public class Main {
  public static void main(String[] args)
                          throws Exception {
    HolaMundoSWService service =
                         new HolaMundoSWService();
    HolaMundoSW port = service.getHolaMundoSW();
    System.out.println("Resultado: " +
                         port.saluda("Miguel"));
  }
}
```

Capas del servicio

- Las capas Stub y Tie
  - Se encargan de componer e interpretar los mensajes SOAP que se intercambian
  - Utilizan la librería JAX-RPC/WS
  - Se generan automáticamente
- El cliente y el servicio
  - No necesitan utilizar JAX-RPC/WS, este trabajo lo hacen las capas anteriores
  - Los escribimos nosotros
  - Para ellos es transparente el método de invocación subyacente
  - El servicio es un componente que implementa la lógica (clase Java)
  - El cliente accede al servicio a través del *stub*, como si se tratase de un objeto Java local que tiene los métodos que ofrece el servicio



Tipos de datos básicos

- Tipos de datos básicos y *wrappers* de estos tipos

`boolean`	`java.lang.Boolean`
`byte`	`java.lang.Byte`
`double`	`java.lang.Double`
`float`	`java.lang.Float`
`int`	`java.lang.Integer`
`long`	`java.lang.Long`
`short`	`java.lang.Short`
`char`	`java.lang.Character`

Otros tipos de datos y estructuras

- Otros tipos de datos

```
java.lang.String          java.util.Calendar
java.math.BigDecimal      java.util.Date
java.math.BigInteger      java.awt.Image
```

- Colecciones y genéricos

Listas: `List`	Mapas: `Map`	Conjuntos: `Set`
`ArrayList`	`HashMap`	`HashSet`
`LinkedList`	`Hashtable`	`TreeSet`
`Stack`	`Properties`	
`Vector`	`TreeMap`	

Otras clases

- Podremos utilizar objetos de clases propias, siempre que estas clases cumplan
  - Deben tener un constructor `void` público
  - No deben implementar `javax.rmi.Remote`
  - Todos sus campos deben
    - Ser tipos de datos soportados por JAXB
    - Los campos públicos no deben ser ni `final` ni `transient`
    - Los campos no públicos deben tener sus correspondientes métodos `get`/`set`.
  - Si no cumplen esto deberemos construir serializadores
- También podemos utilizar *arrays* y colecciones de cualquiera de los tipos de datos anteriores

Fichero JWS

- Forma estándar de definir Servicios Web en Java
  - Clase Java cuyos métodos se ofrecerán como operaciones de un Servicio Web
- Utiliza anotaciones para definir el servicio
  - *Web Services Metadata for the Java Platform* (JSR-181)
- El fichero JWS contendrá:
  - Al menos la anotación `@WebService`
  - Un constructor sin parámetros
  - Por defecto todos sus métodos públicos serán las operaciones del servicio
- Las herramientas utilizadas para generar el servicio dependerán de la plataforma

Ejemplo de fichero JWS

```
package es.ua.jtech.servcweb.conversion;

import javax.jws.WebService;

@WebService
public class ConversionSW {

  public ConversionSW() { }

  public int euro2ptas(double euro) {
    return (int) (euro * 166.386);
  }

  public double ptas2euro(int ptas) {
    return ((double) ptas) / 166.386;
  }
}
```

Publicar servicios con JDK 1.6

- Podemos publicar sin servidor de aplicaciones

```
public class Servicio {
  public static void main(String[] args) {
    Endpoint.publish(
        "http://localhost:8080/ServicioWeb/Conversion",
        new ConversionSW());
  }
}
```

WSIT

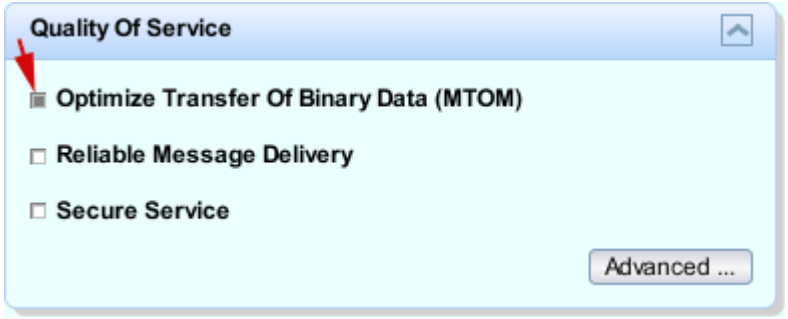
- *Web Services Interoperability Technologies*
- Plataforma Java para servicios web
