XML

JOSUE PEREZ LUCERO

La base de todo: HTML

- Ampliamente utilizado
- Inventado por Tim Berners-Lee, 1991
- Objetivos:
  - Presentar información enlazada
  - Orientado a personas
- Es el lenguaje de **presentación** en la Web
  - Laborioso de procesar por **máquinas**

Carencias semánticas

- Orientado a la presentación
  - Información en base de datos
    - Tiene sentido en su modelo de datos
  - Una vez formateado se pierde el significado
    - Contexto visual para extraer el significado
- Acoplamiento de contenido y presentación
  - “Screen-scraping” para extraer contenido
  - “Metatags” para añadir significado

XML

- Limitaciones de HTML:
  - HTML está siendo extendido por cada desarrollador
  - Necesidad de un estándar
  - W3C: 1996-1998
- W3C XML Working Group:
  - Microsoft, Sun Microsystems, Adobe, IBM,…
  - Formato abierto y libre
  - ¡Desarrollado por empresas competidoras!

¿Qué es XML?

- e**X**tensible **M**arkup **L**anguage
- Estándar W3C para la creación de lenguajes de “etiquetas”
  - Descripción de la información
- Subconjunto de SGML (Standard Generalized Markup Language)
- Etiquetado semántico, no de estilo
  - Gran problema de HTML

Componentes de un Documento XML

- Elementos
  - Delimitados por etiquetas
- Atributos
  - Contenidos en las etiquetas
- Entidades
  - Permiten referirnos a elementos externos
- Componentes avanzados
  - Secciones CDATA y Processing **Instructions**

Normas básicas

- XML es “case sensitive”
- Todos los tags deben cerrarse correctamente.
- Todos los elementos han de anidarse correctamente.
- La primera línea es la declaración XML:
  - <?xml version=“1.0”?>
- Ha de existir siempre un elemento raíz.
- Los valores de los atributos deben estar delimitados por comillas dobles.
- Hay caracteres que no pueden utilizarse

Documento XML

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

<customer id="PELP1234">
    <name>Pepe Le Pew</name>
    <address>Looney Tunes</address>
    <account>
        <id>MERRY MELODIES</id>
        <since>1945</since>
        <balance>-24,98</balance>
    </account>
</customer>
```

¿Qué NO es XML?

- No es un sustituto del HTML
  - Diferentes objetivos: no define presentación
  - HTML debería ser un sublenguaje de XML: XHTML
- No es un lenguaje de marcas
  - las marcas las definimos nosotros: extensible
- No describe la estructura
  - Se hace con un DTD

¿Y qué es un DTD?

- Document Type Definition
  - Descripción lógica de los datos
  - Permite cerrar la estructura del documento
- Validación de la información
  - Documento “bien formado”
  - Documento “validado”
- Está siendo sustituido por XML Schemas
  - La estructura se define en XML

Documento DTD (I)

```
<!DOCTYPE CUSTOMER [

<!ELEMENT customer (name, address, account?,
importantcustomer?)>
<!ATTLIST customer id CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT account (id, since, balance)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT since (#PCDATA)>
<!ELEMENT balance (#PCDATA)>
<!ELEMENT importantcustomer (#PCDATA)>

]>
```

Documento DTD (y II)

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

<!DOCTYPE customer SYSTEM "customer.dtd">

<customer id="PELP1234">
    <name>Pepe Le Pew</name>
    <address>Looney Tunes</address>
    <account>
        <id>MERRY MELODIES</id>
        <since>1945</since>
        <balance>-24,98</balance>
    </account>
</customer>
```

XML Schema

- “Diagrama representativo” de la estructura
- Los DTD están limitados
  - Heredados de SGML
  - Soporte pobre para tipos y espacios de nombres
  - No es XML
- Los “Schemata” permiten una descripción más rica y flexible
  - Incluso rangos de valores de los elementos o atributos

