XML Schema

Tabla de contenidos

1.Introducción a XML Schema....1
1.1.El propósito de XML Schema....1
1.2.La potencia de XML Schema....1
1.3.Un primer vistazo....1
1.3.1.Un XML Schema simple....2
1.4.Validando un documento de instancia XML....2
2.Elementos de tipo simple....4
2.1.Introducción....4
2.2.Tipos simples predefinidos....4
2.2.1.Tipos de datos primitivos....4
2.2.2.Tipos de datos derivados....5
2.2.3.Definiendo un elemento de un tipo simple....5
2.3.Tipos simples derivados por el usuario....6
2.4.Aplicando Facets....7
2.4.1.Controlando la longitud....7
2.4.2.Especificando patrones....8
2.4.3.Trabajando con números....9
2.4.4.Enumeraciones....11
2.4.5.Manejo del espacio blanco....12
2.5.Especificando localmente el tipo de un elemento....13
2.6.Tipos no atómicos....14
2.6.1.Listas....14
2.6.2.Uniones....15
2.7.Declarando elementos de tipos simples globales....16
2.7.1.Elementos de tipo simple globales vs. locales....17
2.8.Valores por defecto....19
2.9.Valores Fijos....20
3.Elementos de tipo complejo....21
3.1.Introducción....21
3.2.Modelos de contenido....21
3.2.1.xs:sequence....21
3.2.2.xs:all....21
3.2.3.xs:choice....22
3.3.Modelos de grupo de contenidos....22
3.4.Restricciones de ocurrencia....23
3.5.Declarando elementos globales de tipo complejo....24
3.6.Contenido mixto....25
3.7.Definiendo tipos complejos globalmente....26
4.Atributos....28
4.1.Introducción....28
4.2.Elementos vacíos....28
4.3.Añadiendo atributos a elementos con contenido complejo....28
4.4.Añadiendo atributos a elementos con contenido simple....29
4.5.Restringiendo los valores de los atributos....30
4.6.Valores por defecto y fijos....33
4.6.1.Valores por defecto....33
4.6.2.Valores fijos....34
4.7.Requiriendo atributos....35
5.Usando claves en XML Schema....37
5.1.Unicidad....37
5.2.Claves....38
6.Apéndice A - Ejercicios....41

1. Introducción a XML Schema

1.1. El propósito de XML Schema

XML Schema es un lenguaje basado en XML que se utiliza para crear otros lenguajes basados en XML y modelos de datos. Un esquema XML define los nombres de los elementos y los atributos de una clase de documentos XML. El esquema también especifica la estructura que deben seguir esos documentos así como el tipo de contenido que puede haber dentro de cada elemento o atributo.

Los documentos XML que intentan seguir un esquema XML se llaman *instancias* de ese esquema. Si siguen el esquema de forma correcta entonces se dice que son *instancias válidas*. Ésto no es lo mismo que ser *bien formados*. Un documento XML bien formado sigue las reglas de sintaxis de XML, pero no sigue ningún esquema en particular. Por lo tanto, un documento XML puede ser bien formado sin ser válido, pero no puede ser válido si no es también bien formado.

1.2. La potencia de XML Schema

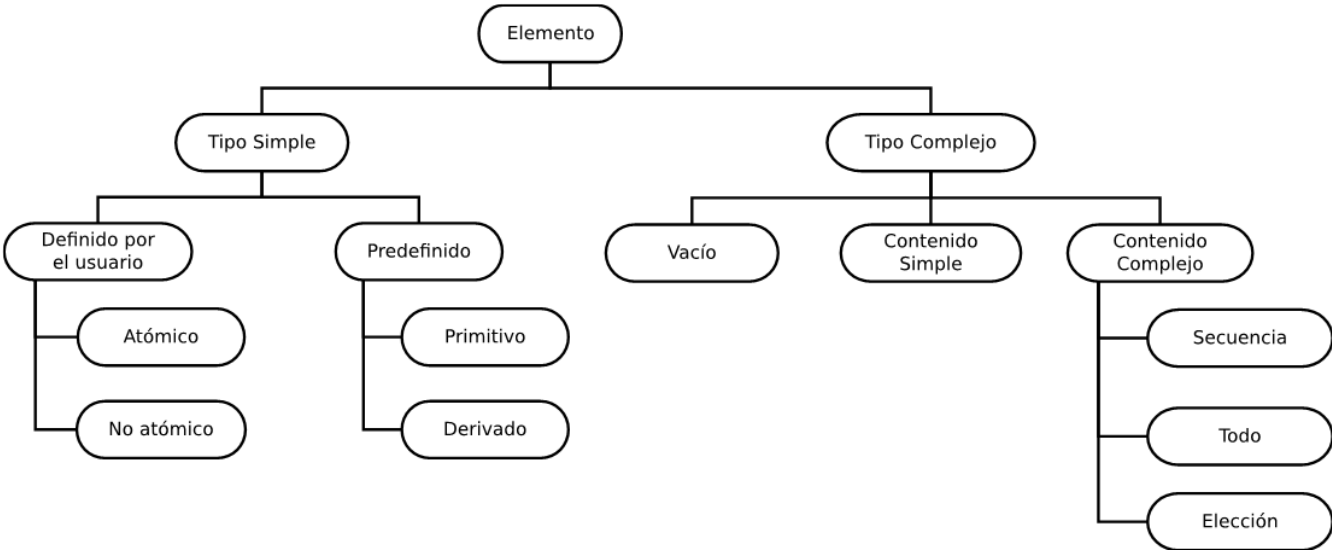
Una tecnología similar son los DTDs. Los DTDs son similares a los esquemas XML en el sentido de que también sirven para especificar clases de documentos XML. Se crearon originalmente para definir lenguajes basados en SGML, la especificación “padre” de XML. Aunque los DTDs son todavía comunes, XML Schema es un lenguaje bastante más potente.

A fin de conocer la potencia de XML Schema, a continuación se enumeran las limitaciones o carencias de los DTDs:

1. Los DTDs no tienen tipos de datos predefinidos.
2. Los DTDs no soportan tipos de datos derivados.
3. Los DTDs permiten un control bastante limitado de la cardinalidad (el número de ocurrencias de un elemento dentro de su elemento padre).
4. Los DTDs no soportan espacios de nombres ni ninguna forma simple de reutilizar o importar de otros esquemas.

1.3. Un primer vistazo

Un esquema XML define la estructura un documento instancia XML definiendo lo que cada elemento puede o debe contener. Un elemento está limitado por su tipo. Por ejemplo, un elemento de un tipo complejo puede contener elementos hijo y atributos, mientras un elemento de un tipo simple sólo puede contener texto. El siguiente diagrama da un primer vistazo a los tipos de elemento de XML Schema.



Los autores de esquemas XML pueden definir sus propios tipos o utilizar los tipos predefinidos. A lo largo de este curso nos referiremos de nuevo a este diagrama conforme se vayan definiendo elementos.