  - Incorpora tecnologías de segunda generación

Optimización	MTOM
Fiabilidad	WS-ReliableMessaging
Seguridad	WS-Security, WS-Trust
Transacciones	WS-AtomicTransaction

  - Mejorar QoS
  - Interoperabilidad con .NET 3.0
- Equivalente a WCF de Microsoft
  - *Windows Communication Foundation*

Optimización de mensajes

- Mejorar eficiencia en el envío
- Los mensajes SOAP son muy extensos
  - La información se envía como texto
  - Gran cantidad de “envoltorios” en el XML
- Especialmente ineficiente para enviar datos binarios
- MTOM
  - *Message Transmission Optimization Mechanism*
  - Optimización del envío de datos binarios
  - Se envían como adjunto

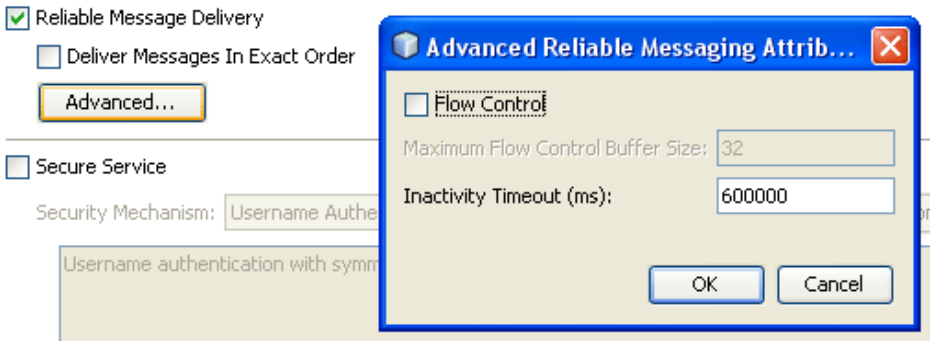


Otras formas de optimización

- *Fast Infoset*
  - Formato binario análogo a XML
  - No hay pérdida de información
  - Los mensajes son más compactos
  - Mayor eficiencia para serializar/deserializar
  - Adecuado para dispositivos limitados
- Transmisión vía TCP
  - Utiliza directamente protocolo TCP en lugar de HTTP
- Ambos métodos se usan cuando son soportados tanto por el cliente como por el servidor

Envío fiable de mensajes

- Las comunicaciones pueden fallar
  - ¿Reintentar la invocación?
  - Peligroso cuando sólo deba ejecutarse una vez
  - En las operaciones *oneway* no habrá confirmación
- *Reliable Messaging*
  - Los mensajes se entregan 1 y sólo 1 vez.
  - Se crea canal de datos
  - Se puede controlar que lleguen en orden (opcional)



Servicios con estado

- Mantienen información de estado de cada cliente
  - Por ejemplo un carrito de la compra

    Cada llamada al servicio añade un producto al carrito
  - Disponible a partir de JAX-WS 2.1
- Cada cliente accede a una instancia del servicio
  - El estado se mantiene mediante variables de instancia
- Basado en *WS-Addressing*. Permite especificar:
  - Dirección del *endpoint*
  - Instancia concreta del servicio a la que acceder

Seguridad en servicios web

- Diferenciamos 3 aspectos
  - Confidencialidad

    Los datos transmitidos no deben poder ser vistos sin autorización

    Solución: cifrado con clave simétrica/asimétrica
  - Integridad

    Los datos no deben poder ser alterados

    Solución: huella/firma digital
  - Autentificación

    Conocer la identidad del otro extremo

    Solución: Credenciales (*login/password*), firma con certificado digital
- Seguridad en la red
  - Se permite invocar operaciones a través de HTTP
  - Los *firewalls* no impiden el acceso

Confidencialidad e integridad

- La información contenida en los mensajes SOAP puede ser confidencial
- Solución:
  - Como los mensajes se envían por HTTP, podemos cifrarlos y firmarlos con SSL
- Problema:
  - Si el mensaje debe atravesar una cadena de servicios, debe ser descifrado dentro de cada uno de ellos

    Los datos estarán inseguros dentro de cada nodo intermedio
  - Solución:

    Cifrar y firmar partes del mensaje por separado

Transporte vs Mensaje

- Seguridad a nivel de transporte
  - SSL (*https*)
  - Sólo ofrece seguridad durante el transporte
  - Más eficiente
  - No plantea problemas con punto-a-punto
- Seguridad a nivel de mensaje
  - *WS-Security*
  - Se protege hasta la llegada al *endpoint*
  - Independiente del protocolo de transporte
  - Requiere que los actores soporten *WS-Security*

Autentificación

- Podemos necesitar identificar al usuario
  - Para prestarle un servicio personalizado
  - Para comprobar si tiene permiso para usar el servicio
  - etc...
