

# Índice

---

<b>1. A2. Validación empregando XML Schema.</b>	<b>2</b>
1.1 Introducción.....	2
1.2 Actividade.....	2
Qué é un esquema XML.....	2
Diferenzas entre DTDs e esquemas XML.....	3
Vinculación dun esquema a un documento XML.....	5
O documento XML non ten o seu propio espazo de nomes.....	5
O documento XML ten o seu propio espazo de nomes.....	5
Construcción de esquemas XML.....	7
Tipos simples.....	7
Atributos.....	8
Tipos complexos.....	9
Modos de declarar os elementos.....	12
Multiplicidade do elemento.....	13
Creación dos seus propios tipos.....	13
Incluir calquera elemento ou atributo.....	19
Os grupos.....	21
Anotacións.....	22
Valor nulo.....	22
Claves e unicidade.....	23
Creación de esquemas a partir doutros esquemas.....	26
Reaprovechar definiciones usando includes.....	26
Redefinición de esquemas.....	27
Importación de esquemas.....	28

---

# 1. A2. Validación empregando XML Schema.

---

## 1.1 Introducción

Esta actividade ten como obxectivos:

- Comprender a necesidade do emprego dun método de validación que permita determinar se un documento XML responde a un formato específico.
- Usar XML Schema na elaboración e na validación de documentos XML.
- Utilizar ferramentas específicas para a validación de documentos XML empregando XML Schema.

## 1.2 Actividade

### Qué é un esquema XML

A descrición de documentos XML mediante o uso de DTDs proporciona un mecanismo para validar documentos XML baseado principalmente na estrutura deste: os elementos e atributos de que consta. Aínda que é un xeito moi útil de especificar cómo se debe organizar un determinado documento XML ou XHTML, as veces, una descrición DTD non proporciona todas as funcionalidades que debería.

O primeiro problema é que a sintaxe dun arquivo DTD é parecido a XML, pero non está ben estruturado segundo as normas da linguaxe XML. Ademais, cunha DTD non podemos dicir moito en canto aos tipos de datos, simplemente se indica que o contido é un elemento PCDATA. Isto significa que, *inda que podamos validar a estrutura dun documento XML empregando unha DTD, non podemos facer nada por validar o seu contido*. Por exemplo, non podemos comprobar se un elemento que debería conter un valor numérico, ten realmente un número no seu interior, nin se un elemento que debería conter unha data ten a información correctamente formatada no seu interior.

Imos ver isto con un exemplo. O seguinte documento XML almacena información dos nenos dunha gardería. Dos nenos gardamos datos de distintos tipos: `Nome` é de tipo carácter, `dataNacemento` é de tipo data, `peso` é de tipo decimal (almacena a información en Kg), `altura` é un número enteiro (almacena a información en cm) e `vacunas` que pode tomar os valores verdadeiro ou falso.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo UD7_1 -->
<!DOCTYPE garderia SYSTEM "garderia.dtd">
<garderia>
  <nenos numeroExpediente="2344">
    <nome>Noa Blanco Coello</nome>
    <dataNacemento>2010-10-12</dataNacemento>
    <peso>13.5</peso>
    <altura>78</altura>
    <vacunas>true</vacunas>
  </nenos>
```

```
</garderia>
```

O seguinte exemplo amosa unha DTD que valida o documento XML anterior:

```
<!ELEMENT garderia (neno*)>
<!ELEMENT neno (nome, dataNacemento, peso, altura, vacunas)>
<!-- ATTLIST neno numeroExpedente CDATA #REQUIRED -->
<!ELEMENT nome (#PCDATA)>
<!ELEMENT dataNacemento (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
<!ELEMENT altura (#PCDATA)>
<!ELEMENT vacunas (#PCDATA)>
```

Empregando este arquivo DTD, o noso exemplo de documento XML sería válido, inda que únicamente se validaría polo DTD a súa estrutura. O problema é que o seguinte documento XML tamén sería válido segundo as definicións da DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo UD7_1 -->
<!DOCTYPE garderia SYSTEM "garderia.dtd">
<garderia>
  <neno numeroExpedente="popeye">
    <nome>cousas</nome>
    <dataNacemento>1.1.1</dataNacemento>
    <peso>non o sei</peso>
    <altura>falso</altura>
    <vacunas>no</vacunas>
  </neno>
</garderia>
```

Aínda que está claro que non ten sentido ningún, unha DTD non nos axuda para asegurar, por exemplo, que o elemento `dataNacemento` é de tipo data, ou que o elemento `altura` debe conter un valor numérico.

Para solucionar isto, implementáronse os esquemas XML, que permiten especificar os tipos de datos dun documento XML. Unha das vantaxes principais disto é que é máis sinxelo relacionar os datos entre documentos XML e compoñentes desenvolvidos en linguaxes de programación que tamén usen regras estritas á hora de definir os tipos de datos.

Unha instancia concreta de esquema XML coñécese como *Definición de esquema XML*, do mesmo xeito que un arquivo DTD coñécese como *Definición de tipo de documento*. A extensión dun arquivo para definición dun esquema XML é `xsd`.

Ademais, un esquema XML é en si mesmo un documento XML → ¡Usamos XML para definir XML!

### Diferenzas entre DTDs e esquemas XML

En definitiva, as principais diferenzas existentes entre un documento DTD e un esquema XML son:

- *A contribución da sintaxe XML* \_ As DTD usan unha linguaxe propia que é necesario comprender. Os esquemas especifican as gramáticas de documentos XML en XML. O documento resultante é moito máis fácil de ler e, polo tanto, de entender.
- *A contribución da tipaxe* \_ Empregando esquemas podemos especificar tipos de datos (por exemplo: número enteiro, data, ...) ou definir tipos propios.

- *Secuencias desordenadas* \_ As DTDs non permiten especificar secuencias que poidan aparecer en calquera orde. Por exemplo:

(nome, idade, peso) ou (idade, nome, peso) ou (nome, peso, idade)  
ou calqueira outra combinación de elementos.

- *Declaracións sensibles ao contexto* \_ Nos DTDs todos os elementos se definen a nivel de documento, co cal non é posible que teñamos contido co mesmo nome cunha estrutura diferente segundo o contexto no que se atope. Cos esquemas XML podemos facer declaracións sensibles ao contexto.
- *Espazos de nomes* \_ As DTDs non soportan os espazos de nomes.
- *Referencias cruzadas* \_ Con DTDs non é posible formar claves a partir de varios atributos ou nomes.

