

DAM

Linguaxes de Marcas e Sistemas de Información

UD3 **DTD
Y
ESQUEMAS**

PROFESORA | CRISTINA PUGA FDEZ

UNIDADE | UD3

TRIMESTRE | 2

MÓDULO | LMSXI

3.1	Introducción.....	3
3.2	DTD	3
3.2.1	Validación de documentos XML	3
3.2.2	Definición de la DTD	5
3.2.2.1	Elementos	5
3.2.2.2	Atributos	6
3.2.2.3	Entidades	8
3.2.3	Carencias de las DTDs	9
3.3	Esquemas XML.....	10
3.3.1	Ventajas	10
3.3.2	Estructura de un XSD	11
3.3.2.1	Vinculación del XSD desde el XML	11
3.3.2.2	Definición del XSD	13
3.3.3	Tipos de datos.....	13
3.3.3.1	Declaración de elementos XML	14
3.3.3.2	Declaración de atributos	15
3.3.3.3	Tipos simples	16
3.3.3.3.1	Restricciones	17
3.3.3.3.2	Listas	19
3.3.3.3.3	Union	20
3.3.3.4	Tipos complejos	21
3.3.3.4.1	Atributos	22
3.3.3.4.2	Subelementos	22
3.3.3.4.3	Elementos con contenido simple	25
3.3.3.4.4	Elementos con contenido complejo	25
3.3.4	Otros elementos del lenguaje	26
3.3.4.1	Grupos	26
3.3.4.2	Anotaciones	27
3.3.4.3	Valores nulos	28
3.3.4.4	Elementos únicos	28
3.3.4.5	Claves	29
3.3.5	Reutilización de esquemas	31
ANEXO I:	Materiales	34

3. DTDs y Esquemas

3.1 Introducción

Con XML se pueden crear nuevos elementos y atributos libremente, por lo que si queremos emplear documentos XML con una estructura fija para intercambiar información, por ejemplo, necesitamos indicar de alguna manera:

- Qué nombres se pueden emplear para los elementos, cuántas veces se pueden usar y en qué orden.
- Qué nombres se pueden emplear para los atributos, qué elementos los usan y si son obligatorios o opcionales.
- Qué tipo de valores admite cada elemento o atributo, y si tienen valores por defecto.

Para ello podemos emplear definiciones **DTD** o **esquemas** XML.

3.2 DTD

El objetivo de una DTD (definición de tipo de documento, document type definition) es definir la estructura de un documento XML, es decir, establecer la gramática que contendrá las restricciones que deben cumplir los elementos del XML.

Cuando asignamos una DTD a un documento podemos verificar si el documento es correcto, es decir, su estructura responde a lo definido en la DTD.

3.2.1 Validación de documentos XML

El proceso de comparar un documento XML con su DTD para comprobar si su estructura es correcta se denomina **validación**, y se lleva a cabo por medio de un analizador que interpreta el documento. Los navegadores web disponen de su propio analizador XML que permite verificar si una página web se ajusta a la estructura definida para HTML.

Un documento XML es **válido** si está bien formado (tiene una sintaxis XML correcta) y además se ajusta a las reglas de un DTD (*Document Type Definition*).

Para poder verificar si un documento XML, el analizador necesita conocer su gramática para validar su construcción. Existen dos maneras de vincular la información de su estructura con un documento XML: mediante la inserción de las reglas de la gramática dentro del archivo XML o añadiendo una referencia a un archivo DTD donde se definirá la gramática del documento.

Tanto en un caso como en otro se debe indicar en el DOCTYPE del documento XML cuál es su elemento raíz y a continuación las reglas que debe cumplir (incluyéndolas directamente o referenciando el documento que las contiene):

```
<!DOCTYPE elemento_raiz SYSTEM "fichero.dtd">
```

En el prólogo de un documento XML bien formado, además de la versión del lenguaje XML se incluye un atributo “standalone”:

```
<?xml version="1.0" standalone="yes"?>
```

Este termino *standalone* permite especificar si el documento es “autosuficiente” o depende de algún otro documento o recurso donde se defina, por ejemplo, su gramática (su DTD).

- Si incluimos la declaración de la gramática dentro del propio fichero XML (en su DOCTYPE), el documento tendrá “standalone=yes”, ya que no depende de ningún otro. La declaración va en el DOCTYPE, insertada entre corchetes.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Declaraciones DTD-->
<!DOCTYPE tema [
  <!ELEMENT tema (autor, apartado+)>
  <!ATTLIST tema
    titulo CDATA #REQUIRED
    unidad CDATA #REQUIRED>
  <!ELEMENT autor (#PCDATA)>
  <!ELEMENT apartado (#PCDATA)>
  <!ATTLIST apartado
    numero CDATA #REQUIRED>
]>
<!-- Datos XML-->
<tema unidad="3" titulo="XML">
  <autor>Cristina Puga</autor>
  <apartado numero="1">Contenido del apartado 1</apartado>
  <apartado numero="2">Contenido del apartado 2</apartado>
</tema>
```

- Si definimos su gramática en un fichero externo (nota.dtd), el atributo standalone tendrá el valor “no”. Para referenciar un documento externo lo haremos también en el DOCTYPE del XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE nota SYSTEM "nota.dtd">
<nota>
<destinatario>Tove</destinatario>
<remitente>Jani</remitente>
<cabeceira>Recordatorio</cabeceira>
<corpo>Chámame!</corpo>
</nota>
```

El fichero donde se incluye la DTD se puede referenciar por medio de su ruta relativa o por medio de una URI:

```
<!DOCTYPE notas SYSTEM
"http://www.servidor.org/dtd/notas.dtd">
```

3.2.2 Definición de la DTD

3.2.2.1 Elementos

Los elementos son los nodos del árbol de un documento XML.

