Introducción al middleware

Versionado de Web Services

Agenda

- Gobernanza de SOA
- Motivación
- Fundamentos
- Versionado de WSDL y XML Schema

Gobernanza

- En general, Gobernanza implica definir y controlar cómo un grupo acuerda trabajar en conjunto

- Consiste en el establecimiento de
  - Cadenas de responsabilidad ante la gente
  - Medición, de forma de controlar la efectividad
  - Políticas, para guiar a la organización, a fin de alcanzar sus metas
  - Mecanismos de control para asegurar cumplimiento
  - Comunicación, para informar a las partes

Gobernanza de SOA

- Pone el foco en el ciclo de vida de los componentes, servicios y procesos de negocio
- Guía el desarrollo de servicios reutilizables, definiendo cómo se diseñan y desarrollan esos servicios y cómo evolucionan con en el tiempo
- No diseña los servicios, pero guía cómo deben ser diseñados

Gobernanza de SOA

- Permite responder preguntas como:
  - Qué servicios hay disponibles?
  - Quién puede utilizarlos?
  - Qué tan confiable son?
  - Qué tanto tiempo estarán disponibles?
  - Puedo contar con que no cambien?
  - Qué pasa cuando cambian para mejorar o corregir una falla?

Gobernanza de SOA

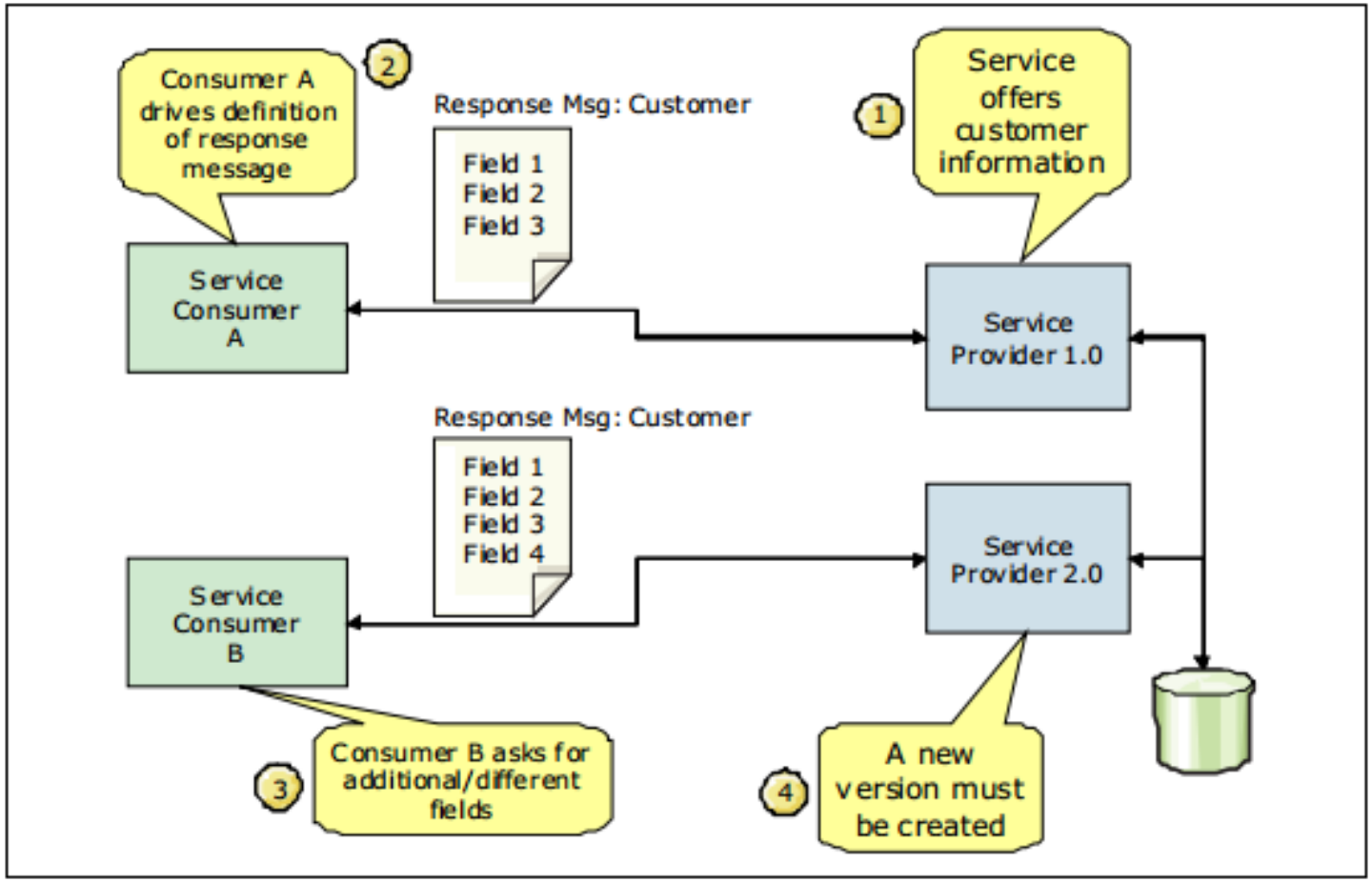
- Definición de servicios
- Ciclo de vida de deployment
- Versionado de servicios
- Migración de servicios
- Registro de servicios
- Modelo de mensajería utilizada
- Monitoreo de servicios
- Propiedad de servicios
- Testing de servicios
- Seguridad de los servicios

