	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

## SERVICIOS WEB

### SOA en el tiempo (desde XML a los Servicios Web para SOA)

Empecemos haciendo un poco de historia de los acontecimientos claves de la industria que han surgido para dar forma a la actual plataforma SOA. A continuación, echemos un vistazo a la forma en que la emancipación de SOA como una arquitectura contemporánea ha alterado las funciones de XML y la tecnología de Servicios Web.

#### XML: una breve historia

Al igual que HTML, el Lenguaje de Marcado Extensible (XML) es una creación de la W3C derivados del popular Lenguaje de Marcado Generalizado (GML – **Generalized Markup Language**) que ha existido desde finales de los 60.

XML ganó popularidad durante el movimiento de comercio electrónico de finales de los años 90. Mediante el uso de XML, los desarrolladores fueron capaces de adjuntar el significado y contexto a cualquier información transmitida a través de los protocolos de Internet.

No sólo se utiliza XML para representar datos en una forma normalizada, el propio lenguaje se ha utilizado como base para una serie de especificaciones adicionales. El Lenguaje de Definición de Esquemas XML (XSD – **XML Schema Definition Language**) y el Lenguaje de Transformación XSL (XSLT – **XSL Transformation Language**) usan XML. Estas especificaciones, de hecho, se han convertido en piezas claves de la tecnología XML.

La representación de los datos con XML representa la capa base de la arquitectura SOA. Dentro de ella, XML establece el formato y la estructura de los mensajes que viajan a lo largo de los servicios. Los esquemas XSD preservan la integridad y la validez del mensaje de datos, y XSLT se emplea para permitir la comunicación entre las diferentes representaciones de datos. En otras palabras, no se puede hacer un movimiento dentro de SOA sin la participación de XML.

#### Servicios Web: una breve historia

En el año 2000, la W3C había recibido un pliego de condiciones para la presentación del Protocolo de Acceso a Objetos Simples (SOAP – **Simple Object Access Protocol**). Esta especificación fue originalmente diseñada para unificar (y en algunos casos sustituir) la propiedad de comunicación RPC. La idea era que cada parámetro de datos transmitidos entre componentes fuera codificado en XML, transportado y, a continuación, decodificado de nuevo en su formato nativo.

Pronto, las empresas y los proveedores de software comenzaron a ver el gran potencial de promover esta comunicación a través de Internet. Esto condujo a crear una tecnología distribuida basada en la Web, llevando a la estandarización de la comunicación entre organizaciones. Este concepto se denomina Servicios Web.

La parte más importante de un Servicio Web es su interfaz pública. Es una pieza central de información que asigna una identidad y permite su invocación posterior. Por lo tanto, una de las primeras iniciativas en apoyo a los Servicios Web fue el Lenguaje de Descripción de Servicios Web (WSDL – **Web Services Description Language**). El W3C recibió la primera presentación del lenguaje WSDL en el año 2001 y desde entonces ha continuado la revisión de esta especificación.

Para permitir la interoperabilidad, los Servicios Web requieren Internet y XML como formato de comunicación que establece un marco normalizado de mensajería. Aunque las alternativas, tales como XML-RPC, fueron examinadas, SOAP ganó como el favorito y en la industria sigue siendo uno de los principales estándares de mensajería para el uso de los Servicios Web.


Para apoyar a SOAP, el W3C respondió con nuevas versiones de la especificación para permitir RPC y hojas de estilos. Finalmente, la palabra "SOAP" ya no se considera un acrónimo de "Simple Object Access Protocol". A partir de la versión 1.2 de la especificación, se convirtió en un término independiente.

Completando la primera generación de los estándares para Servicios Web estaba la especificación UDDI. Originalmente desarrollada por UDDI.org, presentada a OASIS quien continuó su desarrollo en colaboración con UDDI.org. Esta especificación permite la creación de estándares para los registros de Descriptores de Servicios tanto dentro como fuera de los límites de la organización. UDDI proporciona la posibilidad de que los Servicios Web que se registraron puedan ser descubiertos por los solicitantes de servicios. A diferencia de SOAP y WSDL, UDDI todavía no ha alcanzado en la industria una amplia aceptación, y sigue siendo una extensión opcional a SOA.

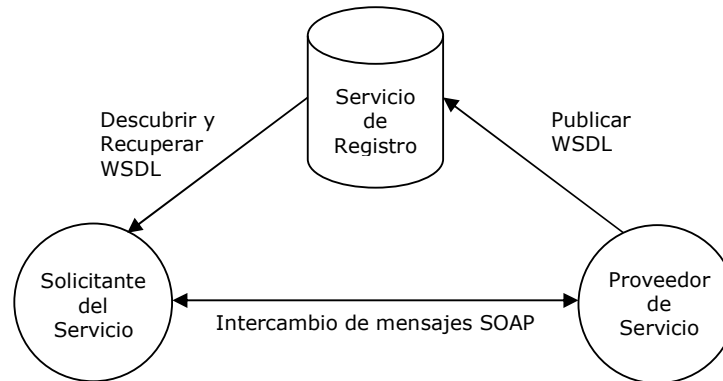
#### SOA: una breve historia

No pasó mucho tiempo antes de que las organizaciones comenzaran a darse cuenta de que en lugar de sólo alojar aplicaciones distribuidas, los Servicios Web podrían convertirse en la base de una arquitectura que podría aprovechar los beneficios de la tecnología de Servicios Web y utilizar el concepto de servicios en la empresa.

Un SOA a menudo se clasifica de diferentes maneras, a menudo dependiendo de la tecnología utilizada para construir los servicios. Uno de los primeros modelos, en su mayoría inspirado en el conjunto inicial de normas de Servicios Web,

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

define SOA como una arquitectura modelada entorno a tres componentes básicos: el Solicitante del Servicio, el Proveedor de Servicio, y el servicio de registro.



La primera generación de Servicios Web cumplen las normas de este modelo de la siguiente manera:

1. WSDL describe el servicio.
2. SOAP el formato de mensajería utilizado por el servicio y su solicitante.
3. UDDI siempre que el servicio esté normalizado con el formato de registro.

Desde una perspectiva de la arquitectura física, la primera variación de los Servicios Web basados en SOA en realidad va más allá del primitivo modelo. SOA primitivo no requiere el uso de un servicio de registro. En lugar de ello, el descubrimiento está clasificado como una de las características contemporáneas de SOA y se promueve, en un nivel de servicio, a través de principios orientados a servicios.

El primitivo modelo de SOA es fácilmente alcanzado hoy, ya que cuenta con el apoyo de todos los principales proveedores de desarrollo y plataformas de tiempo de ejecución. Numerosas de las características contemporáneas de SOA, es el resultado del desarrollo de una serie de extensiones a la primera generación de la plataforma de Servicios Web. Conocida como la "segunda generación" o "WS-\*", estas extensiones tienen como objetivo general, elevar la tecnología de Servicios Web a una plataforma para la empresa.

A través de la orientación a servicios, la lógica de negocio puede ser encapsulada y limpiamente extraída de la automatización de la tecnología subyacente. Esta visión se ha visto apoyada por el crecimiento del lenguaje de definición de procesos de negocio, en particular, WS-BPEL. Esto no sólo para permitir la descomposición de los procesos de negocio en los modelos de una serie de servicios, si no para salvar la brecha entre el análisis y la aplicación, proporcionando un lenguaje capaz de expresar plenamente la lógica de negocio en un formato concreto y ejecutable.

SOA es realmente una evolución. Su importancia hoy es el resultado de numerosas iniciativas interrelacionadas impulsadas por una variedad de organizaciones y proveedores de software. A través de un medio ambiente volátil alimentado por una mezcla de colaboración y competencia, las extensiones se están posicionando estratégicamente.

#### "Estándar" vs. "Especificaciones" frente a "Extensiones"


Estos términos se usan indistintamente. Una especificación es un documento que propone un estándar. No es oficialmente un estándar de la industria hasta que la especificación se presenta, acepta, y publica como tal por una organización.

Para evitar confusiones, estos términos se pueden definir como:

- (1) *Estándar*: Es una norma de la industria aceptada. Toda la primera generación de especificaciones de Servicios Web se considera estándares, así como una serie de especificaciones de XML.
- (2) *Especificación*: Una propuesta o estándar aceptado descrito en un pliego. Los estándares de XML, los estándares de la primera generación de Servicios Web y las extensiones WS-\*, son parte de pliegos.
- (3) *Extensión*: La extensión normalmente representa una WS-\* o especificación de una característica proporcionada por una WS-\*.

#### Framework de los Servicios Web

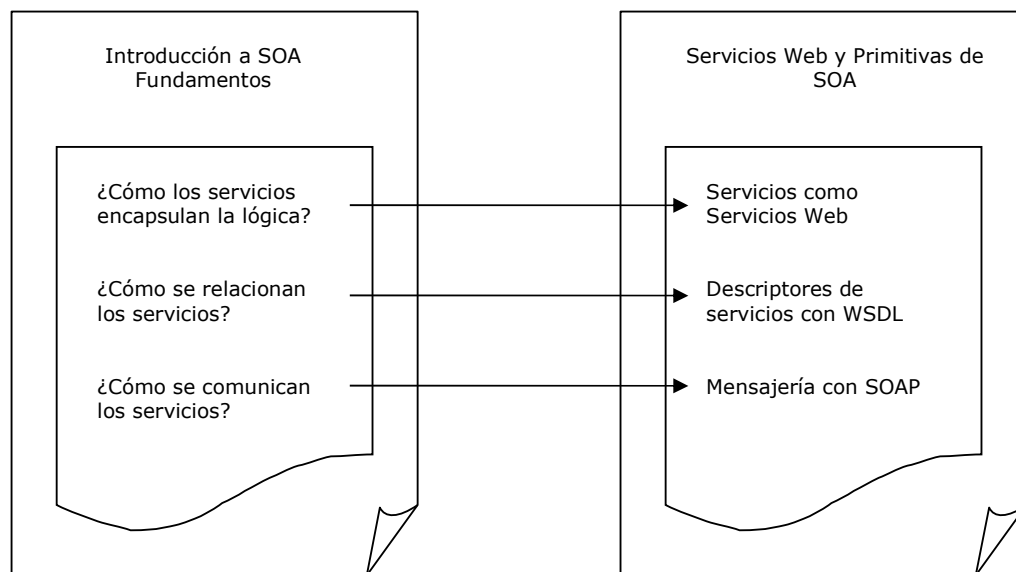
Un framework puede incluir una o más arquitecturas, tecnologías, conceptos, modelos, e incluso subframeworks. El

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

framework establecido para los Servicios Web se compone de todas estas partes. En concreto, este framework se caracteriza por:

- (1) Un resumen de la existencia de normas definidas por las organizaciones y ejecutadas por las plataformas tecnológicas.
- (2) Bloques de construcción básicos, que incluyen Servicios Web, Descriptores de Servicios, y mensajes.
- (3) Comunicaciones centradas alrededor de los Descriptores de Servicios basados en WSDL.
- (4) Un framework de mensajería integrado por tecnología y conceptos SOAP.
- (5) Descriptores de Servicios registrados y descubiertos, a veces, a través de UDDI.
- (6) Una arquitectura que soporta patrones de mensajería y sus composiciones.
- (7) Una segunda generación de extensiones de Servicios Web (también conocidas como las especificaciones WS-\*) las cuales están en constante ampliación.

Otra recomendación adicional es la lista WS-I Basic Profile. Proporciona estándares y las mejores prácticas que rigen el uso de WSDL, SOAP y UDDI. Los frameworks de Servicios Web pueden ser estandarizados a través de este perfil básico.



### Servicios como Servicios Web

Anteriormente, se introdujo el concepto de servicios y la forma en que estos proporcionan un medio de encapsular diversos grados de lógica. Manifestando los servicios del mundo real en soluciones de automatización, se requiere el uso de una tecnología capaz de preservar los fundamentos de la orientación a servicios mientras implementamos funcionalidades de negocio.