A continuación se define un resumen de alto nivel de los tipos de XML Xchema:

1. Los elementos pueden ser de un tipo simple o de un tipo complejo.
2. Los elementos de tipo simple sólo pueden contener texto. No pueden tener elementos hijos ni atributos.
3. Todos los tipos predefinidos son tipos simples.
4. Los autores de esquemas pueden derivar tipos simples restringiendo otro tipo simple. Por ejemplo, un tipo email puede derivarse limitando un tipo cadena a un patrón determinado.
5. Los tipos simples pueden ser atómicos (en general, cadenas y números) o no atómicos (en general, listas).
6. Los elementos de tipo complejo pueden contener otros elementos, atributos y texto.
7. Por defecto, los elementos de tipo complejo tienen contenido complejo, esto es, tienen elementos hijos.
8. Los elementos de tipo complejo pueden ser limitados para que tengan un contenido simple, lo que significa que sólo podrán contener texto. La diferencia con los elementos de tipo simple es que pueden tener atributos.
9. Los elementos de tipo complejo pueden ser limitados para que no tengan contenido, esto es que sean vacíos, pero pueden tener atributos.
10. Los tipos complejos pueden contener contenido mixto, una mezcla de texto y elementos hijos.

1.3.1. Un XML Schema simple

A continuación se muestra un esquema XML simple, que está compuesto de un tipo complejo y dos elementos hijos de tipo simple.

Código de ejemplo:
IntroSchema/Demos/Autor.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="autor">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string" />
        <xs:element name="apellidos" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Como se puede ver, un esquema XML es un documento XML y debe seguir las mismas reglas de sintaxis que cualquier otro documento XML; esto es, debe ser bien formado. Los esquemas XML también tienen que seguir las reglas definidas en “el Esquema de esquemas”, que define, entre otras cosas, la estructura y los nombres de los elementos y atributos de un esquema XML.

Aunque no es obligatorio, es una práctica común el utilizar el calificador `xs` para identificar elementos y tipos de XML Schema.

El elemento raiz de los esquemas XML es `xs:schema`. Necesita un atributo de espacio de nombres para los elementos de XML Schema `xmlns:xs=http://www.w3.org/2001/XMLSchema`.

En este esquema XML se ve un elemento `xs:element` dentro del elemento `xs:schema`. `xs:element` se utiliza para definir un elemento. En este caso define el elemento `autor` como un elemento de tipo complejo, que contiene una secuencia de dos elementos: `nombre` y `apellidos`, los cuales son de un tipo simple, string.

1.4. Validando un documento de instancia XML

En la sección anterior se vio un ejemplo de un esquema XML simple, que definía la estructura de un elemento autor. El ejemplo de debajo muestra una instancia válida de este esquema XML.

Código de ejemplo:

IntroSchema/Demos/MarkTwain.xml

```
<?xml version="1.0"?>
<autor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Autor.xsd">
     <nombre>Mark</nombre>
     <apellido>Twain</apellido>
</autor>
```

Este es un documento XML simple. Su elemento raíz es `autor`, que contiene dos elementos hijo: `nombre` y `apellidos`, tal y como especifica el esquema XML.

El atributo `xmlns:xsi` del elemento raíz indica que este documento XML es una instancia de un esquema XML en formato XML Schema. La localización del esquema XML se da mediante el atributo `xsi:noNamespaceSchemaLocation`.

Hay muchas formas de validar una instancia. Si se utiliza una herramienta de edición de XML es muy probable que disponga de herramientas de validación integradas. Si este no es el caso, se pueden utilizar herramientas de validación online como la disponible en la dirección https://www.corefiling.com/opensource/schemaValidate.html.

2. Elementos de tipo simple

2.1. Introducción

Los elementos de tipo simple no tienen elementos hijos ni atributos. Por ejemplo, el elemento `nombre` de ejemplo siguiente es un elemento de tipo simple mientras `persona` y `paginaWeb` no lo son.

Código de Ejemplo

TiposSimples/Demos/TipoSimple.xml

```
<?xml version="1.0"?>
<persona>
  <nombre>Mark Twain</nombre>
  <paginaWeb URL="http://www.marktwain.com"/>
</persona>
```

Como se vió en el esquema del primer capítulo, un tipo simple puede ser tanto predefinido como definido por el usuario. En este capítulo se describirán ambos.

2.2. Tipos simples predefinidos

XML Schema define 44 tipos predefinidos, 19 de los cuales son primitivos, esto es, tipos básicos.

2.2.1. Tipos de datos primitivos

Los 19 tipos de datos primitivos definidos en XML Schema son:

1. string. Cadena de caracteres sin limitaciones.
2. boolean. Valor booleano. Puede valer `0` ó `1`, `true` ó `false`.
3. decimal. Número real con decimales. Para separar la parte entera de la decimal se requiere un punto (.). Ejemplos: 0, 21.5, 0.32
4. float. Números en notación científica, utilizando la letra e para separar la mantisa del exponente. Ejemplo: 1.35e7, 2.672E-4.
5. double. Igual que float pero con doble precisión.
6. duration. Una duración de tiempo. Sigue el formato `PnYnMnDTnHnMnS`, donde `nY` es el número de años, `nM` el de meses, `nD` el de días, `nH` el de horas, `nM` el de minutos y `nS` el de segundos. Las letras P y T se utilizan para separar.
7. dateTime. Representa una fecha y hora. Sigue el formato `yyyy-mm-ddThh:mm:ss.nnnn:zzz`, donde `yyyy` es el año, el primer `mm` el mes, `dd` el día, `T` separa la fecha de la hora, `hh` son las horas, el segundo `mm` los minutos, `ss` los segundos, `nnnn` las fracciones de segundo y `zzz` la zona horaria.
8. time. Una hora dentro de un día. Se representa como la parte que sigue a la `T` en el tipo `dateTime`.
9. date. Una fecha que se refiere a un día. Se representa como la parte que precede a la `T` en el tipo `dateTime`.
10. gYearMonth. Un mes. Se da en formato `yyyy-mm`, donde `yyyy` es el año y `mm` el mes.
11. gYear. Un año. Se da en formato `yyyy`.
12. gMonthDay. Define un día dentro de un año, como el 12 de marzo. Se da en formato `--mm-dd`, donde `mm` es el mes y `dd` el día. Cuidado con los dos guiones iniciales.
13. gDay. Define un día de cualquier mes, de cualquier año. Se da en formato `---dd`, donde `dd` es el día. Cuidado con los tres guiones iniciales.
14. gMonth. Define un mes del año, por ejemplo, abril. Se da en formato `--mm`, donde `mm` es el mes. Cuidado con los dos guiones iniciales.
15. hexBinary. Información binaria codificada en hexadecimal. Debe tener una longitud par y sólo puede consistir

de números y los caracteres `a` a la `f` (mayúsculas o minúsculas).

16. base64Binary. Información binaria codificada en base64. Sólo se permiten los caracteres de la a a la z (tanto mayúsculas como minúsculas), números, el signo más (+), la barra (/) y el signo igual (=).
17. anyURI. Una URL, tanto absoluta como relativa.
18. QName. Nombre cualificado XML. Es válida cualquier cadena que valga como nombre de un elemento XML.
19. NOTATION. Una notación.

2.2.2. Tipos de datos derivados

Los otros 25 tipos simples predefinidos se derivan de alguno de los tipos simples primitivos, introduciendo alguna restricción.

1. normalizedString
2. token
3. language
4. NMTOKEN
5. NMTOKENS
6. Name
7. NCName
8. ID
9. IDREF
10. IDREFS
11. ENTITY
12. ENTITIES
13. integer
14. nonPositiveInteger
15. negativeInteger
16. long
17. int
18. short
19. byte
20. nonNegativeInteger
21. unsignedLong
22. unsignedInt
23. unsignedShort
24. unsignedByte
25. positiveInteger

2.2.3. Definiendo un elemento de un tipo simple

Un elemento de un tipo simple se define utilizando el atributo `type`.