Motivación

- Los servicios en una SOA están sujetos a un proceso de constante evolución en el cual son mejorados, adaptados o ajustados a lo largo del tiempo para responder a nuevos requerimientos

- Esta evolución implica, en general, cambios en sus contratos para:
  - especificar nuevas capacidades
  - nuevas formas de acceso
  - nuevos datos
  - etc...

Motivación



http://www.redbooks.ibm.com/abstracts/redp4774.html?Open

Motivación

- Una vez que un Web Service se pone en marcha, los consumidores comenzarán a generar dependencias con su contrato

- Cuando es necesario realizar cambios en el mismo es necesario considerar:
  - si estos cambios impactarán negativamente en los consumidores
  - cómo deben implementarse y comunicarse dichos cambios

Motivación

- Estas problemáticas traen como resultado la necesidad del versionado de servicios

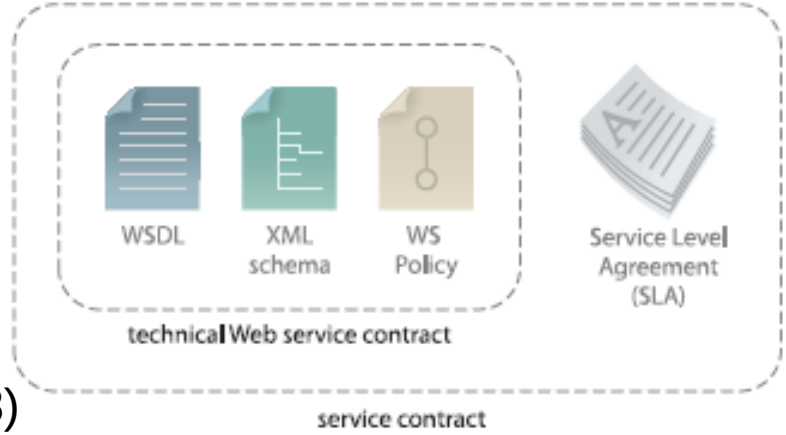
- En este contexto surgen las siguientes preguntas:
  - ¿qué constituye una nueva versión de un contrato?
  - ¿qué indican las distintas partes de un número de versión?
  - ¿la nueva versión del contrato funcionará con consumidores existentes/futuros?
  - ¿cuál es la mejor forma de hacer cambios a contratos?
  - ¿es necesario manejar simultáneamente versiones previas y nuevas de un contrato?

Fundamentos: Versión de un Contrato

- El contrato de un servicio se compone de uno o más documentos publicados que describen al servicio y pueden existir en un único o varios archivos físicos.
- La parte fundamental de un contrato son los documentos que expresan la interfaz técnica del servicio.
- Esta interfaz especifica una API a través de la cual el servicio ofrece sus funcionalidades.

