Conclusión, validación con XML Schemas

En este documento desarrollaremos un ejemplo de principio a fin con todas las explicaciones necesarias para la compresión de los elementos utilizados.

Usaremos como ejemplo un escenario en el que se intenta crear una aplicación de reserva de habitaciones en un hotel. Las reservas vienen por la red, y permiten indicar un conjunto de habitaciones y las personas que se alojarán en éstas; asimismo, se indica el mecanismo mediante el cual se hará el pago de las mismas.

Ejemplo de Hotel

```
<?xml version="1.0"?>
 <reserva version="1">
   <habitaciones>
     <habitacion numero="202">
         <desde>2009-11-22</desde>
         <hasta>2009-11-24</hasta>
         <residentes>
             <residente responsable="true">
                 <nombres>Pedro</nombres>
                 <apellido>Camacho</apellido>
                 <pasaporte>552321</pasaporte>
                 <nacionalidad>ES</nacionalidad>
         </residente>
         <residente>
                 <nombres>Julia</nombres>
                 <apellido>Urquidi</apellido>
                 <pasaporte>664423</pasaporte>
                 <nacionalidad>BO</nacionalidad>
         </residente>
         </residentes>
     </habitacion>
  </habitaciones>
  <mediopago>
     <tarjeta_credito>
         <tipo>AMEX</tipo>
         <numero>38822050023042</numero>
     </tarjeta_credito>
  </mediopago>
</reserva>
```

Asimismo, el sistema del hotel responderá satisfactoriamente así:

Repuesta satisfactoria del hotel

```
<?xml version="1.0"?>
<confirmacion version="1">
   <satisfactorio>true</satisfactorio>
   <comentario>todo conforme</comentario>
</confirmacion>
```

o en caso de no ser satisfactorio el requerimiento, la respuesta será algo como esto:

Respuesta del hotel errónea

```
<?xml version="1.0"?>
<confirmacion version="1">
  <satisfactorio>false</satisfactorio>
  <comentario>la habitacion 202 es para una sola persona</comentario>
</confirmacion>
```

Pero estos movimientos ¿están adecuadamente configurados? validaremos los movimientos a través de XML-Schemas

Empecemos por los mensajes de confirmación. Primero podríamos hacer una validación XML para asegurarnos de que los documentos están bien formados:

```
$ xmlstarlet val conf_no_reserva.xml
conf_no_reserva.xml - valid
```

Ahora construiremos un schema para este archivo. En primer lugar, por convención se emplea la extensión xsd, por lo que crearemos **"confirmacion.xsd":**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
</xsd:schema>
```

Los esquemas normalmente se construyen a partir de un conjunto de tags definidos el "espacio de nombres de esquemas" identificado por el URI que se indicó (el uso de los URIs es una convención, análoga a los nombres de paquete Java: ¡no se supone que realmente ingresemos a ese URL!) Las herramientas que procesan esquemas ya tienen precargados los tags especiales de este espacio de nombres (en otras palabras, asociados a este URI), por lo que la directiva simplemente los pone a nuestra disposición asociándoles el prefijo "xsd". Esto también es convencional; se podría haber usado: *xmlns:otroprefijo="http://www.w3.org..."* pero esto sería confuso para quien leyera el schema. Este prefijo debe utilizarse para todos los tags "especiales" del espacio de nombres de esquemas, empezando por el mismo tag "schema", por lo que la declaración con "otroprefijo" tendría que haber sido:

```
<otroprefijo:schema xmlns:otroprefijo="http://www.w3.org/2001/XMLSchema">
```

En lo que sigue usaremos "xsd" para los tags del espacio de nombres de esquemas.

Ahora bien, el elemento raíz es "confirmacion", el cual está constituido por dos elementos ("satisfactorio" y "comentario"). En tanto que "confirmacion" tiene elementos "hijos", entonces se dice que es "complejo", mientras que los otros al no tenerlos, son "simples".

Los elementos simples se especificarán mediante:

```
<xsd:element name="satisfactorio" type="xsd:boolean"/>
<xsd:element name="comentario" type="xsd:string"/>
```

Como se aprecia, se ha utilizado los tipos "boolean" y "string" del especio de nombres de esquemas. Ahora bien, se suele exigir que los elementos aparezcan en una secuencia

ordenada (aunque no es obligatorio), por lo que podríamos envolverlos en una "secuencia":

```
<xsd:sequence>
<xsd:element name="satisfactorio" type="xsd:boolean"/>
<xsd:element name="comentario" type="xsd:string"/>
</xsd:sequence>
```

Todo esto representa el contenido que deberíamos asociar al elemento "confirmación"; en otras palabras, lo anterior define el "tipo" del elemento complejo "confirmación". Los "tipos" pueden tener un nombre, tal como "conf_tipo", y ser definidos así:

```
<xsd:complexType name="conf_tipo">
<xsd:sequence>
<xsd:element name="satisfactorio" type="xsd:boolean"/>
<xsd:element name="comentario" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="version" type="xsd:integer"/>
</xsd:complexType>
```

Aquí hemos agregado el atributo "version" de tipo entero, que es parte del elemento "confirmacion". Con todo esto, "confirmacion" quedará totalmente definido mediante:

```
<xsd:element name="confirmacion" type="conf_tipo"/>
```

Y el esquema total queda así:

```
$ more confirmacion.xsd
```

Validación de las confirmaciones. “confirmación.xsd”

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:complexType name="conf_tipo">
          <xsd:sequence>
               <xsd:element name="satisfactorio" type="xsd:boolean"/>
               <xsd:element name="comentario" type="xsd:string"/>
          </xsd:sequence>
     <xsd:attribute name="version" type="xsd:integer"/>
     </xsd:complexType>
          <xsd:element name="confirmacion" type="conf_tipo"/>
</xsd:schema>
```

Para validar el documento XML, éste requiere hacer referencia al esquema. Esto lo logramos así:

Documento de confirmación. Confirmacion.xml

```
<?xml version="1.0"?>
<confirmacion version="1"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="confirmacion.xsd">
     <satisfactorio>true</satisfactorio>
     <comentario>todo conforme</comentario>
</confirmacion>
```

En el elemento principal hemos utilizado el atributo "noNamespaceSchemaLocation" para especificar el nombre de nuestro esquema (que deberá ubicarse en el mismo directorio).

Luego se explicará la razón del nombre tan extraño de este atributo.

Es importante anotar que este atributo también pertenece a otro espacio de nombres estandarizado denominado "instancias de esquemas", identificado con el URI que se aprecia, y asociado a los documentos típicamente con el prefijo "xsi". Nótese que el "xsi" sólo se emplea en los atributos del elemento raíz con este propósito, y raramente aparece en el resto del documento.

La validación a continuación:

```
$ xmlstarlet val -e -s confirmacion.xsd conf_reserva_schema.xml
conf_reserva_schema.xml - valid
```

Ahora bien, para este esquema concreto podríamos ahorrar algo de texto definiendo el tipo del elemento complejo "confirmacion" en el interior de la declaración de dicho elemento de este modo:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="confirmacion">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="satisfactorio" type="xsd:boolean"/>
<xsd:element name="comentario" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="version" type="xsd:integer"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Esto es más eficiente para este caso dado que el tipo "conf_tipo" sólo se utiliza una única vez.

Ahora pasemos a la solicitud de reserva. El elemento raíz es "reserva" y contiene los elementos "habitaciones" y "mediopago" (en dicho orden):

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema">
<xsd:element name="reserva">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="habitaciones">
...
</xsd:element>
<xsd:element name="mediopago">
...
</xsd:element>
</xsd:sequence>
<xsd:attribute name="version" type="xsd:integer"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Si bien nuestro ejemplo sólo introduce una "habitacion", es de esperarse que se puedan reservar varias de una vez (pero al menos una). Por lo tanto, nuestro elemento "habitaciones" contiene una secuencia de elementos "habitacion" que admite repeticiones; esto se indica agregando el atributo *maxOccurs="unbounded"* al tag xsd:element:

```
<xsd:element name="habitaciones">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="habitacion" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="desde"
type="xsd:date"/>
<xsd:element name="hasta"
type="xsd:date"/>
<xsd:element name="residentes">
...
</xsd:element>
</xsd:sequence>
<xsd:attribute name="numero" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

"habitacion" también resulta ser un elemento complejo conteniendo los elementos "desde", "hasta", "residentes" y el atributo "numero" (de habitación).

Dado que el elemento "residentes" también es complejo, a fin de no hacer tantos anidamientos, crearemos un tipo explícitamente:

```
<xsd:complexType name="tipo_residentes">
<xsd:sequence>
<xsd:element name="residente" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nombres" type="xsd:string"/>
<xsd:element name="apellido" type="xsd:string"/>
<xsd:element name="pasaporte" type="xsd:string"/>
<xsd:element name="nacionalidad" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="responsable" type="xsd:boolean"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

Con esto el elemento "residentes" queda así:

```
<xsd:element name="residentes" type="tipo_residentes"/>
```

Para el caso de "mediopago" hemos indicado el caso de "tarjeta_credito", pero obviamente podrían haber otros más. Asumamos que también puede haber "efectivo", el cual tiene como atributo “moneda”. Normalmente el medio de pago es uno u otro, pero no ambos, por lo que la "secuencia" de elementos no tiene mucho sentido aquí, usaremos “choice”.