Código de ejemplo:

TiposSimples/Demos/Autor.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="autor">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellidos" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

En el ejemplo anterior, los elementos `nombre` y `apellidos` no se definen de forma explícita como elementos de tipo simple. En lugar de eso se define su tipo utilizando el atributo `type`. Dado que el valor de este atributo (en ambos casos `string`) es un tipo simple, los elementos también son de un tipo simple.

Ejercicio 1. Creando un esquema simple

TiposSimples/Ejercicios/Ejercicio1.zip

2.3. Tipos simples derivados por el usuario

Un autor de esquemas XML puede derivar un nuevo tipo simple utilizando el elemento `xs:simpleType`. Este tipo simple puede ser utilizado a continuación de la misma forma en que se utilizan los tipos simples predefinidos.

Los tipos simples se derivan a base de *restringir* los tipos simples predefinidos o otros tipos simples derivados por el usuario. Por ejemplo, se podría querer crear un nuevo tipo simple llamado password, con una cadena de caracteres de 8 caracteres de longitud. Para hacer esto, hay que partir del tipo base `xs:string` y restringir su longitud a 8 caracteres. Esto se hace incluyendo el elemento `xs:restriction` dentro del elemento `xs:simpleType`.

Código de ejemplo:

/TiposSimples/Demos/Password.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="password">
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="usuario">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PW" type="password"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código de ejemplo:

/TiposSimples/Demos/Password.xml

```
<?xml version="1.0"?>
<usuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Password.xsd">
  <PW>MyPasWrd</PW>
</usuario>
```

2.4. Aplicando Facets

Los tipos simples se pueden derivar a base de aplicar una o mas de los siguientes *facets* (facetas):

- `length`. Especifica una longitud exacta.
- `minLength`. Especifica una longitud mínima.
- `maxLength`. Especifica una longitud máxima.
- `pattern`. Especifica un patrón mediante una expresión regular.
- `enumeration`. Especifica un conjunto de valores válidos. Sólo se podrá especificar uno de los valores dados.
- `whiteSpace`. Especifica como se tratarán los espacios en blanco dentro de una cadena.
- `minInclusive`. Permite definir un rango numérico dando el valor mínimo del rango, incluido el dado.
- `minExclusive`. Igual que el anterior pero excluyendo el número dado.
- `maxInclusive`. Permite definir un rango numérico dando el valor máximo del rango, incluyendo el dado.
- `maxExclusive`. Igual que el anterior, pero excluyendo el número dado.
- `totalDigits`. Permite restringir el total de dígitos de un número.
- `fractionDigits`. Permite restringir el número de dígitos *decimales* de un número.

2.4.1. Controlando la longitud

La longitud de una cadena se puede restringir utilizando los facets `length`, `minLength` y `maxLength`. Ya se utilizó el facet `length` en el ejemplo anterior para crear una un tipo simple *password* como una cadena de 8 caracteres. Se puede utilizar `minLength` y `maxLength` para permitir contraseñas que tengan entre seis y doce caracteres de longitud.

Código de Ejemplo:

TiposSimples/Demos/Password2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="password">
      <xs:restriction base="xs:string">
         <xs:minLength value="6"/>
         <xs:maxLength value="12"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="usuario">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="PW" type="password"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/Password2.xml

```
<?xml version="1.0"?>
<usuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Password.xsd">
   <PW>MyPasWrd</PW>
</usuario>
```

Código de Ejemplo:

TiposSimplesDemos/Password2b.xml

```
<?xml version="1.0"?>
<usuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Password2.xsd">
   <PW>MyPassWord</PW>
</usuario>
```

2.4.2. Especificando patrones

Los patrones se especifican utilizando el elemento `xs:pattern` y expresiones regulares. Por ejemplo, se podría utilizar el elemento `xs:pattern` para restringir el tipo simple *password* para que consista en entre seis y doce caracteres que sólo pueden ser letras mayúsculas y minúsculas y el carácter subrayado (_).

Código de Ejemplo:

TiposSimples/Demos/Password3.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="password">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Za-z_]{6,12}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="usuario">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PW" type="password"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/Password3.xml

```
<?xml version="1.0"?>
<usuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Password3.xsd">
  <PW>MyPassword</PW>
</usuario>
```

2.4.3. Trabajando con números

Los tipos simples numéricos se pueden derivar a base de limitar el valor a un determinado rango utilizando `minInclusive`, `minExclusive`, `maxInclusive` y `maxExclusive`. También se podría limitar el número total de dígitos y el número de dígitos de la parte decimal utilizando `totalDigits` y `fractionDigits`, respectivamente.

2.4.3.1. Mínimos y máximos

El ejemplo siguiente muestra como se deriva un tipo simple llamado Salario, que es un número decimal entre 10000 y 90000.

Código de Ejemplo:

TiposSimples/Demos/Empleado.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="salario">
      <xs:restriction base="xs:decimal">
         <xs:minInclusive value="10000"/>
         <xs:maxInclusive value="90000"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="salario" type="salario"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/JuanGomez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado.xsd">
   <salario>55000</salario>
</empleado>
```

2.4.3.2. Número de dígitos

Utilizando `totalDigits` y `fractionDigits`, se puede especificar además que el tipo Salario debe consistir de siete dígitos, dos de los cuales van detrás del punto decimal. Tanto `totalDigits` como `fractionDigits` son máximos, esto es, si `totalDigits` vale 5 y `fractionDigits` vale 2, un número válido no puede tener más de cinco dígitos en total y no más de dos después del punto decimal.

Código de Ejemplo:

TiposSimples/Demos/Empleado2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="salario">
      <xs:restriction base="xs:decimal">
         <xs:minInclusive value="10000"/>
         <xs:maxInclusive value="90000"/>
         <xs:fractionDigits value="2"/>
         <xs:totalDigits value="7"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="salario" type="salario"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/MariaSanchez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado2.xsd">
   <salario>55000.00</salario>
</empleado>
```

2.4.4. Enumeraciones

Un tipo derivado puede ser una lista de los posibles valores. Por ejemplo, el elemento `puestoTrabajo` podría ser una lista de nombres de puestos de trabajo predefinida.