Fundamentos: Versión de un Contrato

- Cuando los servicios se implementan como Web Services, los documentos más comunes para la descripción de su interfaz técnica son:
  - las definiciones WSDL
  - las definiciones XML Schema
  - las definiciones WS-Policy



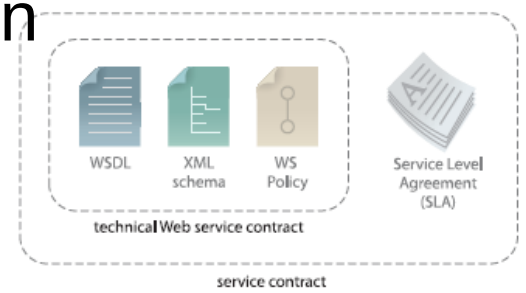
(Erl et al., 2008)

Fundamentos: Versión de un Contrato

- Las definiciones WSDL permiten especificar:
  - las operaciones que provee el servicio
  - el formato de los mensajes a intercambiar
  - las ubicaciones en donde puede accederse

- Las definiciones XML Schema permiten especificar la estructura detallada de los mensajes definiendo:
  - qué elementos y atributos se permiten
  - en qué orden deben aparecer y
  - qué tipo de datos pueden contener

(Erl et al., 2008)

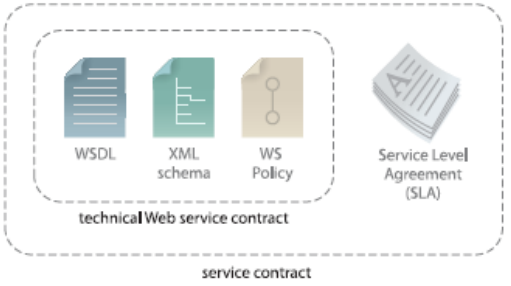


Fundamentos: Versión de un Contrato

- Las definiciones WS-Policy permiten extender las definiciones anteriores para especificar requerimientos o características adicionales, por ejemplo, a nivel de seguridad.

- Estas definiciones (WSDL, XML Schema y WS-Policy) pueden además ser compartidas por varios contratos

(Erl et al., 2008)



Fundamentos: Versión de un Contrato

- Cuando hablamos de una nueva versión de un contrato podemos asumir que ha habido entonces un cambio en alguna de estas definiciones
  - Definiciones WSDL
  - Definiciones XML Schema
  - Definiciones WS-Policy

Fundamentos: Compatibilidad de Contratos

- Una de las principales problemáticas al introducir cambios en el contrato de un servicio es el impacto que tendrán en los consumidores

- Esto está muy relacionado con la compatibilidad entre la versión previa y la nueva versión del contrato

Fundamentos: Compatibilidad de Contratos

- Una nueva versión de un contrato es compatible hacia atrás (*backwards-compatible*) cuando continúa soportando las aplicaciones diseñadas para trabajar con su versión anterior

- Esto significa que los cambios en el contrato no impactan a los consumidores que ya estaban utilizando el contrato

- Ejemplos: Nuevos contratos que únicamente agreguen:
  - una operación en el WSDL
  - un elemento opcional en XML Schema

Fundamentos: Compatibilidad de Contratos

- Ejemplo: Nueva operación en las definiciones WSDL

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

        <operation name="getWeatherByCity">
            <input message="getWeatherByCityIn"/>
            <output message="getWeatherByCityOut"/>
        </operation>
</portType>
```

Fundamentos: Compatibilidad de Contratos

- Ejemplo: Nuevo elemento opcional en las definiciones XML Schema

```
<xsd:schema targetNamespace="http://..../weatherService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <xsd:element name="City" type="xsd:string"/>
        <xsd:element name="Country" type="xsd:string"/>

        <xsd:element name="Region" type="xsd:string" minOccurs="0"/>

</schema>
```

Fundamentos: Compatibilidad de Contratos

- Una nueva versión de un contrato es "compatible hacia adelante" (*forwards-compatible*) cuando fue diseñado para soportar futuros consumidores

- Un servicio con un contrato "compatible hacia adelante" puede aceptar un rango más amplio de datos, desconocidos en tiempo de diseño

Fundamentos: Compatibilidad de Contratos

- La forma más común de dar soporte a la compatibilidad hacia adelante es a través de wildcards