Los elementos se definen de la siguiente manera:

```
<!ELEMENT nombreElemento contenido>
```

- Donde contenido puede ser:

		DTD	XML
EMPTY	Elemento sin contenido, aunque podría tener atributos	<!ELEMENT br EMPTY>	
ANY	Permite cualquier tipo de XML como contenido (no se recomienda su uso).		
(#PCDATA)	Contenido textual, un conjunto de caracteres	<!ELEMENT modulo (#PCDATA)>	<modulo>LMSXI</modulo>
Subelementos	Entre paréntesis y separados por comas (indica orden obligatorio) o por el operador (indica opcionalidad entre ellos)	<!ELEMENT correo (#PCDATA imagen)*> <!ELEMENT imagen EMPTY>	<correo><imagen src="foto.jpg"/> </correo>

Existen dos tipos de elementos: los elementos terminales (las hojas del árbol XML) y los elementos no terminales (las ramas, que son elementos anidados).

- Los elementos anidados o no terminales son los que tienen subelementos:

```
<!ELEMENT listadeproductos (producto)>
<!ELEMENT producto (nombre,codigo,precio)>
```

- Los elementos terminales tienen algún tipo de datos:

```
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT codigo (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
```

Cuando definimos un agrupamiento de subelementos, los datos textuales #PCDATA deben aparecer siempre en primera posición.

Sobre los subelementos y los agrupamientos de subelementos podemos aplicar los siguientes operadores:

- ? - indica que puede incluirse o no (0 o 1 vez)
- * - indica que puede repetirse o no (0 o más veces)
- + - indica que debe incluirse y puede repetirse (1 o más veces)

```
<!ELEMENT
  modulo(nombre,(horas|créditos),profesor*,aula,taller?)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT horas (#PCDATA)>
<!ELEMENT profesor (#PCDATA)>
<!ELEMENT aula (#PCDATA)>
<!ELEMENT taller (#PCDATA)>
```

3.2.2.2 Atributos

Los atributos de un elemento se definen de la siguiente manera:

```
<!ATTLIST nombreElemento
  Atributo1 tipo valor
  Atributo2 tipo valor
  ...
>
```

El tipo del atributo puede ser:

- CDATA: character data, puede tomar cualquier valor de texto.

```
<!ATTLIST profesor nombre CDATA #REQUIRED>
```

- Lista entre paréntesis con los posibles valores que puede tomar separados por el operador |.

```
<!ATTLIST tipo dietetico (si|no) #REQUIRED>
```

- ID: identificador del elemento. Debe ser único en el documento y debe empezar por una letra o por el carácter de subrayado:

```
<!ATTLIST profesor id ID #REQUIRED>
```

- IDREF: son referencias a identificadores (atributos definidos como ID). El valor del atributo IDREF debe corresponder a un identificador de elemento existente en el documento.

```
<!ATTLIST alumno tutor IDREF #IMPLIED>
```

- IDREFS: permite especificar varias referencias a identificadores dentro de la declaración de un atributo. Las distintas referencias irán separadas por un espacio en blanco.

```
<!ELEMENT libro (receita | ingrediente)*>
<!ELEMENT receita #PCDATA>
<!ATTLIST receita id ID #REQUIRED>
<!ELEMENT ingrediente #PCDATA>
<!ATTLIST ingrediente ref IDREFS #IMPLIED>
<libro>
  <receita id="rec_1">Filloas</receita>
  <receita id="rec_2"> Flan de queixo </receita>
  <receita id="rec_3"> Torta de mazá </receita>
  <ingrediente ref="rec_1 rec_3">ovos</ingrediente>
  <ingrediente ref="rec_2">Queixo</ingrediente>
  <ingrediente ref="rec_1 rec_2">Leite</ingrediente>
</libro>
```

El valor puede ser:

- #REQUIRED: obliga a especificar el atributo.
- #IMPLIED: permite no especificar el atributo, sin que tome un valor por defecto.
- #FIXED valor: el valor del atributo permanece constante para ese elemento en todo el documento.

```
<!ATTLIST tipo dietetico CDATA #FIXED "desconocido">
```

- valor predefinido: se especifica directamente el valor por defecto que tomará el atributo si no indica un valor

```
<!ATTLIST tipo dietetico CDATA "no">
```

Los atributos de un elemento se pueden indicar unos a continuación de otros o en distintas etiquetas:

```
<!ATTLIST ciclo codigo  
CDATA #REQUIRED grao  
CDATA #REQUIRED>
```

```
<!ATTLIST ciclo codigo CDATA  
#REQUIRED>  
<!ATTLIST ciclo grao CDATA  
#REQUIRED >
```

3.2.2.3 Entidades

Las entidades permiten hacer referencia a otros contenidos que se incrustarán en el documento en el momento de ser procesados, y que no deben ser analizados sintácticamente según las reglas de XML. Esto incluye "abreviaturas" para términos que se repiten mucho o fragmentos de código que no debe interpretar el parseador, por lo que se suelen utilizar para definir macros, scripts o constantes.

Tal como se vio anteriormente, XML tiene algunas entidades internas predefinidas para permitir insertar caracteres que tienen un significado especial en XML, como los símbolos <, >, etc.

Carác	Enti
&	&am
<	<
>	>
"	&qu
'	&ap

Las entidades se declaran de la siguiente manera:

```
<!ENTITY nombreEntidad "cadena a reemplazar">
```

Por ejemplo:

```
<!ENTITY ciudad "Santiago de Compostela">
```

Para usarlas en el documento tendremos que poner: &nombreEntidad;


```
&ciudad;
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE tarjetaVisita [
<!ENTITY ciudad "Santiago de Compostela">
]>
<tarjetaVisita>
  <apellidos>Graña Arias</apellidos>
  <nombre>Rodrigo</nombre>
  <profesion>Doctor</profesion>
  <direccion>
    <calle>Ponzano</calle>
    <numero>66</numero>
    <poblacion>&ciudad;</poblacion>
    <cp>15891</cp>
  </direccion>
</tarjetaVisita>
```

3.2.3 Carencias de las DTDs

Las DTDs se usan cada vez menos gracias sobre todo a la existencia de lenguajes de validación más modernos. Los principales problemas de las DTDs son:

- Ausencia de tipos de datos: únicamente podemos decir que un valor es de tipo PCDATA, por lo que podemos validar la estructura del documento pero no su contenido. Con DTDs no es posible decirle, por ejemplo:
 - que el contenido de la etiqueta <cantidad> sea un número positivo.
 - que el contenido de la etiqueta <mes> sea un número entero entre 1 y 12.
 - que el texto de la etiqueta <title> sea una cadena de hasta 255 caracteres.

```
<!--DTD -->
<!ELEMENT guarderia (niño*)>
<!ELEMENT niño (nombre, fecnacimiento, peso, altura, vacunas)>
<!ATTLIST niño numeroExpediente CDATA #REQUIRED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT fecnacimiento (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
<!ELEMENT altura (#PCDATA)>
<!ELEMENT vacunas (#PCDATA)>
<!-- XML -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE guarderia SYSTEM "guarderia.dtd">
```

```
<guarderia>
  <niño numeroExpediente="popeye">
    <nombre>cosas</nombre>
    <fecnacimiento>1.1.1</fecnacimiento>
    <peso>no lo se</peso>
    <altura>falso</altura>
    <vacunas>no</vacunas>
  </niño>
</guarderia>
```

Este documento sería válido según su DTD, a pesar de que los valores del documento no son correctos.