Código de Ejemplo

TiposSimples/Demos/Empleado3.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="salario">
      <xs:restriction base="xs:decimal">
         <xs:minInclusive value="10000"/>
         <xs:maxInclusive value="90000"/>
         <xs:fractionDigits value="2"/>
         <xs:totalDigits value="7"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="puestoTrabajo">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Gestor de Ventas"/>
         <xs:enumeration value="Vendedor"/>
         <xs:enumeration value="Recepcionista"/>
         <xs:enumeration value="Desarrollador"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="salario" type="salario"/>
            <xs:element name="puesto" type="puestoTrabajo"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/EstebanGonzalez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado3.xsd">
   <salario>90000.00</salario>
   <puesto>Gestor de Ventas</puesto>
</empleado>
```

2.4.5. Manejo del espacio blanco

Por defecto, el espacio blanco en los elementos del tipo `xs:string` se preserva en un documento XML; sin embargo, esto se puede cambiar para los tipos derivados de `xs:string`. Ésto se hace utilizando el elemento `xs:whiteSpace`, el cual puede tomar uno de los siguientes valores:

- `preserve`. El espacio no se normaliza, esto es, se queda como esté.
- `replace`. Todos los tabuladores, saltos de línea y retornos de carro se reemplazan por espacios en blanco.
- `collapse`. Todos los tabuladores, saltos de línea y retornos de carro se reemplazan por espacios en blanco y a continuación todos los grupos de espacios en blanco se reemplazan por un único espacio en blanco. Finalmente se eliminan todos los espacios al inicio y al final.

En el ejemplo /TiposSimples/Demos/Password.xsd se pretendía restringir la longitud de un tipo Password a ocho caracteres, utilizando el elemento `xs:length`. Si el espacio se preserva, entonces los espacios al inicio y al final se consideran parte de la contraseña. En el siguiente ejemplo, ponemos `xs:whiteSpace` a `collapse` y descontando, por lo tanto, cualquier espacio al principio y al final. Como se puede ver, esto puede permitir al autor de una instancia XML el formatear el documento sin tener que preocuparse por los espacios.

Código de Ejemplo:

TiposSimples/Demos/Password4.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="password">
      <xs:restriction base="xs:string">
         <xs:length value="8"/>
         <xs:whiteSpace value="collapse"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="usuario">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="PW" type="password"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/Password4.xml

```
<?xml version="1.0"?>
<usuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Password4.xsd">
   <PW>
      12345678
   </PW>
</usuario>
```

Ejercicio 2. Restringiendo el contenido de un elemento

TiposSimples/Ejercicios/Ejercicio2.zip

2.5. Especificando localmente el tipo de un elemento

Hasta ahora en esta tema se han venido definiendo los tipos de forma *global* y posteriormente cambiando el atributo `type` de las declaraciones de los elementos para que sean del tipo simple derivado apropiado. Esto hace que sea muy simple el reutilizar un tipo ya definido a lo largo de múltiples elementos, tal y como se vio en el ejercicio anterior con el tipo nombrePropio.

También es posible el definir el tipo de un elemento localmente. En este caso el tipo no tiene nombre y sólo se aplica al elemento en el que se ha definido. La única razón para hacer esto es que se quiera dejar claro que el tipo es específico para este elemento y no se pretende reutilizarlo.

Código de Ejemplo:

TiposSimples/Demos/PasswordLocal.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="usuario">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="PW">
           <xs:simpleType>
             <xs:restriction base="xs:string">
               <xs:length value="8"/>
               <xs:whiteSpace value="collapse"/>
             </xs:restriction>
           </xs:simpleType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

2.6. Tipos no atómicos

Todos los tipos predefinidos de XML Schema son atomicos, esto es, no se pueden descomponer en trozos que tengan sentido. XML Schema proporciona dos tipos no atómicos: listas y uniones.

2.6.1. Listas

Los tipos lista son secuencias de tipos atómicos separados por espacio blanco; se puede tener una lista de enteros o una lista de fechas. Las listas no deben confundirse con las enumeraciones. Las enumeraciones proporcionan los valores que puede tomar un elemento, pero éste sólo puede tomar uno de ellos. Las listas representan uno o más valores *dentro* de un elemento.

Código de Ejemplo:

TiposSimples/Demos/ListaEmpleados.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="salario">
      <xs:restriction base="xs:decimal">
         <xs:minInclusive value="10000"/>
         <xs:maxInclusive value="90000"/>
         <xs:fractionDigits value="2"/>
         <xs:totalDigits value="7"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="puestoTrabajo">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Gestor de Ventas"/>
         <xs:enumeration value="Vendedor"/>
         <xs:enumeration value="Recepcionista"/>
         <xs:enumeration value="Desarrollador"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="listaFechas">
      <xs:list itemType="xs:date"/>
   </xs:simpleType>
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="salario" type="salario"/>
            <xs:element name="puesto" type="puestoTrabajo"/>
            <xs:element name="diasVacaciones" type="listaFechas"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/SandraLopez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ListaEmpleados.xsd">
   <salario>44000</salario>
   <puesto>Vendedor</puesto>
   <diasVacaciones>2006-08-13 2006-08-14 2006-08-15</diasVacaciones>
</empleado>
```

2.6.2. Uniones

Los tipos union son agrupaciones de tipos, permitiendo esencialmente que el valor de un elemento sea de más de un tipo. En el ejemplo siguiente se derivan dos tipos simples atómicos: carrera y gimnasia. Un tercer tipo simple, evento, se deriva entonces como una unión de los dos anteriores. El elemento evento es del tipo evento, lo que significa que puede ser tanto del tipo carrera como gimnasia.

Código de Ejemplo:

TiposSimples/Demos/Programa.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="carrera">
      <xs:restriction base="xs:string">
         <xs:enumeration value="100 metros"/>
         <xs:enumeration value="10 kilometros"/>
         <xs:enumeration value="Media maraton"/>
         <xs:enumeration value="Maraton"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="gimnasia">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Potro"/>
         <xs:enumeration value="Suelo"/>
         <xs:enumeration value="Anillas"/>
         <xs:enumeration value="Barra"/>
         <xs:enumeration value="Barras asimétricas"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="evento">
      <xs:union memberTypes="carrera gimnasia"/>
   </xs:simpleType>
   <xs:element name="programa">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="evento" type="evento"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposSimples/Demos/100Metros.xml

```
<?xml version="1.0"?>
<programa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Programa.xsd">
   <evento>100 metros</evento>
</programa>
```

Ejercicio 3 - Añadiendo tipos no atómicos

TiposSimples/Ejercicios/Ejercicio3.zip

2.7. Declarando elementos de tipos simples globales

Cuando la declaración de un elemento es un hijo directo del elemento raíz `xs:schema`, el elemento declarado se considera *global*. Los elementos globales pueden ser referenciados en las declaraciones de otros elementos, permitiendo la reutilización de elementos. Véase el siguiente ejemplo:

Código de Ejemplo:

TiposSimples/Demos/AutorGlobal.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="nombre" type="xs:string"/>
   <xs:element name="apellidos" type="xs:string"/>
   <xs:element name="autor">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="nombre"/>
            <xs:element ref="apellidos"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

En este ejemplo tanto el elemento `nombre` como el elemento `apellidos` se declaran ambos de forma global. Los elementos globales se referencian a continuación como hijos de la secuencia `autor`.

2.7.1. Elementos de tipo simple globales vs. locales

La principal ventaja de declarar un elemento globalmente es que el elemento puede ser a continuación referenciado a lo largo del esquema. Esto hace que el esquema sea más modular y fácil de mantener. Por ejemplo, supongamos que el esquema *cancion* contuviera los elementos *compositor*, *escritor* y *cantante*. Cada uno de estos elementos podría tener un elemento hijo llamado *nombre*. Si se declara el elemento nombre de forma global, cualquier cambio que quiera realizar sobre él sólo habría que hacerlo en un sitio, en lugar de en tres, como sería el caso si se declarara localmente dentro de compositor, escritor y cantante.

En cambio, la principal desventaja de declarar elementos globales es que todos los elementos globales deben tener un nombre único, esto es, no puede haber dos elementos globales con el mismo nombre. Esto es así por razones obvias: Si hay dos elementos globales con el mismo nombre y se encuentra una referencia con dicho nombre, ¿cual de los dos se usaría?