Los Servicios Web posibilitan cumplir con estos requisitos, pero tienen que ser intencionalmente diseñado para hacerlo. Esto es así porque el framework de los Servicios Web es flexible y adaptable. Los Servicios Web pueden ser diseñados para duplicar el comportamiento y la funcionalidad que encontramos en los sistemas distribuidos, o pueden ser diseñados para ser plenamente compatibles con SOA. Esta flexibilidad ha permitido que los Servicios Web puedan ser parte de muchos de los entornos de aplicaciones y ha sido una de las razones de su popularidad. También revela el hecho de que los Servicios Web no son intrínsecamente orientados a servicios.

Empecemos con una visión del conjunto de conceptos de diseño elementales para los Servicios Web. Fundamentalmente, todos los Servicios Web pueden estar asociados con:

- (1) Una clasificación temporal basada en el rol que asume el servicio durante el procesamiento en tiempo de ejecución de un mensaje.
- (2) Una clasificación permanente basada en la lógica de la aplicación que proporciona, y los roles que se asumen dentro de la arquitectura de la solución.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

A continuación, evaluaremos ambos diseños de clasificación:

- (1) Roles del servicio (clasificación temporal)
- (2) Modelos de servicios (clasificación permanente)

#### *Roles del servicio*

Un Servicio Web es capaz de asumir diferentes roles, dependiendo del contexto en el que esté involucrado. Por ejemplo, un servicio puede actuar como emisor, intermediario, o receptor de un mensaje. En contraste con la arquitectura Cliente-Servidor clásica, un servicio no actúa por tanto exclusivamente con el rol de cliente o el rol de servidor; en vez de eso, actúa como una unidad de software capaz de modificar su rol, dependiendo de su responsabilidad de procesamiento dentro del escenario en que actúa.

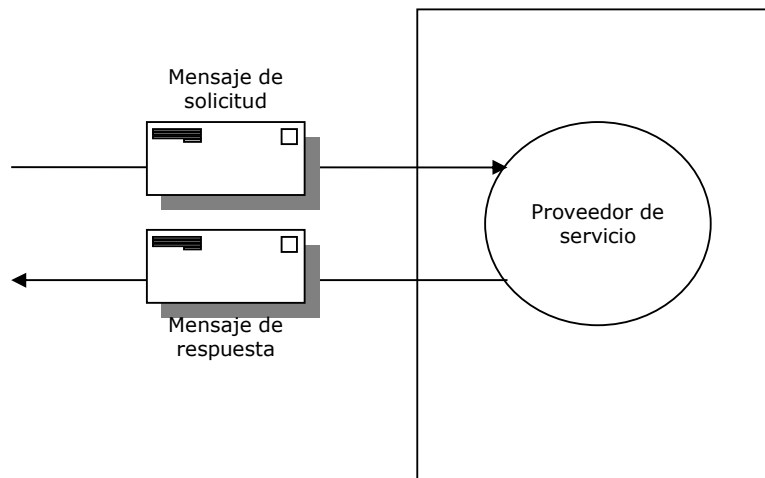
El que un Servicio Web cambie su rol más de una vez dentro de un proceso de negocio es bastante común dentro de una arquitectura SOA; el servicio asume diferentes roles dependiendo de las diferentes tareas que conllevan ese proceso de negocio.

#### Proveedor de Servicio (Service Provider)

El rol de Proveedor de Servicio es asumido por un Servicio Web bajo las siguientes condiciones:

- (1) El Servicio Web es invocado a través de una fuente externa, como un Solicitante de Servicio (Service Requestor) (ver gráfico).
- (2) El servicio proporciona una Descripción del Servicio (Service Description) con las características y el comportamiento ofrecido.

El rol de Proveedor de Servicio es análogo al rol de servidor en una arquitectura Cliente-Servidor. Dependiendo del tipo de mensaje utilizado cuando se invoca al servicio proveedor (Request Message), este podría proporcionar una respuesta a través de un mensaje de respuesta (Response Message).



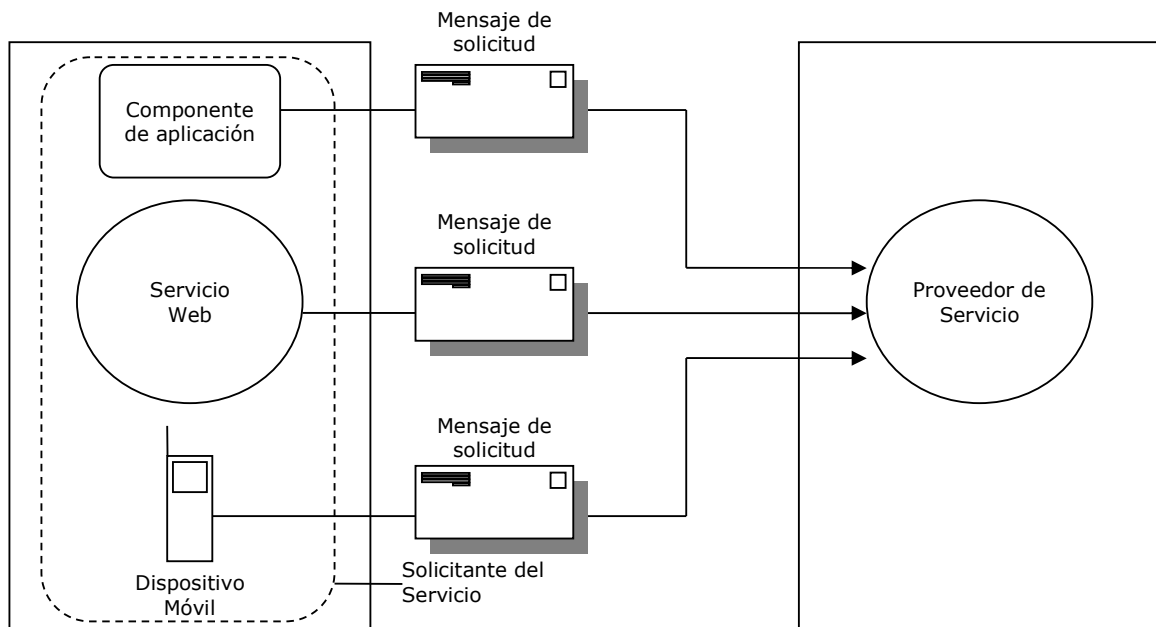
#### Solicitante del Servicio (Service Requestor)

Cualquier unidad de procesamiento lógica capaz de emitir un mensaje de solicitud (Message Request) que pueda ser procesado y entendido por el Proveedor de Servicio está clasificado como un Solicitante del Servicio. Un Servicio Web es siempre un Proveedor de Servicio pero también podría actuar como un Solicitante del Servicio.

Un Servicio Web cumple la función de Solicitante del Servicio en las siguientes circunstancias:

- (1) El Servicio Web invoca un Proveedor de Servicio mediante el envío de un mensaje (ver gráfico).
- (2) El Servicio Web busca y evalúa el más adecuado Proveedor de Servicio mediante el estudio de los Descriptores de Servicios disponibles.

El Solicitante del Servicio es el homólogo natural al Proveedor de Servicio, comparable al cliente en una típica arquitectura Cliente-Servidor. Un Solicitante del Servicio se ve mejor como un programa de software que inicia una conversación con un Proveedor de Servicio.

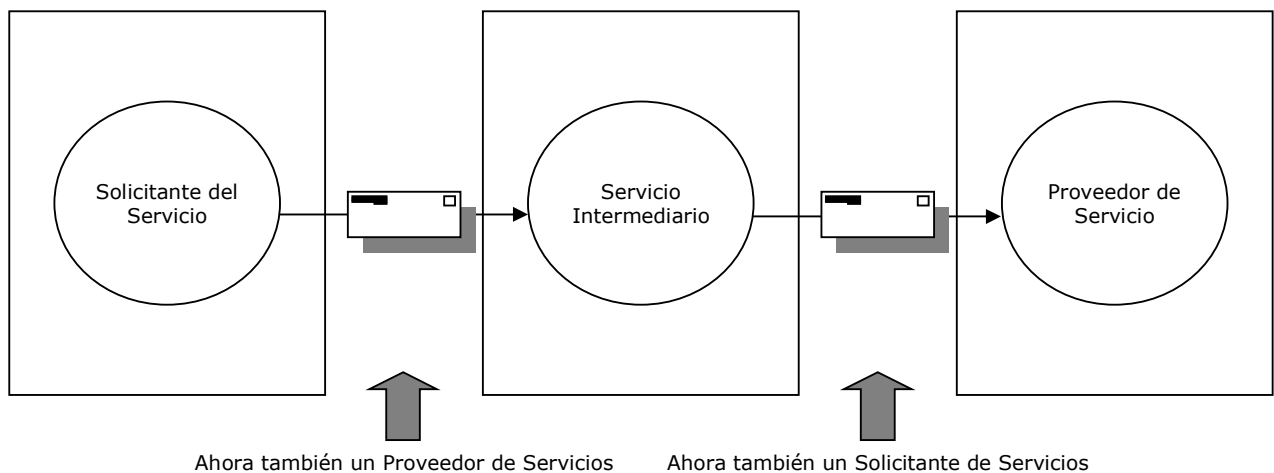


#### Servicios Intermediarios (Intermediaries)

La infraestructura de comunicación en la arquitectura de Servicios Web contrasta con la naturaleza predecible de los canales de comunicación tradicionales punto a punto. Aunque es menos flexible y escalable, la comunicación punto a punto es sencilla y fácil de implantar.

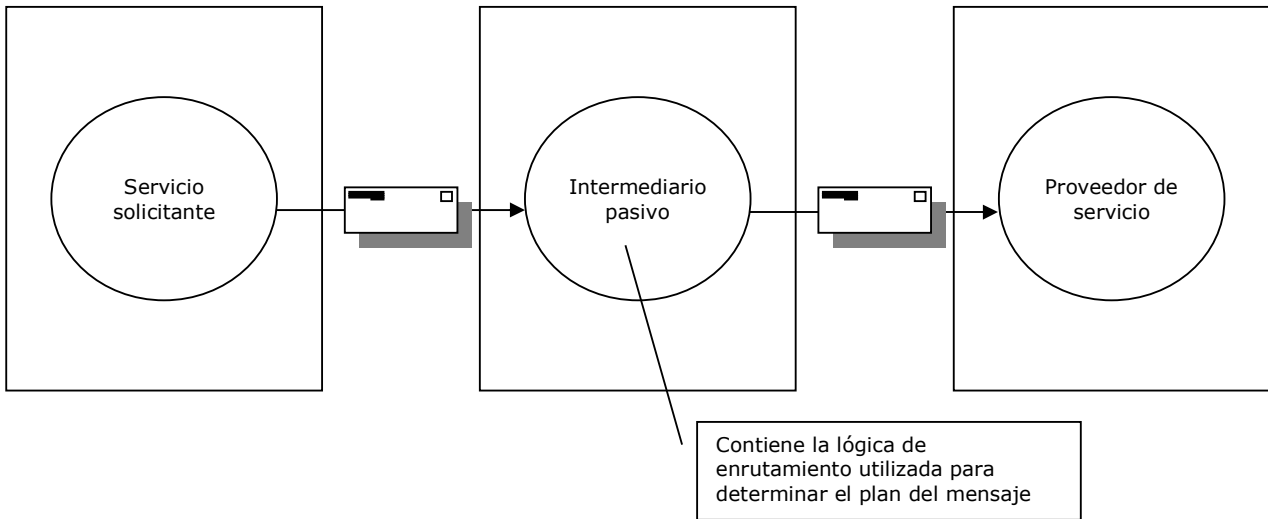
La comunicación utilizando Servicios Web está basada en la utilización de rutas de mensajes (Messaging Paths), que podrían ser descritas como rutas punto-a-\*. La razón es que una vez que un Proveedor de Servicio emite un mensaje, este puede ser procesado por múltiples rutas intermedias y agentes de procesamiento antes de llegar a destino.