- No se definen en XML, lo que implica la necesidad de tener distintos analizadores y APIs para procesar el documento: uno para la DTD y para el documento.
- No permite definir secuencias de atributos no ordenadas
- No es posible formar claves a partir de varios atributos o elementos.
- Son muy poco extensibles, y es difícil combinar diferentes DTDs entre sí.
- No son compatibles con los espacios de nombres XML.

3.3 Esquemas XML

El XML Schema Definition Language (XSDL) es un lenguaje XML recomendado por el W3C, creado para describir la estructura y restricciones del contenido de un documento XML. XSDL permite especificar los tipos de datos de los elementos de un documento XML y mejora en muchos aspectos la validación de documentos de DTDs.

Los archivos en los que definimos los esquemas XML tienen extensión xsd (XML Schema Definition), y son en sí mismos documentos XML. Cuando se valida un documento XML que tiene vinculado un esquema XML en realidad se están realizando dos validaciones, ya que el documento xml se valida con el esquema xml, y este a su vez se validaría con un hipotético XMLSchema.xsd, ya que tiene que cumplir la sintaxis de los esquemas XML.

3.3.1 Ventajas

Los esquemas XML pretenden solucionar los principales problemas de las DTD, por lo que tienen varias ventajas:

- Permite validar el contenido de los campos del documento
- Facilita el procesamiento de los documentos XML por parte de lenguajes de programación que usen reglas estrictas para definir los tipos de datos utilizados.
- Únicamente necesitamos un parseador para cotejar el documento XML y la definición de su esquema, ya que ambos vienen indicados en XML.
- Permiten definir secuencias de elementos que pueden aparecer en cualquier orden: (nombre, edad, peso), (edad, nombre, peso), (nombre, peso, edad) o cualquier otra combinación.
- En las DTDs los elementos se definen a nivel de documento, por lo que no podemos tener en un mismo documento dos elementos con hijos que tengan el mismo nombre. Los esquemas XML permiten realizar declaraciones “sensibles al contexto”.
- Los esquemas XML soportan el uso de espacios de nombres y permiten formar claves a partir de varios atributos o nombres.

3.3.2 Estructura de un XSD

3.3.2.1 Vinculación del XSD desde el XML

Tenemos dos formas de vincular un documento XML con su xsd en función de si el documento tiene o no su propio espacio de nombres:


- Si el documento XML no tiene su propio espacio de nombres:
 - En el esquema (en el archivo xsd) usaremos el espacio de nombres XMLSchema (<http://www.w3.org/2001/XMLSchema>), que se referencia por convención con el prefijo **xs** (se podría usar cualquier otro), para definir los elementos y tipos de datos simples predefinidos (por ejemplo `xs:string`, para poder identificarlos como elementos del lenguaje del esquema XML, que sería como identificarlos como “palabras reservadas”).

```
<?xml version="1.0" ... ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> ← Esquema XSD
  ... definición del esquema ...
</xs:schema>
```

- En el documento XML tendremos que utilizar el espacio de nombres XMLSchema-instance (<http://www.w3.org/2001/XMLSchema-instance>), que se referencia con el

prefijo xsi, para poder usar en el elemento raíz el atributo **xsi:noNamespaceSchemaLocation**, que nos permite indicar en qué archivo está definido el esquema

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


 Ubicación del archivo xsd

Por ejemplo:

```
esquemaGuarderia.xsd
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="guarderia">
    ...
  </xs:schema>
Documento guarderia.xml
<?xml version="1.0"?>
<guarderia xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
    xsi:noNamespaceSchemaLocation="esquemaGuarderia.xsd">
<niño numeroExpediente="323">
  ...
</guarderia>
```


- Si queremos que los elementos definidos en el esquema pertenezcan a un cierto espacio de nombres, usaremos el atributo **targetNamespace** en la definición del elemento raíz para indicar el espacio de nombres que usaremos para validarlo.

```
<?xml version="1.0" ... ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="URIdelEspacioDeNombres">
  ... definición del esquema ...
</xs:schema>
```

 Esquema XSD

y en el documento crearemos y utilizaremos en los elementos un prefijo para el espacio de nombres.

```
<?xml version="1.0" ... ?>
<pref:elementoRaíz
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:pref="URIdelEspacioDeNombres"
  xsi:schemaLocation="URIdelEspacioDeNombres esquema.xsd">
  ...
</pref:elementoRaíz>
```

 Documento XML

3.3.2.2 Definición del XSD

El esquema XML consta de un elemento raíz (guardería) y un conjunto de subelementos (niños) que podrían contener a su vez otros subelementos (nombre, fechnacimiento, peso, altura) o un valor de un tipo de datos simple.

Los elementos que contienen subelementos o atributos se denominan tipos complejos, y los que tienen valores son tipos simples.

Los atributos de los elementos son siempre de algún tipo de datos simple.