A continuación vemos un exemplo de esquema XML:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="garderia">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="neno" minOccurs="1" maxOccurs="50"
          type="tipoNeno"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tipoNeno">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="dataNacemento" 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="numeroExpedente" type="xs:unsignedShort"/>
  </xs:complexType>
</xs:schema>
```

Elemento raíz schema e espazo de nomes do esquema XML

Permite especificar rangos

Permite definir tipos propios

Permite especificar tipos

O esquema XML consta dun elemento raíz schema e un conxunto de subelementos. Cada un dos elementos do esquema ten o prefixo xs: que está asociado co espazo de nomes do esquema XML a través da declaración `xmlns:xsd="http://www.w3.org/2001/XMLSchema"` que aparece no elemento schema. O prefixo xs: é usado por convención para denotar o espazo de nomes do esquema XML, inda que se pode usar calquera outro prefixo. O mesmo prefixo, e polo tanto a mesma asociación, tamén aparece nos nomes dos tipos simples predefinidos, por exemplo `xs:string`. O propósito desta asociación é identificar os elementos e tipos simples como pertencentes ao vocabulario da linguaxe Esquema XML e non ao vocabulario do autor do esquema.



No Texto de apoio 1 da UD6 temos o método para validar un documento XML empregando DTDs, que é o mesmo que deberemos seguir para comprobar que un documento XML responde a un XML Schemas dado.

## Vinculación dun esquema a un documento XML


Cando se valida un documento XML que ten vinculado un esquema XML, en realidade estanse a realiza dúas validacións, xa que o documento xml validase co esquema xml, e este a súa vez validarase nun hipotético XMLSchema.xsd, xa que o esquema ten unha sintaxe definida no seu propio esquema de esquemas e tena que cumprir.

Hai dous tipos de vínculos dependendo de se o noso documento XML ten ou non o seu propio espazo de nomes.

### O documento XML non ten o seu propio espazo de nomes


Neste caso, simplemente indicamos no esquema un único espazo de nomes: o dos seus propios elementos. Os elementos e tipos de datos que usaremos para construír o esquema están definidos no espazo de nomes `http://www.w3.org/2001/XMLSchema` (ao que nos referiremos co prefixo `xs:`)

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

 Esquema XSD

e como o documento non ten o seu propio espazo de nomes, enlazárase ao seu esquema a través do atributo `xsi:noNamespaceSchemaLocation`. Para ter acceso a este atributo, creamos anteriormente o prefixo `xsi` que fai referencia ao espazo de nomes especificado na etiqueta raíz.

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

 Documento XML

Por exemplo:

**esquemaGarderia.xsd**

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

**garderia.xml**

```
<?xml version="1.0"?>
<garderia xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="esquemaGarderia.xsd">
  <nenho numeroExpediente="323">
    ...
  </nenho>
</garderia>
```

### O documento XML ten o seu propio espazo de nomes

Se queremos que os elementos definidos no esquema pertenczan a un determinado espazo de nomes, que imos usar nos documentos validados por dito esquema, teremos que usar o atributo `targetNamespace` na definición do elemento raíz.

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

 Esquema XSD

E no documento XML crearemos e usaremos nos elementos un prefixo para o espazo de nomes:

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

 Documento XML

Se engadimos o atributo `elementFormDefault="qualified"` todos os elementos e atributos declarados globalmente no esquema deberán aparecer cualificados cun espazo de nomes que usemos no esquema.

Por exemplo:

**esquemaGarderia.xsd**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=http://www.concello.es/garderias
    elementFormDefault="qualified">
<xs:element . . ."/>
...
</xs:schema>
```

**garderia.xml**

```
<?xml version="1.0"?>
<xg:garderia xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xg="http://www.concello.es/garderias"
    xsi:schemaLocation="http://www.concello.es/garderias esquemaGarderia.xsd">
    <nenos numeroExpediente="323"> ...
    ...
</xg:garderia>
```

Así, no esquema XML:

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` → Indica que os elementos e tipos de datos usados no esquema proveñen do espazo de nomes: `http://www.w3.org/2001/XMLSchema`
- `targetNamespace=http://www.concello.es/garderias` → Indica que os elementos definidos por este esquema (garderia, neno, nome, ...) proveñen do espazo de nomes: <http://www.concello.es/garderias>
- `elementFormDefault="qualified"` → Indica que todos os elementos e atributos declarados globalmente no esquema deberán aparecer cualificados cun espazo de nomes que usemos no esquema.



Na tarefa 1 vincularemos un esquema XML para validar o documento XML que contén a información da gardería.

## Construcción de esquemas XML

Un documento XML consta dun elemento principal (no nos exemplo *garderia*), e unha serie de subelementos (*nenos*). Estes subelementos, poderían conter a súa vez outros subelementos (*nome*, *dataNacemento*, *peso*, *altura*), e así sucesivamente ata que o elemento contén un valor en lugar de máis subelementos.

Os elementos que conteñen subelemento ou teñen atributos son denominados *tipos complexos*, mentras que os elementos que conteñen valores (números, datas, cadeas, etc.) pero non constan de subelementos denomínanse *tipos simples*.

Algúns elementos teñen atributos, e estes sempre terán un tipo simple.

### Tipos simples

Como xa dixemos, un tipo simple é aquel que unicamente ten como contido texto. Non pode ter ningún subelemento, nin atributos. Defínese da forma:

```
<xs:element name="xxx" type="yyy"/>
```

Por exemplo, o elemento *nome* do noso exemplo da *gardería* defínese:

```
<xs:element name="nome" type="xs:string"/>
```

Os elementos simples poden ter un valor por defecto:

```
<xs:element name="cidade" type="xsd:string" default="Vigo"/>
```

Ademais, pódese asignar un valor previamente fixado (co cal non se poderá cambiar):

```
<xs:element name="color" type="xsd:string" fixed="blanco"/>
```

### Tipos simples predefinidos

Os tipos simples predefinidos ou *built-in* son os que están definidos na especificación. A seguinte táboa amosa os tipos predefinidos máis empregados:

Categoría	Tipo	Exemplo	Observacións
Texto	xs:string	Calquera cadea de caracteres alfanumérica	
	xs:normalizedString	Calquera cadea de caracteres alfanumérica	Os caracteres de nova liña, tabulador e retorno de carro son convertidos a espazos en branco antes do procesamento do esquema.
	Xs:token	Calquera cadea de caracteres alfanumérica	Os espazos en branco adxacentes son comprimidos a un único espazo en branco, e os espazos en branco de diante e detrás son eliminados
	xs:anyURI	http://www.example.com	
Números	xs:byte	125, -2	
	xs:unsignedByte	0, 45	
	xs:short	-1, 12534	
	xs:unsignedShort	0, 12534	
	xs:integer	-1, 126789675	
	xs:unsignedInt	0, 126789675	
	xs:positiveInteger	1, 12345	Maior que 0 (>0)

Categoría	Tipo	Exemplo	Observacións
	xs:nonPositiveInteger	0, -1, -12345	Menor igual que 0 ( $\leq 0$ )
	xs:negativeInteger	-1, -126789	Menor que 0 ( $< 0$ )
	xs:nonNegativeInteger	0, 1, 126789	Maior igual que 0 ( $\geq 0$ )
	xs:long	-1, 12678967541111	
	xs:unsignedLong	0, 12678967541111	
	xs:decimal	-1.23, 0, 123.4	Separador decimal é o punto
	xs:float	-1E4, 12.78E-2	equivalente a punto flotante de precisión simple de 32-bit,
	xs:double	-1E4, 12.78E-2	equivalente a punto flotante de precisión dobre de 64-bit,
Data/Hora	xs:dateTime	2010-12-31T13:20:00	formato especificado no estándar ISO/8601: Primeiro a data e despois a hora separados por unha T .12 de Decembro de 2010 ás 1.20pm
	xs:date	2012-12-31	formato "AAAA-MM-DD"
	xs:time	13:20:00	formato "hh:mm:ss"
	xs:gDay	--31	O día 31
	xs:gMonth	--05--	O mes de maio
	xs:gYear	2013	
	xs:gMonthDay	--04-18	Cada 18 de abril
	xs:gYearMonth	2013-02	O mes de Febreiro de 2013, sen importar o número de días
Lóxicos	xs:boolean	true, false 1, 0	
Varios	xs:language	ES, EN, FR	especifica un idioma, no formato de dous caracteres establecido pola norma ISO639.
	xs:NMTOKEN	diaSemana	é unha cadea de caracteres composta por letras, números, puntos, dous puntos, guións e suñados. Non pode ter espazos.
	xs:NMTOKENS	Nome completo	é unha cadea de caracteres composta por letras, números, puntos, dous puntos, guións e suñados.. Pode conter espazos.

## Atributos

Os atributos en si, decláranse como tipos simples:

```
<xs:attribute name="xxx" type="yyy"/>
```

Por exemplo, para o xml:

```
<nen numberExpedente="2344">
```

a definición do esquema xml sería:

```
<xs:attribute name="numeroExpedente" type="xs:unsignedShort"/>
```

Os atributos tamén poden ter un valor por defecto, ou un valor fixo. No caso de que teñamos atributos cun valor fixo: se no xml non se indica este, o valor será o indicado coa palabra clave *fixed* no esquema XML; e no caso de darlle un valor, este debe coincidir co establecido no esquema XML.



```
<xs:attribute name="provincia" type="xsd:NMTOKEN" default="Lugo"/>
<xs:attribute name="provincia" type="xsd:NMTOKEN" fixed="Lugo"/>
```

Por defecto os atributos son opcionais. Se queremos establecer que sexan obrigatorios, usaremos `required`:

```
<xs:attribute name="nenos" type="numeroExpediente" use="optional"/>
<xs:attribute name="nenos" type="numeroExpediente" use="required"/>
```

## Tipos complexos

Os tipos complexos permiten definir elementos que conteñen atributos e/ou elementos anidados e defínense empregando o elemento `complexType`.

Os elementos decláranse co elemento `element`, e como xa vimos, para declarar atributos teremos que usar o elemento `attribute`. Por exemplo, `tipoEnderezo` está definido como un tipo complexo, e dentro da definición de `endereço` vemos tres declaracións de elementos e unha declaración de atributo:

```
<xs:complexType name="tipoEnderezo">
  <xs:sequence>
    <xs:element name="rua" type="xsd:string"/>
    <xs:element name="numero" type="xsd:unsignedShort"/>
    <xs:element name="localidade" type="xsd:string"/>
  </xs:sequence>
  <xs:attribute name="provincia" type="xsd:NMTOKEN" fixed="Lugo"/>
</xs:complexType>
```

Este tipo `tipoEnderezo` podemos asocialo a un ou máis elementos que respondan a este formato (p.e. `endereçoPrincipal` e `endereçoVerán`). E tendo en conta que deben manterse a orde e os nomes dos elementos e atributos tal cal están definidos no tipo. Os elementos `rua` e `localidade` conterán unha cadea de caracteres e `numero` un dato numérico. O elemento que se declare como `tipoEnderezo` pode aparecer con un atributo chamado `provincia` que debe conter a cadea `Lugo`.

## Tipos de elementos complexos

Como xa temos visto, un elemento complexo é un elemento xml que consta de outros elementos e/ou atributos. Existen catro tipos de elementos complexos:

### ■ Elementos baleiros

Algúns dos elementos do documento xml poden estar baleiros. Son os "empty elements". Por exemplo, supoñamos que temos un elemento `distancia` que consta da unidade de medida e a distancia propiamente dita como valores de atributos en lugar de telos como valores separados de atributo e contido:

```
<distancia unidade="Km" valor="444,23"/>
```

O correspondente esquema XML sería:

```
<xs:element name="distancia">
  <xs:complexType>
    <xs:attribute name="unidade" type="xsd:string"/>
    <xs:attribute name="valor" type="xsd:decimal"/>
  </xs:complexType>
</xs:element>
```

- *Elementos contedores de outros elementos*

Un elemento pode estar formado por outros elementos, cada un deles co seu propio tipo.

```
<xs:complexType name="nomeCompleto">
  <xs:sequence>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="apelido1" type="xs:string"/>
    <xs:element name="apelido2" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Un exemplo de documento XML que responde a este esquema sería:

```
<nomeCompleto>
  <nome>Mar</nome>
  <apelido1>Blanco</apelido1>
</nomeCompleto>
```

- *Elementos que constan unicamente de texto*

Alguns elementos poden conter só texto ou texto e atributos. Por exemplo:

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

Ao cal respondería o seguinte XML:

```
<distancia unidade="Km">32</distancia>
```

- *Elementos que constan de texto e de outros elementos (mixtos).*

Alguns elementos complexos poden conter texto, subelementos e atributos. Estes decláranse especificando o atributo `mixed="true"` da etiqueta `<xs:complexType>`. Por exemplo:

```
<xs:element name="carta">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="fillo" type="xs:string"/>
      <xs:element name="numero" type="xs:unsignedShort"/>
      <xs:element name="data" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Un exemplo de documento XML que responde a este esquema es:

```
<carta>
  Estimado Sr.<nome>Pedro Barro</nome>.
  A presente carta é para notificarlle que o seu fillo
  <fillo>Xoel</fillo> ten un total de <numero>3</numero> faltas
```

```
    dende o día <data>13/4/30</data>.  
</carta>
```

### Indicadores de orde

Ata agora só temos visto o compositor `sequence`, que define un grupo ordenado de elementos, pero existen outros dous tipos de compositores: `choice` e `all`.