Documento XML Scherma

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="customer" type="CustomerType"/>
  <xsd:complexType name="CustomerType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="address" type="xsd:string"/>
      <xsd:element name="account" type="AccountType‟ minOccurs="0"/>
      <xsd:element name="importantcustomer" type="YesNoType‟ minOccurs="0"
default="no"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="AccountType">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
      <xsd:element name="since" type="xsd:gYear"/>
      <xsd:element name="balance" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="YesNoType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="yes"/>
      <xsd:enumeration value="no"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Espacios de nombres

- También conocidos como “Namespaces”
  - Similar los de C++/C# y los paquetes de Java
- Puede que coincidan nombres de elementos en diferentes esquemas
- A través de “namespaces” podemos identificar unívocamente el significado de los elementos
- Se definen a través de una URI (normalmente una URL Web)

Ejemplo de “Namespaces”

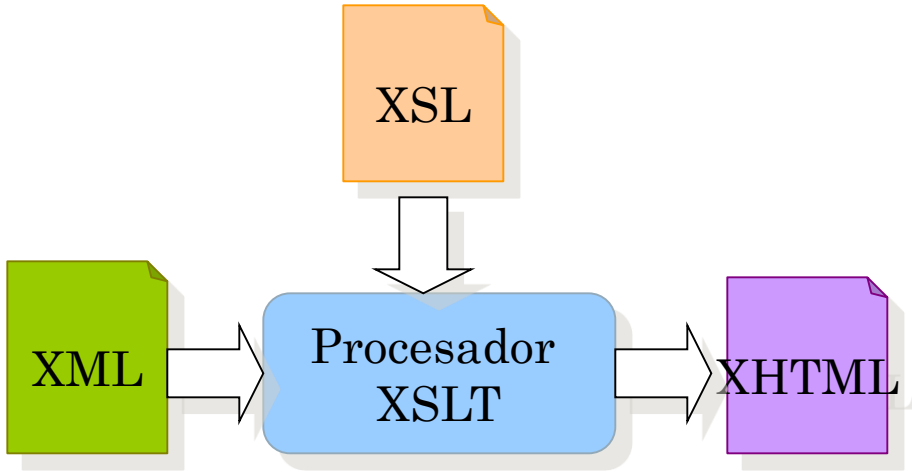
```
<mycust:customer id="AYX1234"
xmlns:mycust="http://www.captiva.es/schema/customer">
  <mycust:name>Christopher</mycust:name>
  <mycust:address>none</mycust:address>
  <mycust:account>
    <mycust:id>1234AD</mycust:id>
    <mycust:since>1997</mycust:since>
    <mycust:balance>-24,98</mycust:balance>
  </mycust:account>
</mycust:customer>
```

XSL

- Podemos transformar los documentos XML a otros formatos:
  - Basados en XML
  - O no basados en XML: binarios y texto
- Se definen dos lenguajes (XML)
  - XSL Transformation (XSLT)
  - XSL Formating Objects (XSL:FO)

XSLT

- Se trata de un lenguaje que define un conjunto de comandos para transformar los documentos
  - Selección de la información
  - Operaciones condicionales
  - Bucles
  - Combinación de documentos



DTDs.

El control de la DTD.

- La DTD proporciona:
  - Una sintaxis formal que sirva de guía a un intérprete/analizador (*parser*).
  - La habilidad de definir valores predeterminados para los atributos.
  - Especificaciones para la estructura.
- Una DTD es una buena manera (pero no la única) de controlar la creación de datos.

Declaración de elemento **ELEMENT.**

`<!ELEMENT`	Apertura y palabra clave.
`Nombre_elemento`	Nombre del elemento.
`( . . . )`	Modelo de contenido o
`PALABRACLAVE`	contenido declarado.
`>`	Cierre.

Palabras clave para el contenido declarado:

`EMPTY`	Sin elemento o contenido.
`ANY`	Cualquier combinación de elementos descendientes y datos caracter.

Modelo de contenido.

- Elementos o #PCDATA.
- Conectores.

,	seguido de	`(a,b)`
|	uno u otro	`(a|b)`

- Indicadores de ocurrencia.

