

T4.1. XML Schema

Los esquemas XML son un mecanismo radicalmente distinto de crear reglas para validar ficheros XML. Se caracterizan por:

- Estar escritos en XML. Por lo tanto, las mismas bibliotecas que permiten procesar ficheros XML de datos permitirían procesar ficheros XML de reglas.
- Son mucho más potentes: ofrecen soporte a tipos de datos con comprobación de si el contenido de una etiqueta es de tipo `integer`, `date` o de otros tipos. También se permite añadir restricciones como indicar valores mínimo y máximo para un número o determinar el patrón que debe seguir una cadena válida
- Ofrecen la posibilidad de usar *espacios de nombres*. Los espacios de nombres son similares a los paquetes Java: permiten a personas distintas el definir etiquetas con el mismo nombre pudiendo luego distinguir etiquetas iguales en función del espacio de nombres que importemos.

Un ejemplo

Supongamos que deseamos tener ficheros XML con un solo elemento llamado `<cantidad>` que debe tener dentro un número.

```
<cantidad>20</cantidad>
```

Un posible esquema sería el siguiente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="xsd:integer"/>
</xsd:schema>
```

¿Qué contiene este fichero?

1. En primer lugar se indica que este fichero va a usar unas etiquetas ya definidas en un espacio de nombres (o XML Namespace, de ahí `xmlns`). Esa definición se hace en el espacio de nombres que aparece en la URL. Nuestro validador no descargará nada, esa URL es oficial y todos los validadores la conocen. Las etiquetas de ese espacio de nombres van a usar un prefijo que en este caso será `xsd`. Nótese que el prefijo puede ser como queramos (podría ser «abcd» o «zztop»), pero la costumbre es usar `xsd`.
2. Se indica que habrá un solo elemento y que el tipo de ese elemento es `<xsd:integer>`. Es decir, un entero básico.

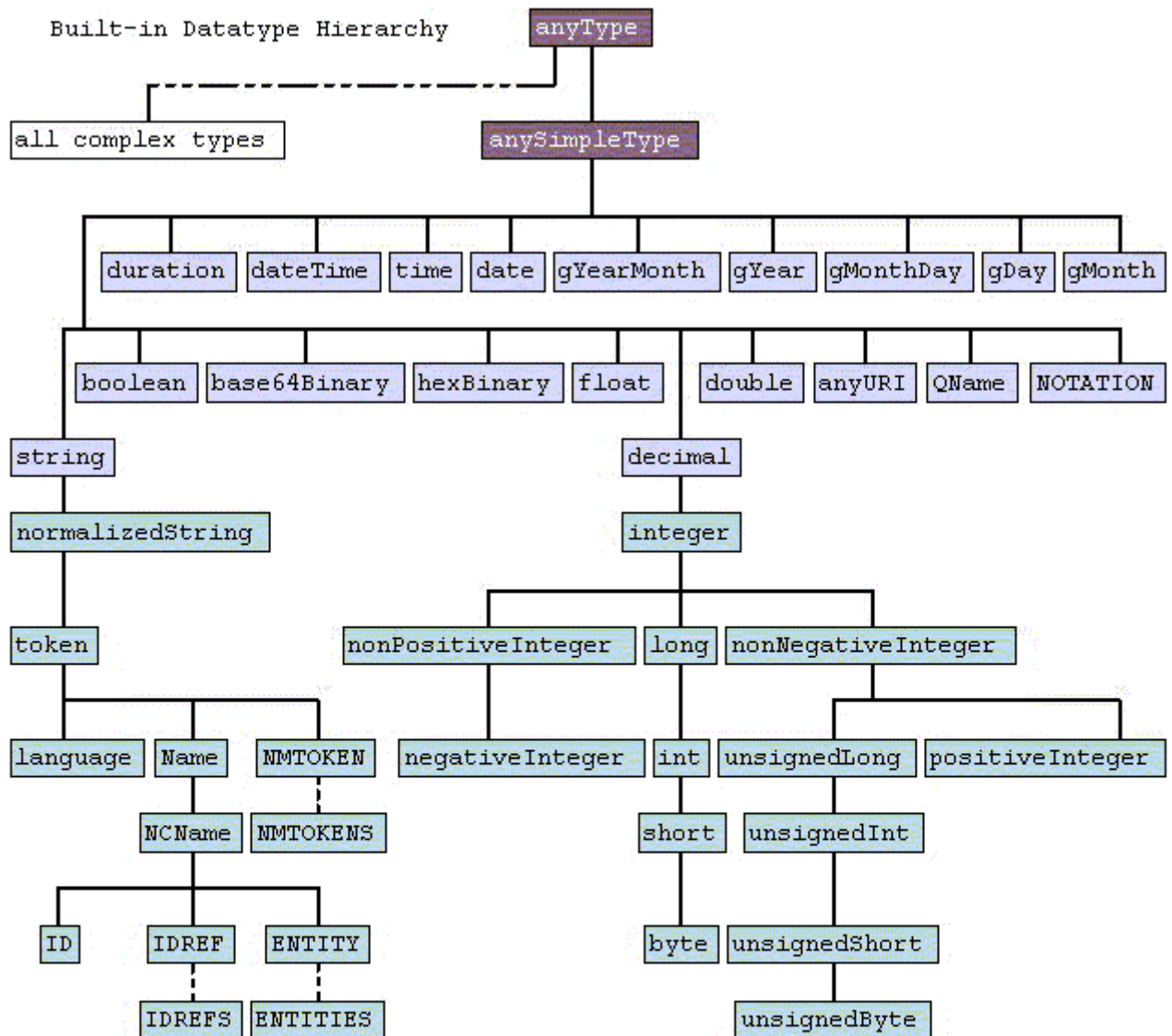
Si probamos el fichero de esquema con el fichero de datos que hemos indicado veremos que efectivamente el fichero XML de datos es válido. Sin embargo, si en lugar de una cantidad incluyésemos una cadena, veríamos que el fichero **no se validaría**








T4.2. Tipos de datos básicos

Podemos usar los siguientes tipos de datos:

- `xsd:byte`: entero de 8 bits.
- `xsd:short`: entero de 16 bits
- `xsd:int`: número entero de 32 bits.
- `xsd:long`: entero de 64 bits.
- `xsd:integer`: número entero sin límite de capacidad.
- `xsd:unsignedByte`: entero de 8 bits sin signo.
- `xsd:unsignedShort`: entero de 16 bits sin signo.
- `xsd:unsignedInt`: entero de 32 bits sin signo.
- `xsd:unsignedLong`: entero de 64 bits sin signo.
- `xsd:string`: cadena de caracteres en la que los espacios en blanco se respetan.
- `xsd:normalizedString`: cadena de caracteres en la que los espacios en blanco no se respetan y se reemplazarán secuencias largas de espacios o fines de línea por un solo espacio.
- `xsd:date`: permite almacenar fechas que deben ir **obligatoriamente** en formato AAAA-MM-DD (4 dígitos para el año, seguidos de un guión, seguido de dos dígitos para el mes, seguidos de un guión, seguidos de dos dígitos para el día del mes)
- `xsd:time`: para almacenar horas en formato HH:MM:SS.C
- `xsd:dateTime`: mezcla la fecha y la hora separando ambos campos con una T mayúscula. Esto permitiría almacenar `2020-09-22T10:40:22.6`.
- `xsd:duration`. Para indicar períodos. Se debe empezar con «P» y luego indicar el número de años, meses, días, minutos o segundos. Por ejemplo «P1Y4M21DT8H» indica un período de 1 año, 4 meses, 21 días y 8 horas. Se aceptan períodos negativos poniendo -P en lugar de P.
- `xsd:boolean`: acepta solo valores «true» y «false».
- `xsd:anyURI`: acepta URIs.
- `xsd:anyType`: es como la clase `Object` en Java. Será el tipo del cual heredaremos cuando no vayamos a usar ningún tipo especial como tipo padre.

La figura siguiente (tomada de la web del W3C) ilustra todos los tipos así como sus relaciones de herencia:



- | | | | |
|---|--------------------------|---|-------------------------------------|
|  | ur types |  | derived by restriction |
|  | built-in primitive types |  | derived by list |
|  | built-in derived types |  | derived by extension or restriction |
|  | complex types | | |

T4.3. Tipos simples y complejos

Todo elemento de un esquema debe ser de uno de estos dos tipos.

- Un elemento es de tipo simple si no permite dentro ni elementos hijo ni atributos.
- Un elemento es tipo complejo si permite tener dentro otras cosas (que veremos en seguida). Un tipo complejo puede a su vez tener contenido simple o contenido complejo:
 - Los que son de contenido simple no permiten tener dentro elementos hijo pero sí permiten atributos.
 - Los que son de contenido complejo sí permiten tener dentro elementos hijo y atributos.

Así, por ejemplo un tipo simple que no lleve ninguna restricción se puede indicar con el campo **type** de un **element** como hacíamos antes:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="xsd:integer"/>
</xsd:schema>
```