Los Servicios Web y los agentes que enrutan y procesan un mensaje después que este es enviado y antes de que llegue a su destino final, se dice que son Servicios Intermediarios. Como un intermediario siempre recibe y emite mensajes, desempeñará también ambos roles: Proveedor de Servicio y Solicitante de Servicio (ver gráfico).

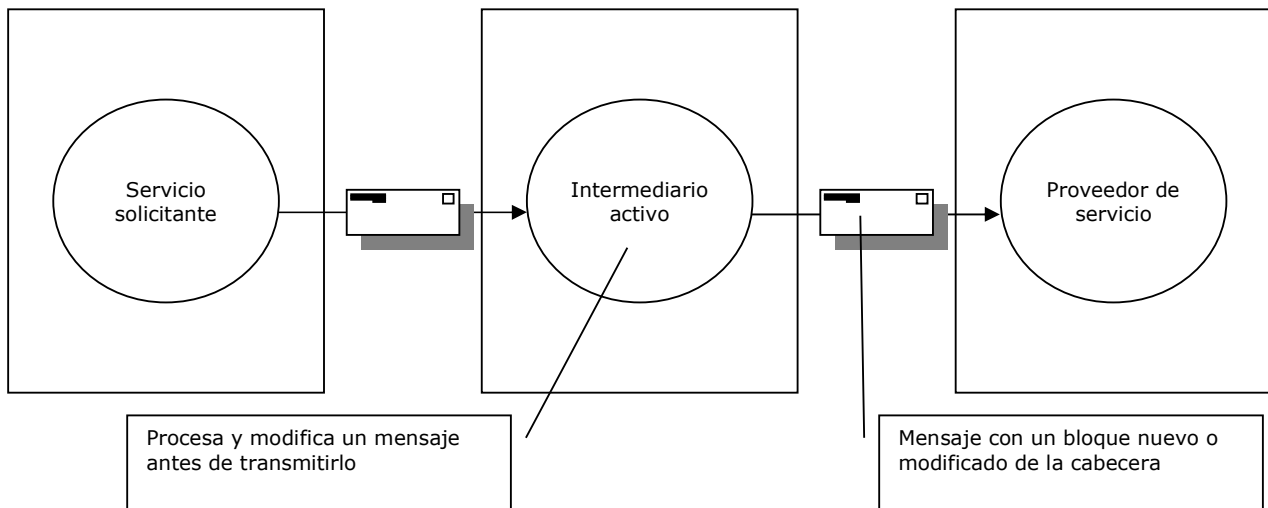


Existen dos tipos de Servicios Intermediarios:

- (1) *Servicio Intermediario Pasivo*: Responsables de enrutar los mensajes al servicio siguiente. Suelen utilizar información de la cabecera del mensaje SOAP para determinar la ruta a seguir, o utilizar lógica de enrutamiento nativa para lograr algún nivel de balanceo de carga. Lo más importante es que este tipo de servicios no modifican el mensaje.



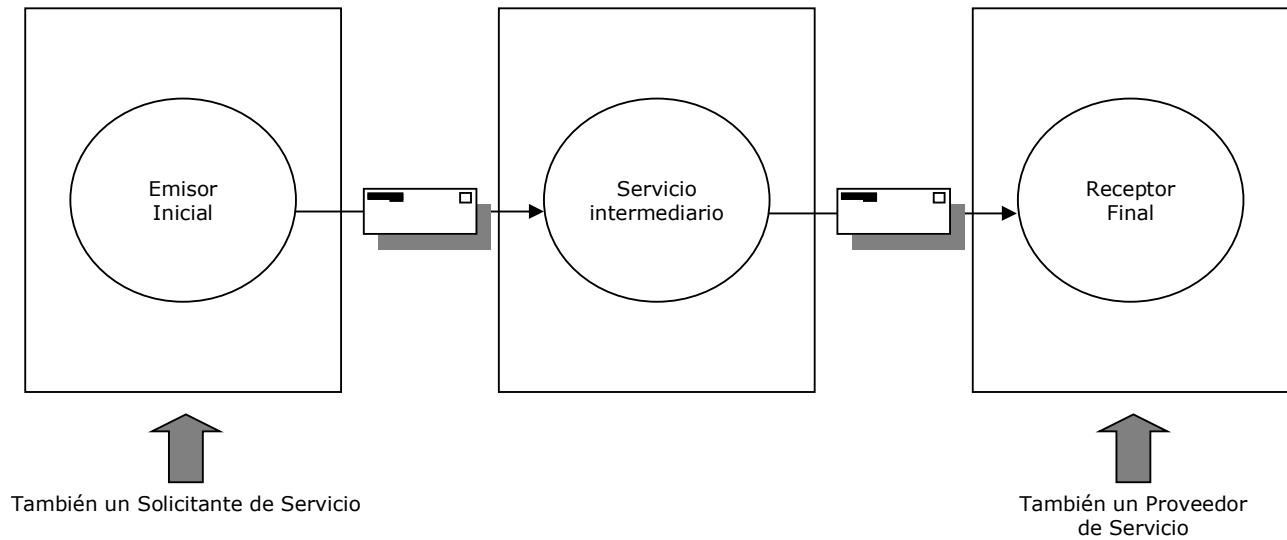
- (2) *Servicio Intermediario Activo:* A similitud de los Servicios Pasivos, los Servicios Intermediarios Activos también enrutan los mensajes al destino siguiente. Pero antes de realizar el envío del mensaje, estos servicios procesan y pueden alterar el contenido del mensaje. Típicamente, los Intermediarios Activos buscan un bloque de la cabecera SOAP particular y realizan alguna acción en base a la información que encuentran. Los datos de la cabecera suelen ser modificados, y se pueden insertar o incluso borrar bloques completos de la cabecera.



#### Emisor Inicial y Receptor Final (Initial Sender / Ultimate Receiver)

Los Emisores Iniciales son simplemente los Solicitantes de Servicios que inician la transmisión de un mensaje. Por lo tanto, el Emisor Inicial es siempre el primer Servicio Web en la ruta de un mensaje (Message Path). La contrapartida a este rol es el Receptor Final. El Receptor en realidad se corresponderá con el Proveedor de Servicio existente al final de la ruta del mensaje (ver gráfico).

Es importante darse cuenta que los Servicios Intermedios no pueden ser ni Emisores Iniciales ni Receptores Finales, dentro del contexto de esta arquitectura.

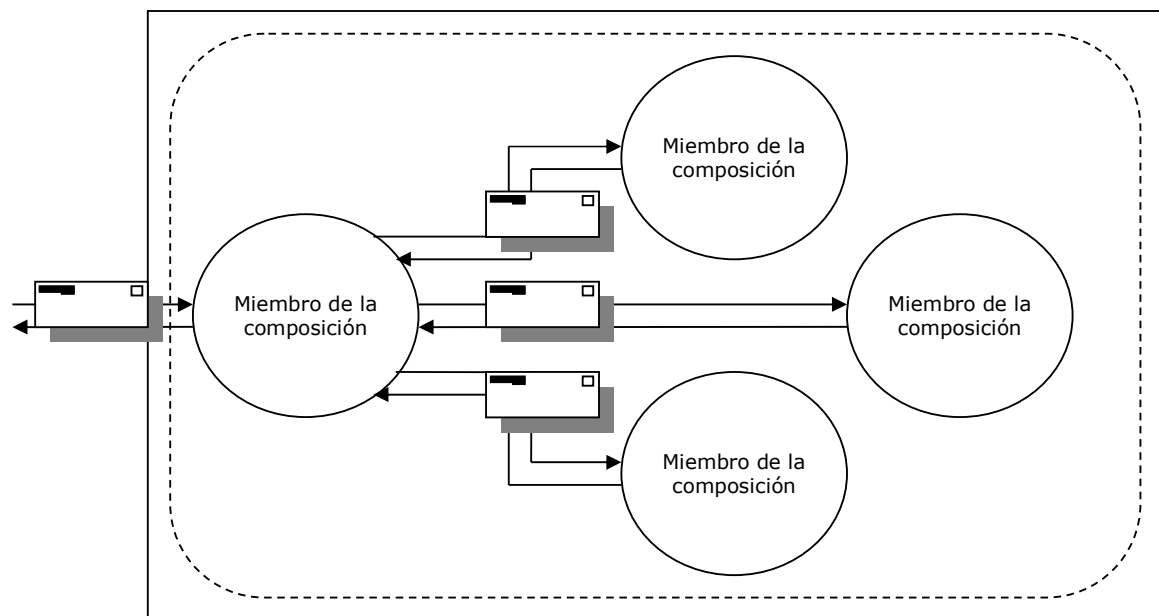


### Composición de Servicios

Tal como sugiere el nombre, este término se aplica no a un único servicio, sino a la relación existente entre una colección de servicios. Un servicio tiene la posibilidad de englobar a su vez a uno o más servicios para completar una tarea determinada. Es más, cada servicio de este conjunto tiene la posibilidad de englobar a otro conjunto de servicios para realizar una serie de subtarear. Por lo tanto, cada servicio que participa en una composición desempeña el rol individual de Miembro de la Composición de Servicios (Composition member) (ver gráfico).

Normalmente, los Servicios Web deben ser diseñados teniendo en cuenta la Composición de Servicios para ser un miembro eficaz de dicha composición. Los principios de la orientación a servicios hacen énfasis en la composición, permitiendo que algunos Servicios Web sean diseñados de tal manera que puedan ser utilizados en futuras composiciones sin un conocimiento de cómo se utilizarán.

El concepto de Composición de Servicios es muy importante para los entornos orientados a servicios. De hecho, son con frecuencia regidos por las extensiones WS-\*, tales como WS-BPEL y WS-CDL, que introducen los conceptos de orquestación y coreografía, respectivamente.



### *Modelos de servicios*

Los roles descritos al momento no ofrecen ningún tipo de información sobre la naturaleza de la funcionalidad que los Servicios Web proporcionan. Se limitan a describir estados genéricos en los que se pueden encontrar los servicios, dentro de un contexto genérico también.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

Para reflejar la forma en la que los servicios se utilizan en el mundo real, se ha creado una clasificación basada en la naturaleza de la lógica de la aplicación que proporcionan, así como sus roles de negocio dentro de la solución completa. Esta clasificación es conocida como Modelos de Servicios.

#### Modelo de Servicios de Negocio

Dentro de SOA, los Servicios de Negocio representan los pilares básicos de su construcción. Encapsulan un conjunto de la lógica de negocio dentro de una funcionalidad bien definida. Son totalmente autónomos pero no están pensados para su ejecución de manera individual, sino para su ejecución dentro de una Composición de Servicios.

Los Servicios de Negocio se usan para:

- (1) Ser bloques de construcción que representa la lógica de negocio.
- (2) Representar a una entidad corporativa o conjunto de información.
- (3) Representar la lógica de un proceso de negocio.
- (4) Ser miembro de una Composición de Servicios.

#### Modelo de Servicios Utilitarios

Cualquier Servicio Web genérico o agente de servicio diseñado para una posible reutilización puede ser clasificado como un Servicio Utilitario. La clave para lograr esta clasificación es que las funciones reutilizables sean totalmente genéricas y no específicas de la naturaleza de su aplicación.

Los Servicios Utilitarios se usan para:

- (1) Ser servicios que posibilitan la reusabilidad dentro de SOA.
- (2) Ser servicios intermediarios sin un propósito de negocio concreto.
- (3) Ser servicios que promuevan la interoperabilidad, característica intrínseca de la arquitectura SOA.
- (4) Ser servicios con el más alto grado de autonomía.

#### Modelo de Servicios Controladores

Las Composiciones de Servicios están formadas por un conjunto de servicios independientes, cada uno de ellos contribuyendo a la ejecución de una tarea de negocio global. La organización y coordinación de estos servicios es con frecuencia una tarea en sí misma, que puede ser implementada en un servicio dedicado, o como tarea secundaria en uno de los servicios del conjunto. El Servicio Controlador es el que desempeña este rol, actuando como servicio Maestro para los miembros de la Composición de Servicios.