	Uno y solo uno	`configuracion`
?	Cero o uno	`Nombre?`
+	Uno o más	`Controlador+`
*	Cero o más	`Opciones*`

Ejemplos de modelos de contenido.

```
(Titulo, Seccion+)
```

```
(Titulo, (Parrafo+ | Seccion+))
```

```
(Titulo, (Parrafo | Seccion)+)
```

```
(Nombre, Numero, (Articulo, (Cantidad | Lote), Descripcion,
 precio)+, Descuento*)
```

```
<!ELEMENT Capitulo (Titulo, Seccion+)>
```

Contenido mixto.

- Caracteres (#PCDATA) que aparecen solos o en combinación con elementos descendientes en un modelo de contenido.
- Pueden ser expresados en combinaciones como un grupo o un contenido repetible:

```
(#PCDATA | grafico | tabla | lista)
```

- El mismo elemento descendiente no puede aparecer más de una vez en el grupo.

```
<!ELEMENT parrafo (#PCDATA | lista)*>
```

Comentarios XML.

- Los comentarios pueden aparecer en cualquier parte del documento fuera de otros marcajes.
- Pueden aparecer dentro de la declaración de tipo de documento.
- Un procesador XML puede, pero no requiere, ser capaz de leer y recuperar los comentarios.

```
<!-- Articulos secundarios para BD,
                revisado el 2000/I/29 -->
```

Ejemplos de declaraciones de elementos.

```
<!--    Nombre          Modelo contenido                -->
<!ELEMENT       clima   (ciudad+)                       >
<!ELEMENT       ciudad  (nombre, reporte)               >
<!ELEMENT       nombre  (#PCDATA)                       >
<!ELEMENT       reporte (alta, baja, precip?)           >
<!ELEMENT       alta    (#PCDATA)                       >
<!ELEMENT       baja    (#PCDATA)                       >
<!ELEMENT       precip  EMPTY                           >
```

Declaración de atributos **ATTLIST.**

`<!ATTLIST`	Apertura y palabra clave.
`Nombre_elemento`	Nombre del elemento.
`Nombre_atributo`	Nombre del atributo.
`( . . . )`	Lista de valores o
`PALABRACLAVE`	valor declarado.
`" . . . "`	Valor predeterminado o
`#PALABRACLAVE`	palabra clave de valor predeterminado.
`>`	Cierre.

Ejemplos de declaracions de atributos.

```
<!ELEMENT Novela       (titulo, parrafo+)>

<!ATTLIST Novela

  Copyright             CDATA          #REQUIRED

  PalabraClave          CDATA          #IMPLED

        type    (original|revisada|adaptada) “original”

  Estante               CDATA          #REQUIRED>

...

<Novela Copyright=“1998 Ed. Diana” Estante=“i1022”>

...

</Novela>
```

Ejemplos de elementos con atributos.

```
<!--    Nombre          Modelo contenido                    -->
<!ELEMENT       clima   (ciudad+)                           >
<!ELEMENT       ciudad  (nombre, reporte)                   >
<!ELEMENT       nombre  (#PCDATA)                           >
<!ELEMENT       reporte (alta, baja, precip?)   >
<!ELEMENT       alta    (#PCDATA)                           >
<!ELEMENT       baja    (#PCDATA)                           >
<!ELEMENT       precip  EMPTY                               >
<!ATTLIST       precip  total_dia       CDATA   #REQUIRED
  tipo          (lluvia | nieve)        “lluvia”
  fuerza (ligera | fuerte)      #IMPLIED        >
```

Declaracion de documento **DOCTYPE.**

<!DOCTYPE	Apertura y palabra clave.
Elemento_raiz	Nombre del elemento raíz.
PALABRACLAVE	SYSTEM o PUBLIC y
“dtd.dtd”	una DTD XML externa o
[. . .]	declaraciones internas.
>	Cierre.

Ejemplos de declaraciones de documento.

```
<!DOCTYPE novela [
<!ELEMENT      novela  (titulo, parrafo+)          >
<!ELEMENT      titulo  (#PCDATA)                   >
<!ELEMENT      parrafo (#PCDATA)                   >
]>
```

```
<!DOCTYPE novela      SYSTEM  "novela.dtd"   >
```

Mitos de las DTDs de XML.