- *Tokens* de autentificación

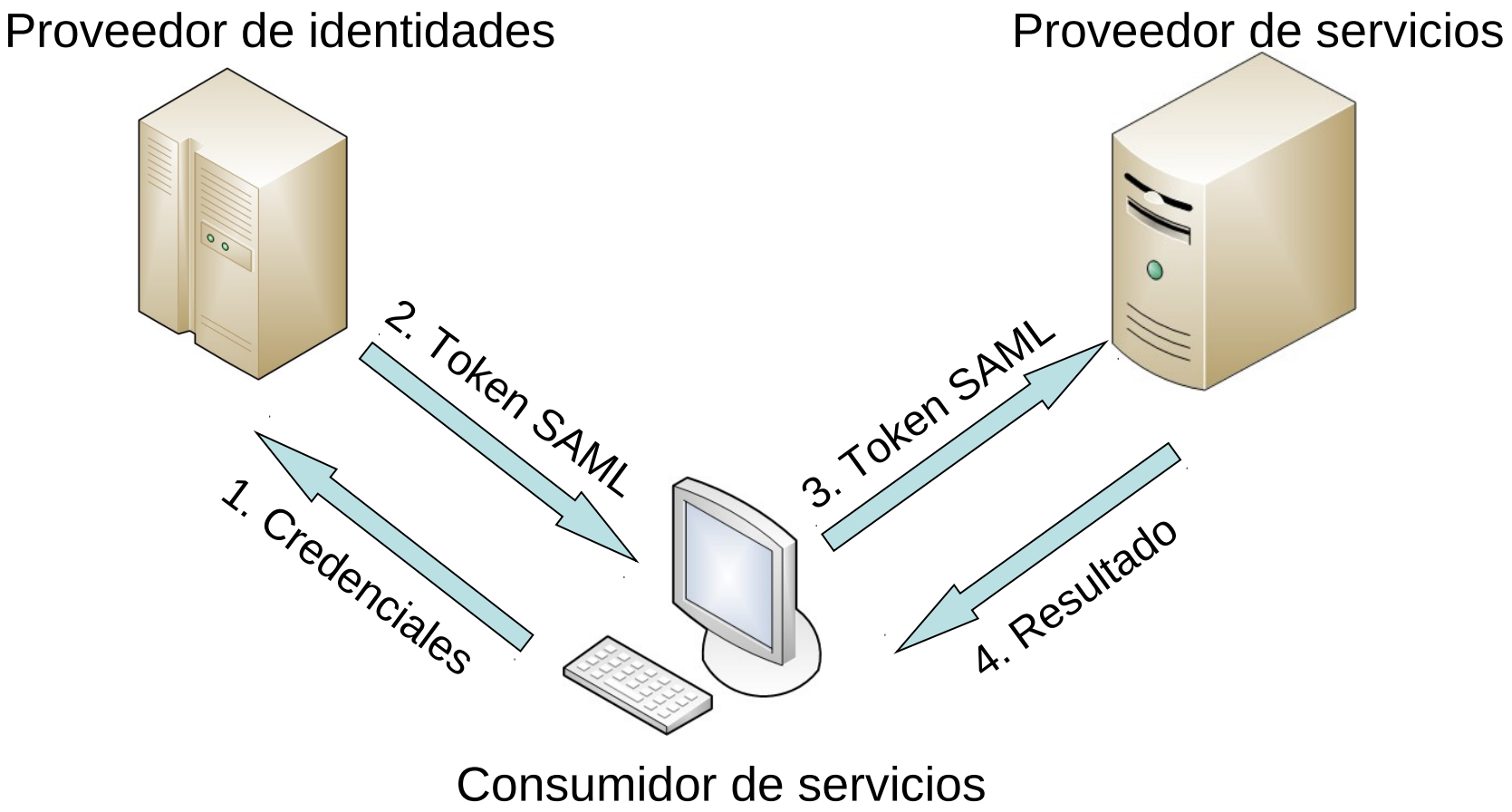
Username token	Login/password
X.509 token	Certificado digital
SAML token	Autentificación y autorización

Contexto compartido

- Problema:
  - Si utilizamos servicios de distintos servidores, tendremos que autentificarnos para cada uno por separado
- Solución:
  - Crear contexto compartido (proveedor de identidades) donde puedan consultar información sobre la autentificación

    MS Passport, Liberty Project
- Sun Access Manager