Un esquema XML tendría el siguiente aspecto:



```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="guarderia">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="niño" minOccurs="1" maxOccurs="50"
type="tipoNiño"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tipoNiño">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="fechnacimiento" type="xs:date"/>
      <xs:element name="peso" type="xs:decimal"/>
      <xs:element name="altura" type="xs:unsignedByte"/>
      <xs:element name="vacunas" type="xs:boolean"/>
    </xs:sequence>
    <xs:attribute name="numeroExpediente"
type="xs:unsignedShort"/>
  </xs:complexType>
</xs:schema>

```

Annotations in the diagram:

- Elemento raíz schema y espacio de nombres del esquema XML (points to `<xs:schema>`)
- Permite definir tipos propios (points to `<xs:complexType>`)
- Permite especificar rangos (points to `minOccurs="1" maxOccurs="50"`)
- Permite usar tipos predefinidos (points to `type="xs:string"`)
- Los atributos tienen tipos simples (points to `<xs:attribute name="numeroExpediente" type="xs:unsignedShort"/>`)

3.3.3 Tipos de datos

Existen cuatro construcciones básicas en las que se basa un esquema:

- Declaración de elementos: Sirve para asociar el nombre de un elemento a un tipo predefinido, a un tipo simple o a un tipo complejo.
- Declaración de atributos: Sirve para asociar el nombre de un atributo a un tipo simple (el valor de los atributos sólo puede contener texto).
- Tipos simples: existen varios tipos simples predefinidos (string, integer, decimal, etc), pero es posible también definir nuevos tipos basados en los tipos simples predefinidos,

aplicándoles ciertas restricciones. En los tipos simples definidos su valor siempre es de tipo texto y no pueden contener otros elementos ni atributos.

- Tipos complejos: Sirven para definir nuevos tipos de datos cuyos valores constan de atributos y/o elementos (también pueden incluir texto en su contenido).

3.3.3.1 Declaración de elementos XML

Para declarar un elemento XML en el esquema se utiliza la etiqueta `<xs:element ...>`.

Podemos declarar los tipos de datos de manera global o local:

- En una declaración **global** los elementos son hijos del elemento raíz del esquema XML (`xs:schema`), por lo que pueden utilizarse en cualquier lugar del documento. Cuando declaramos el elemento, indicamos su tipo de datos por medio de los atributos `type` o `ref`:

```
<xs:element name="direccionPrincipal" type="tipoDireccion"/>
<xs:element name="direcciones">
  <xs:sequence>
    <xs:element ref="tipoDireccion" minOccurs="0"/>
  </xs:sequence>
</xs:element>
<xs:complexType name="tipoDireccion">
  <xs:sequence>
    <xs:element name="calle" type="xs:string"/>
    <xs:element name="numero" type="xs:unsignedShort"/>
    <xs:element name="localidad" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="provincia" type="xs:NMTOKEN"
    fixed="Lugo"/>
</xs:complexType>
```

- Una declaración **local** se define dentro de un elemento, por lo que aplica únicamente a ese elemento. Se dice que son tipos anónimos ya que no tienen un nombre.

```
<xs:element name="nombreElemento">
  <xs:simpleType>
    ...
  </xs:simpleType>
</xs:element>
```

Dependiendo del tipo de elemento que se trate podemos declararlo de alguna de las siguientes maneras:

- **Basados en un tipo predefinido:** sólo pueden tomar un valor que cumpla las restricciones del tipo en el que se basa. No se permiten ni (sub)elementos ni atributos.

```
<xs:element name="nombreElemento" type="tipoPredefinido" />
```

Por ejemplo:

```
<xs:element name="nombreCurso" type="xs:string"/>
<xs:element name="númeroAsistentes"
  type="xs:nonNegativeInteger"/>
```

- **Basados en un tipo simple definido por el usuario:** sólo puede tomar un valor que cumpla las restricciones impuestas por el tipo simple definido por el usuario. Tampoco permiten ni (sub)elementos ni atributos.
- **Basados en un tipo complejo:** Cuando un elemento contenga atributos u otros elementos, será necesario definir un tipo complejo. La declaración de un elemento basado en un tipo complejo es exactamente igual a la de un tipo simple, cambiando `<xs:simpleType>` por `<xs:complexType>`, pero además permite definir subelementos y atributos.

3.3.3.2 Declaración de atributos

En la declaración de un elemento `<xs:element ... >` podemos especificar atributos. Los más usados son:

- **name:** el nombre del elemento, el que se usará en los documentos XML encerrado entre los símbolos `<>`.
- **type:** el tipo de datos que asignamos al elemento
- **ref:** su valor es el nombre de otro elemento definido en el esquema (o en otro esquema especificando su espacio de nombres). Si se especifica este atributo, no se podrán indicar los atributos `default`, `fixed` o `type`, ni contener los elementos `<simpleType>` o `<complexType>`.

```
<xs:element ref="nombreDelElementoReferenciado"/>
```

- **maxOccurs:** número máximo de veces que el elemento puede aparecer dentro del elemento que a su vez lo contiene. Su valor debe ser de tipo `nonNegativeInteger` o bien la cadena `'unbounded'` para que no tenga límite. Por defecto es 1.

```
<xs:element name="línea" type="xs:string"
  maxOccurs="unbounded"/>
```

- **minOccurs:** número mínimo de veces que el elemento debe aparecer dentro del elemento que a su vez lo contiene. Su valor debe ser de tipo `nonNegativeInteger`. Si se especifica el valor 0 se convierte en un elemento opcional. Por defecto es 1.

```
<xs:element name="fechaDeNacimiento" type="xs:date" minOccurs="0"/>
```

- **default:** permite especificar un valor por defecto para el elemento.

```
<xs:element name="númeroDeCopias" type="xs:positiveInteger" default="1"/>
```

- **fixed:** su valor es automáticamente asignado al contenido del elemento, sin poder cambiarse en el documento. Este atributo no es compatible simultáneamente con el atributo `default`.
- **use:** para determinar si el elemento o atributo es obligatorio (`required`) u opcional (`optional`)

3.3.3.3 Tipos simples

Son tipos de datos que contienen un solo valor, que puede ser de un tipo predefinido (entero, cadena, fecha, etc) o bien de un tipo de datos más específico definido por el usuario.