- *O compositor `choice`*

O compositor `choice` define un conxunto de elementos dos cales únicamente pode aparecer un destes elementos. Dacordo coa seguinte construción, para o elemento `persoa` debería aparecer ou o elemento `alcume` ou o nome e apelidos desta:

```
<xs:complexType name="persoa">  
  <xs:choice>  
    <xs:element name="alcume" type="xs:string"/>  
    <xs:sequence>  
      <xs:element name="nome" type="xs:string"/>  
      <xs:element name="apelido1" type="xs:string"/>  
      <xs:element name="apelido2" type="xs:string" minOccurs="0"/>  
    </xs:sequence>  
  </choice>  
</xs:complexType>
```

- *O compositor `all`*

Se queremos que un elemento poida conter outros elementos en calquera orde, podemos agrupar eses elementos internos cun grupo `all`. É similar ao compositor `sequence`, só que os elementos non teñen que aparecer en ningunha orde determinada. Por exemplo:

```
<xs:element name="persoa">  
  <xs:complexType>  
    <xs:all>  
      <xs:element name="nif" type="xs:string"/>  
      <xs:element name="nome" type="nomeCompleto"/>  
    </xs:all>  
  </xs:complexType>  
</xs:element>
```

Tendo en conta que o tipo `nomeCompleto` sería o definido cando explicabamos os tipos de elementos complexos, serían instancias válidas para este esquema:

```
<persoa>  
  <nif>44444444M</nif>  
  <nomeCompleto>  
    <nome>Mar</nome>  
    <apelido1>Blanco</apelido1>  
  </nomeCompleto>  
</persoa>
```

ou ben:

```
<persoa>  
  <nomeCompleto>  
    <nome>Mar</nome>  
    <apelido1>Blanco</apelido1>  
  </nomeCompleto>  
  <nif>44444444M</nif>
```

```
</persoa>
```

O esquema XML estipula que un grupo `all` debe aparecer como fillo único no nivel máis alto do modelo de contido. Noutras palabras, o seguinte *non é legal*:

```
<xs:element name="persoa">
  <xs:complexType>
    <xs:sequence>
      <xs:all>
        <xs:element name="nif" type="xs:string"/>
        <xs:element name="nome" type="nomeCompleto"/>
      </xs:all>
      <xs:element name="nss" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

### Modos de declarar os elementos

Os elementos ou tipos poden declararse globales ao documento ou locais:

- Nunha *declaración global* os elementos deben ser fillos do elemento raíz do esquema XML (`xs:schema`) o que permite a reutilización de código.
- Unha declaración local está anidada dentro da estrutura do esquema XML no interior doutra declaración.

Independentemente de se estamos a declarar tipos simples ou complexos temos varias opcións á hora de declarar os elementos:

- *Tipos anónimos: Introducendo a declaración do tipo dentro da definición do elemento.* Este xeito de declarar elementos dise que é *local*, xa que o elemento ao estar anidado dentro de outros elementos, non pode ser empregado noutros puntos do esquema.

```
<xs:element name="endereçoPrincipal">
  <xs:sequence>
    <xs:element name="rua" type="xs:string"/>
    <xs:element name="numero" type="xs:unsignedShort"/>
    <xs:element name="localidade" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="provincia" type="xs:NMTOKEN" fixed="Lugo"/>
</xs:element>
```

- *Facendo a declaración do tipo de xeito independente, e declarando elementos deste tipo.* Inda que a primeira opción sexa máis lexible non permite a reutilización de tipos, o que si sería posible usando este modo de declaración. Este tipo de declaración ou definición é *global*, fillo inmediato do elemento `schema`, e polo tanto reutilizable no contexto do esquema.

```
... ..
<xs:element name="endereçoPrincipal" type="tipoEndereço"/>
... ..
<xs:complexType name="tipoEndereço">
  <xs:sequence>
    <xs:element name="rua" type="xs:string"/>
    <xs:element name="numero" type="xs:unsignedShort"/>
    <xs:element name="localidade" type="xs:string"/>
  </xs:sequence>
```

```
<xs:attribute name="provincia" type="xs:NMTOKEN" fixed="Lugo"/>
</xs:complexType>
```

- *Empregando referencias.* Ao igual que coa opción anterior definimos o tipo e despois podemos facer referencia a este as veces que sexa necesario.

```
<xs:element name="enderezos">
  <xs:sequence>
    <xs:element ref="tipoEnderezo" minOccurs="0"/>
  </xs:sequence>
</xs:element>
... ..
<xs:complexType name="tipoEnderezo">
  <xs:sequence>
    <xs:element name="rua" type="xs:string"/>
    <xs:element name="numero" type="xs:unsignedShort"/>
    <xs:element name="localidade" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="provincia" type="xs:NMTOKEN" fixed="Lugo"/>
</xs:complexType>
```

## Multiplicidade do elemento

Para indicar a multiplicidade dun elemento temos os atributos `minOccurs` e `maxOccurs`. O valor por defecto destes atributos é 1, o que significa que o elemento é *obligatorio* e só pode aparecer unha vez.

Os valores posibles para ambos atributos son enteiros positivos, e é importante asegurarse de que o atributo `minOccurs` non sexa maior que o atributo `maxOccurs` para o mesmo elemento.

Se queremos indicar que un elemento é *opcional* temos que asignar a `minOccurs` o valor 0. O seguinte elemento é opcional: pode aparecer 0 ou 1 vez:

