Arquitectura Orientada a Servicios

XML Schema

Edwin Salvador

15 de mayo de 2015

Sesión 6

Contenido I

XML Schema para SOA

- XML es el lenguaje oficial de SOA.
- Utilizado para el envío de mensajes, configuración de la aplicación, implementación, descubrimiento, políticas de ejecución y para representar lenguajes de ejecución como BPEL que son muy importantes dentro de SOA.
- Las interfaces de los servicios web son representadas con XML (WSDL).
- La flexibilidad de XML y su expresividad lo hacen perfecto para diseñar nuestro modelos de datos para SOA ya que esto nos brinda independencia de tecnologías, marcas, lenguajes, etc.

XML Schema para SOA

- Según los expertos, los nuevos dispositivos de comunicación (celulares, SMS, FB, email, Whatsapp, etc) no cambia solamente ***como*** nos comunicamos sino también ***que*** comunicamos.
- El formato del mensaje determina también el contenido del mensaje.
- La principal ventaja de XML en su *flexibilidad*.
- Muchas empresas sufren de los problemas del *aislamiento de procesos*.
- Cada una de estas aplicaciones está atada a un modelo de datos y cada uno debe ser actualizado cuando existe un cambio.
- Son estas empresas las que están decidiendo la implementación de SOA.
- XML ofrece un modelo de datos que con una vista contextual que fluye, que puede ser transformada y puede interactuar fácilmente a través de diferentes servicios.

XML Schema para SOA

- KISS *Keep It Simple (and) Stupid*
- Al desarrollar los modelos de datos con XML Schema es recomendable mantener las cosas lo más simple posible.
- Esto maximiza la interoperabilidad y aprovecha la flexibilidad y apertura de XML.
- Si decidimos hacer las cosas complejas, esto nos atará a herramientas específicas o a ciertas marcas lo cuál escapa del objetivo de SOA.
- La orientación a objetos no tiene mucho espacio dentro de SOA. Debemos evitar encapsular todo en un modelo.
- XML Schema es muy poderoso y nos ayuda a representar modelo de datos en lugar de un modelo de objetos (**no orientado a objetos**).
- A pesar de que XML Schemas permiten crear modelos de datos similares a objetos (enumeration e incluso polimorfismo) se debe mantener los servicios lo más generalizados posible para **permitir la comunicación entre lenguajes**.

XML Schema para SOA

- Usaremos varias herramientas para crear XML Schemas para representar estructuras de datos.
- Tendremos que saber elegir el diseño apropiado para asegurarnos que nuestro modelo de datos está estructurado de una manera que represente nuestras necesidades y que mantenga la flexibilidad en SOA.

Contenido I

Diseñando un esquema para SOA

- Diseñaremos un modelo de datos utilizando XML Schema para utilizarlo en nuestra solución SOA.
- Debido a la flexibilidad permitida por XML es difícil saber como escribir nuestro modelo en una forma consistente y clara que nos brinde una buena combinación de expresividad, flexibilidad y definición de tipos de datos lo suficientemente fuerte.
- Veremos algunos patrones de definición aceptados y las mejores prácticas para Schema que mejor se adaptan al contexto de SOA.
- Los patrones que veremos son:
  - Russian Doll,
  - Salami Slice
  - Venetian Blind
  - Garden of Eden
  - Chameleon
- Estos patrones se diferencian entre sí, de la manera que definimos nuestros tipos (globales o locales).

Diseñando un esquema para SOA

- Schema nos permite definir los bloques básicos para definir nuestras entidades: tipos simples, tipos compuestos, elementos y atributos.
- Existen muchas elecciones al definir tipos locales o globales, *namespace* y más.
- Hacer elecciones apresuradas podría destruir nuestra solución SOA limitando la flexibilidad.
- Sin un cuidadoso diseño podríamos descubrir muy tarde que los servicios en nuestra solución están muy atados entre ellos, lo cual es perjudicial para SOA.
- En estos casos un simple cambio podría forzar a tener que rediseñar servicios enteros.

Elementos o tipos globales y locales

- **Globales** son hijos del nodo schema.
- **Locales** están anidados dentro de otro elemento o tipo. No pueden ser usados en otra parte.

Contenido I

Russian doll (Matryoshka)

- Una Matryoshka es una muñeca de madera que contiene dentro más muñecas idénticas más pequeñas y cada una contiene otras dentro de ellas.

Russian doll

- El patrón de diseño sigue esta misma idea por lo tanto debería ser fácil de recordar.
- Consiste en crear un solo elemento raíz global que contiene todos lo tipos que lo componen.
- Todos las demás declaraciones de elementos son anidadas dentro del elemento raíz (locales).
- Solo el elemento raíz tiene acceso al espacio de nombre (namespace) global.

Características

- Un solo elemento global.
- Todos lo tipos son locales (anidados dentro del raíz)
- Soporta esquemas diseñados **en un solo archivo**.
- Alta cohesión (la relación de sus elementos), mínima atadura (**Coupling**. dependencia de otros modelos).
- Alto encapsulamiento (sus tipos no están expuestos).
- El patrón más fácil de leer y escribir.