  - Puede ser instalado como *addon* de Glassfish

Single Sign-On (SSO)

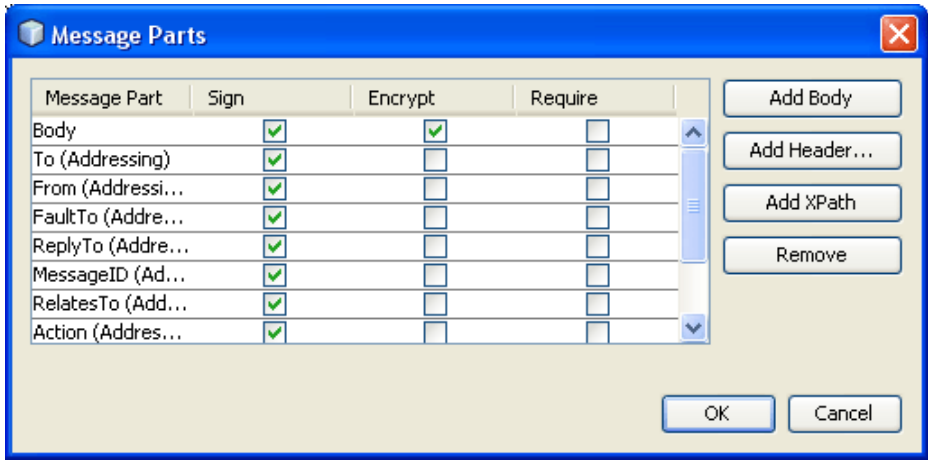


Modos de seguridad con WSIT

- *Username Authentication with Symmetric Keys*
  - Seguridad a nivel de mensaje.
  - Autentificación mediante *login* y *password*.
  - Clave simétrica para cifrar y firmar.
- *Mutual Certificates Security*
  - Seguridad a nivel de mensaje.
  - Autentificación mediante un certificado X.509.
- *Transport Security (SSL)*
  - Seguridad a nivel de transporte
  - Se protege la ruta en `web.xml`.