```
<xs:element name="observacions" type="xs:string" minOccurs="0"/>
```

`maxOccurs` pode ter o valor `unbounded`, o que significa que non ten límite superior en canto ás veces que pode ocorrer o elemento.



Na tarefa 2 imos crear esquemas XML que respondan as especificacións que se determinan que deben cumprir os documentos XML que se queren validar.

## Creación dos seus propios tipos

Os esquemas ofrécennos a posibilidade de definir os nosos propios tipos, xa sexan estes datos simples ou complexos. Os tipos permiten dividir o esquema en partes e facer vínculos entre elas.

Como xa temos visto, os tipos de datos simples son elementos `simpleType` e os tipos de datos complexos, elementos `complexType`. A estas definicións podemos aplicarlle as seguintes restricións:

## Restriccións

Ao definir un tipo simple podemos especificar, ademais do tipo de dato que se espera como contido dos elementos e atributos, unha gran variedade de restriccións que nos permiten garantir que o contido dos elementos e atributos validan as condicións que nelas se indican.

As restriccións sempre se aplican sobre un tipo existente (denominado tipo base), e permiten restrinxir o conxunto de valores que estes poden tomar. A súa sintaxe é a seguinte:

```
<xs:simpleType name="nomeTipoSimple">
  <xs:restriction base="nomeTipoBase">
    ...
  </xs:restriction>
</xs:simpleType>
```

Na seguinte táboa vemos as posibles restriccións sobre un tipo base:

RESTRICCIÓN	DESCRIPCIÓN
length	Número exacto de caracteres da cadea ou número mínimo de elementos da lista permitidos. Debe ser igual ou maior que cero
minLength	Número mínimo de caracteres da cadea ou número mínimo de elementos da lista permitidos. Debe ser igual ou maior que cero
maxLength	Número máximo de caracteres da cadea ou número máximo de elementos da lista permitidos. Debe ser igual ou maior que cero
pattern	O contido debe encaixar cunha expresión regular
enumeration	Define unha lista de valores aceptables
whiteSpace	Controla a normalización dos espazos en branco (saltos de liña, tabuladores, espazos e retornos de carro)
minInclusive	Indica o límite inferior do valor numérico (debe ser maior ou igual ao valor indicado)
minExclusive	Indica o límite inferior do valor numérico (debe ser maior ao valor indicado)
maxInclusive	Indica o límite superior do valor numérico (debe ser menor ou igual ao valor indicado)
maxExclusive	Indica o límite superior do valor numérico (debe ser menor ao valor indicado)
totalDigits	Indica o máximo número de díxitos nos tipos numéricos. Debe ser maior que 0
fractionDigits	Indica o máximo número de decimais nos tipos numéricos. Debe ser maior ou igual que 0

Dependendo do tipo base poderase aplicar un posible conxunto de restriccións. Por exemplo, as restriccións sobre un tipo `string` son: `length`, `minLength`, `maxLength`, `pattern`, `enumeration` e `whiteSpace`. As restriccións sobre un tipo de datos `integer` son: `totalDigits`, `pattern`, `whiteSpace`, `enumeration`, `maxInclusive`, `maxExclusive`, `minInclusive` e `minExclusive`.

### Restriccións referidas á lonxitude

A continuación imos definir un tipo de datos (`nomeCorto`) que corresponda a unha cadea de caracteres que acepte un máximo de 15 caracteres. O tipo creado será:

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

- O elemento `simpleType` indica que o tipo que se vai crear será simple.

- O elemento `restriction` permite anunciar que se vai crear un novo tipo a partir do tipo especificado no atributo `base` (neste caso o tipo deriva de `string`).
- O elemento `maxLength` establece a restricción para que o número máximo de caracteres sexa 15 no atributo `value`.

#### Restricións sobre os valores

O seguinte exemplo limita a idade dos alumnos dunha gardería de 0 a 3 anos:

```
<xs:element name="idade">
  <xs:simpleType>
    <xs:restriction base="xs:byte">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="3"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

#### Restricións sobre o conxunto de valores que pode tomar: enumeration

Imos ver un exemplo no que especificamos os posibles valores para un elemento utilizando elementos `enumeration`. Para elo, crearemos un tipo chamado `tipoGrao` que se crea a partir do tipo base `string` e que pode tomar dous valores: `medio` e `superior`.

```
<xs:simpleType name="tipoGrao">
  <xs:restriction base="xs:string">
    <xs:enumeration value="medio"/>
    <xs:enumeration value="superior"/>
  </xs:restriction>
</xs:simpleType>
```

#### Forzar que o texto se adapte a un patrón

As veces queremos que os datos se axusten a un patrón determinado, como por exemplo o NIF, un código postal, ou o código dun ciclo. Para crear un patrón úsase unha linguaxe especial que establece onde poden aparecer un tipo determinado de caracteres. Isto son as "expresións regulares".

Nos esquemas defínense empregando a restricción `pattern`. Na seguinte táboa vemos

Patrón	Descrición	Exemplo		Non validaría
<b>^</b>	Coincidencia ao principio da cadea	<code>^Isto</code>	Isto é unha cadea	É isto de aquí
<b>\$</b>	Coincidencia ao final da cadea	<code>final\$</code>	Chegamos ao final	
<b>*</b>	O carácter que o precede pode aparecer 0, 1 ou máis veces	<code>ma*</code>	maaaaa ou ma ou m	
<b>+</b>	O carácter que o precede debe aparecer polo menos una vez (1 ou máis veces)	<code>ma+</code>	maaaaa ou ma	m
<b>?</b>	O carácter que o precede pode aparecer como moito unha vez (0 ou 1 vez)	<code>ma?</code>	ma ou m	maaaaa
<b>[...]</b>	Calquera carácter dentro dos paréntesis	<code>a[px]e</code>	ape ou axe	ale
<b>[ - ]</b>	Indica un rango de caracteres	<code>[c-k]a</code>	ca ou ea	pa

Patrón	Descrición	Exemplo		Non validaría
[^...]	Calquera carácter menos os que están entre paréntesis	a[^px]e	ale ou a1e	ape ou Axe
[^ - ]	Calquera carácter menos os que estean nese rango	[^c-k]a	pa	ea
{n}	O carácter que o precede debe aparecer exactamente n veces	bua{2}	buaa	bua
{n,}	O carácter que o precede debe aparecer n veces ou máis veces	bu{2,}	buuuuu ou buu	bu
{n,m}	O carácter que o precede debe aparecer como mínimo n e como máximo m veces	[0-9]{2,4}	32 ou 4444	8 ou 78978
(...)	Podemos agrupar con paréntese	(co){2}liso	cocoliso	Coliso
	Unha barra vertical separa alternativas	M[0-9]   MP	MP ou M9	MA 9M
\d	Calquera dígito ( é equivalente a [0-9])	\d{2}	23	S7
\D	Calquera non dígito	\D{2}M	L_M ou ABM	4CM
\w	Calquera letra ou dígito	ala\w	ala4 ou alag	ala_
\W	Calquera carácter diferente de letra ou dígito	MWa	M-a	M1a
.	Calquera carácter salvo salto de liña			
\n	Salto de liña			
\s	Carácter en branco, tabulador ou salto de liña			