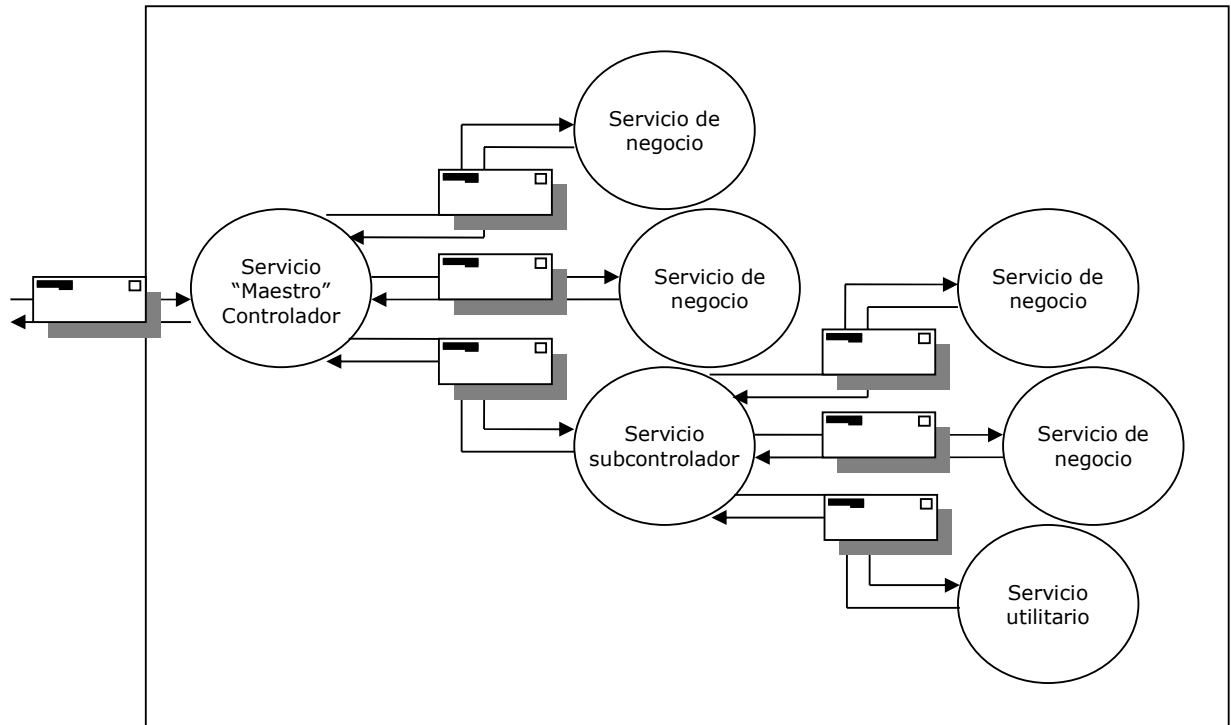
Los servicios controladores se usan dentro de SOA para:

- (1) Soportar e implementar el principio de composición.
- (2) Aumentar las posibilidades de reutilización.
- (3) Apoyar la autonomía en otros servicios.

Debe tenerse en cuenta que los Servicios Controladores pueden convertirse en miembros subordinados de la Composición de Servicios. En este caso, la composición es coordinada por un controlador, es decir, en su totalidad está integrado a una composición más amplia. En esta situación puede haber un servicio maestro que actúa como controlador de la totalidad de los servicios de composición, así como un subcontrolador, responsable de la coordinación de una parte de la composición (ver gráfico).

El modelo de Servicios Controladores se utiliza con frecuencia cuando se construye SOA con servicios especializados con distintas capas de abstracción.

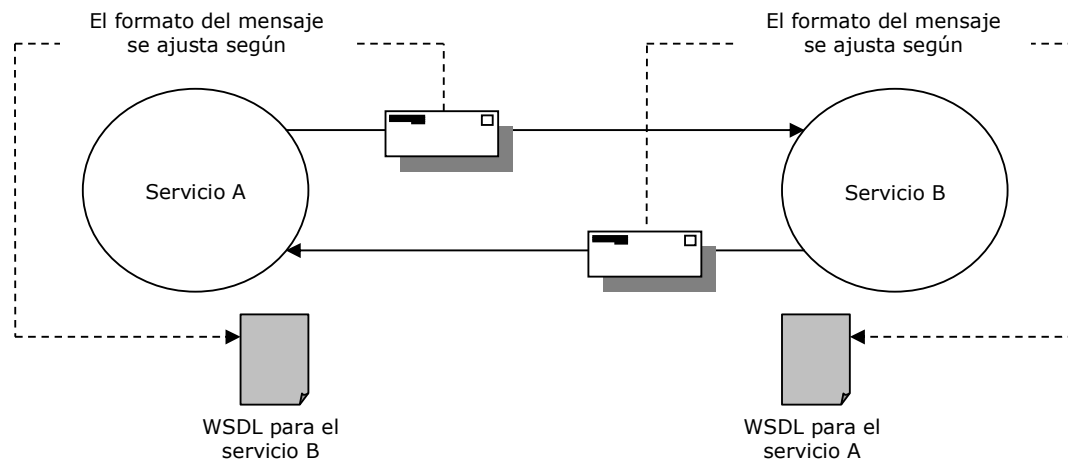




### Descriptores de Servicios

Los Descriptores de Servicios, son la parte de la arquitectura SOA que proporciona el ingrediente clave para establecer una comunicación débilmente acoplada entre Servicios Web.

Con este fin, los documentos de descripción se necesitan para acompañar a cualquier servicio que desee actuar como un receptor final. El documento principal de descripción de servicios es el WSDL (ver gráfico).




### Endpoints y Descriptores de Servicios

Un documento WSDL proporciona una vía de comunicación con el Proveedor de Servicio; este concepto se denomina normalmente Service Endpoint o simplemente Endpoint. Así, esta descripción da la información necesaria al Solicitante del Servicio acerca de la estructura de los Mensajes de Solicitud (Request Messages) y le informa acerca de la localización física (Address) del servicio.

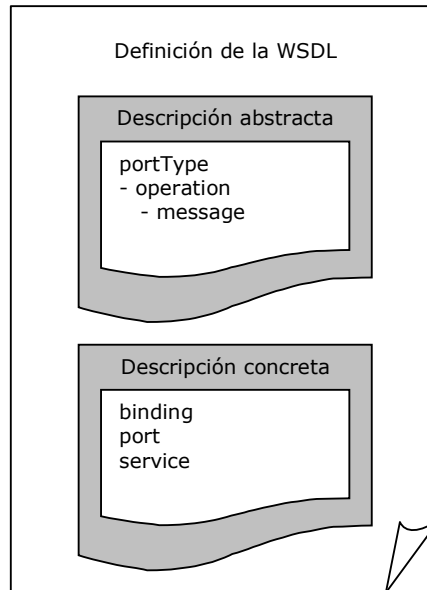
El contenido de la descripción de un servicio por medio del WSDL tiene dos partes:

- (1) Descripción Abstracta (Abstract Description)

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

## (2) Descripción Concreta (Concrete Description)

Como muestra el gráfico cada una de estas partes conforman la definición completa del WSDL. Debe tenerse en cuenta la lógica jerárquica establecida dentro de las partes de la definición abstracta.



### Descripción abstracta

La descripción abstracta describe cuales son las características del Servicio Web sin ninguna referencia a la tecnología utilizada para albergar el servicio o para transmitir los mensajes. Al separar esta información, la integridad de la descripción del servicio se puede conservar a pesar de que se produzcan cambios internos en la tecnología posteriormente.

A continuación describe cuales son las principales partes que componen la descripción abstracta.

#### **portType, operation, message**

Cada **portType** proporciona una vista de alto nivel de la interfaz del servicio, ordenando los mensajes que puede procesar en grupos de funcionalidades conocidas como **operaciones**.

Cada **operación** representa una acción específica que puede realizar el servicio. Una operación de servicio es algo similar a los métodos públicos utilizados en las arquitecturas de desarrollo distribuidas basadas en componentes. Al igual que estas arquitecturas, las operaciones tienen parámetros de entrada y salida. Dado que los Servicios Web deben basarse exclusivamente en una comunicación a través de **mensajes**, los parámetros están representados como mensajes. Por lo tanto, una operación consta de un **conjunto de mensajes de entrada y salida**.

Debe tenerse en cuenta que la secuencia de transmisión de estos mensajes puede estar regulada por un intercambio de mensajes predeterminado que también se asocia con la operación.

### Descripción concreta


Para que un Servicio Web pueda funcionar en la práctica es necesaria que su definición abstracta sea conectada a una tecnología real. Debido a que la ejecución de la lógica de la aplicación del servicio siempre involucra una comunicación, la interfaz abstracta del servicio necesita ser conectada a un protocolo de transporte "físico" concreto. Esta conexión es lo que se especifica en la descripción concreta del documento WSDL.

Está compuesta de tres partes a su vez:

#### **binding, port, service**

El elemento **binding** en la descripción WSDL determina los requerimientos para que el servicio abra una conexión "física" o para que se puedan conectar al servicio. En otras palabras, binding representa una tecnología de transporte que el servicio puede utilizar para comunicarse. El protocolo más común utilizado para este fin es SOAP, pero es posible la utilización de otros protocolos. Un binding puede ser utilizado para la interfaz completa o únicamente para una operación específica.

Muy relacionado con el binding se encuentra el **port**, que representa la dirección física por la cual se accede al servicio utilizando un protocolo específico. Se mantiene de manera separada esta dirección física para permitir que la

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.3 VIGENCIA: 19-09-2007
APUNTE DE SERVICIOS WEB Y REST		

información acerca de la localización del servicio se “mantenga” independientemente de otros aspectos de la descripción concreta. Dentro del lenguaje WSDL, el término **service** se utiliza para designar un grupo de endpoints relacionados.