```
<xsd:element name="cliente">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="nombre" type="xs:string"/>
        <xsd:element name="apellido" type="xs:string"/>
        <xsd:any minOccurs="0"/>
      </xsd:sequence>
      <xsd:anyAttribute/>
    </xsd:complexType>
  </xsd:element>
```

http://www.w3schools.com/schema/schema_complex_any.asp
http://www.w3schools.com/schema/schema_complex_anyattribute.asp

Fundamentos: Compatibilidad de Contratos

- En el ejemplo anterior se incluyen los elementos xsd:any y xsd:anyAttribute para permitir que el contrato del Web Service acepte un mayor rango de elementos y datos

- El contrato se está diseñando entonces para ajustarse a cambios en el futuro

Fundamentos: Compatibilidad de Cambios

- Cuando se realiza un cambio en el contrato de un servicio y éste no afecta negativamente a sus consumidores, el cambio se denomina “cambio compatible” (hacia atrás)

- Algunos cambios compatibles son:
  - Agregar una operación
  - Agregar un parámetro opcional a una operación

Fundamentos: Compatibilidad de Cambios

- Si el cambio en el contrato de un servicio hace que el contrato ya no sea compatible con sus consumidores, el cambio se denomina “cambio incompatible”

- Algunos cambios incompatibles son:
  - Eliminar una operación
  - Renombrar una operación

Fundamentos: Identificador de Versión

- El primer paso para establecer una estrategia de versionado efectiva es decidir cómo las versiones se identifican y representan en los contratos de los servicios

- Las versiones se comunican en general a través de números de versiones:
  - 2.0, 2.1.3, 3.2

Fundamentos: Identificador de Versión

- El significado común de estos números es la medida o significado del cambio
  - Incrementar el número principal de versión en general indica un cambio mayor
  - Incrementar los números secundarios de versión en general indica cambios menores

Fundamentos: Estrategias de Versionado

- No existe una estrategia de versionado que sea adecuada para todas las organizaciones dado que depende de las convenciones y requerimientos de cada organización
- Sin embargo, han surgido algunos enfoques comunes:
  - Strict Strategy
  - Flexible Strategy
  - Loose Strategy

Fundamentos: Estrategias de Versionado

- La estrategia STRICT requiere una nueva versión del contrato cuando se realiza cualquier tipo de cambio en cualquier parte del contrato

- Esto generalmente se implementa cambiando el “targetNamespace” de una definición WSDL cada vez que se realiza un cambio al contenido del WSDL o XML Schema

targetNamespace

```
<wsdl:definitions name="HelloWorldService"
                  targetNamespace="http://webservices.samples.jboss.org/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tns="http://webservices.samples.jboss.org/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xs:schema ----------
        ----------
        <xs:complexType name="sayHello">
            <xs:sequence>
                <xs:element minOccurs="0" name="arg0" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
```