Configuración de la seguridad

- Crear usuario en *realm file (username token)*
- Instalar certificados X.509 v3
  - *Keystore* (certificados propios)
  - *Truststore* (certificados raíz)
- Partes del mensaje



Otros modos

- Basados en *tokens* SAML
  - *SAML Authorization over SSL*
  - *SAML Sender Vouches with Certificates*
  - *SAML Holder of Key*
  - Se pueden utilizar junto a *Sun Access Manager*
- Basados en STS (*Secure Token Service*)
  - Crear un servicio de tipo STS
  - El STS proporciona un *token* de seguridad
  - Accedemos al servicio proporcionando dicho *token*

Frameworks Java para SWS

- Metro
  - Desarrollada por Sun
  - Fácil desarrollo con NetBeans
- Axis2
  - Requiere módulo Rampart
  - Entorno de ejecución de servicios para servidores de aplicaciones Java EE
- CXF (Celtix + XFire)
  - Integración con sistemas existentes
  - Utiliza Spring

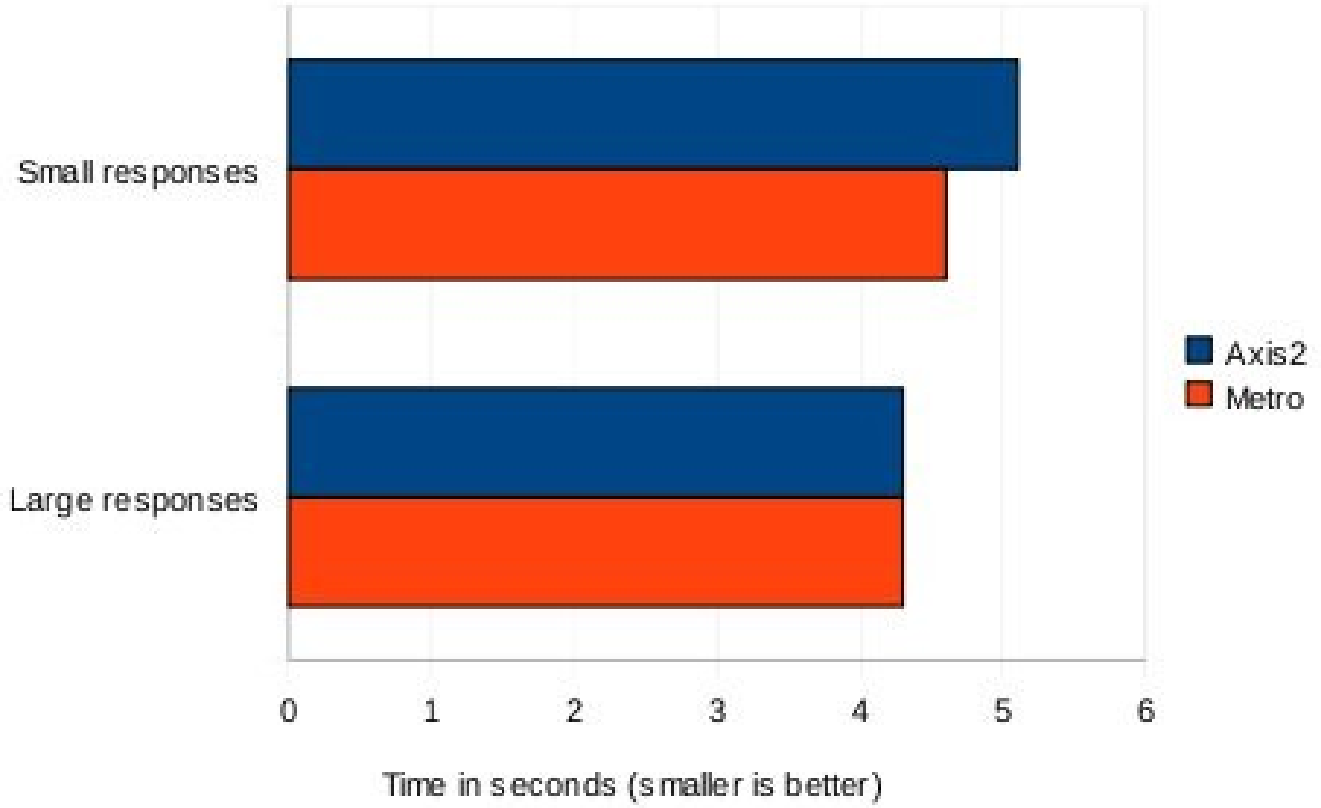
Comparativa

Standards (*)	Axis2	CXF	JAX-WS/Metro
WS-Addressing	X	X	X
WS-Coordination	X(2)		X
WS-MetadataExchange			X
WS-Policy	X	X	X
WS-ReliableMessaging	X(3)	X	X
Web Services Security	X(1)	X(4)	X
WS-SecureConversation	X(1)		X
WS-SecurityPolicy			X
WS-Transaction	X(2)		X
WS-Trust	X		X
WS-Federation			

- (1) Necesita el módulo adicional Apache Rampart
- (2) Necesita el módulo adicional Apache Kandula2
- (3) Necesita el módulo adicional Apache Sandesha2
- (4) Proporcionado por Apache WSS4J Interceptor
- (*) Extraído del informe "Apache Axis2, CXF and Sun JAX-WS RI in comparison", de Thomas Bayer, 20/10/2008

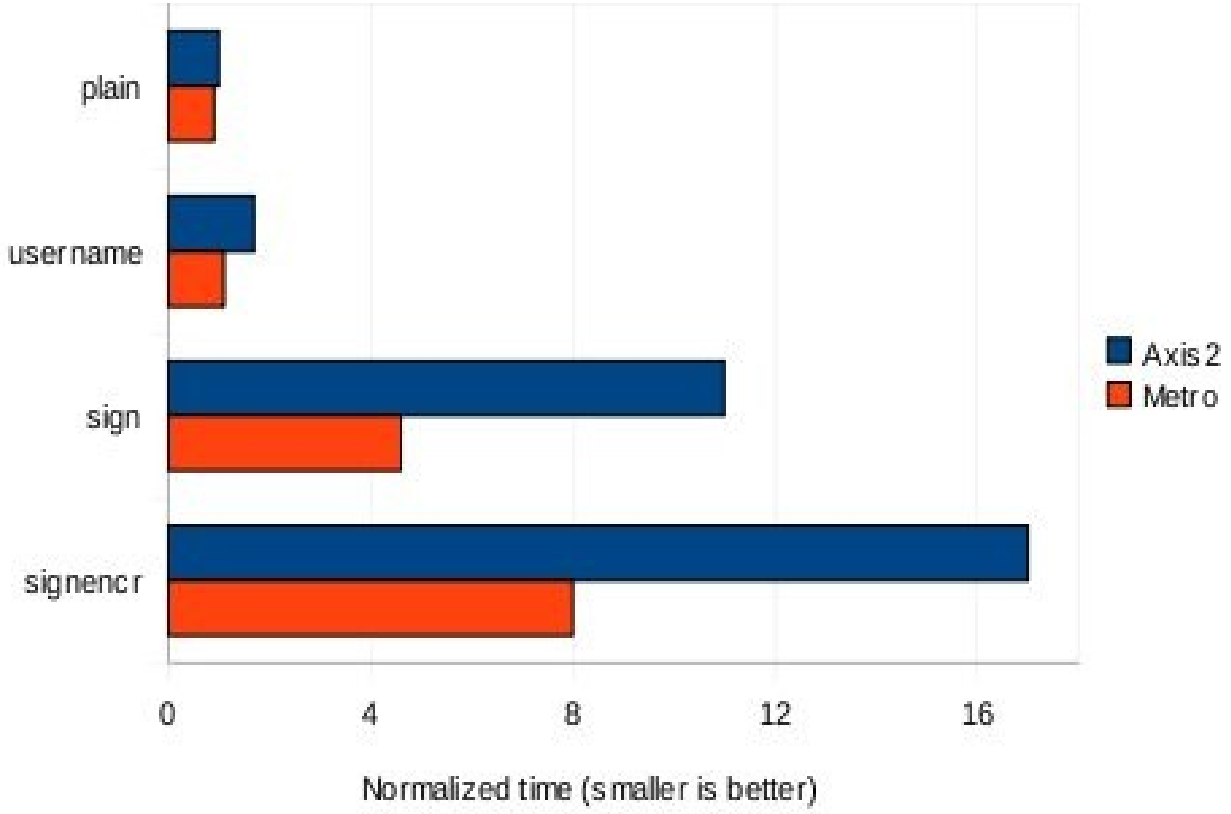
Comparativa

- Tiempos de respuesta sin seguridad



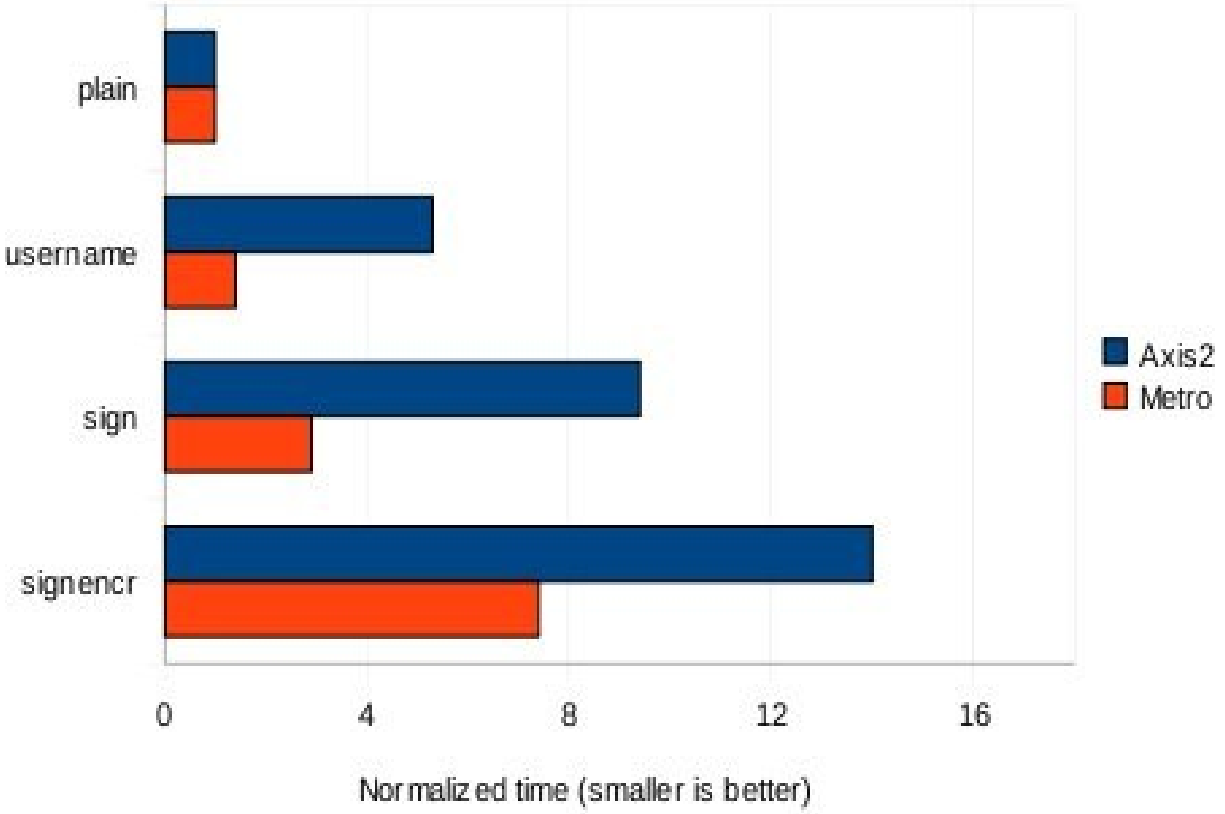
Comparativa