A continuación amósase un patrón para un número de teléfono:

```
<xs:simpleType name="tipoTelefono">
  <xs:restriction base="xs:string">
    <xs:pattern value="([0-9]{3})\s[0-9][0-9]-[0-9][0-9]-[0-9][0-9]"/>
  </xs:restriction>
</xs:simpleType>
```

A expresión regular que definimos para o `tipoTelefono` significa:

- Debe empezar con paréntese de apertura (.
- Seguido por tres dígitos.
- Seguidos por un paréntese de peche ) e un espazo.
- Seguidos por dous dígitos máis.
- Seguidos por un guión.
- Seguidos por dous dígitos máis.
- Seguidos por un guión.
- Seguidos por dous dígitos máis.

[Restricións referidas ao tratamento dos espazos en branco](#)

Empregando o elemento `whiteSpace` podemos indicar cómo queremos que o analizador XML trate os espazos en branco. Este elemento pode tomar os valores:

- `preserve` \_ Conserva os caracteres en branco (saltos de liña, tabuladores, espazos e retornos de carro)



- `replace _` Substitue todos os caracteres en branco por espazos simples
- `collapse _` Reemplaza secuencias de saltos de liña, tabuladores e múltiples espazos en branco consecutivos por espazos simples. Elimina os espazos iniciais e finais.

Por exemplo:

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Na tarefa 3 usaremos restriccións para garantir que a información que se introduce no documento XML axústase adecuadamente ao contido esperado para ese elemento ou atributo.

### Tipos lista

Unha lista consiste nunha secuencia de valores dun determinado tipo, separada por espazos. A súa sintaxe é a seguinte:

```
<xs:simpleType name="nomeTipoLista">
  <xs:list itemType="tipoDosValores"/>
</xs:simpleType>
```

Por exemplo, para crear unha lista de elementos `listaDeNotas` baseada no tipo `nota` que vimos anteriormente:

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

Onde temos definido o tipo `nota` como:

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

No documento XML teríamos o elemento `listaNota` coa lista de notas como vemos a continuación:

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

Aos tipos lista poden aplicárselles as seguintes propiedades:

- `length`: lonxitude
- `minLength`: lonxitude mínima
- `maxLength`: lonxitude máxima, y

- **enumeration:** enumeración de posibles valores

Por exemplo, imaxinemos que para a 1ª avaliación temos unha lista de 5 notas (notas1Ev), definimos primeiro a lista de notas listaNotas a partir do tipo nota, e entón creamos notas1Ev, restrinxindo listaDeNotas a só cinco elementos:

```
<xsd:simpleType name="listaNotas">
  <xsd:list itemType="nota"/>
</xsd:simpleType>
<xsd:simpleType name="notas1Ev">
  <xsd:restriction base="listaNotas">
    <xsd:length value="5"/>
  </xsd:restriction>
</xsd:simpleType>
```

Os elementos que teñan como tipo notas1Ev deben ter cinco elementos, e cada un destes debe ser do tipo nota. Por exemplo:

```
<notas1Ev>5 9 4 8 6</notas1Ev>
```

As listas non sempre son un bon recurso, xa que se perden as vantaxes do formato XML. É máis recomendable engadir máis marcas como vemos a continuación:

```
<notas1Ev>
  <nota>5</nota>
  <nota>9</nota>
  <nota>4</nota>
  <nota>8</nota>
  <nota>6</nota>
</notas1Ev>
```

### Tipos union

As unións, permiten crear novos tipos de datos formados pola unión de outros tipos. Por exemplo, para definir un tipo moeda, que poda tomar valores integer ou float:

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

Pódense combinar varios tipos simples nun tipo único con distintos tipos de datos. A súa sintaxe é:

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

Por exemplo, as posibles notas dun alumno son un número enteiro do 1 ao 10, ou un NP no caso de non terse presentado ao exame:

```
<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>

```

### Incluir calquera elemento ou atributo

A estrutura `any` permite incluír elementos non declarados inicialmente no documento XML, isto é, que o esquema acepte nese lugar calquera contido dun determinado espazo de nomes; ou con `anyAttribute` calquera atributo dun espazo de nomes.

En xeral, un elemento `any` especifica que calquera XML ben formado é permisible nese modelo de contido. No seguinte exemplo, requerimos que o contido do elemento `comentario` pertenza ao espazo de nomes `http://www.w3.org/1999/xhtml`, isto é, debe ser HTML. Ademais, tamén require que exista polo menos un elemento para este espazo de nomes (ver os valores de `minOccurs` y `maxOccurs`):

```

<?xml version = "1.0" encoding = "utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persoa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome" type="xs:string"/>
        <xs:element name="apelido1" type="xs:string"/>
        <xs:element name="apelido2" 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>

```

O documento XML que responde a este esquema podería ser:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- tarefa 1 UD7 -->

<persoa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xsi:noNamespaceSchemaLocation="a.xsd">
  <nome>Nerea</nome>
  <apelido1>Solla</apelido1>
  <apelido2>Solla</apelido2>

```