Para los casos de opciones excluyentes se emplea "xsd:choice", y su uso es análogo a sequence:

```
<xsd:element name="mediopago">
<xsd:complexType>
<xsd:choice>
```

```
<xsd:element name="tarjeta_credito">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="tipo"
type="xsd:string"/>
<xsd:element name="numero"
type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="efectivo">
<xsd:complexType>
<xsd:attribute name="moneda"
type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
```

No se puede olvidar que debemos modificar el XML para que haga referencia a nuestro esquema:

```
<reserva version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="solicitud.xsd">
<habitaciones>
...
```

La validación debe funcionar ahora:

```
$ xmlstarlet val -e -s solicitud.xsd sol_reserva_ef_schema.xml
sol_reserva_ef_schema.xml - valid
```

También si alteramos el medio de pago:

```
<mediopago>
<efectivo moneda="EUR"/>
</mediopago>
```

Típicamente los establecimientos sólo aceptan ciertas monedas para las operaciones. Por ejemplo, asumamos que sólo se aceptarán euros (EUR) y dólares (USD), podemos utilizar para el efectivo:

```
<xsd:attribute name="moneda">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="EUR"/>
<xsd:enumeration value="USD"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
```

Probemos a invalidar el documento:

```
<mediopago>
<efectivo moneda="PEk"/>
</mediopago>
```

Tenemos ahora:

```
$ xmlstarlet val -e -s solicitud.xsd sol_reserva_ef_schema.xml
sol_reserva_ef_schema.xml:26: element efectivo: Schemas validity
error : Element 'efectivo', attribute 'moneda': [facet
'enumeration'] The value 'PEk' is not an element of the set
{'PEN', 'USD'}.
sol_reserva_ef_schema.xml:26: element efectivo: Schemas validity
error : Element 'efectivo', attribute 'moneda': 'PEk' is not a
valid value of the local atomic type.
sol_reserva_ef_schema.xml - invalid
Atributos Default
```

Entre las personas que se alojan por lo general hay un responsable de hacer las reservas y que sirve de intermediario con el hotel. Esto lo hemos señalado mediante el atributo "responsable" del elemento "residente". Sin embargo, es interesante notar que este atributo no fue fijado para todos los residentes, salvo para el primero de ellos.

En ocasiones, es conveniente que los atributos sean obligatorios. Esto lo podríamos haber señalado así en nuestro esquema:

```
<xsd:attribute name="responsable" type="xsd:boolean" use="required"/>
```

Con esto la validación fallaría:

```
$ xmlstarlet val -e -s solicitud_atob.xsd
sol_reserva_ef_schema.xml
sol_reserva_ef_schema.xml:16: element residente: Schemas validity
error : Element 'residente': The attribute 'responsable' is
required but missing.
sol_reserva_ef_schema.xml - invalid
```

Para efectos de la aplicación que procesa el documento, es más conveniente recibir siempre un atributo "responsable" (en vez de considerar separadamente el caso en que hay o no hay tal atributo). Del mismo modo, puede ser conveniente para quien redacta el documento el no tener que suministrarlo; para esta situación el atributo "default" proporciona un compromiso:

```
<xsd:attribute name="responsable" type="xsd:boolean" default="false"/>
```

Los valores default también pueden aplicarse a elementos. Por ejemplo, asumiendo que la mayoría de residentes son españoles, se podría definir el elemento nacionalidad así:

```
<xsd:element name="nacionalidad" type="xsd:string"
default="ES"/>
```

Con esto, el documento se podría simplificar así:

```
<nombres>Pedro</nombres>
<apellido>Camacho</apellido>
<pasaporte>552321</pasaporte>
<nacionalidad/>
```

Sin embargo, si se elimina el elemento fallaría:

```
<nombres>Pedro</nombres>
<apellido>Camacho</apellido>
```

```
<pasaporte>552321</pasaporte>

$ xmlstarlet val -e -s solicitud_atob.xsd
sol_reserva_ef_schema.xml
sol_reserva_ef_schema.xml:10: element residente: Schemas validity
error : Element 'residente': Missing child element(s). Expected is
( nacionalidad ).
sol_reserva_ef_schema.xml - invalid
```

Es decir, el valor "default" de un elemento aplica sólo para aquellos que están vacíos, pero que sí existen.

Elementos Opcionales

Suponiendo que no se dispone del pasaporte, pero igual deseamos hacer la reserva:

```
<nombres>Julia</nombres>
<apellido>Urquidi</apellido>
<!--<pasaporte>664423</pasaporte>-->
<nacionalidad>BO</nacionalidad>
```

Para que esto sea válido podemos modificar el esquema así:

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

Por omisión minOccurs="1" y maxOccurs="1" por lo que normalmente todo elemento debe aparecer exactamente una vez.

Espacios de Nombres

En casos más complejos es frecuente la reutilización de distintos esquemas para validar un mismo documento. Los esquemas se pueden construir reutilizando tipos definidos en otros esquemas. Esto puede dar lugar a conflictos de nombres en los que distintos creadores de esquemas eligieron el mismo tag para designar algún elemento.