Por ejemplo,

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://ws/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://ws/" name="FactorialService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws/"
schemaLocation="http://localhost:8080/FactorialService/FactorialService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="factorial">
    <part name="parameters" element="tns:factorial"/>
  </message>
  <message name="factorialResponse">
    <part name="parameters" element="tns:factorialResponse"/>
  </message>
  <portType name="FactorialService">
    <operation name="factorial">
      <input wsam:Action="http://ws/FactorialService/factorialRequest" message="tns:factorial"/>
      <output wsam:Action="http://ws/FactorialService/factorialResponse" message="tns:factorialResponse"/>
    </operation>
  </portType>
  <binding name="FactorialServicePortBinding" type="tns:FactorialService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="factorial">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="FactorialService">
    <port name="FactorialServicePort" binding="tns:FactorialServicePortBinding">
      <soap:address location="http://localhost:8080/FactorialService/FactorialService"/>
    </port>
  </service>
</definitions>
```

#### *Metadatos y los contratos de servicios*

Hasta el momento hemos establecido que la descripción abstracta y concreta proporcionada por el WSDL expresa la información técnica de cómo el servicio puede ser accedido y qué tipo de intercambio de datos puede soportar.

La definición del WSDL se basa con frecuencia en esquemas XSD para formalizar la estructura de ingreso y egreso de mensajes. Otra descripción suplementaria es un documento de políticas. Las políticas pueden proporcionar normas, preferencias, y el procesamiento de datos más allá de lo que se expresa a través del WSDL y el esquema XSD.

Así es que tenemos hasta tres documentos separados donde cada uno describe un aspecto del servicio:

- (1) Definición del WSDL
- (2) Esquema XSD
- (3) Política

Cada uno de estos tres documentos puede ser clasificado como los **metadatos** del servicio, ya que cada uno proporciona información sobre el servicio.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

Los documentos de descripción del servicio pueden ser vistos colectivamente como el establecimiento de un **contrato de servicio**, el cual es un conjunto de condiciones que debe cumplir y aceptar un solicitante potencial del servicio a fin de tener éxito en la comunicación.

#### *Descripciones semánticas*

La mayoría de los metadatos que actualmente prestan los servicios se centran en expresar información técnica relacionada con la representación de datos y procesamiento de los requisitos. Sin embargo, la descripción del servicio en estos documentos no suele resultar útil para explicar detalles sobre las características del comportamiento del servicio. De hecho, la parte más difícil de proporcionar en una descripción completa de un Servicio Web está en la comunicación de sus cualidades semánticas.

Ejemplos de la semántica de un servicio incluyen:

- (1) Cómo se comporta un servicio bajo ciertas condiciones.
- (2) Cómo un servicio responde a una condición específica.
- (3)Cuál es el servicio más adecuado para ciertas tareas específicas.

La información sobre la semántica es generalmente de mayor importancia cuando se trata con proveedores de servicios externos, cuando el conocimiento de la otra parte del servicio se limita a la información que el servicio propietario decide publicar. Pero incluso dentro de los límites organizativos, las características semánticas tienden a tener mayor relevancia a medida que la cantidad de Servicios Web internos crece.

#### *La publicación y descubrimiento del descriptor del servicio*

Como hemos establecido, el único requisito para que un servicio contacte a otro, es el acceso al descriptor del otro servicio. Como la cantidad de servicios aumenta dentro y fuera de las organizaciones, los mecanismos de publicación y descubrimiento de los Descriptores de Servicios pueden ser necesarios. Por ejemplo, los directorios y registros centralizados pueden convertirse en una opción para realizar un seguimiento de los numerosos Descriptores de Servicios que estén disponibles. Estos repositorios permiten:

- (1) Localizar la última versión del Descriptor del Servicio.
- (2) Descubrir nuevos Servicios Web que cumplan con ciertos criterios.


Cuando el conjunto inicial de normas de Servicios Web surgió, esta eventualidad se tuvo en cuenta. Esta es la razón por la que UDDI forma parte de la primera generación de los estándares de Servicios Web. Aunque todavía no se aplica comúnmente, UDDI nos proporciona un modelo de registro a considerar.

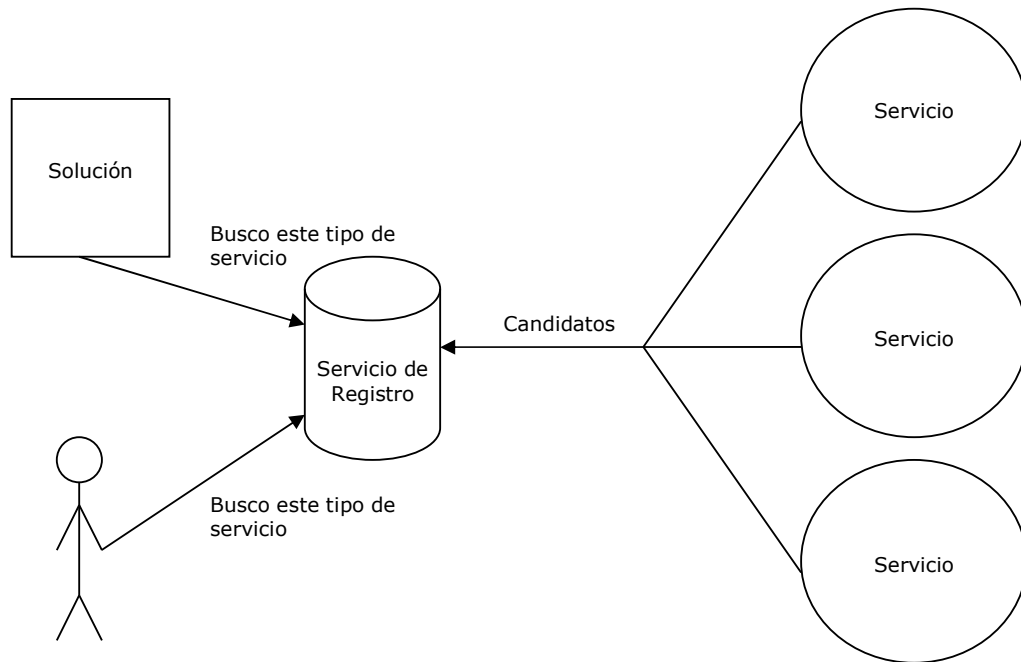
#### Registros privados y públicos

UDDI especifica un nivel relativamente aceptado para la estructuración de los registros asociados a los Descriptores de Servicios para su seguimiento correspondiente (ver gráfico). Estos registros se pueden buscar manualmente y acceder a través de la programación normalizada de la API.

Los **registros públicos** aceptan la suscripción de cualquier organización, independientemente de si tienen Servicios Web por ofrecer. Una vez suscriptas, las organizaciones en calidad de Proveedores de Servicios pueden registrar los mismos.

Los **registros privados** pueden ser implementados dentro de los límites de una organización para proporcionar un repositorio centralizado de todos los Descriptores de Servicios que la organización desarrolla, vende o compra.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007



### Mensajería con SOAP

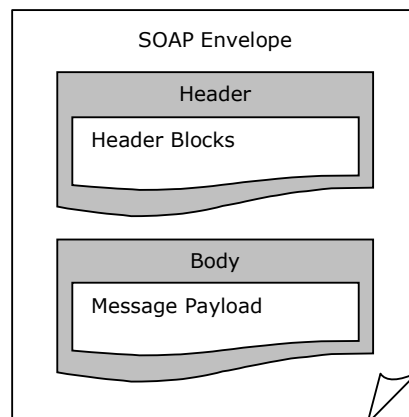
Debido a que todas las comunicaciones entre los servicios se realizan mediante mensajes, la mensajería debe ser normalizada para que todos los servicios, independientemente de su origen, utilicen el mismo formato y protocolo de transporte.

La especificación SOAP fue la elegida, y desde entonces ha sido universalmente aceptada como el estándar para el transporte de los mensajes procesados por los Servicios Web. Desde su versión inicial, SOAP se ha revisado para dar cabida a estructuras de mensajes más sofisticadas.

#### Mensajes

A pesar de que nombre originalmente contiene la palabra Protocolo (lo cual a partir de la versión 1.2 ha cambiado y se establece SOAP como nombre con entidad por sí mismo), el objetivo principal de la especificación de SOAP es definir un formato de mensaje estandarizado. La estructura en sí del formato es bastante simple, pero es la habilidad de ser extendido y "personalizado" lo que ha logrado que este estándar sea parte de las principales soluciones comerciales para la adaptación de SOA en la empresa.

Cada mensaje SOAP se encuentra dentro de un "contenedor" denominado **Envelope**. Como su nombre indica, proporciona una "envoltura" que engloba el resto de partes del mensaje (ver gráfico).



	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

Cada mensaje puede contener una cabecera, cuya función es albergar “metainformación”. En la mayoría de soluciones orientadas a servicios, esta sección es una parte vital de la arquitectura y, aunque es opcional, muy pocas veces se omite. Su importancia radica en que esta cabecera contiene los **Header Blocks** a través de los cuales se pueden implementar las diferentes extensiones.

El contenido del mensaje actual se guarda dentro del **Body**, que suele estar compuesto de datos XML con un formato común. A este contenido se le suele denominar **Message Payload** o simplemente Payload.

#### Header Blocks

Una de las principales características que se persiguen en el Framework de comunicación de SOAP es que los mensajes tengan la suficiente “inteligencia” para ser tan “autosuficientes” como sea posible. Esto implica que los mensajes SOAP tengan un nivel de independencia que a su vez incrementa la robustez y extensibilidad del framework. Es muy importante esta característica para lograr uno de los principios fundamentales de SOA, que es el bajo acoplamiento de los servicios.

Esta independencia se consigue a través de los Header Blocks, que no son otra cosa que paquetes de “metainformación” adicional que se guardan en el área de la cabecera del mensaje. Los Header Blocks “equipan” un mensaje con toda la información necesaria para que cualquier servicio en el que se encuentre el mensaje sea capaz de procesarlo y enrutarlo, acorde a sus reglas, instrucciones y propiedades.

La utilización del mecanismo anterior evita que los servicios tengan que almacenar y mantener la lógica específica de los mensajes, lo que refuerza las características de SOA asociadas a la reutilización, interoperabilidad y escalabilidad.

La utilización de los Header Blocks ha permitido que las arquitecturas basadas en Servicios Web adquieran una dimensión extensible y escalable en las arquitecturas empresariales. Prácticamente casi todas las extensiones WS-\* son implementadas utilizando Header Blocks.

Algunos ejemplos de las características que se pueden incluir en los Header Blocks son:

- (1) Instrucciones de procesamiento que pueden ser ejecutadas en los Servicios Intermediarios o el Receptor Final.
- (2) Información de routing o workflow (flujo de trabajo) asociada al mensaje.
- (3) Características acerca de la seguridad del mensaje.
- (4) Información de administración del contexto y transacciones.
- (5) Información de “correlación” (un identificador utilizado para asociar un mensaje de solicitud con un mensaje de respuesta).

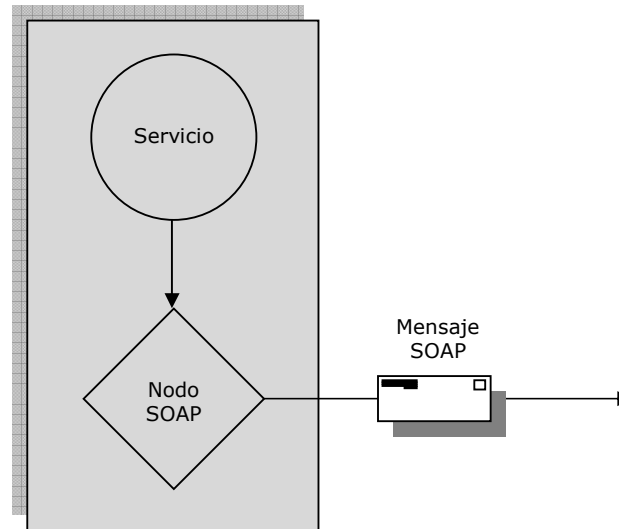
#### Body

El cuerpo contiene la parte principal del mensaje SOAP, es decir, la parte destinada al receptor final del mensaje SOAP.

#### Nodos

Aunque los Servicios Web existen como unidades de procesamiento lógico “autosuficientes”, no quita que sean dependientes de una infraestructura de comunicación física, que se encarga de procesar y administrar el intercambio de mensajes SOAP. Cada proveedor tiene su propia implementación del servidor de comunicaciones SOAP, y como resultado cada solución tiene sus propias variaciones en cuanto a los componentes software. Un nodo intermedio puede proveer servicios adicionales tales como una transformación de datos contenidos en el mensaje o ejecutar operaciones relacionadas con la seguridad. Como ya vimos, la cabecera SOAP se puede utilizar para indicar algún tipo de procesamiento adicional en un nodo intermedio, es decir, un procesamiento independiente del procesamiento realizado en el destino final.

En alto nivel, denominamos nodos SOAP (SOAP nodes) a los programas que los servicios utilizan para transmitir y recibir mensajes SOAP (ver gráfico).



Independientemente de cómo sean implementados, los nodos SOAP deben cumplir el conjunto de estándares de procesamiento de la especificación SOAP que están utilizando. Esta característica crítica es lo que hace que se pueda mantener la comunicación independientemente del Framework SOA del proveedor utilizado. Además garantiza que un mensaje SOAP pueda ser enviado por un nodo SOAP desde un servicio hacia otro nodo SOAP con la misma especificación de SOAP y poder ser procesado.

#### Transporte

Una decisión acertada hecha por los autores de SOAP consiste en la separación de la definición del mensaje y de su transporte. El resultado es una lista de posibles transportes para el mensaje SOAP, algunos más probables que otros: HTTP, SMTP, MQSeries, Raw Sockets, Files, Named Pipes, Carrier Pigeon. La mayoría de los desarrolladores se enfocaron en HTTP como el protocolo estándar de transporte para los mensajes.

HTTP es el transporte más utilizado para SOAP porque es ampliamente aceptado y difundido. La combinación de HTTP, el protocolo de transporte estándar para la Web, y SOAP, el formato de mensaje estándar, constituye una poderosa herramienta.

Hay un par de reglas básicas para el uso de HTTP como transporte de SOAP. Los mecanismos para enviar un mensaje SOAP sobre HTTP son a través del método POST. Un HTTP POST envía un bloque de datos a una URI particular en un servidor Web. El bloque de datos es el propio mensaje SOAP. Porque el mensaje SOAP es XML, el encabezado Content-Type del HTTP POST debe ser "application/soap". Si hay una respuesta al mensaje, este retorna una respuesta HTTP.

#### SOAP Request