Código de Ejemplo:

TiposSimples/Demos/LibroLocal.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="tituloPersona">
     <xs:restriction base="xs:string">
       <xs:enumeration value="Sr."/>
       <xs:enumeration value="Sra."/>
       <xs:enumeration value="Dr."/>
     </xs:restriction>
   </xs:simpleType>
   <xs:element name="libro">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="titulo" type="xs:string"/>
         <xs:element name="autor">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="titulo" type="tituloPersona"/>
               <xs:element name="nombre" type="xs:string"/>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

Nótese que hay dos elementos llamados `titulo`, que aparecen en distintas partes de la instancia XML y que son de tipos diferentes. Cuando el elemento `titulo` aparezca como elemento raíz de la instancia XML, su valor podría ser cualquier cadena de texto; sin embargo, cuando aparezca como hijo de `autor`, su valor sólo puede ser “Sr.”, “Sra.” y “Dr.”.

El ejemplo siguiente define un modelo de contenido similar; sin embargo, dado que los elementos son declarados de forma global, el nombre `titulo` no se puede utilizar dos veces.

Código de Ejemplo:

TiposSimples/Demos/LibroGlobal.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="tituloPersona">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Sr."/>
         <xs:enumeration value="Sra."/>
         <xs:enumeration value="Dr."/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="tituloLibro" type="xs:string"/>
   <xs:element name="titulo" type="tituloPersona"/>
   <xs:element name="nombre" type="xs:string"/>
   <xs:element name="libro">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="tituloLibro"/>
            <xs:element name="autor">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="titulo"/>
                     <xs:element ref="nombre"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Ejercicio 4 - Convirtiendo declaraciones de elementos de tipo simple de local a global

TiposSimples/Ejercicios/Ejercicio4.zip

2.8. Valores por defecto

Los elementos que no tienen elementos hijos pueden tener valores por defecto. Para especificar un valor por defecto, hay que utilizar el atributo `default` del elemento `xs:element`.

Código de Ejemplo:

TiposSimples/Demos/EmpleadoDefecto.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- CODIGO OMITIDO -->
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="salario" type="salario"/>
            <xs:element name="puesto" type="puestoTrabajo" default="Vendedor"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Cuando se utilizan valores por defecto en un esquema, se aplican las siguientes reglas a las instancias:

1. Si el elemento aparece en la instancia y tiene contenido, el valor por defecto se ignora.

2. Si el elemento aparece en la instancia y NO tiene contenido, se le aplica el valor por defecto.
3. Si el elemento NO aparece, el elemento se queda fuera. Dicho de otra forma, proporcionar un valor por defecto no implica que el elemento se inserte si el autor de la instancia no lo ha incluido.

Si se examina la instancia siguiente, el elemento `puesto` no puede quedarse vacío; requiere que se utilice uno de los valores de la enumeración definida en el tipo simple `puestoTrabajo`. Sin embargo, y de acuerdo a la regla número 2 anterior, el procesador de XML Schema aplicaría el valor por defecto `Vendedor` al elemento puesto, de forma que la instancia se valida correctamente.

Código de Ejemplo:
TiposSimples/Demos/MiguelSanchez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="EmpleadoDefecto.xsd">
  <salario>90000</salario>
  <puesto/>
</empleado>
```

2.9. Valores Fijos

Los valores de un elemento pueden fijarse, lo que significa que, de aparecer en una instancia, deben contener el valor indicado. Los elementos de valor fijo se utilizan normalmente como opciones de tipo verdadero / falso.

Código de Ejemplo:
TiposSimples/Demos/EmpleadoFijo.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="salario">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="10000"/>
      <xs:maxInclusive value="90000"/>
      <xs:fractionDigits value="2"/>
      <xs:totalDigits value="7"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="puestoTrabajo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Gestor de Ventas"/>
      <xs:enumeration value="Vendedor"/>
      <xs:enumeration value="Recepcionista"/>
      <xs:enumeration value="Desarrollador"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="empleado">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="salario" type="salario"/>
        <xs:element name="puesto" type="puestoTrabajo"/>
        <xs:element name="estado" type="xs:string"
      fixed="actualizado" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

El atributo `minOccurs` (que veremos más tarde en detalle) se utiliza para indicar que el elemento `estado` es opcional. Sin embargo, si se utiliza, deberá contener el valor `actualizado` o debe estar vacío, en cuyo caso se supone que tiene el valor fijo. El archivo TiposSimples/Demos/LauraSanchez.xml valida contra este esquema.

3. Elementos de tipo complejo

3.1. Introducción

Los elementos de tipo complejo tienen atributos, elementos hijos o una combinación de ambos. Por ejemplo, los elementos `nombreCompleto` y `paginaWeb` del siguiente ejemplo son ambos de tipo complejo.

Código de Ejemplo:
TiposComplejos/Demos/TipoComplejo.xml

```
<?xml version="1.0"?>
<persona>
  <nombreCompleto>
    <nombre>Mark</nombre>
    <apellidos>Twain</apellidos>
  </nombreCompleto>
  <paginaWeb URL="http://www.marktwain.com"/>
</persona>
```

Un elemento de tipo complejo puede estar vacío, contener contenido simple como una cadena de caracteres o puede contener contenido complejo como una secuencia de elementos.

Aunque no es necesario declarar de forma explícita que un elemento de tipo simple es un tipo simple, si que es necesario el indicar que un elemento de tipo complejo ES un tipo complejo. Ésto se realiza mediante el elemento `xs:complexType` que se muestra a continuación:

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <!--La especificacion del contenido va aqui-→
  </xs:complexType>
</xs:element>
```

3.2. Modelos de contenido

Los modelos de contenido se utilizan para indicar la estructura y el orden en el cual pueden aparecer los elementos hijos dentro de su elemento padre. Los modelos de contenido se realizan a partir de modelos de grupo. Los tres tipos de modelo de grupo se muestran a continuación:

1. `xs:sequence`: Los elementos deben aparecer en el orden especificado.
2. `xs:all`: Los elementos deben aparecer, pero el orden no es importante.
3. `xs:choice`: Sólo uno de los elementos debe aparecer.

3.2.1. xs:sequence

El siguiente código muestra la sintaxis que se utiliza para declarar un elemento de tipo complejo como una secuencia, lo que significa que los elementos deberán aparecer en el mismo orden en el que se declaran.

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ElementoHijo1" type="xs:string"/>
      <xs:element name="ElementoHijo2" type="xs:string"/>
      <xs:element name="ElementoHijo3" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.2.2. xs:all

El siguiente código muestra la sintaxis para declarar un elemento de tipo complejo como una conjunción, lo que significa que los elementos pueden aparecer en cualquier orden, aunque deben aparecer todos.

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <xs:all>
      <xs:element name="ElementoHijo1" type="xs:string"/>
      <xs:element name="ElementoHijo2" type="xs:string"/>
      <xs:element name="ElementoHijo3" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

3.2.3. xs:choice

El siguiente código muestra la sintaxis a seguir para declarar un elemento de tipo complejo como una elección, lo que significa que uno y sólo uno de ellos puede aparecer como hijo del elemento padre.