```
<soapenv:Envelope
  xmlns:q0="http://webservices.samples.jboss.org/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    .....>
    <soapenv:Body>
        <q0:sayHello>
            <arg0>test</arg0>
        </q0:sayHello>
    </soapenv:Body>
</soapenv:Envelope>
```

targetNamespace

- El cambio del targetNamespace fuerza un cambio en todos los consumidores

- Ejemplo de respuesta si se cambiara el targetNamespace del ejemplo anterior:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Client</faultcode>
            <faultstring>
                Unexpected wrapper element {http://webservices.samples.jboss.org/}sayHello found.
                Expected {http://webservices2.samples.jboss.org/}sayHello.
            </faultstring>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Fundamentos: Estrategias de Versionado

- Si bien la estrategia STRICT no es muy práctica, algunas veces es garantía cuando hay implicancias legales en los cambios en los contratos de los servicios

- Dado que cualquier cambio implica una nueva versión, esta estrategia no soporta la compatibilidad hacia atrás ni la compatibilidad hacia adelante

Fundamentos: Estrategias de Versionado

- Un beneficio de la estrategia STRICT es que se tiene control total sobre la evolución del servicio

- Además, no hay que preocuparse cómo los cambios impactan en los consumidores ya que todos los cambios “rompen” el contrato

Fundamentos: Estrategias de Versionado

- La estrategia STRICT puede aumentar el trabajo de gobierno dado que se tienen que establecer estrategias de transición de forma cuidadosa

- También puede incrementar la existencia de distintas versiones de un servicio, por lo que la infraestructura (Ej, Registro de Servicios) debe estar preparada para eso

Fundamentos: Estrategias de Versionado

- La estrategia FLEXIBLE propone que sólo los cambios incompatibles generen una nueva versión del contrato del servicio

- De esta forma pueden ocurrir cambios compatibles en un contrato sin necesidad de generar una nueva versión del mismo

Fundamentos: Estrategias de Versionado

- La principal ventaja de la estrategia FLEXIBLE es que el contrato puede soportar varios cambios manteniendo la compatibilidad hacia atrás

- Sin embargo, es necesario un proceso que permita asegurar que los contratos de los servicios no se vuelvan complejos y sobrecargados

Fundamentos: Estrategias de Versionado

- La estrategia LOOSE propone que los cambios incompatibles sean los que generen una nueva versión del contrato

- Además se propone que los contratos sean diseñados para dar soporte a hacia delante (por ejemplo, a través de wildcards)

Fundamentos: Estrategias de Versionado

- La estrategia LOOSE tiene como ventaja que permite enviar al servicio contenido indefinido, por lo que brinda la posibilidad de que el contrato se ajuste a nuevos requerimientos

- Sin embargo, esto lleva a diseñar contratos vagos y que pueden requerir realizar validaciones a nivel de la implementación del servicio

Fundamentos: Estrategias de Versionado

	Estrategias		
	STRICT	FLEXIBLE	LOOSE
Strictness	high	medium	low
Governance Impact	high	medium	high
Complexity	low	medium	high

(Erl et al., 2008)

Versionado de Definiciones WSDL

- Los cambios en las definiciones WSDL son los que en general tienen mayor impacto visible

- En particular vamos a ver:
  - Versionando operaciones
  - Versionando port types

Versionado de Definiciones WSDL

- Se utilizará el siguiente esquema para el versionado:

1. El número de versión (principal y secundario) se incluirá en el elemento “documentation” que sigue al elemento “definitions” de apertura

```
<definitions name=“”
        ……
        <documentation> Version 2.1 </documentation>

</definitions>
```

2. El número de versión principal se añadirá al target namespace con un prefijo “v”

```
targetNamespace=“http://...../contract/MiServicio/v2”
```

Versionado de Definiciones WSDL

- Se utilizará la siguiente estrategia para el versionado:

3. La excepción a la regla 2 es cuando se trata de la primera versión del servicio

4. Un cambio compatible incrementa el número de versión secundario y no cambia el targetNamespace

5. Un cambio incompatible incrementa el número de versión principal y cambia el targetNamespace

Versionado de Definiciones WSDL

- La mayoría de los cambios en los documentos WSDL están centrados en las operaciones
- En particular se verán los siguientes cambios:
  - Agregar una Operación
  - Renombrar una Operación
  - Quitar una Operación

Agregar una Operación

- Cuando una definición WSDL está en uso, probablemente varios consumidores hayan generado dependencias con sus operaciones
- Extender esta definición WSDL con una nueva operación, no impacta estas dependencias
- Agregar una operación es entonces un cambio compatible hacia atrás

Agregar una Operación

- Suponiendo que la versión del contrato es la 2.1 y siguiendo la estrategia descripta:
  - ¿cuál es el nuevo número de versión luego del cambio?
  - ¿cómo se especifica este nuevo número de versión en el documento WSDL?

Agregar una Operación

- Como “Agregar una Operación” es un cambio compatible:
  - no se cambia el número principal de versión
  - se incrementa el número secundario de versión

- El nuevo número de versión es entonces 2.2 y se especifica en el elemento <documentation> de la definición WSDL
  - <documentation>Version 2.2</documentation>

- No se cambia el targetNamespace

Renombrar una Operación

- Si el nombre de una operación necesita cambiarse, esto claramente impactará a consumidores que utilicen esta operación

- Suponiendo que la versión del contrato era la 2.1 y siguiendo la estrategia descripta:
  - ¿cuál es el nuevo número de versión luego del cambio?
  - ¿cómo se especifica este nuevo número de versión en el documento WSDL?

Renombrar una Operación

- Dado que Renombrar una Operación es un cambio no compatible
  - Se debe cambiar el número de versión principal

- El nuevo número de versión es entonces 3.0 y se especifica en el WSDL
  - A través del elemento <documentation>
  - Cambiando el targetNamespace

Renombrar una Operación

<definitions name="MyService"

targetNamespace=http://www.fing.edu.uy/contract/mysrv/v3

…>

<documentation>Version 3.0</documentation>

……

</definitions>

Quitar una Operación

- Quitar una operación del contrato de un servicio claramente impactará a los consumidores que estén utilizándola, dado que las llamadas a dicha operación fallarán

- Suponiendo que la versión del contrato era la 2.1 y siguiendo la estrategia descripta:
  - ¿cuál es el nuevo número de versión luego del cambio?
  - ¿cómo se especifica este nuevo número de versión en el documento WSDL?

Quitar una Operación

<definitions name="MyService"

targetNamespace=http://www.fing.edu.uy/contract/mysrv/v3

…>

<documentation>Version 3.0</documentation>

……

</definitions>

Versionando portTypes

- Una definición WSDL puede tener varios ***portTypes***

- Veremos cómo un único contrato puede actuar como contenedor de múltiples portTypes, cada uno representando una versión distinta

Versionando portTypes

- Una alternativa a crear una nueva definición WSDL cada vez que ocurre un cambio incompatible, es crear un nuevo portType

- Dado que en una definición WSDL es posible incluir varios portTypes, se puede crear un documento WSDL con múltiples interfaces

Versionando portTypes