```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:factorial xmlns:ns2="http://ws/">
      <nro>10</nro>
    </ns2:factorial>
  </S:Body>
</S:Envelope>
  
```

#### SOAP Response

```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:factorialResponse xmlns:ns2="http://ws/">
      <return>3628800</return>
    </ns2:factorialResponse>
  </S:Body>
</S:Envelope>
  
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.3 VIGENCIA: 19-09-2007
APUNTE DE SERVICIOS WEB Y REST		

## Extensión WS-\*

### XML-Signature

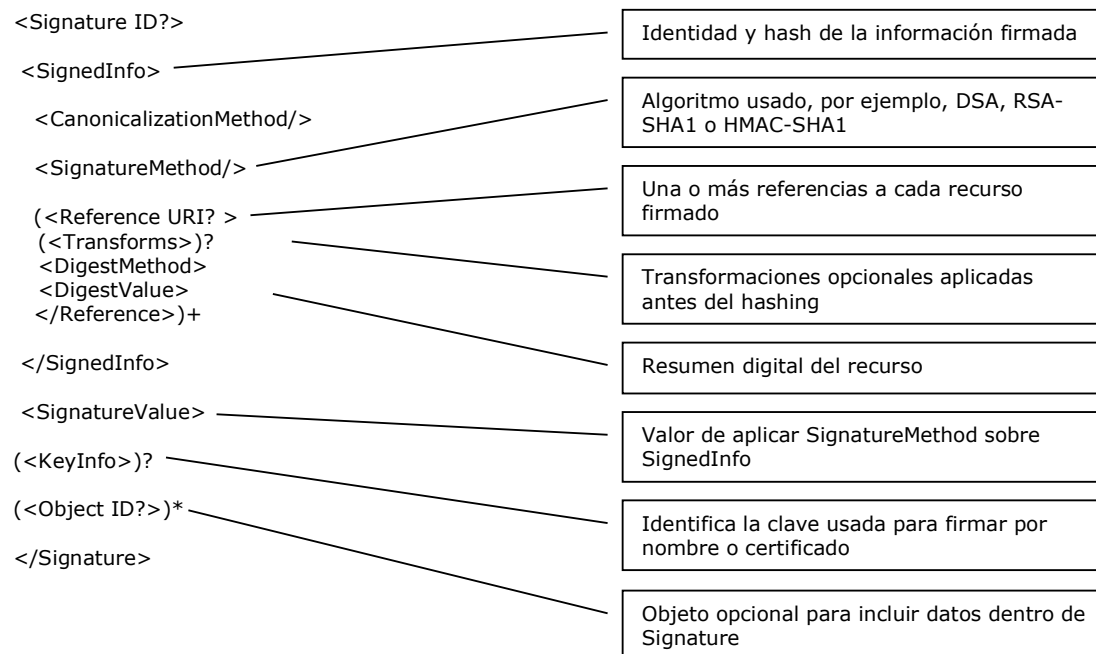
XML Digital Signature (xml-dsig) es un estándar propuesto por el IETF (Internet Engineering Task Force) y el W3C (World-Wide Web Consortium).

Define los medios para convertir una firma digital en formato XML, para firmar cualquier tipo de contenido digital y no sólo XML, así como verificar la firma realizada, proporcionando autenticación, integridad y opcionalmente no repudio. Las ventajas principales de este estándar son:

- Hacen las firmas digitales fácilmente legibles por el ojo humano.
- Son fácilmente analizables.
- Independientes de la plataforma.
- Permite realizar la firma de múltiples documentos.
- Obliga a firmar la información de los algoritmos de cifrado usados en la firma.

Una firma XML puede ser de tres tipos diferentes. Si es una firma envuelta (enveloped), ésta se encuentra en el mismo documento XML que se firma, pero no incluye su valor en la firma realizada. También puede ser envolvente (enveloping), en cuyo caso la firma envuelve a todo el documento XML que se firma. Por último, la firma puede ser independiente (detached), utilizada cuando se firma un recurso externo al elemento de firma (referenciado por una URI) o bien cuando la firma y el documento firmado residen en el mismo documento XML y son hermanos.


A continuación se muestra la estructura de una firma XML, sobre la que explicaremos el proceso de firmado y verificación de la misma:



El proceso realizado para realizar una firma XML <ds:Signature> es el siguiente:

- (1) Se aplican las transformaciones especificadas en <ds:Transforms> al documento que se quiere firmar.
- (2) Se genera el resumen digital de los objetos de datos que se quieren firmar y se almacena en el elemento <ds:DigestValue>, indicando el método de hash utilizado en <ds:DigestMethod>.
- (3) Se canoniza el elemento <ds:SignedInfo> según <ds:CanonicalizationMethod>.
- (4) El elemento <ds:SignedInfo> es firmado según <ds:SignatureMethod> y su resultado se introduce en el elemento <ds:SignatureValue>.



	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

(5) En <ds:KeyInfo> (si está presente) se introduce información para validar la firma.

(6) El elemento <ds:Object> (si existe) contiene la información firmada en el caso de usar firma envolvente.

El proceso seguido para realizar la validación de una firma XML se puede dividir en dos partes, la validación del hash de cada elemento <ds:Reference> y la validación de la firma calculada sobre el elemento <ds:SignedInfo>. A continuación se describen los pasos necesarios:

- (1) Convertir a forma canónica el contenido del elemento <ds:SignedInfo>, haciendo uso del método indicado en el elemento <ds:CanonicalizationMethod>.
- (2) Para cada elemento <ds:Reference>, se obtiene el elemento firmado y se obtiene su hash según el método indicado en su <ds:DigestMethod>.
- (3) Se compara el resultado del hash realizado con el valor del elemento <ds:DigestValue> en el elemento <ds:Reference>, en caso de no coincidir se produce un fallo en la validación.
- (4) Se obtiene la información de la clave necesaria para descifrar la firma.
- (5) Según el algoritmo indicado en <ds:SignatureMethod> se descifra la firma y se obtiene el hash inicial.
- (6) Se compara el hash obtenido con el resultado de realizar el hash al elemento <ds:SignedInfo> siguiendo el método de hash indicado en <ds:DigestMethod>.

El estándar OASIS Digital Signature Service se basa en XML-Signature para definir un protocolo de preguntas y respuestas que permiten firmar y verificar documentos tanto en formato XML como en otro cualquiera, así como añadirles sellos de tiempo.

#### XML-Encryption

Este estándar ha sido desarrollado por el W3C y su objetivo es definir un mecanismo para el cifrado de cualquier tipo de datos, y representar el resultado como un documento XML. Además, permite indicar una referencia a los datos cifrados junto con las transformaciones necesarias para recuperarlos, expresar el algoritmo de cifrado utilizado junto a sus parámetros y la información que identifica la clave necesaria para descifrar el documento (usando el elemento <ds:KeyInfo> de XML-Signature).

Un documento XML cifrado consiste en el documento original en claro, en el que el contenido cifrado se ha sustituido por un elemento EncryptedData que contiene los datos cifrados, tal y como define la especificación de XML-Encryption. De esta forma, si lo que se ha cifrado es un documento XML en su totalidad, el documento cifrado consistirá en un único elemento EncryptedData. La estructura del elemento EncryptedData es la que se muestra a continuación:

<EncryptedData Id? Type? MimeType? Encoding?>

<EncryptionMethod/>?

Algoritmo de cifrado usado, por ejemplo, AES, 3DES, RSA

<ds:KeyInfo>

<EncryptedKey>?

<AgreementMethod>?

<ds:KeyName>?

<ds:RetrievalMethod>?

<ds:\*>?

</ds:KeyInfo>

KeyInfo proporciona la clave:

- cifrada, o
- usando negociación de claves, o
- por identificación de una clave conocida, o
- mediante un enlace al lugar donde se encuentra la clave cifrada, o
- de otra forma (normalmente por la clave pública)

<CipherData>

<CipherValue>?

<CipherReference URI?>?

</CipherData>

Datos cifrados, bien su valor o bien una referencia a su localización

<EncryptionProperties>?


Información adicional sobre la generación de los datos cifrados, por ejemplo, timestamp

</EncryptedData>

El elemento <xenc:CipherData> ha de contener la secuencia de octetos cifrada y codificada en Base 64, o una referencia al lugar donde estos datos se encuentran contenida en el elemento <xenc:CipherReference>.

Tras realizar el cifrado del contenido correspondiente, el almacenamiento del resultado se puede realizar en un documento XML de dos formas, dependiendo de lo que se esté cifrando:

- (1) Si se cifra un elemento XML. El elemento <xenc:EncryptedData> resultante del proceso de cifrado pasará a reemplazar a dicho elemento XML cifrado.

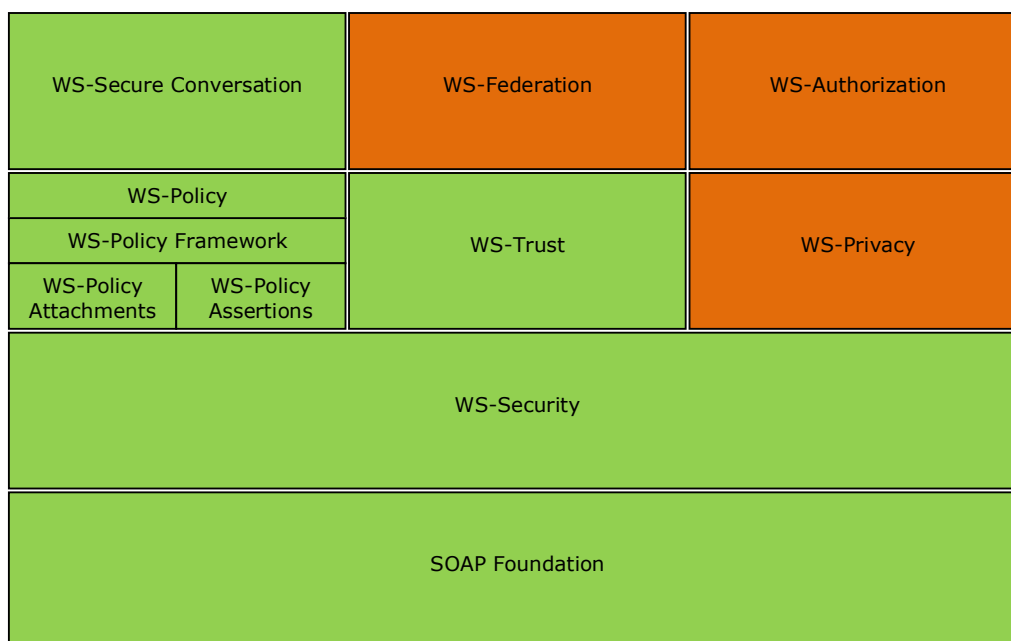
	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

- (2) Si el contenido cifrado no es un elemento XML. En esta situación el elemento <xenc:EncryptedData> resultante puede situarse como raíz de un nuevo documento XML o bien como subelemento de algún documento XML existente.