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <xs:choice>
      <xs:element name="ElementoHijo1" type="xs:string"/>
      <xs:element name="ElementoHijo2" type="xs:string"/>
      <xs:element name="ElementoHijo3" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

3.3. Modelos de grupo de contenidos

En los ejemplos anteriores, los modelos de grupo estaban formados todos por elementos de tipo simple. Sin embargo, los elementos de tipo complejo pueden contener otros elementos de tipo complejo.

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <xs:choice>
      <xs:element name="ElementoHijo1" type="xs:string"/>
      <xs:element name="ElementoHijo2">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ElementoNieto1" type="xs:string"/>
            <xs:element name="ElementoNieto2" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="ElementoHijo3" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Además, los modelos de grupo pueden ir anidados unos dentro de otros. El siguiente ejemplo ilustra este punto. Hay

que destacar que el modelo de grupo `choice`, que permite tanto un elemento `salario` como un elemento `comision` está anidado dentro de un modelo de grupo `sequence`. Ambas de las instancias que se muestran detrás son válidas de acuerdo a este esquema.

Código de Ejemplo:

TiposComplejos/Demos/Empleado.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="salario">
      <xs:restriction base="xs:decimal">
         <xs:minInclusive value="10000"/>
         <xs:maxInclusive value="90000"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="empleado">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="nombreCompleto">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="nombre"/>
                     <xs:element name="Apellidos"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:choice>
               <xs:element name="salario" type="salario"/>
               <xs:element name="comision" type="xs:decimal"/>
            </xs:choice>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Código de Ejemplo:

TiposComplejos/Demos/DavidSanchez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado.xsd">
   <nombreCompleto>
      <nombre>David</nombre>
      <apellidos>Sanchez</apellidos>
   </nombreCompleto>
   <salario>90000</salario>
</empleado>
```

Código de Ejemplo:

TiposComplejos/Demos/JuliaSanchez.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado.xsd">
   <nombreCompleto>
      <nombre>Julia</nombre>
      <apellidos>Sanchez</apellidos>
   </nombreCompleto>
   <comision>20.50</comision>
</empleado>
```

3.4. Restricciones de ocurrencia

Por defecto, los elementos que declaran de forma local deben aparecer una vez y sólo una dentro de su elemento padre. Esta restricción se puede cambiar utilizando los atributos `minOccurs` y `maxOccurs`. El valor por defecto para cada

uno de estos atributos es de 1. El valor de `minOccurs` puede ser cualquier entero no negativo. El valor de `maxOccurs` puede ser cualquier entero positivo o el valor especial `unbounded`, que significa que el elemento puede aparecer cualquier número de veces.

El siguiente ejemplo muestra como se puede utilizar `minOccurs` para hacer que un elemento sea opcional y como se puede utilizar `maxOccurs` para que un elemento se repita de forma indefinida.

Código de Ejemplo:

TiposComplejos/Demos/Empleado2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="salario">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="10000"/>
      <xs:maxInclusive value="90000"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="empleado">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombreCompleto">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre"/>
              <xs:element name="primerApellido"/>
              <xs:element name="segundoApellido" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:choice>
          <xs:element name="salario" type="salario"/>
          <xs:element name="comision" type="xs:decimal"/>
        </xs:choice>
        <xs:element name="responsabilidades">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="responsibilidad" type="xs:string"
                maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Hay que tener en cuenta también que tanto `minOccurs` como `maxOccurs` se pueden aplicar también los modelos de grupo (e.g. `sequence`) para controlar el número de veces que se puede repetir un modelo de grupo.

Ejercicio 5 - Añadiendo elementos de tipo complejo

TiposComplejos/Ejercicios/Ejercicio5.zip

3.5. Declarando elementos globales de tipo complejo

Tal y como ocurre con los elementos de tipo simple, los elementos de tipo complejo se pueden declarar globalmente colocando la declaración del elemento como elemento hijo de `xs:schema`.

Los elementos declarados globalmente *no* pueden llevar restricciones de ocurrencia. Sin embargo, las referencias a los elementos declarados globalmente *si* pueden llevarlas. Para ilustrar este punto a continuación se muestra un ejemplo. En él todos los elementos, tanto los de tipo simple como los de tipo complejo se declaran globalmente y se referencian

desde dentro de los modelos de grupo. Algunas de las referencias, por ejemplo, responsabilidades, tienen restricciones de occurencia.

Código de Ejemplo:

TiposComplejos/Demos/Empleado3.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="salario">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="10000"/>
      <xs:maxInclusive value="90000"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="nombreCompleto">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <xs:element ref="primerApellido"/>
        <xs:element ref="segundoApellido" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="nombre"/>
  <xs:element name="primerApellido"/>
  <xs:element name="segundoApellido"/>
  <xs:element name="comision" type="xs:decimal"/>
  <xs:element name="salario" type="salario"/>
  <xs:element name="responsabilidades">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="responsabilidad" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="responsabilidad" type="xs:string"/>
  <xs:element name="empleado">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombreCompleto"/>
        <xs:choice>
          <xs:element ref="salario"/>
          <xs:element ref="comision"/>
        </xs:choice>
        <xs:element ref="responsabilidades" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ejercicio 6 - Convirtiendo elementos de tipo complejo locales en globales

TiposComplejos/Ejercicios/Ejercicio6.zip

3.6. Contenido mixto

A veces un elemento debe contener tanto otros elementos como texto. Por ejemplo, un elemento p (de HTML) puede contener tanto texto como otros elementos, como por ejemplo em o strong, mezclados con el texto.

Por ejemplo, tenemos el siguiente documento XML:

Código de Ejemplo:

TiposComplejos/Demos/PaulMcCartney.xml

```
<?xml version="1.0"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Empleado4.xsd">
   <nombreCompleto>
      <nombre>Paul</nombre>
      <primerApellido>McCartney</primerApellido>
   </nombreCompleto>
   <salario>90000</salario>
   <bio>
      Trabajó para <empresa>the Beatles</empresa> como
         <puesto>Cantante</puesto>.
      Trabajó para <empresa>the Beatles</empresa> como
         <puesto>Bajo</puesto>.
      Trabajó para <empresa>the Wings</empresa> como
         <puesto>Cantante</puesto>.
   </bio>
</empleado>
```

Se puede comprobar que el elemento `bio` tiene los elementos hijos `empresa` y `puesto` así como texto. De este tipo de elementos se dice que tienen contenido mixto. La sintaxis para declarar un elemento con contenido mixto se muestra a continuación.

```
<xs:element name="elementoPadre">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="elementoHijo1" type="xs:string"/>
      <xs:element name="elementoHijo2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El siguiente ejemplo ilustra como hacer esto en el esquema empleado.

Código de Ejemplo:

TiposComplejos/Demos/Empleado4.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- CODIGO OMITIDO -->
          <xs:element name="bio">
            <xs:complexType mixed="true">
              <xs:sequence maxOccurs="unbounded">
                <xs:element name="empresa" type="xs:string"/>
                <xs:element name="puesto" type="xs:string"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
<!-- CODIGO OMITIDO -->
</xs:schema>
```

3.7. Definiendo tipos complejos globalmente

Al igual que los tipos simples, los tipos complejos también pueden ser definidos globalmente. El ejemplo siguiente muestra como se puede hacer esto.