A fin de evitar estos conflictos, es conveniente que los tipos definidos en los esquemas tengan un "espacio de nombres" propio. Por ejemplo, para nuestro esquema de reservas podríamos definir el espacio de nombres: *"http://www.hotel-esquematico.com/reservas"* (convencionalmente se emplea un URI), y empleamos el atributo targetNamespace:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas">
<xsd:complexType name="tipo_residentes">

...

<xsd:sequence>
<xsd:element name="desde"
type="xsd:date"/>
<xsd:element name="hasta"
type="xsd:date"/>
<xsd:element name="residentes"
type="nuevo:tipo_residentes"/>
</xsd:sequence>
```

Nuestro esquema hace uso al interior del tipo "tipo_residentes" el cual ya no se puede referenciar por mediante type="tipo_residentes" dado que los tipos pertenecen al espacio de nombres que estamos creando, por lo tanto debemos hacer referencia al mismo usando un prefijo; por este motivo hemos asociado un prefijo ("nuevo") con el mismo

espacio de nombres con la directiva:

```
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
```

Así podemos definir el elemento "residentes" satisfactoriamente.

A continuación, el documento XML sufre estos cambios:

```
<res:reserva version="1"
xmlns:res="http://www.hotel-esquematico.com/reservas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="solicitud_ns.xsd">
...
</res:reserva>
```

Esto es, se ha añadido un prefijo "res" asociado al espacio de nombres que se creó en el esquema, y con este prefijo estamos declarando al elemento "reserva" (es decir, "res:reserva").

Calificación de elementos

Como se vio en la sección anterior, el documento sólo requirió prefijar al elemento "reserva". De acuerdo al esquema este es el único elemento que es declarado como hijo inmediato de <schema>. El resto de elementos del esquema (por ejemplo, "habitaciones", etc.). se denominan "locales".

Por omisión, el prefijo es necesario sólo para los elementos raíz de los documentos XML.

Si se desea forzar el prefijo a los elementos hijos (además de los raíz), se debe emplear la cualificación de elementos “qualified”:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas"
elementFormDefault="qualified">
```

Con esto nuestro documento tendrá esta forma:

```
<?xml version="1.0"?>
<res:reserva version="1"
xmlns:res="http://www.hotel-esquematico.com/reservas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="solicitud_ns.xsd">
<res:habitaciones>
<res:habitacion numero="202">
<res:desde>2009-11-22</res:desde>
<res:hasta>2009-11-24</res:hasta>
<res:residentes>
<res:residente responsable="true">
<res:nombres>Pedro</res:nombres>
...
```

Se puede exigir también que los atributos sea cualificados mediante el atributo de "schema": attributeFormDefault="qualified"

Cuando se tiene un documento XML en el que todos o la mayoría de elementos comparten un mismo prefijo, puede ser conveniente utilizar el "default namespace", que corresponde al espacio de nombre asociado a los elementos sin prefijo. Así, el documento anterior

equivale a este:

```
<?xml version="1.0"?>
<reserva version="1"
xmlns="http://www.hotel-esquematico.com/reservas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="solicitud_ns.xsd">
<habitaciones>
<habitacion numero="202">
<desde>2009-11-22</desde>
<hasta>2009-11-24</hasta>
<residentes>
<residente responsable="true">
<nombres>Pedro</nombres>
```

El uso de *xmlns="http://www.hotel-esquematico.com/reservas"*, esto es, el default namespace asumido para todos los elementos sin prefijo.

Dividiendo el esquema en partes menores

El esquema ahora será dividido en dos partes. Una de ellas contendrá la definición del tipo "tipo_residente":

```
$ more solicitud_nsq_p2.xsd
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas"
elementFormDefault="qualified">
<xsd:complexType name="tipo_residentes">
<xsd:sequence>
<xsd:element name="residente" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nombres" type="xsd:string"/>
<xsd:element name="apellido" type="xsd:string"/>
<xsd:element name="pasaporte" type="xsd:string"/>
<xsd:element name="nacionalidad" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="responsable" type="xsd:boolean"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

La otra parte incluirá la anterior:

```
$ more solicitud_nsq_p1.xsd

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas"
elementFormDefault="qualified">
<xsd:include schemaLocation="solicitud_nsq_p2.xsd"/>
<xsd:element name="reserva">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="habitaciones">
...
```

Aquí el elemento clave es "xsd:include", que permite incluir otro archivo de esquema. Con esto, todas las definiciones del primer esquema se suman al del segundo. El documento XML sólo requiere apuntar al documento "top".

Esto solo es válido si el documento incluido define elementos en el mismo espacio de nombres del inclusor (es decir, ambos comparten el mismo targetNamespace).