#### *Oasis Web Services Security*

WS-Security es un estándar propuesto por IBM, Microsoft y Verisign para construir Servicios Web seguros, que permitir integridad, confidencialidad, federación, autorización y gestión de políticas en los mensajes SOAP utilizados. Se define un modelo que pretende soportar, integrar y unificar diferentes modelos de seguridad, mecanismos y tecnologías de amplia aplicación (Kerberos y X.509 entre otros).

WS-Security hace uso de las especificaciones XML Signature y XML Encryption mencionadas anteriormente, así como de tokens de seguridad, para definir cómo incluir la firma digital, resumen digital y cifrado en un mensaje SOAP. Seguidamente se muestra una figura con la estructura de la arquitectura OASIS Web Services Security, cuyos elementos serán descritos con detalle a continuación.



Los mecanismos de integridad están diseñados para soportar múltiples firmas, realizadas por varios actores SOAP y para extenderse con soporte para nuevos formatos de firma.


Los mecanismos de confidencialidad están diseñados para soportar procesos de encriptación adicionales y operaciones de cifrado/descifrado realizadas por varios actores SOAP.

#### *WS-Security*

Esta es la especificación base de la arquitectura. Es un estándar definido por OASIS y proporciona integridad, confidencialidad y opcionalmente no repudio a los mensajes SOAP intercambiados entre Servicios Web.

#### *WS-Policy*

Esta especificación define un marco de trabajo para administrar las políticas para Servicios Web. Su objetivo es manejar la seguridad de Servicios Web de cierta complejidad, debemos tener un amplio conocimiento del contrato XML del Servicio además de cualquier requisito, capacidad, y preferencias adicionales (también llamado política). El cliente de este tipo de Servicios necesita conocer si un Servicio concreto utiliza realmente WS-Security, y si es así, también necesita conocer qué tokens de seguridad es capaz de procesar (tokens de seguridad de tipo UserName, tickets Kerberos o certificados X.509), y cuál de ellos prefiere. El cliente debe determinar también si el Servicio requiere o no mensajes firmados. Y, finalmente, el cliente debe determinar cuándo cifrar los mensajes, qué algoritmos utilizar, y cómo intercambiar una clave secreta con el Servicio. Intentar la integración con un Servicio sin conocer todos estos detalles supone un grandísimo esfuerzo del que será muy difícil obtener compensación adecuada. Sin una forma estándar de acordar las políticas, los desarrolladores quedan abandonados a su suerte y se ven en la situación de tener que desarrollar soluciones ad-hoc que no se integren ni sean escalables. Un marco de trabajo de políticas permitiría a los desarrolladores expresar las políticas de los Servicios de una forma procesable por las computadoras. La infraestructura de los Servicios Web puede verse mejorada para entender ciertas políticas y forzar su uso en tiempo

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.3 VIGENCIA: 19-09-2007
APUNTE DE SERVICIOS WEB Y REST		

de ejecución. Por ejemplo, un desarrollador podría escribir una política que indique que un Servicio dado requiere tickets Kerberos, firmas digitales y cifrado. Otros desarrolladores podrían utilizar la información de las políticas para determinar si pueden o no utilizar el Servicio, y todo ello en tiempo de ejecución sin la necesidad de ninguna intervención externa adicional. La infraestructura podría imponer estos requisitos sin obligar al desarrollador a tener que escribir ni una sola línea de código. Por lo tanto, un marco de trabajo para definición y administración de políticas como el que aquí se describe no sólo provee una capa de descripción adicional, sino que también ofrecerá a los desarrolladores un modelo de programación más práctico y declarativo. WS-Policy define un modelo genérico y una sintaxis para describir y comunicar las políticas de los Servicios Web.

#### *WS-Trust*

La especificación WS-Trust define extensiones construidas sobre WS-Security y que consisten en ampliar la capacidad de los mecanismos de seguridad definidos por ésta, permitiendo la solicitud, emisión e intercambio de tokens de seguridad y la gestión de las relaciones de confianza.

Esta especificación, a partir de estos mecanismos básicos, define primitivas adicionales junto con extensiones en los mensajes SOAP para el intercambio de tokens de seguridad. Este intercambio permite la emisión y propagación de las credenciales dentro de diferentes dominios de confianza y gestionar así las relaciones de confianza. Para garantizar la comunicación segura entre las partes, ambas deben intercambiar credenciales de seguridad (directa o indirectamente), que son la base de la confianza entre los distintos Servicios que se comunican. Aunque cada interlocutor, de forma independiente, determina si puede confiar en las credenciales presentadas por el otro. La especificación define además un protocolo agnóstico que permite emitir, renovar e intercambiar tokens de seguridad. El objetivo principal de esta especificación es, por lo tanto, habilitar a los sistemas para que puedan crear patrones de intercambio de mensajes que sean confiables.

Las extensiones que WS-Trust proporciona sobre WS-Security son:

- Métodos para emitir, renovar y cambiar (un tipo por otro) tokens de seguridad.
- Métodos para establecer y acceder a las relaciones de confianza presentes.

Mediante el uso de estas extensiones, las aplicaciones pueden dedicarse al diseño de las infraestructuras necesarias para obtener una comunicación segura que se integre dentro del marco de trabajo general de los Servicios Web. Por ejemplo, incluyendo descripciones de Servicios WSDL, estructuras de entradas UDDI businessServices y bindingTemplates, y mensajes SOAP.

El establecimiento de un token de contexto de seguridad o la derivación e intercambio de claves quedan fuera del alcance de la especificación WS-Trust. De hecho, ambas características son desarrolladas en la especificación WS-SecureConversation.

#### *WS-SecureConversation*

La especificación WSSecureConversation define un conjunto de extensiones para permitir el establecimiento, compartición y derivación de contextos y claves de seguridad entre los interlocutores de una comunicación. Mientras que la especificación WS-Security se centra en un modelo de autenticación de mensajes, que aunque resultará muy útil en muchos casos, se encuentra sujeto a múltiples formas de ataque, WS-SecureConversation introduce el concepto de seguridad contextual y describe como utilizarlo. Esto permite que se establezcan contextos y que se realicen intercambios más eficientes de material de criptográfico, consiguiendo el incremento global del rendimiento y de la seguridad en los intercambios posteriores.

El modelo de autenticación basado en el establecimiento de contextos de seguridad autentica una serie de mensajes requiriendo una serie de comunicaciones adicionales como paso previo a los intercambios de la información de negocio entre los Servicios Web. Para implementar este modelo, WS-SecureConversation define una serie de cabeceras y extensiones nuevas del marco de trabajo de mensajería SOAP.

#### *WS-Federation*


Esta especificación define los mecanismos necesarios para conseguir la federación de dominios de seguridad distintos o similares y lo consigue permitiendo e intermediando la confianza de las identidades, atributos, y autenticación de los Servicios Web participantes.

Define un modelo y un marco de trabajo general para conseguir la federación. Se basa en perfiles que detallarán cómo los distintos solicitantes encajan en este modelo.

Es por tanto un objetivo prioritario de ésta especificación habilitar la federación de la información de identidades, sus atributos, autenticación y autorización.

Los requisitos que guían la elaboración de esta especificación son los siguientes:

- (1) Habilitar una compartición apropiada de los datos de la identidad, atributos, autenticación y autorización utilizando diferentes mecanismos.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

- (2) Intermediación de la confianza e intercambios de tokens de seguridad.
- (3) Las identidades locales de un dominio de seguridad no tienen porqué encontrarse en los otros Servicios con lo que se realizará la federación. Es decir, es requisito que mi dominio de seguridad foo.com defina un usuario A y que yo pueda federar su identidad y atributos en otro dominio de seguridad abc.ar sin la necesidad de que éste último tenga que conocer la identidad del usuario A.
- (4) Ocultamiento opcional de la información de la identidad y otros atributos. Es decir, el usuario A en el dominio foo.com puede acceder a distintos recursos en los dominios abc.ar y xyz.org y su navegación no debería poder ser trazable a través de estos dominios y recursos ya que su identidad real "A" está oculta mediante algún mecanismo de alias o pseudónimos.

#### WS-Authorization

El principal objetivo de esta especificación es cómo definir y gestionar las políticas de control de acceso a los Servicios Web. Se centra especialmente en describir como las sentencias de autorización pueden especificarse dentro de los tokens de seguridad y como estas sentencias son interpretadas en las entidades finales.

La especificación WS-Authorization está diseñada para ser flexible y extensible con respecto a los lenguajes y formatos de token manejados en los procesos de autorización. Con esto se pretende ampliar al máximo el rango de escenarios aplicables y asegurar la viabilidad del marco de trabajo de seguridad a largo plazo.

#### WS-Privacy

Esta especificación utiliza los mecanismos definidos por la especificación WS-Security, WS-Policy y WS-Trust para permitir la transmisión de las políticas de privacidad. Estas políticas de privacidad serán definidas por la organización propietaria del Servicio Web y los mensajes SOAP recibidos deberán ser conformes con dichas políticas de privacidad. WS-Privacy indica cómo incluir afirmaciones de políticas de privacidad mediante WS-Policy y utilizará WS-Trust para evaluar las sentencias de privacidad encapsuladas en los mensajes SOAP contra las preferencias de los usuarios y las políticas de la organización.

### Orquestación y coreografía de servicios


Las tecnologías básicas para el desarrollo de los Servicios básicos y la interconexión de estos, han alcanzado ya un nivel de madurez aceptable, aunque siguen quedando algunas cuestiones por resolver. Sobre todo a la hora de gestionar aquellas interacciones de Servicios que van más allá de las simples secuencias de mensajes de peticiones y respuestas o que involucran a un gran número de participantes.

<b>Composición</b> BPEL, BPELJ, WS-CDL	
<b>Coordinación – Contexto – Transacciones – Seguridad</b> WS-Coordination, WS-AtomicTransactions, WS-Security,...	
<b>Descripción</b> WSDL, WS-Policy	<b>Descubrimiento</b> UDDI
<b>Mensajes</b> SOAP, WS-Addressing, WS-ReliableMessaging	
<b>Transporte</b> HTTP, SMTP, HTTPS	<b>Formato</b> XML

La capa de composición de la figura anterior es la que se encarga de la interconexión entre Servicios y procesos de negocios. La creciente preocupación por el desarrollo de estándares para esta capa refleja los enlaces que existen entre la Gestión de Procesos de Negocio (BPM) y SOA. Por un lado, las técnicas de BPM se apoyan en SOA como paradigma de gestión de recursos, descripción de procesos o captura de interacciones entre un proceso y su entorno. Por otro lado, un Servicio debe servir como punto de entrada para un proceso de negocio subyacente, de ese modo podemos ver como existe una relación inherente entre un modelo de Servicio y un modelo de proceso. Además, los Servicios deben participar en interacciones con otros Servicios en el contexto de procesos de negocio colaborativo. Si observamos dentro de la composición de Servicios nos encontramos con dos estándares, BPEL y WS-CDL.

#### BPEL

BPEL está basado en el concepto de orquestación de Servicios Web. La orquestación se utiliza principalmente en procesos de negocio privadas. En este tipo de integración un proceso central (que puede ser otro Servicio Web) toma el control de los diferentes Servicios Web y se encarga de su coordinación. Los Servicios Web implicados en la

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

orquestración no tienen conocimiento de formar parte de dicha integración. Solamente el coordinador central conoce el objetivo que debe alcanzar el proceso.

Es este tipo de integración el que implemente el lenguaje Business Process Execution Language (BPEL), basado en XML y que permite definir procesos de negocio con Servicios Web.

Cada compañía tiene una forma única de definir los flujos de procesos de negocio. El objetivo principal de BPEL es estandarizar el formato de dichos flujos de procesos de negocio para permitir homogeneizar las interacciones entre las compañías.

BPEL extiende el modelo de interacción de los Servicios Web y permite el soporte a las transacciones. Está basado en los Servicios Web desde el punto de vista de que cada proceso de negocio implicado está definido como un Servicio Web.