Código de Ejemplo:

TiposComplejos/Demos/Autor.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:complexType name="persona">
      <xs:sequence>
         <xs:element name="nombre" type="xs:string"/>
         <xs:element name="apellidos" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="autor" type="persona"/>
</xs:schema>
```

Como se puede ver, un tipo complejo se define utilizando el elemento `xs:complexType`. La ventaja principal de definir un tipo complejo de forma global es que puede ser reutilizado. Por ejemplo, un esquema puede permitir tanto un elemento `ilustrador` como un elemento `autor`. Los dos elementos pueden ser del tipo `persona`. De esta forma, si el tipo `persona` se cambia más tarde, el cambio se aplicará simultáneamente a los dos elementos.

El documento de instancia que se muestra debajo se debe validar correctamente contra el esquema anterior.

Código de Ejemplo:

TiposComplejos/Demos/MarkTwain.xml

```
<?xml version="1.0"?>
<autor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Autor.xsd">
   <nombre>Mark</nombre>
   <apellidos>Twain</apellidos>
</autor>
```

4. Atributos

4.1. Introducción

Aunque los atributos deben ser de un tipo simple, sólo los elementos de tipo complejo pueden tener atributos.

4.2. Elementos vacíos

Un elemento vacío es aquel que no tiene ningún contenido, pero si que puede tener atributos. El elemento `paginaWeb` del siguiente documento de instancia es un elemento vacío. Bajo la instancia se muestra el trozo de código del esquema Autor.xsd que declara el elemento `paginaWeb`.

Código de Ejemplo:

Atributos/Demos/MarkTwain.xml

```
<?xml version="1.0"?>
<autor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Autor.xsd">
  <nombreCompleto>
    <nombre>Mark</nombre>
    <primerApellido>Twain</primerApellido>
  </nombreCompleto>
  <paginaWeb URL="http://www.marktwain.com"/>
</autor>
```

Código de Ejemplo:

Atributos/Demos/Autor.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- CODIGO OMITIDO -->
         <xs:element name="paginaWeb">
           <xs:complexType>
             <xs:attribute name="URL" type="xs:anyURI"/>
           </xs:complexType>
         </xs:element>
<!-- CODIGO OMITIDO -->
</xs:schema>
```

4.3. Añadiendo atributos a elementos con contenido complejo

De los elementos que tienen elementos hijos se dice que tienen contenido complejo. Los atributos para dichos elementos se declaran DESPUES del modelo de grupos del elemento. Por ejemplo, el elemento `nombreCompleto` en la instancia siguiente tiene dos elementos hijos y dos atributos. Bajo la instancia se muestra el esquema correspondiente.

Código de Ejemplo:

Atributos/Demos/MarkTwain2.xml

```
<?xml version="1.0"?>
<autor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Autor2.xsd">
  <nombreCompleto pseudonimo="true" paginaWeb="http://www.marktwain.com">
    <nombre>Mark</nombre>
    <primerApellido>Twain</primerApellido>
  </nombreCompleto>
</autor>
```

Código de Ejemplo:

Atributos/Demos/Autor2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="autor">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombreCompleto">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="primerApellido" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="pseudonimo" type="xs:boolean"/>
            <xs:attribute name="paginaWeb" type="xs:anyURI"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.4. Añadiendo atributos a elementos con contenido simple

Un elemento con contenido simple es aquel que sólo contiene texto. Si ese elemento contuviera uno o más atributos, entonces pasaría a convertirse en un elemento con contenido complejo. Los elementos con contenido simple y atributos se declaran utilizando el elemento `xs:simpleContent` y posteriormente *extendiendo* el elemento utilizando el elemento xs:extension, que debe especificar el tipo del contenido simple utilizando el atributo base. A continuación se muestra la sintaxis.

Código de Ejemplo:

```
<xs:element name="nombreElemento">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="nombreAtributo" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Por ejemplo, el elemento nombre de la instancia XML siguiente contiene sólo contenido simple y tiene un único atributo. Bajo la instancia se muestra el esquema.

Código de Ejemplo

Atributos/Demos/NatHawthorne.xml

```
<?xml version="1.0"?>
<autor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Autor3.xsd">
   <nombreCompleto pseudonimo="true"
paginaWeb="http://www.nathanielhawthorne.com">
     <nombre completo="false">Nat</nombre>
     <primerApellido>Hawthorne</primerApellido>
   </nombreCompleto>
</autor>
```

Código de Ejemplo:

Atributos/Demos/Autor3.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="autor">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="nombreCompleto">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="nombre">
                 <xs:complexType>
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute name="completo" type="xs:boolean"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
               <xs:element name="primerApellido" type="xs:string"/>
             </xs:sequence>
             <xs:attribute name="pseudonimo" type="xs:boolean"/>
             <xs:attribute name="paginaWeb" type="xs:anyURI"/>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

4.5. Restringiendo los valores de los atributos

Los valores de los atributos se pueden restringir de la misma forma en que se hace con los valores de los elementos de tipo simple. A continuación hay tres ejemplos.

Este primer ejemplo muestra como restringir el valor de un atributo definiendo su tipo de forma local. Puede probar el archivo Atributos/Demos/HuckFinn.xml contra este esquema.

Código de Ejemplo:

Atributos/Demos/Libro.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="titulo">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="Sr."/>
                  <xs:enumeration value="Sra."/>
                  <xs:enumeration value="Dr."/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

El segundo ejemplo muestra como restringir el valor de un atributo aplicando un tipo simple definido globalmente. Se puede probar si el archivo /Atributos/Demos/TomSawyer.xml valida contra este esquema.

Código de Ejemplo:

Atributos/Demos/Libro2.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="tituloPersona">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Sr."/>
         <xs:enumeration value="Sra."/>
         <xs:enumeration value="Dr."/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="libro">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="titulo" type="xs:string"/>
            <xs:element name="autor">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="nombre" type="xs:string"/>
                  </xs:sequence>
                  <xs:attribute name="titulo" type="tituloPersona"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

El tercer ejemplo muestra cómo declarar un atributo de forma global. Se puede probar si valida el documento Atributos/Demos/LifeInTheMississippi.xml.

Código de Ejemplo:

Artibutos/Demos/Libro3.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="titulo">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Sr."/>
        <xs:enumeration value="Sra."/>
        <xs:enumeration value="Dr."/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
            </xs:sequence>
            <xs:attribute ref="titulo"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.6. Valores por defecto y fijos

4.6.1. Valores por defecto

Los atributos pueden tener valores por defecto. Para especificar un valor por defecto se utiliza el atributo `default` del elemento `xs:attribute`. Los valores por defecto para los atributos funcionan de una manera ligeramente distinta a como lo hacen en los elementos. Si el atributo no aparece en la instancia, el procesador de XML Schema lo inserta con el valor por defecto. Puede probar la instancia Atributos/Demos/NatHawthorne2.xml contra este esquema.

Código de Ejemplo:

Atributos/Demos/Autor4.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="autor">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="nombreCompleto">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="nombre">
                 <xs:complexType>
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute name="completo" type="xs:boolean"
default="true"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
               <xs:element name="primerApellido" type="xs:string"/>
             </xs:sequence>
             <xs:attribute name="pseudonimo" type="xs:string" default="false"/>
             <xs:attribute name="paginaWeb" type="xs:anyURI"/>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

4.6.2. Valores fijos

El valor de un atributo puede fijarse, lo que implica que, si aparece en el documento de instancia, debe contener el valor especificado. Al igual que para los elementos de tipo simple, esto se realiza utilizando el atributo fixed. Puede probar el archivo de instancia Atributos/Demos/NatHawthorne3.xml contra este esquema.

Código de Ejemplo:

Atributos/Demos/Autor5.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="autor">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="nombreCompleto">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="nombre">
                        <xs:complexType>
                           <xs:simpleContent>
                              <xs:extension base="xs:string">
                                 <xs:attribute name="completo" type="xs:boolean"
default="true"/>
                              </xs:extension>
                           </xs:simpleContent>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="primerApellido" type="xs:string"/>
                  </xs:sequence>
                  <xs:attribute name="pseudonimo" type="xs:boolean" fixed="true"/>
                  <xs:attribute name="paginaWeb" type="xs:anyURI"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

4.7. Requiriendo atributos

Por defecto, los atributos son opcionales, pero se pueden hacer que sean obligatorios cambiando el valor del atributo `use` de `xs:attribute` a `required`, tal y como se muestra a continuación.

Código de Ejemplo:

/Atributos/Demos/Autor6.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="autor">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="nombreCompleto">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="nombre">
                 <xs:complexType>
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute name="completo" type="xs:boolean"
default="true"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
               <xs:element name="primerApellido" type="xs:string"/>
             </xs:sequence>
             <xs:attribute name="pseudonimo" type="xs:boolean" fixed="true"/>
             <xs:attribute name="paginaWeb" type="xs:anyURI" use="required"/>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

Ejercicio 7 - Añadiendo atributos a los elementos

Atributos/Ejercicios/Ejercicio7.zip

5. Usando claves en XML Schema

5.1. Unicidad

XML Schema proporciona un mecanismo para requerir que haya elementos que sean únicos de alguna forma respecto a los demás.

Esto se ilustra mejor con un ejemplo:

Código de Ejemplo:

Claves/Demos/Unico.xsd