No se debe usar este patrón cuando se desea reusar los tipos

Ejemplo

Russian doll

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ns.soacookbook.com/russiandoll"
    xmlns:tns="http://ns.soacookbook.com/russiandoll"
    elementFormDefault="unqualified">
<xsd:annotation>
  <xsd:documentation>
    Book schema as Russian Doll design.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="price" type="xsd:decimal"/>
      <xsd:element name="category" type="xsd:NCName"/>
      <xsd:choice>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="authors">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author"
                  type="xsd:string"
                  maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Observaciones

Russian doll

- Tiene un solo elemento `book`
- Los tipos para crear un elemento `book` están anidados.
- Los elementos y tipos no pueden ser referenciados. El elemento `authors` redefine el elemento `author`. Lo cual podría causar problemas de mantenimiento.
- El espacio de nombres es local.

Ventajas

Russian doll

- Ideal para Schemas simples, claridad leer y escribir (no referencias, no mezclas, las definiciones están donde son utilizadas).
- Es menos flexible que otros lo que lo hace predecible.
- Facilidad de entender las intenciones del autor.
- Es auto-contenido lo que lo hace desacoplado de otros schemas.

Desventajas

Russian doll

- No adecuado para schemas grandes.
- Los tipos y elementos no pueden ser reutilizados.

Contenido I

Salami slice

- Representa lo contrario que el Russian doll
- Todos los **elementos** se declaran como **globales**.
- Todos los **tipos** se declaran **locales**.
- Todos los elementos se ubican en el namespace global, esto permite que el schema sea utilizado por otros schemas.
- Cada elemento viene a ser una definición que puede ser combinada con otras.
- Es completamente abierto a todas las posibles combinaciones, se los puede organizar de cualquier manera.
- Este patrón de diseño ofrece la mayor posibilidad de reutilización de todos los patrones.

Características

Salami slice

- Todos los elementos son globales
- Todos los elementos son definidos en el namespace global
- Todos los tipos son locales
- Las declaraciones de los elementos nunca se anidan
- Las declaraciones de los elementos son reutilizables
- Es complicado determinar cuál es el elemento raíz.

Ejemplo

Salami slice

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ns.soacookbook.com/salami"
  xmlns:tns="http://ns.soacookbook.com/salami"
  elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:documentation>
      Book schema as Salami Slice design.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:title"/>
        <xsd:element ref="tns:author"/>
        <xsd:element ref="tns:category"/>
        <xsd:element ref="tns:price"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="title"/>
  <xsd:element name="price"/>
  <xsd:element name="category"/>
  <xsd:element name="author"/>
</xsd:schema>
```

Ventajas y Desventajas

Salami slice

Ventajas

- Reutilización de elementos (globales).
- Todo está bien organizado.

Desventajas

- Altamente acoplados debido a que si se cambia un elemento esto afectará a todos los elementos que lo utilicen.
- Los schemas con este patrón suelen contener mucho texto.
- Si se utiliza este schema para diseñar los servicios web, estos también estarán altamente acoplados.

Contenido I

Venetian blind

- Ofrece lo mejor de los anteriores patrones de diseño.
- Es una extensión del Russian doll. Contiene un solo elementos raíz.
- Permite la reutilización de todos lo tipos y el elemento raíz global.
- Se define un elemento raíz global y se lo compone con tipos definidos externamente. Esto maximiza la reutilización.

Características

Venetian blind

- Tiene un solo elemento raíz
- Mezcla declaraciones locales y globales. En Russian doll todos los tipos son locales y en Salami slice todos los tipos son globales.
- Alta cohesión y alto acoplamiento.
- Maximiza la reutilización (tipos y raíz pueden ser recombinados).
- Bajo encapsulamiento (los tipos están expuestos).
- Permite utilizar varios archivos para definir un schema.
- Suelen ser archivos largos. Se permite tener mejor control de cada aspecto individual de los elementos.

Se debe elegir Venetian blind cuando se desea maximizar la reutilización y flexibilidad y tomar ventaja de la exposición del namespace.

Ejemplo

Venetian blind

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://ns.soacookbook.com/venetianblind"
        xmlns:tns="http://ns.soacookbook.com/venetianblind"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">
<xsd:annotation>
  <xsd:documentation>
      Book schema as Venetian Blind design.
  </xsd:documentation>
</xsd:annotation>
<!-- Single global root element exposed -->
<xsd:element name="book" type="tns:BookType"/>

<!-- The root is given a type that is defined here,
     using all externally defined elements. -->
<xsd:complexType name="BookType">
    <xsd:sequence>
        <xsd:element name="title" type="tns:TitleType"/>
        <xsd:element name="author" type="tns:AuthorType"/>
        <xsd:element name="category" type="tns:CategoryType"/>
        <xsd:element name="price" type="tns:PriceType" />
    </xsd:sequence>
</xsd:complexType>
.....
.....
.....
.....
</xsd:schema>
```

Contenido I

Garden of Eden

- Combina Salami slice con Venetian blind
- Se debe definir todos los elementos y tipos en el namespace global y referenciar los elementos cuando sea necesario.
- Ofrece la mayor reutilización y flexibilidad de todos los patrones de diseño.
- Se puede reutilizar elementos y tipos libremente.
- No existe encapsulación.
- Muchos elementos raíz potenciales (globales).
- Documentos XML son difíciles de leer.
- No se tiene claras las intenciones del autor (no se sabe cual es el elemento raíz).