Los procesos escritos en BPEL pueden orquestar interacciones entre Servicios Web utilizando documentos XML de una manera estandarizada. Estos procesos pueden ser ejecutados en cualquier plataforma o producto que soporte la especificación BPEL.

#### *WS-CDL*

WS-CDL se basa en el concepto de coreografía de Servicios Web. Una coreografía describe una colaboración entre una colección de Servicios con el fin de lograr un objetivo común. Describe las interacciones en las que participan los diferentes Servicios para conseguir este objetivo y las dependencias entre dichas interacciones, ocasionadas por el control de flujo, los mensajes correlacionados, restricciones de tiempo, etc.

Otra característica de una coreografía es que no describen las acciones internas que ocurre dentro de los Servicios que participan si estas no producen ningún efecto visible externamente (como podrían ser una transformación o cálculo de un dato). En otras palabras, una coreografía comprende todas las interacciones entre los participantes que son relevantes con respecto al objetivo de la coreografía.

El estándar WS-CDL es el lenguaje utilizado para la especificación de protocolos punto a punto donde cada participante quiere actuar de forma autónoma y ninguno de ellos es líder sobre ningún otro. Es decir, a diferencia de la orquestración, se trata de un escenario donde no existe ningún punto de centralización o coordinador.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

## REST

### ¿Qué es REST?

REST (Representational State Transfer) define un set de principios arquitectónicos por los cuales se diseñan servicios Web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP.

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST está basado en estándares:

- HTTP
- URL
- Representación de los recursos: XML/HTML/GIF/JPEG/...
- Tipos MIME: text/xml, text/html, ...

Los objetivos de este estilo de arquitectura se listan a continuación:

- (1) *Escalabilidad de la interacción con los componentes.* La Web ha crecido exponencialmente sin degradar su rendimiento. Una prueba de ellos es la variedad de clientes que pueden acceder a través de la Web: estaciones de trabajo, sistemas industriales, dispositivos móviles,...
- (2) *Generalidad de interfaces.* Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial. Esto no es del todo cierto para otras alternativas, como SOAP para los Servicios Web. Quienes hayan usado SOAP para Servicios Web, sabrán que es bien fácil de diseñar, pero algo complicado de consumir: se necesita toda una API para construir los clientes utilizando el WSDL. Por ejemplo, para PHP se necesita de la biblioteca NuSOAP.
- (3) *Puesta en funcionamiento independiente.* Este hecho es una realidad que debe tratarse cuando se trabaja en Internet. Los clientes y servidores pueden ser puestas en funcionamiento durante años. Por tanto, los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido.
- (4) *Compatibilidad con componentes intermedios.* Los más populares intermediarios son varios tipos de proxys para Web. Algunos de ellos, las caches, se utilizan para mejorar el rendimiento. Otros permiten reforzar las políticas de seguridad: firewalls. Y por último, otro tipo importante de intermediarios, gateway, permiten encapsular sistemas no propiamente Web. Por tanto, la compatibilidad con intermediarios nos permite reducir la latencia de interacción, reforzar la seguridad y encapsular otros sistemas.

REST logra satisfacer estos objetivos aplicando cuatro restricciones:

- (1) *Identificación de recursos y manipulación de ellos a través de representaciones.* Esto se consigue mediante el uso de URIs. HTTP es un protocolo centrado en URIs. Los recursos son los objetos lógicos a los que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Cuando se utiliza un método PUT para enviar información, se toma como una representación de lo que nos gustaría que el estado del recurso fuera. Internamente el estado del recurso puede ser cualquier cosa desde una base de datos relacional a un fichero de texto.
- (2) *Mensajes autodescriptivos.* REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos que HTTP logra esto es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. Por ejemplo, las cachés Web saben que por defecto el comando GET es cacheable (ya que es side-effect-free) en cambio POST no lo es. Además saben cómo consultar las cabeceras para controlar la caducidad de la información. HTTP es un protocolo sin estado y cuando se utiliza adecuadamente, es posible es posible interpretar cada mensaje sin ningún conocimiento de los mensajes precedentes.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

- (3) *Hipermedia como un mecanismo del estado de la aplicación.* El estado actual de una aplicación Web debería ser capturada en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce sobre el estado de sus recursos, aunque no intenta seguirle la pista a las sesiones individuales de los clientes. Esta es la misión del navegador, el sabe como navegar de recurso a recurso, recogiendo información que el necesita o cambiar el estado que él necesita cambiar.

Por tanto, una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- (1) Utiliza los métodos HTTP de manera explícita.
- (2) No mantiene estado.
- (3) Expone URIs con forma de directorios.
- (4) Transfiere XML, JavaScript Object Notation (JSON), o ambos.

*REST utiliza los métodos HTTP de manera explícita*

Una de las características claves de los servicios Web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor Web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- Se usa POST para crear un recurso en el servidor.
- Se usa GET para obtener un recurso.
- Se usa PUT para cambiar el estado de un recurso o actualizarlo.
- Se usa DELETE para eliminar un recurso.

Como un principio de diseño general, ayuda seguir las reglas de REST que aconsejan usar sustantivos en vez de verbos en las URIs. En los servicios web REST, los verbos están claramente definidos por el mismo protocolo: POST, GET, PUT y DELETE. Idealmente, para mantener una interfaz general y para que los clientes puedan ser explícitos en las operaciones que invocan, los servicios web no deberían definir más verbos o procedimientos remotos, como ser /agregarusuario y /actualizarusuario. Este principio de diseño también aplica para el cuerpo de la petición HTTP, el cual debe usarse para transferir el estado de un recurso, y no para llevar el nombre de un método remoto a ser invocado.

*REST no mantiene estado*

Los servicios web REST necesitan escalar para poder satisfacer una demanda en constante crecimiento. Se usan clusters de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways de manera de conformar una topología de servicio, que permita transferir peticiones de un equipo a otro para disminuir el tiempo total de respuesta de una invocación al servicio web. El uso de servidores intermedios para mejorar la escalabilidad hace necesario que los clientes de servicios web REST envíen peticiones completas e independientes; es decir, se deben enviar peticiones que incluyan todos los datos necesarios para cumplir el pedido, de manera que los componentes en los servidores intermedios puedan redireccionar y gestionar la carga sin mantener el estado localmente entre las peticiones.


Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o cliente de servicio web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

*REST expone URIs con forma de directorios*

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el web service REST, y si el servicio va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URI de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.



	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.3 VIGENCIA: 19-09-2007
APUNTE DE SERVICIOS WEB Y REST		

Una forma de lograr este nivel de usabilidad es definir URIs con una estructura al estilo de los directorios. Este tipo de URIs es jerárquica, con una única ruta raíz, y va abriendo ramas a través de las subrutas para exponer las áreas principales del servicio. De acuerdo a esta definición, una URI no es solamente una cadena de caracteres delimitada por barras, sino más bien un árbol con subordinados y padres organizados como nodos. Por ejemplo, en un servicio de hilos de discusiones que tiene temas varios, se podría definir una estructura de URIs como esta:

<http://www.ubp.edu.ar/discusion/temas/{tema}>

La raíz, /discusion, tiene un nodo /temas como hijo. Bajo este nodo hay un conjunto de nombres de temas (como ser tecnología, actualidad, y más), cada uno de los cuales apunta a un hilo de discusión. Dentro de esta estructura, resulta fácil recuperar hilos de discusión al tipear algo después de /temas/.

Podemos también enumerar algunas guías generales más al momento de crear URIs para un servicio web REST:

- (1) Ocultar la tecnología usada en el servidor que aparecería como extensión de archivos (.jsp, .php, .asp), de manera de poder portar la solución a otra tecnología sin cambiar las URI.
- (2) Mantener todo en minúsculas.
- (3) Sustituir los espacios con guiones o guiones bajos (uno u otro).
- (4) Evitar el uso de query.
- (5) En vez de usar un 404 Not Found si la petición es una URI parcial, devolver una página o un recurso predeterminado como respuesta.

Las URI deberían ser estáticas de manera que cuando cambie el recurso o cambie la implementación del servicio, el enlace se mantenga igual. Esto permite que el cliente pueda generar "favoritos". También es importante que la relación entre los recursos que está explícita en las URI se mantenga independiente de las relaciones que existen en el medio de almacenamiento del recurso.

*REST transfiere XML, JSON, o ambos*

La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples "fotos" en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y etiquetas XML, donde los valores de los elementos en el XML contienen los valores de las filas. O, si el sistema tiene un modelo de datos, la representación de un recurso es una fotografía de los atributos de una de las cosas en el modelo de datos del sistema. Estas son las cosas que se proporciona con servicios web REST.

La última restricción al momento de diseñar un servicio web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las peticiones/respuestas. Aquí es donde realmente vale la pena mantener las cosas simples, legibles por humanos, y conectadas.


Los objetos del modelo de datos generalmente se relacionan de alguna manera, y las relaciones entre los objetos del modelo de datos (los recursos) deben reflejarse en la forma en la que se representan al momento de transferir los datos al cliente. En el servicio de hilos de discusión anterior, un ejemplo de una representación de un recurso conectado podría ser un tema de discusión raíz con todos sus atributos, y links embebidos a las respuestas al tema.

```
<discusion fecha="{fecha}" tema="{tema}">
  <comentario>{comentario}</comentario>
  <respuestas>
    <respuesta de="mpastarini@ubp.edu.ar" href="/discusion/temas/{tema}/mpastariniUBP"/>
    <respuesta de="mpastar@gmail.com" href="/discusion/temas/{tema}/mpastariniGMAIL"/>
  </respuestas>
</discusion>
```

Por último, es bueno construir los servicios de manera que usen el atributo HTTP Accept del encabezado, en donde el valor de este campo es el tipo MIME. De esta manera, los clientes pueden pedir por un contenido en particular que mejor pueden analizar. Algunos de los tipos MIME más usados para los servicios web REST son:

MIME-Type	Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml




	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	VERSIÓN: 1.3 VIGENCIA: 19-09-2007
APUNTE DE SERVICIOS WEB Y REST		

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado HTTP Accept es un mecanismo conocido como negociación de contenido, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

### ¿Cómo diseñar un servicio Web basado en REST?

De todo lo antes expuesto, podemos determinar que las pautas a seguir en el diseño de servicios Web basados en REST son:

- (1) Identificar todas las entidades conceptuales que se desean exponer como servicio.
- (2) Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos (acciones). Por ejemplo no utilizar esto:  
  
<http://www.service.com/entities/getEntity?id=001>  
  
 Como podemos observar, getEntity es un verbo. Mejor utilizar el estilo REST, un nombre:  
  
<http://www.service.com/entities/001>
- (3) Categorizar los recursos de acuerdo con si los clientes pueden obtener una representación del recurso o si pueden modificarlo. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.
- (4) Todos los recursos accesibles mediante GET no deberían tener efectos secundarios. Es decir, los recursos deberían devolver la representación del recurso. Por tanto, invocar al recurso no debería ser el resultado de modificarlo.
- (5) Ninguna representación debería estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.
- (6) Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema,...). Para los servicios que requieran un POST o un PUT es aconsejable también proporcionar un esquema para especificar el formato de la respuesta.
- (7) Describir como nuestro servicio ha de ser invocado, mediante un documento WSDL/WADL o simplemente HTML.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO DE SOFTWARE – 10º CUATRIMESTRE	
	APUNTE DE SERVICIOS WEB Y REST	VERSIÓN: 1.3 VIGENCIA: 19-09-2007

#### **BIBLIOGRAFÍA**

[Costello] Roger L. Costello. "Building Web Services the REST Way". <http://www.xfront.com/REST-Web-Services.html>

[Erl05] Thomas Erl (2005). "Service-Oriented Architecture: Concepts, technology and design" Prentice Hall. USA

[Fielding00] Roy T. Fielding (2000). "Architectural Styles and the Design of Network-based Software Architectures". USA. <http://roy.gbiv.com/pubs/dissertation/top.htm>