Los tipos de datos simples predefinidos más empleados son los siguientes:

- **Texto**
 - `xs:string`: cadena de caracteres alfanumérica
 - `xs:normalizedString`: los caracteres de nueva línea, tabulador y retorno de carro son convertidos a espacios en blanco antes del procesamiento del esquema.
 - `xs:anyURI`: una URI
- **Números**
 - `xs:byte`, `xs:short`, `xs:integer` y `xs:long`: tipos numéricos de menor a mayor tamaño
 - `xs:unsignedByte`, `xs:unsignedInt`, `xs:unsignedShort`, `xs:unsignedLong`: números sin signo
 - `xs:positiveInteger`: valor estrictamente mayor que 0
 - `xs:negativeInteger`: valor estrictamente menor que 0
 - `xs:nonPositiveInteger`: valor menor o igual a 0
 - `xs:nonNegativeInteger`: valor mayor o igual a 0

- xs:decimal, xs:float, xs:double: tipos decimales. Como separador decimal se utiliza el punto.
- Fecha/hora
 - xs:date: fecha con formato “AAAA-MM-DD”
 - xs:time: hora con formato “hh:mm:ss”
 - xs:dateTime: fecha y hora, con formato ISO 8601 (separados por una T: 2010-04-14T00:19:25)
 - xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:gMonthDay: día, mes, año, mes del año y día del mes
- Lógicos
 - xs:boolean: acepta los valores true o 1 y false o 0

Además de los tipos de datos simples predefinidos se pueden definir tipos de datos simples propios. Existen tres mecanismos para crear tipos simples personalizados: restricciones, listas y uniones.

3.3.3.3.1 Restricciones

La sintaxis para definir un tipo simple con restricciones es la siguiente:

```
<xs:simpleType name="nombreTipoSimple">
  <xs:restriction base="tipoBase">
    ...
  </xs:restriction>
</xs:simpleType>
```

Las restricciones que podemos aplicar son las siguientes:

RESTRICCIÓN	DESCRIPCIÓN
length	Número exacto de caracteres de la cadena o número mínimo de elementos de la lista permitidos. Debe ser igual o mayor que cero
minLength	Número mínimo de caracteres de la cadena o número mínimo de elementos de la lista permitidos. Debe ser igual o mayor que cero
maxLength	Número máximo de caracteres de la cadena o número máximo de elementos de la lista permitidos. Debe ser igual o mayor que cero
pattern	El contenido debe encajar con una expresión regular
enumeration	Define una lista de valores válidos

whiteSpace	Permite indicar cómo queremos que el validador trate los espacios en blanco, saltos de línea, tabuladores, espacios y retornos de carro. Podemos darle tres valores: <u>preserve</u> (los conserva), <u>replace</u> (sustituye grupos de espacios en blanco por espacios en blanco simples) o <u>collapse</u> (sustituye grupos de espacios en blanco, tabuladores y saltos de línea por espacios en blanco simples, y elimina espacios iniciales y finales)
minInclusive	Límite inferior incluido del valor numérico (debe ser mayor o igual al valor indicado)
minExclusive	Límite inferior del valor numérico (debe ser mayor al valor indicado)
maxInclusive	Límite superior incluido del valor numérico (debe ser menor o igual al valor indicado)
maxExclusive	Límite superior del valor numérico (debe ser menor al valor indicado)
totalDigits	Máximo número de dígitos en los tipos numéricos. Debe ser mayor que 0
fractionDigits	Máximo número de decimales en los tipos numéricos. Debe ser mayor o igual que 0

Dependiendo del tipo base se podrán aplicar unas restricciones u otras:

- **Sobre cadenas:** `length`, `minLength`, `maxLength`, `pattern`, `enumeration` y `whitespace`

```
<xs:simpleType name="nombreCorto">
  <xs:restriction base="xs:string">
    <xs:maxLength value="15"/>
  </xs:restriction>
</xs:simpleType>
```

- **Sobre números:** `totalDigits`, `pattern`, `whitespace`, `enumeration`, `maxInclusive`, `maxExclusive`, `minInclusive` y `minExclusive`.

```
<xs:element name="nota">
  <xs:simpleType>
    <xs:restriction base="xs:byte">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Cuando utilizamos la restricción “pattern”, usaremos expresiones regulares para determinar qué caracteres podemos usar, cuantos o en qué lugar irá cada caracter:

Patrón	Descripción	Ejemplo	
^	Coincidencia al principio de la cadena	^Esto	Esto es una cadena
\$	Coincidencia al final de la cadena	final\$	Coincidencia al final
*	El caracter que lo precede puede aparecer 0, 1 o más veces	ma*	m, ma, maa, etc
+	El carácter que lo precede debe aparecer 1 o más veces	ma+	ma, maa, etc
?	El carácter que lo precede puede aparecer 0 o 1 vez	ma?	ma o m

Patrón	Descripción	Ejemplo	
[...]	Cualquier caracter dentro de los paréntesis	a[px]e	ape o axe
[-]	Indica un rango de caracteres	[c-k]a	ca,da,ea,fa,ga,ha, ia o ka
[^...]	Cualquier caracter menos los que están entre paréntesis	a[^px]e	ale, a1e, etc
[^ -]	Cualquier caracter menos los que estén en ese rango	[^c-k]a	ba, la, etc
{n}	El carácter que lo precede debe aparecer exactamente n veces	ma{2}	maa
{n,}	El carácter que lo precede debe aparecer n o más veces	bu{2,}	maa, maaa, etc
{n,m}	El carácter que lo precede debe aparecer como mínimo n y como máximo m veces	[0-9]{2,4}	32, 4444
(...)	Permite agrupar caracteres para aplicarles los patrones anteriores	(ca){2}túa	cacatúa
	Permite separar alternativas válidas	M[0-9] MP	MP o M9
\d	Cualquier dígito (es equivalente a [0-9])	\d{2}	23
\D	Cualquier no dígito	\D{2}M	L_M o ABM
\w	Cualquier letra o dígito	ala\w	ala4 o alag
\W	Cualquier caracter diferente de letra o dígito	M\Wa	M-a
.	Cualquier caracter salvo salto de línea		
\n	Salto de línea		
\s	Caracter en blanco, tabulador o salto de línea		

3.3.3.3.2 Listas

Una lista es una secuencia de valores de un determinado tipo, separados por espacios. La sintaxis para crear una lista es la siguiente:

```
<xs:simpleType name="nombreTipoLista">
  <xs:list itemType="tipoDatos"/>
</xs:simpleType>
```

En el tipo de datos podríamos usar cualquier tipo de datos, predefinido o definido por el usuario:

```
<xs:simpleType name="nota">
  <xs:restriction base="xs:byte">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="listaNotas">
  <xs:list itemType="nota"/>
</xs:simpleType>
```

Para asignarle valores al elemento en el documento XML:

```
<listaNotas>2 5 7 5 9</listaNotas>
```

A los tipos lista se le pueden aplicar las siguientes propiedades:

- `length`: longitud de la lista
- `minLength`: longitud mínima
- `maxLength`: longitud máxima
- `enumeration`: enumeración de posibles valores

Si queremos una lista de 3 notas, para guardar las notas de cada evaluación de un alumno, podemos crear un nuevo tipo de datos simple restringiendo el tipo “listaNotas” para que solo tenga 3 elementos:

```
<xsd:simpleType name="listaNotas">
  <xsd:list itemType="nota"/>
</xsd:simpleType>
<xsd:simpleType name="notasCurso">
  <xsd:restriction base="listaNotas">
    <xsd:length value="3"/>
  </xsd:restriction>
</xsd:simpleType>
```

Aunque utilizar listas puede facilitar las cosas en ciertos aspectos, no se recomienda su utilización en XML, ya que pierden las ventajas del lenguaje. En su lugar se deberían añadir más marcas para definir el elemento:

```
<notasCurso>
  <nota>5</nota>
  <nota>9</nota>
  <nota>4</nota>
</notasCurso>
```

3.3.3.3.3 Union

Las uniones permiten crear elementos que pueden tomar valores de distintos tipos de datos. Su sintaxis es la siguiente:

```
<xs:simpleType name="nombreTipoUnion">
  <xs:union>
    <xs:simpleType> ...</xs:simpleType>
    <xs:simpleType> ...</xs:simpleType>
    ...
  </xs:union>
</xs:simpleType>
```

```
</xs:union>
</xs:simpleType>
```

Por ejemplo, para definir un tipo moneda, que pueda tomar valores integer o float:

```
<xs:element name="moneda">
  <xs:simpleType>
    <xs:union memberTypes="integer float" />
  </xs:simpleType>
</xs:element>
```

Las posibles notas de un alumno, que pueden tomar un valor entero del 1 al 10, o un NP (no presentado):

```
<xs:simpleType name="tipoNota">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:maxInclusive value="1"/>
        <xs:minInclusive value="10"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="NP"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

3.3.3.4 Tipos complejos

Los tipos complejos permiten definir elementos que contienen atributos y/o elementos anidados, y se definen por medio del tipo **complexType**.

Los elementos se declaran con el elemento **element** y los atributos con el elemento **attribute**:

```
<xs:complexType name="tipoDireccion">
  <xs:sequence>
    <xs:element name="calle" type="xsd:string"/>
    <xs:element name="numero" type="xsd:unsignedShort"/>
    <xs:element name="localidad" type="xsd:string"/>
  </xs:sequence>
  <xs:attribute name="provincia" type="xsd:NMTOKEN"
    fixed="Lugo"/>
</xs:complexType>
```

Varios elementos de un XML pueden tener el mismo tipo complejo (podríamos tener un XML con **direccionCasa** y **direccionTrabajo** y podríamos asignar a ambas el tipo **direccion** del ejemplo).

Un elemento complejo puede contener atributos, subelementos e incluso contenido propio en forma de texto.

3.3.3.4.1 Atributos

Los atributos de un tipo complejo se definen dentro del tipo complejo. Dado que los atributos solo pueden tener valores de tipo simple, su definición se puede hacer asignándole un tipo de datos simple predefinido:

```
<xs:attribute name="nombreAtributo" type="tipoDelAtributo" [default="valorPorDefecto" |
fixed="valorFijo"] [use="required"] />
```

o asignándole un tipo de datos definido por el usuario:

```
<xs:attribute name="nombreAtributo" [default="valorPorDefecto" | fixed="valorFijo"]
[use="required"] >
  <xs:simpleType>
    <xs:restriction type="tipoSimple">
      ...
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

```
<xs:element name="distancia">
  <xs:complexType>
    <xs:attribute name="unidad" type="xsd:string"/>
    <xs:attribute name="valor" type="xsd:decimal"/>
  </xs:complexType>
</xs:element>
---
<distancia unidad="Km" valor="444,23"/>
```

3.3.3.4.2 Subelementos

Los tipos complejos pueden tener subelementos. Si alguno de ellos puede aparecer varias veces o aparecer al menos un número determinado de veces, podemos configurarlo por medio de los atributos **minOccurs** y **maxOccurs**. El valor de ambos por defecto es 1 (es obligatorio y solo puede aparecer una vez), pero podemos modificarlos para indicar el número de apariciones que queramos (de forma similar a los operadores + y * de la DTD).

```
<xs:element name="observaciones" type="xs:string"
minOccurs="0" maxOccurs="4"/>
```

Cuando un tipo complejo tiene varios subelementos, éstos pueden agregarse de distintas formas.

- **sequence:** Una secuencia ordenada de elementos que deben aparecer siempre en el mismo orden.
- **choice:** Un grupo de elementos de los cuales solo puede aparecer uno:

```
<xs:complexType name="persona">
  <xs:choice>
    <xs:element name="apodo" type="xs:string"/>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido1" type="xs:string"/>
      <xs:element name="apellido2" type="xs:string"
minOccurs="0"/>
    </xs:sequence>
  </choice>
</xs:complexType>
```

- **all**: Una secuencia desordenada de elementos que pueden aparecer en cualquier orden. Si un elemento complejo tiene un grupo de subelementos de tipo “all”, éste debe aparecer como hijo único del elemento principal (no puede tener delante un sequence y después el all, por ejemplo, o aparece un sequence y dentro del sequence el all).

```
<xs:element name="persona">
  <xs:complexType>
    <xs:all>
      <xs:element name="nif" type="xs:string"/>
      <xs:element name="nombre" type="nombreCompleto"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Serían válidos:

```
<persona>
  <nombreCompleto>
    <nombre>Mar</nombre>

  <apellido1>Blanco</apellid
ol>
  </nombreCompleto>
  <nif>44444444M</nif>
</persona>
```

```
<persona>
  <nif>44444444M</nif>
  <nombreCompleto>
    <nombre>Mar</nombre>

  <apellido1>Blanco</apellid
ol>
  </nombreCompleto>
</persona>
```

- **any**: La estructura <xs:any> permite que el esquema acepte en ese lugar cualquier elemento y su contenido.