```

    <comentario>
      <html:p>
        <html:emph>Repetidor</html:emph> con un módulo pendente.
      </html:p>
    </comentario>
  </persoa>

```

Se queremos que se valide que o código HTML é válido, o atributo `processContents` (proceso de contidos) debería ser igual a `strict` (o valor por defecto). Nese caso, o procesador XML está obrigado a obter o esquema asociado co espazo de nomes requirido, e validar o HTML que aparece dentro do elemento `comentario`. Se lle damos a este atributo o valor `lax`, só validaría se é posible.

O atributo `namespace`, ademáis de indicar un espazo de nomes podería tomar os seguintes valores:

- `##any`, para indicar que o contido do elemento debe ser calquera XML ben formado de calquera espazo de nomes. Este é o valor por defecto.
- `##local`, para indicar que o contido debe ser calquera XML ben formado e sen calificar, é dicir, non declarado como pertencente a un espazo de nomes.
- `##other`, neste caso o contido pode ser calquera XML ben formado que non é do espazo de nomes do tipo que está sendo definido.
- `##targetNamespace`, permite un novo elemento proporcionado polo espazo de nomes que o esquema XML está definindo.

Con `anyAttribute` indicamos a presenza de calquera atributo dentro dun elemento, polo tanto no seguinte exemplo o elemento `persoa` podería ter calquera tipo de atributo.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persoa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome" type="xs:string"/>
        <xs:element name="apelido1" type="xs:string"/>
        <xs:element name="apelido2" minOccurs="0" type="xs:string"/>
      </xs:sequence>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Un documento XML que responde a este esquema podería ser:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- exemplo AnyAttribute -->

<persoa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploAnyAttribute.xsd"
  sexo="femenino">
  <nome>Nerea</nome>
  <apelido1>Solla</apelido1>
  <apelido2>Solla</apelido2>
</persoa>

```

## Os grupos

Os grupos poden empregarse para definir conxuntos relacionados de elementos con `group`, ou conxuntos relacionados de atributos con `attributeGroup` o que nos permite posteriormente facer referencia a eles dende outros elementos.

Un exemplo de grupo de elementos sería:

```
<xs:group name="nomeCompleto">
  <xs:sequence>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="apelido1" type="xs:string"/>
    <xs:element name="apelido2" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:group>
```

Posteriormente poderíamos facer referencia a él da seguinte maneira:

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

Un exemplo completo que emprega *grupo de atributos* sería:

```
<?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="grao">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <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="nome" type="xs:string"/>
        <xs:element name="familia" type="xs:string"/>
      </xs:sequence>
      <xs:attributeGroup ref="caracteristicas"/>
    </xs:complexType>
  </xs:element>
```

Un documento XML que responde a este esquema podería ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<ciclo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="exemploAttributeGroup.xsd"
      duracion="2000">
```

```

        grao="superior">
    <nome>Desenvolvimento de aplicacións web</nome>
    <familia>IFC</familia>
</ciclo>

```

Utilizando un grupo de atributos deste modo podemos aumentar a lexibilidade dos esquemas, y facilitar a actualización dos esquemas xa que un grupo de atributos defínese nun único lugar e pode ser referenciado en múltiples definicións e declaracións.

Un grupo de atributos pode conter outros grupos de atributos.

## Anotacións

O elemento `annotation` úsase para documentar o esquema XML, tanto para as persoas como para os programas. Dependendo de a quen vaia destinado terá diferentes elementos fillo:

- `documentation`, cando a documentación vai dirixida as persoas. Este elemento pode ter dous atributos:

- `source`, que contén a URL dun arquivo que contén información complementaria.
- `lang`, que indica a linguaxe na que está escrita a documentación.

```

<xs:documentation source=http://www.web.com/documentacion.txt
                  xml:lang="ES"/>

```

- `appinfo`, cando a documentación vai dirixida a programas. Este elemento pode ter un atributo:

- `source`, que contén a URL dun arquivo que contén información complementaria.

```

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

```

As anotacións non teñen efecto na validación do esquema XML, podendo ser calquera sempre que estean ben formadas en XML. Pero unicamente poden ir antes ou despois dun compoñente global: non se poden poñer en calquera parte do documento.

Por exemplo:

```

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

    <xsd:annotation>
        <xsd:appInfo source="http://www.web.com/documAxenda.xml">
            Axenda telefónica
        </xsd:appInfo>
        <xsd:documentation>
            Verifica que os números telefónicos son válidos
            Os nomes e apelidos escribíranse en maiúsculas
        </xsd:documentation>
    </xsd:annotation>
    . . .
</xsd:schema>

```

## Valor nulo

Debemos diferenciar entre un valor de "cadea baleira" e un valor descoñecido ou nulo. As veces ten importancia indicar que non coñecemos a información, ou que a información non

é aplicable de xeito explícito a un elemento. Isto non sería equivalente á ausencia do elemento.

Ademáis, debe ser posible representar os valores nulos (`NULL`) que se reciben ou envían a unha base de datos relacional cun elemento que sí está presente. Para estes casos temos o atributo `nillable` que nos permite indicar que ese elemento pode tomar o valor `NULL`. A ausencia de valor para un elemento no documento XML indícarase co atributo `xsi:nil`. Imos ver un exemplo donde no caso de non coñecer a idade dun neno queremos que esta tome o valor `NULL`:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="garderia">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nenos" minOccurs="1" maxOccurs="50"
          type="tipoNeno"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="tipoNeno">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="idade" type="xs:unsignedByte" nillable="true"/>
    </xs:sequence>
    <xs:attribute name="numeroExpediente" type="xs:unsignedShort"/>
  </xs:complexType>
</xs:schema>
```

O documento XML, no caso de non coñecer o valor, sería o seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- exemplo valor NULO -->

<garderia xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploNillable.xsd">
  <nenos numeroExpediente="2344">
    <nome>Nuno Cruz Vila</nome>
    <idade xsi:nil="true"></idade>
  </nenos>
</garderia>
```

## Claves e unicidade

Nos DTDs podíamos establecer a unicidade dun elemento empregando o atributo `ID`, co cal validábase que o valor deste atributo tiña que ser único para todo o documento. Ademáis, empregando os atributos `ID` e `IDREF` conxuntamente establecíamos relacións entre os elementos.

Cos esquemas XML temos mecanismos máis potentes, de xeito que podemos:

- Indicar que un elemento é único
- Indicar que un atributo é único
- Definir combinacións de elementos e atributos como únicos

- Definir claves, onde, ademáis de ser único debe existir obrigatoriamente e non pode ser nulo.

### Unicidade

A unicidade especificase empregando o elemento `unique`, que permite definir valores únicos para elementos e atributos. O seu valor pode ser nulo, pero no caso de introducir dous valores iguais daría error. O elemento `unique` consta de:

- O elemento `selector`, que empregando `xpath` selecciona un subconxunto do documento.
- O elemento `field`, no que empregando outra expresión `xpath` seleccionamos o campo que vai ser clave dentro do subconxunto seleccionado con `selector`.

Imos ver isto cun exemplo dunha axenda, onde os nosos contactos contan cun nome e un teléfono. Queremos impedir que o nome dos contactos se repita para evitar confusións e para isto, ímolo declarar como único. O esquema quedaría como sigue:

```
<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="nomeUnico">
      <xs:selector xpath="contacto"/>
      <xs:field xpath="nome"/>
    </xs:unique>
  </xs:element>

  <xs:complexType name="tipoContacto">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="telefono" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Onde con `selector` temos seleccionado o subgrupo `contacto` dentro do documento, e con `field` indicamos que é o nome o que non se pode repetir. Un XML que validaría este esquema sería:

```
<?xml version='1.0' encoding='UTF-8' ?>
<agenda xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploUnique.xsd">
  <contacto>
    <nome>Iria</nome>
    <telefono>988988988</telefono>
  </contacto>

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



Se repetísemos o nome dun contacto o documento xml non validaría ao estar definido como unique.

### Claves

As claves defínense con `key`, e teñen a mesma estrutura que `unique`, coa diferenza de que non pode tomar valores nulos.

Para facer referencia a unha clave definida previamente úsase o elemento `keyref`, que so seu atributo `ref` fai referencia á clave correspondente.

Imos ver isto cun exemplo dun documento XML para almacenar os empregados e os departamentos dunha empresa. Así, definimos unha clave para o atributo código dos departamentos e facemos referencia a este para indicar o departamento ao que pertence cada empregado, impedindo así que se asocien a departamentos que non existen. O esquema XML sería o seguinte:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="empresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="empregado" 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="empregado"/>
      <xs:field xpath="departamento"/>
    </xs:keyref>
  </xs:element>

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

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

Un documento XML que validaría este esquema sería:

```
<?xml version='1.0' encoding='UTF-8' ?>
<empresa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploKey.xsd">
  <empregado>
    <nome>Iria</nome>
```

```

        <departamento>IFC</departamento>
    </empregado>
</empregado>
    <nome>Xoel</nome>
    <departamento>CON</departamento>
</empregado>
<departamento codigo="IFC">
    <nome>Informática</nome>
</departamento>
<departamento codigo="CON">
    <nome>Contabilidade</nome>
</departamento>
</empresa>

```

Onde no caso de meter un valor que non exista para o código de departamento daría unha mensaxe de erro.

Unha clave podería estar formada por varios elementos. Por exemplo:

```

<xs:key name="nomeUnico">
    <xs:selector xpath="contacto"/>
    <xs:field xpath="nome"/>
    <xs:field xpath="apelidos"/>
</xs:key>

```



**Na tarefa 4** partiremos dunha DTD e crearemos un documento XML válido para ésta, e un esquema XML que incorpore algunhas características adicionais para comprobar algunha das vantaxes dos esquemas XML fronte ás DTDs.

## Creación de esquemas a partir doutros esquemas

### Reaprovechar definiciones usando includes

As veces, temos documentos XML que teñen elementos similares. Nestos casos, é posible que queiramos reaproveitar as definicións dos elementos que son comúns a varios esquemas evitando ter que escribilas en cada un dos esquemas XML.

Isto pode facerse creando un esquema coas definicións comúns e enlazando este dende os outros esquemas empregando o elemento `include`. Os elementos deben estar no mesmo espazo de nomes. É como se estivesen todos tecleados nun mesmo ficheiro.

Imos ver un exemplo no que declaramos un elemento para nomes propios (o primeiro carácter en maiúsculas e os seguintes en minúsculas) nun esquema XML chamado `definicionsComuns.xsd` para a continuación enlazalo dende outro esquema XML:

#### `definicionsComuns.xsd`

```

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

```

```
</xs:schema>
```

### exemploInclude.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="definicionsComuns.xsd"/>
  <xs:element name="persoa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome" type="tipoNomePropio"/>
        <xs:element name="apelido1" type="tipoNomePropio"/>
        <xs:element name="apelido2" minOccurs="0" type="tipoNomePropio"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- exemplo Include -->
<persoa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploInclude.xsd">
  <nome>Nerea</nome>
  <apelido1>Solla</apelido1>
  <apelido2>Mouriño</apelido2>
</persoa>
```

### Redefinición de esquemas

`include` facilita o uso de compoñentes de esquema creados externamente tal cual, isto é, sen ningunha modificación. Con `redefine` podense redefinir os tipos complexos, simples, grupos, ou grupos de atributos que se obtieñen de arquivos de esquema externos.

Ao igual que con `include`, `redefine` require que os compoñentes externos estean no mesmo espazo de nomes que o esquema que os vai redefinir, inda que os compoñentes externos que non teñen espazo de nomes algún, tamén poden ser redefinidos. Neste caso, os compoñentes redefinidos pasan a formar parte do espazo de nomes do esquema que os redefine.

Imos ver un exemplo onde temos un esquema XML (`exemploRedefine1.xsd`) onde temos unha definición para un tipo complexo chamado `nomeCompleto` que inclúe dous elementos: `nome` e `apelido1`. No noso documento queremos engadir outro elemento (`apelido2`) co cal no esquema XML (`exemploRedefine2.xsd`) redefinimos o tipo para que o inclúa:

### exemploRedefine1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="nomeCompleto">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="apelido1" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

## exemploRedefine2.xsd

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

  <xs:redefine schemaLocation="exemploRedefine1.xsd">
    <xs:complexType name="nomeCompleto">
      <xs:complexContent>
        <xs:extension base="nomeCompleto">
          <xs:sequence>
            <xs:element name="apelido2" type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:redefine>

  <xs:element name="persoa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nomeC" type="nomeCompleto"/>
        <xs:element name="alcume" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- exemplo Redefine -->
<persoa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemploRedefine2.xsd">
  <nomeC>
    <nome>Nieves</nome>
    <apelido1>Pérez</apelido1>
    <apelido2>Pérez</apelido2>
  </nomeC>
  <alcume>Niní</alcume>
</persoa>
```

## Importación de esquemas

`import` úsase para mezclar esquemas XML con diferentes espazos de nomes. Ao importar as declaracións, os dous esquemas poden usarse conjuntamente nun mesmo documento instancia.

```
<xs:schema ...>
  <xs:import namespace="A" schemaLocation="esquemaA.xsd"/>
  <xs:import namespace="P" schemaLocation="esquemaB.xsd"/>
  . . .
</xs:schema>
```



**Na tarefa 5** imos crear esquemas XML que respondan as especificacións que se determinan que deben cumprir os documentos XML que se queren validar

■ empregando todos os conceptos explicados na UD.