```
<?xml version="1.0"?>
<!-- CODIGO OMITIDO -->
          <xs:element name="artistas">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="artista" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="unID" type="xs:string"
use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="letra">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="estrofa" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="linea" type="xs:string"
maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="artista" type="xs:string"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
      <xs:unique name="claveArtista">
        <xs:selector xpath="artistas/artista"/>
        <xs:field xpath="@unID"/>
      </xs:unique>
    </xs:element>
</xs:schema>
```

El elemento `artista` tiene un atributo `unID`, el cual se quiere utilizar para identificar de forma única a cada artista. El elemento `xs:unique` de XML Schema se utiliza para forzar a que esto sea así. Tiene dos elementos hijo:

- `xs:selector`: Tiene un atributo `xpath` que contiene una expresión de la especificación XPath 1.0 la cual referencia los elementos que se verán afectados por la restricción.

- `xs:field`: Tiene también un atributo `xpath` que contiene asimismo una expresión XPath 1.0 que especifica qué parte del elemento tiene que ser única.

En el ejemplo anterior, la expresión XPath de `xs:selector` identifica todos los elementos `artista` que son hijos de elementos `artistas`. La expresión XPath de `xs:field` identifica al atributo `unID` como la parte del elemento `artista` que debe ser única.

En la instancia XML siguiente, cada artista debe tener un atributo unID único. Para probarlo se podrían cambiar para que fueran iguales y validar.

Código de Ejemplo:

Claves/Demos/Unico.xml

```
<?xml version="1.0"?>
<cancion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Unico.xsd">
   <titulo tipo="dueto">The Girl Is Mine</titulo>
   <año>1983</año>
   <duracion>media</duracion>
   <artistas>
      <artista unID="MJ">Michael Jackson</artista>
      <artista unID="PM">Paul McCartney</artista>
   </artistas>
<!-- CODIGO OMITIDO -->
</cancion>
```

5.2. Claves

XML Schema también proporciona un mecanismo para utilizar claves y referencias a claves, esto es, para crear una relación entre elementos a través del valor de un atributo o elemento hijo. Los elementos xs:key y xs:keyref se utilizan para crear este tipo de relación.

Código de Ejemplo:

Claves/Demos/Claves.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!--CODIGO OMITIDO -->
         <xs:element name="artistas">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="artista" maxOccurs="unbounded">
                 <xs:complexType>
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute name="unID" type="xs:string"
use="required"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="letra">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="estrofa" maxOccurs="unbounded">
                 <xs:complexType>
                   <xs:sequence>
                     <xs:element name="linea" type="xs:string"
maxOccurs="unbounded"/>
                   </xs:sequence>
                   <xs:attribute name="artista" type="xs:string"/>
                 </xs:complexType>
               </xs:element>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
     <xs:key name="claveArtista">
       <xs:selector xpath="artistas/artista"/>
       <xs:field xpath="@unID"/>
     </xs:key>
     <xs:keyref name="refClaveArtista" refer="claveArtista">
       <xs:selector xpath="letra/estrofa"/>
       <xs:field xpath="@artista"/>
     </xs:keyref>
   </xs:element>
</xs:schema>
```

Al igual que el elemento `xs:unique`, también los elementos `xs:key` y `xs:keyref` contienen cada uno elementos hijos `xs:selector` y `xs:field`.

El elemento `xs:key` se utiliza para identificar los elementos que van a ser referenciados por los elementos especificados mediante el elemento `xs:keyref`.

En el ejemplo anterior, el atributo `artista` del elemento `estrofa` debe referenciar a un atributo `unID` de un elemento `artista` que debe ser único.

En la instancia siguiente, cada artista debe tener un atributo unID único y cada elemento estrofa debe tener un atributo artista con el mismo valor que el de alguno de los atributos unID de un elemento artista. Si se intenta cambiar el valor del atributo artista de alguna de las estrofas a un valor que no existe en alguno de los atributos unID de algún artista o se

intenta dar el mismo valor a los dos atributos unID de los elementos artista, se producirá un error.

Código de Ejemplo:

Claves/Demos/Claves.xml

```
<?xml version="1.0"?>
<cancion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Claves.xsd">
   <titulo tipo="dueto">The Girl Is Mine</titulo>
   <año>1983</año>
   <duracion>media</duracion>
   <artistas>
     <artista unID="MJ">Michael Jackson</artista>
     <artista unID="PM">Paul McCartney</artista>
   </artistas>
   <letra>
     <estrofa artista="MJ">
       <linea>Every night she walks right in my dreams</linea>
       <linea>Every night she walks right in my dreams</linea>
<!-- CODIGO OMITIDO -->
     </estrofa>
     <estrofa artista="PM">
       <linea>I don't understand the way you think</linea>
       <linea>Saying that she's yours not mine</linea>
<!-- CODIGO OMITIDO -->
     </estrofa>
     <estrofa artista="MJ">
       <linea>I know she'll tell you I'm the one for her</linea>
       <linea>'Cause she said I blow her mind</linea>
<!-- CODIGO OMITIDO -->
     </estrofa>
   </letra>
</cancion>
```

Por lo tanto, el mecanismo de claves asegura dos premisas:

- Los valores de las claves deben ser únicos, esto es, no puede haber dos claves con el mismo valor.
- Debe haber integridad referencial, esto es, no puede haber una referencia a una clave por un valor que no exista entre las claves.

6. Apéndice A - Ejercicios

En esta sección se incluyen ejercicios que utilizan todas o parte de las herramientas que se han definido a lo largo del curso.

Ejercicio A1 - Esquema a partir de definición escrita

Apendice A/Ejercicio A1