Ejemplo

Garden of Eden

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ns.soacookbook.com/eden"
  xmlns:tns="http://ns.soacookbook.com/eden"
  elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:documentation>
      Book schema as Garden of Eden design.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="book" type="tns:bookType"/>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  <xsd:element name="category" type="xsd:string"/>
  <xsd:element name="price" type="xsd:double"/>

  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element ref="tns:title"/>
      <xsd:element ref="tns:author"/>
      <xsd:element ref="tns:category"/>
      <xsd:element ref="tns:price"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Contenido I

Deber

- Consultar y escribir un informe sobre la definición del **Modelo de datos canónico (*Canonical data Model (CDM)*)**, explicar sus ventajas, desventajas y cuando debe ser utilizado y cuando no. **Fecha límite Jueves 21 de mayo 23:59**. Utilizar el formato de deberes.
- Verificar ingreso a PeerWise `https://peerwise.cs.auckland.ac.nz`
- Realizar al menos 5 preguntas en PeerWise relacionadas con la materia. Preguntas de opción múltiple. Pueden ser ejercicios similares a los deberes. **Fecha límite Jueves 21 de mayo 23:59**
- Responder al menos 5 preguntas (que no hayan sido realizadas por ustedes mismo) de PeerWise. **Fecha límite Jueves 28 de mayo 23:59**
- La nota será definida según la participación que hayan tenido en PeerWise, el tipo de preguntas que hayan realizado y el número de preguntas que hayan respondido y la dificultad de las preguntas respondidas.

Contenido I

Contenido I

Ejercicio 1

- Escribir el Schema `"Shiporder.xsd"` de la clase anterior utilizando cada uno de los 4 patrones de diseño que vimos hoy.

Contenido I

Ejercicio 2

- Indicar que caracteres son aceptados por la siguiente expresión regular y escribir 1 ejemplo de elemento válidos y 1 ejemplo de elemento no válido para cada uno de estos schemas:
  Referencia: `http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html#sum`
  Validación:
  `http://www.regexplanet.com/advanced/java/index.html`

```
<xs:element name="ejercicioUno">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="2"/>
      <xs:maxLength value="30"/>
      <xs:pattern value="[A-Za-z\-. ']{2,30}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio 2.2

```
<xs:element name="ejercicioDos">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="\(?\d{3}\)?[ \-\.]?\d{3}[ \-\.]?\d{4}"/>
      <xs:minLength value="10"/>
      <xs:maxLength value="16"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio 2.3

```
<xs:element name="ejercicioTres">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="(\w+\.)*\w+@(\w+\.)+[A-Za-z]{2,9}"/>
      <xs:minLength value="6"/>
      <xs:maxLength value="255"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio 2.4

```
<xs:element name="ejercicioCuatro">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{5}(-\d{4})?"/>
      <xs:minLength value="5"/>
      <xs:maxLength value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio 2.5

```
<xsd:simpleType name="ejercicioCinco">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}(-)?\d{2}(-)?\d{4}"/>
    <xsd:minLength value="9"/>
    <xsd:maxLength value="11"/>
  </xsd:restriction>
</xsd:simpleType>
```

Ejercicio 2.6

```
<xsd:simpleType name="ejercicioSeis">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2" />
    <xsd:enumeration value="AB"/>
    <xsd:enumeration value="BC"/>
    <xsd:enumeration value="MB"/>
    <xsd:enumeration value="NB"/>
    <xsd:enumeration value="NL"/>
    <xsd:enumeration value="NS"/>
    <xsd:enumeration value="NT"/>
    <xsd:enumeration value="NU"/>
    <xsd:enumeration value="ON"/>
    <xsd:enumeration value="PE"/>
    <xsd:enumeration value="QC"/>
    <xsd:enumeration value="SK"/>
    <xsd:enumeration value="YT"/>
  </xsd:restriction>
</xsd:simpleType>
```

Ejercicio 2.7

```
<xs:element name="ejercicioSiete">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="(https?://)?(www.)?[-\w]+(\.\w{2,3})+(:\d{2,5})?
        (/([\w/_.]*)?)?"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio 2.8

```
<xs:element name="ejercicioOcho">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="(((\d{0,2})|(1(\d){0,2})|(2[0-4]\d)|(25[
        0-5]))\.){3} ((\d{0,2})|(1(\d){0,2})|(2[0-4]\d)|(25[0-5]))"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```