Esta estructura tiene atributos que permiten configurar, por ejemplo, que acepte únicamente elementos de ciertos espacios de nombres, o bien que la validación del contenido sea más o menos rígida.

Por ejemplo, si quisiésemos definir un elemento que contenga un código HTML, sea cual sea, podemos definirlo de forma que acepte cualquier elemento del espacio de nombres

“<http://www.w3.org/1999/xhtml>”:

```
<?xml version = "1.0" encoding = "utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```

<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido1"
type="xs:string"/>
      <xs:element name="apellido2" minOccurs="0"
type="xs:string"/>
      <xs:element name="comentario"
type="tipoComentario"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="tipoComentario">
  <xs:sequence>
    <xs:any namespace="http://www.w3.org/1999/xhtml"
minOccurs="1" maxOccurs="unbounded"
processContents="skip" />
  </xs:sequence>
</xs:complexType>
</xs:schema>
---
<?xml version="1.0" encoding="UTF-8"?>
<persona xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xsi:noNamespaceSchemaLocation="a.xsd">
  <nombre>Nerea</nombre>
  <apellido1>Gómez</apellido1>
  <apellido2>Rodríguez</apellido2>
  <comentario>
    <html:p>
      <html:b>Repetidor</html:b> con un módulo pendiente.
    </html:p>
  </comentario>
</persona>

```

Podemos utilizar también “anyAttribute” para indicar que el elemento puede contener cualquier atributo:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persona">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellido1" type="xs:string"/>
        <xs:element name="apellido2" minOccurs="0"
type="xs:string"/>
      </xs:sequence>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```



```

    </xs:complexType>
  </xs:element>
</xs:schema>
---
<?xml version="1.0" encoding="UTF-8"?>
<persona xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="ejemploAnyAttribute.xsd"
    sexo="femenino">
  <nombre>Nerea</nombre>
  <apellido1>Gómez</apellido1>
  <apellido2>Rodríguez</apellido2>
</persona>

```

3.3.3.4.3 Elementos con contenido simple

Son elementos que tienen texto y atributos. En el interior del contenido indicaremos si queremos extender o restringir el tipo base. Para ello utilizaremos las etiquetas `<xs:extension base="tipoBaseExtendido">` o `<xs:restriction base="tipoBaseRestringido">`.

```

<xs:complexType name="distanciaConUnidades">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="unidad" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:element name="distancia" type="distanciaConUnidades"/>
---
<distancia unidad="Km">32</distancia>

```

3.3.3.4.4 Elementos con contenido complejo

El elemento puede contener texto, subelementos y atributos. Este tipo de elementos se denominan mixtos y deben declararse con el atributo: **mixed="true"**.

```

<xs:element name="carta">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="hijo" type="xs:string"/>
      <xs:element name="numero" type="xs:unsignedShort"/>
      <xs:element name="fecha" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
---

```

```
<carta>
  Estimado Sr.<nombre>Pedro Barro</nombre>.
  La presente carta es para notificarle que su hijo
  <hijo>Xoel</hijo> tiene un total de <numero>3</numero> faltas
  desde el día <fecha>13/4/30</fecha>.
</carta>
```

3.3.4 Otros elementos del lenguaje

3.3.4.1 Grupos

Los grupos pueden emplearse para definir conjuntos relacionados de elementos (group), o conjuntos relacionados de atributos (attributeGroup). Son útiles para hacer referencia a ellos desde otros elementos.

Un grupo de atributos puede contener otros grupos de atributos.

Un ejemplo de grupo de elementos sería:

```
<xs:group name="nombreCompleto">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido1" type="xs:string"/>
    <xs:element name="apellido2" type="xs:string"
minOccurs="0"/>
  </xs:sequence>
</xs:group>
```

Posteriormente podríamos hacer referencia a él de la siguiente manera:

```
<xs:element name="alumno" type="tipoAlumno"/>
<xs:complexType name="tipoAlumno">
  <xs:all>
    <xs:group ref="nombreCompleto" />
    <xs:element name="numeroExpediente" type="xs:string"/>
    <xs:element name="curso" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

Un ejemplo completo, con un grupo de atributos:

```
XSD:
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:attributeGroup name="caracteristicas">
  <xs:attribute name="duracion" type="xs:unsignedShort"/>
  <xs:attribute name="grado">
    <xs:simpleType>
      <xs:restriction base="xs:string">
```

```

        <xs:enumeration value="básico"/>
        <xs:enumeration value="medio"/>
        <xs:enumeration value="superior"/>
    </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:attributeGroup>

<xs:element name="ciclo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="familia" type="xs:string"/>
        </xs:sequence>
        <xs:attributeGroup ref="caracteristicas"/>
    </xs:complexType>
</xs:element>
XML:
<?xml version="1.0" encoding="UTF-8"?>
<ciclo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:noNamespaceSchemaLocation="ejemploAttributeGroup.xsd"
        duracion="2000"
        grado="superior">
    <nombre>Desenvolvimiento de aplicaciones web</nombre>
    <familia>IFC</familia>
</ciclo>

```

Utilizando un grupo de atributos de este modo podemos aumentar la legibilidad de los esquemas y facilitar la actualización de los esquemas ya que un grupo de atributos se define en un único lugar y puede ser referenciado en múltiples definiciones y declaraciones.

3.3.4.2 Anotaciones

Se usan para documentar el esquema XML y no tienen efecto en su validación siempre que estén bien formadas. No pueden ir en cualquier parte del documento, solo antes o después de un componente global. Existen dos tipos de anotaciones:

- Las dirigidas a personas (documentation), que pueden tener los atributos source (URL con información adicional) o lang (idioma de la documentación)

```

<xs:documentation source=http://www.web.com/documentacion.txt
    xml:lang="ES"/>
    ... texto explicativo de la documentación ...
</xs:documentation>

```

- Las dirigidas a programas (appinfo). Solamente tienen el atributo source:

```
<xs:appinfo source="http://www.web.com/docum.xml"/>
```

3.3.4.3 Valores nulos

Permiten representar valores nulos (no confundir con el valor de la cadena vacía). Para definir el atributo como nullable en el esquema, usaremos el atributo “nilable” asignándole el valor “true”. En el documento XML usaríamos el atributo `xsi:nil="true"` para asignar el valor null al elemento.