```
<definitions name="PurchaseOrder" targetNamespace=http://actioncon.com/contract/po
                                  xmlns:tns=http://actioncon.com/contract/po ….>

        <documentation>Versions 1.0, 2.0, 3.0</documentation>

        <portType name="ptPurchaseOrder-v1">
            <operation name="opSubmitOrder“>…..</operation>
            <operation name="opCheckOrderStatus“>…..</operation>
        </portType>

        <portType name="ptPurchaseOrder-v2“>
            <operation name="opSubmitOrders“>…..</operation>
            <operation name="opCheckOrderStatus“>…..</operation>
        </portType>

        <portType name="ptPurchaseOrder-v3“>
            <operation name="opSubmitOrder“>…..</operation>
            <!-- opCheckOrderStatus Removed 01/12 -->
        </portType>
```

Versionando portTypes

- Dado que el versionado se representa a través de portTypes, el targetNamespace no cambia y los números de versiones principales se incorporan en el portType

- El elemento <documentation> contiene las versiones soportadas por el contrato

- Hay que tener en cuenta que este enfoque puede dar lugar a definiciones WSDL de gran tamaño y difíciles de gobernar

Versionado de Esquemas de Mensajes

- Los cambios en las definiciones de tipos y estructuras de datos utilizadas por los mensajes definidos en el WSDL pueden tener un gran impacto en los consumidores

- Veremos cuatro cambios comunes:
  - Agregar un componente al esquema
  - Quitar un componente del esquema
  - Renombrar un componente del esquema
  - Modificar las restricciones de un componente del esquema

Ejemplo Base

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
        targetNamespace=http://actioncon.com/schema/po
        xmlns=http://actioncon.com/schema/po
        elementFormDefault="qualified" version="1.0">

    <xsd:element name="LineItem" type="LineItemType"/>
    <xsd:complexType name="LineItemType">
        <xsd:sequence>
            <xsd:element name="productID" type="xsd:string"/>
            <xsd:element name="productName" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

```
<LineItem xmlns="http://actioncon.com/schema/po">
        <productID>AY2345</productID>
        <productName>Service Blaster 2000</productName>
</LineItem>
```

Versionado de Esquemas de Mensajes

- Convenciones de Versionado:
  - Como en el caso del WSDL los números de versión principal se especificarán a través del targetNamespace
  - Sin embargo, para los números de versión secundarios se utilizará el atributo versión

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
        targetNamespace="http://actioncon.com/schema/po"
        xmlns="http://actioncon.com/schema/po"
        version="1.0">
        ...
</xsd:schema>
```