- Tiempos de respuesta con seguridad (respuestas cortas):



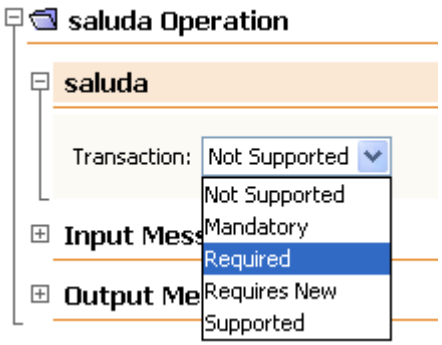
Comparativa

- Tiempos de respuesta con seguridad (respuestas largas):



Transacciones en servicios web

- Basadas en *WS-AtomicTransaction*
- No se necesitan nuevas APIs
  - Se usa JTA
  - Se comportan igual que en los EJBs
- Se configuran a nivel de operación
  - Mismos modos que en EJBs



Cliente con JTA

- Desde servlet o EJB
  - `UserTransaction`
- Sólo desde EJB
  - *Container Manager Transaction* (CMT)

```
@Resource UserTransaction ut;
HotelSW pHotel = sHotel.getHotelSW();
VuelosSW pVuelos = sVuelos.getVuelosSW();
ut.begin();
boolean hotelReservado = pHotel.reservaHotel(datosHotel);
boolean vueloReservado = pVuelos.reservaVuelo(datosVuelo);
if(!hotelReservado || !vueloReservado) {
  ut.rollback();
} else {
  ut.commit();
}
```

¿Preguntas...?