```
-- Esquema
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="guarderia">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="niño" minOccurs="1" maxOccurs="50"
type="tipoNiño"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tipoNiño">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="edad" type="xs:unsignedByte"
nilable="true"/>

```

3.3.4.4 Elementos únicos

Se especifican por medio del elemento “unique” (que permitiría asignar valores nulos, pero no podría haber dos elementos con valor nulo). El elemento unique consta de dos subelementos:

- Selector: selecciona un subconjunto del elemento por medio de xpath

- Field: dentro del subconjunto seleccionados indicamos también con xpath cual será el elemento o elementos que no se pueden repetir

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="agenda">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="contacto" type="tipoContacto"
maxOccurs="100"/>
      </xs:sequence>
    </xs:complexType>
    <xs:unique name="nombreUnico">
      <xs:selector xpath="contacto"/>
      <xs:field xpath="nombre"/>
    </xs:unique>
  </xs:element>

  <xs:complexType name="tipoContacto">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="telefono" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
-- Documento XML:
<?xml version='1.0' encoding='UTF-8' ?>
<agenda xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ejemploUnique.xsd">
  <contacto>
    <nombre>Iria</nombre>
    <telefono>988988988</telefono>
  </contacto>

  <contacto>
    <nombre>Xoel</nombre>
    <telefono>988988999</telefono>
  </contacto>
</agenda>
```

3.3.4.5 Claves

Se definen con el elemento “key” y tienen la misma estructura que unique, con la diferencia de que no admiten valores nulos. Para definir una clave foránea podemos usar el elemento keyref, que nos permitirá hacer referencia a la clave a la que apunta.

XSD

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

```

<xs:element name="empresa">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="empleado" type="tipoEmpleado"
maxOccurs="200"/>
      <xs:element name="departamento"
type="tipoDepartamento"
maxOccurs="8"/>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="depUnico">
    <xs:selector xpath="departamento"/>
    <xs:field xpath="@codigo"/>
  </xs:key>
  <xs:keyref name="departamento" refer="depUnico">
    <xs:selector xpath="empleado"/>
    <xs:field xpath="departamento"/>
  </xs:keyref>
</xs:element>

<xs:complexType name="tipoEmpleado">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="departamento" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="tipoDepartamento">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="codigo" type="xs:string"
use="required"/>
</xs:complexType>
</xs:schema>

```

XML

```

<?xml version='1.0' encoding='UTF-8' ?>
<empresa xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:noNamespaceSchemaLocation="ejemploKey.xsd">
  <empleado>
    <nombre>Iria</nombre>
    <departamento>IFC</departamento>
  </empleado>
  <empleado>
    <nombre>Xoel</nombre>
    <departamento>CON</departamento>
  </empleado>
  <departamento codigo="IFC">
    <nombre>Informática</nombre>

```

```

</departamento>
<departamento codigo="CON">
  <nombre>Contabilidad</nombre>
</departamento>
</empresa>

```

En caso de indicar un valor inexistente para el código del departamento, daría un mensaje de error.

3.3.5 Reutilización de esquemas

Existen tres formas de reutilizar declaraciones realizadas en un esquema en otros esquemas que hagan uso de la misma información:

- Incluir la definición de un esquema en otro (**xs:include**): en este caso los elementos deben estar en el mismo espacio de nombres (es como si estuviesen todos en el mismo fichero). Esta opción permite reutilizar las definiciones de los elementos del esquema tal cual están creadas.
- Redefinir los tipos complejos, simples, grupos o grupos de atributos existentes en un xsd (**xs:redefine**). También requiere que los componentes externos no tengan espacio de nombres o que estén en el mismo espacio de nombres que el esquema que los va a redefinir.
- Importar declaraciones realizadas en un esquema en otro esquema diferente (**xs:import**). En este caso las declaraciones pueden pertenecer a espacios de nombres diferentes.

El siguiente ejemplo muestra como incluir, redefinir o importar la definición de un tipo de dato “nombrePropio” declarado en un fichero “definicionesComunes.xsd” que estará formado por una letra mayúscula seguido de letras minúsculas indefinidas:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="tipoNombrePropio">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-ZÑÁÉÍÓÚ][a-zñáéíóú]*/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>

```

- Include (fichero “ejemploInclude.xsd”)

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="definicionesComunes.xsd"/>
  <xs:complexType name="tipoNombreCompleto">
    <xs:sequence>

```

```

        <xs:element name="nombre"
type="tipoNombrePropio"/>
        <xs:element name="apellido1" type="tipoNombrePropio"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="persona" type="tipoNombreCompleto"/>
</xs:schema>

```

- Redefine (fichero “ejemploRedefine.xsd”):

```

<?xml version='1.0' encoding='UTF-8' ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:redefine schemaLocation="ejemploInclude.xsd">
    <xs:complexType name="tipoNombreCompleto">
      <xs:complexContent>
        <xs:extension base="tipoNombreCompleto">
          <xs:sequence>
            <xs:element name="apellido2"
type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:redefine>

  <xs:element name="persona">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombreC"
type="tipoNombreCompleto"/>
        <xs:element name="apodo" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

- import

```

<xs:schema ...>
  <xs:import namespace="A"
schemaLocation="ejemploInclude.xsd"/>
  <xs:import namespace="P"
schemaLocation="ejemploRedefine.xsd"/>
  . . .
</xs:schema>

```


ANEXO I: Materiales

I. Textos de apoyo o de referencia

- Wikipedia. <http://www.wikipedia.org>
- W3schools XML: <https://www.w3schools.com/xml/default.asp>

II. Recursos didácticos

- Apuntes en el aula virtual.
- Ordenador personal, con navegador web y conexión a internet.
- Software para elaboración de documentos de texto.