Versionado de Esquemas de Mensajes

- Convenciones de Versionado:
  - Si un esquema tiene un cambio que requiere un nuevo targetNamespace, este cambio es propagado a nivel de WSDL (nuevo targetNamespace para el WSDL)
  - La mejor forma de entender esta relación es ver el XML Schema como una extensión al WSDL
  - En los siguientes ejemplos se considerará un esquema de versionado flexible

Agregar un Componente

- Agregar un nuevo componente a un esquema se considera un cambio:
  - Incompatible, si el componente es requerido
  - Compatible, si el componente es opcional

Agregar un Componente

- Componente requerido

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(Erl et al., 2008)

Agregar un Componente

- Componente opcional

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po"
  xmlns="http://actioncon.com/schema/po"
  elementFormDefault="qualified"
  version="1.1">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="available" type="xsd:boolean"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(Erl et al., 2008)

Eliminar un Componente

- Quitar la declaración de un componente de una definición XML Schema es un cambio:
  - incompatible

- Se requiere entonces incrementar el número principal de versión y cambiar el targetNamespace

Eliminar un Componente

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <!-- productName Removed 09/12 -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(Erl et al., 2008)

Renombrar un Componente

- Renombrar un componente es un cambio:
  - incompatible
- Requiere entonces:
  - un nuevo targetNamespace
  - un nuevo número de versión principal del contrato

Renombrar un Componente

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  elementFormDefault="qualified"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:string"/>
      <xsd:element name="productName2" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(Erl et al., 2008)

Renombrar un Componente

- Una alternativa para hacer que este cambio sea compatible:

```
<xsd:complexType name="LineItemType">
  <xsd:sequence>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:choice>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="productName2" type="xsd:string"/>
    </xsd:choice>
    <xsd:element name="available" type="xsd:boolean"/>
  </xsd:sequence>
</xsd:complexType>
```

(Erl et al., 2008)

Modificar Restricciones

- Ajustar las reglas de validación para un componente de un esquema es un tipo cambio común

- Algunos ejemplos de estos cambios son:
  - Cambios en los tipos de datos
  - Cambios en la cantidad de veces máxima o mínima que un determinado elemento puede aparecer

Modificar Restricciones

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://actioncon.com/schema/po/v2"
  xmlns="http://actioncon.com/schema/po/v2"
  elementFormDefault="qualified"
  version="2.0">
  <xsd:element name="LineItem" type="LineItemType"/>
  <xsd:complexType name="LineItemType">
    <xsd:sequence>
      <xsd:element name="productID" type="xsd:integer/>
      <xsd:element name="productName" type="xsd:string"
       maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(Erl et al., 2008)

Modificar Restricciones

- En el ejemplo anterior, modificar el tipo de datos es un cambio incompatible dado que el tipo ***integer*** es más restrictivo que el ***string***

- Pero el nuevo valor para maxOccurs es un cambio compatible ya que es menos restrictivo

Modificar Restricciones

- Por lo anterior, un mensaje que contenga lo siguiente no será válido dado que si bien es posible recibir más de un productName, el productID no es un valor entero

```
<LineItem xmlns="http://actioncon.com/schema/po">
  <productID>AY2345</productID>
  <productName>Service Blaster 2000</productName>
  <productName>Service Blaster 2010</productName>
</LineItem>
```

(Erl et al., 2008)

Resumen

- Versionado de servicios es una parte de la Gobernanza SOA
- Compatibilidad hacia atrás/hacia adelante
  - Estrategia strict, loose, flexible
- Cómo representar cambios en WSDL y XML Schemas

Referencias

- Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, L.U., Liu, K., Orchard, D., Tost, A., Pasley, J.: Web Service Contract Design and Versioning for SOA. Prentice Hall (2008).
- IBM Redbooks | Building a Service Versioning Gateway with WebSphere ESB and WebSphere Service Registry and Repository V7.5, http://www.redbooks.ibm.com/abstracts/redp4774.html?Open.
- Bean, J.: SOA and Web Services Interface Design: Principles, Techniques, and Standards. Morgan Kaufmann (2009).
- Introduction to SOA governance. http://www.ibm.com/developerworks/library/ar-servgov/
- SOA Governance in Action (Jos Dirksen)