Sin embargo, si queremos indicar alguna restricción adicional ya no podremos usar el atributo **type**. Debemos reescribir nuestro esquema así:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType>
    Aquí irán las restricciones, que hemos omitido por ahora.
  </xsd:simpleType>
</xsd:schema>
```

T4.4. Restricciones y atributos

Restricciones

Como se ha dicho anteriormente la forma más común de trabajar es crear tipos que en unos casos aplicarán modificaciones en los tipos ya sea añadiendo cosas o restringiendo posibilidades. En este apartado se verá como aplicar restricciones.

Si queremos aplicar restricciones para un tipo simple las posibles restricciones son:

- **minInclusive** para indicar el menor valor numérico permitido.
- **maxInclusive** para indicar el mayor valor numérico permitido.
- **minExclusive** para indicar el menor valor numérico que ya no estaría permitido.
- **maxExclusive** para indicar el mayor valor numérico que ya no estaría permitido.
- **totalDigits** para indicar cuantas posibles cifras se permiten.
- **fractionDigits** para indicar cuantas posibles cifras decimales se permiten.
- **length** para indicar la longitud exacta de una cadena.
- **minLength** para indicar la longitud mínima de una cadena.
- **maxLength** para indicar la longitud máxima de una cadena.
- **enumeration** para indicar los valores aceptados por una cadena.
- **pattern** para indicar la estructura aceptada por una cadena.

Si queremos aplicar restricciones para un tipo complejo con contenido las posibles restricciones son las mismas de antes, pero además podemos añadir el elemento <attribute> así como las siguientes.

- **sequence** para indicar una secuencia de elementos
- **choice** para indicar que se debe elegir un elemento de entre los que aparecen.

Atributos

En primer lugar es muy importante recordar que **si queremos que un elemento tenga atributos entonces ya no se puede considerar que sea de tipo simple. Se debe usar FORZOSAMENTE un complexType**. Por otro lado en los XML Schema todos los atributos **son siempre opcionales, si queremos hacerlos obligatorios habrá que añadir un «required»**.

Un atributo se define de la siguiente manera:

```
<xsd:attribute name="fechanacimiento" type="xsd:date" use="required"/>
```

Esto define un atributo llamado **nombre** que aceptará solo fechas como valores válidos y que además es obligatorio poner siempre.