- El DTD clarifica el significado del documento.
  - No necesariamente. La DTD solo especifica el orden de los elementos de un documento, no su significado.
- Es posible intercambiar información ciegamente usando una DTD.
  - No. La DTD sirve para asegurarse de que todos los involucrados usan la misma estructura.

Ejemplo completo (XML+DTD).

```
<?xml version="1.0"?>
<!DOCTYPE clima [
<!--                  Nombre    Modelo contenido    -->
<!ELEMENT clima      (ciudad+)                  >
<!ELEMENT ciudad     (nombre, reporte)          >
<!ELEMENT nombre     (#PCDATA)                  >
<!ELEMENT reporte    (alta, baja, precip?)      >
<!ELEMENT alta       (#PCDATA)                  >
<!ELEMENT baja       (#PCDATA)                  >
<!ELEMENT precip     EMPTY                      >
<!ATTLIST precip     total_dia       CDATA      #REQUIRED
     tipo       (lluvia | nieve)    "lluvia"
     fuerza     (ligera | fuerte)   #IMPLIED   >
]>
```

DTD incluida con el XML.

```
<clima>
   <ciudad>
      <nombre>Mexico DF</nombre>
      <reporte>
         <alta>27</alta>
         <baja>18</baja>
         <precip total_dia="0" tipo="lluvia" fuerza="ligera"/>
      </reporte>
   </ciudad>
   <ciudad>
      <nombre>Monterrey</nombre>
      <reporte>
         <alta>42</alta>
         <baja>36</baja>
         <precip total_dia="0" tipo="lluvia" fuerza="fuerte"/>
      </reporte>
   </ciudad>
</clima>
```

Código XML.

2 elementos de segundo nivel de ejemplo.

Ejemplo completo (XML+DTD).

```
...
<!DOCTYPE clima [
<!--                Nombre    Modelo contenido    -->
<!ELEMENT clima     (ciudad+)                >
<!ELEMENT ciudad    (nombre, reporte)        >
<!ELEMENT nombre    (#PCDATA)                >
<!ELEMENT reporte   (alta, baja, precip?)    >
<!ELEMENT alta      (#PCDATA)                >
<!ELEMENT baja      (#PCDATA)                >
<!ELEMENT precip    EMPTY                    >
<!ATTLIST precip    total_dia      CDATA     #REQUIRED
     tipo      (lluvia | nieve)    "lluvia"
     fuerza    (ligera | fuerte)   #IMPLIED  >
]>
...
```

Ejemplo completo (XML+DTD).

```
...
<clima>
   <ciudad>
      <nombre>Mexico DF</nombre>
      <reporte>
         <alta>27</alta>
         <baja>18</baja>
         <precip total_dia="0" tipo="lluvia" fuerza="ligera"/>
      </reporte>
   </ciudad>
   <ciudad>
      <nombre>Monterrey</nombre>
      <reporte>
         <alta>42</alta>
         <baja>36</baja>
         <precip total_dia="0" tipo="lluvia" fuerza="fuerte"/>
      </reporte>
   </ciudad>
</clima>
```

```
clima     (ciudad+)
ciudad    (nombre, reporte)
nombre    (#PCDATA)
reporte   (alta, baja, precip?)
alta      (#PCDATA)
baja      (#PCDATA)

precip    EMPTY
precip    total_dia CDATA #REQUIRED
tipo      (lluvia | nieve) "lluvia"
fuerza    (ligera | fuerte) #IMPLIED
```

Alternativas a las DTDs.

- Para datos generados automáticamente:
  - Scripts.
  - Programas.
- Para datos generados por personas:
  - Formularios.
  - Scripts de conversión.
  - Editores restringidos.
  - “Guías de uso y estilo”.
- Esquemas W3C.
  - Nueva especificación del W3C.

XML Schemas

Definición de URL

- URL = Uniform Resource Locator.

- Es la extensión a una red de computadores del sistema de direccionamiento de un sistema de ficheros.

- No solo nombra directorios y ficheros, sino también consultas (“queries”), es decir, recursos “dentro” de bases de datos.

- “Si está ahí fuera podemos apuntar hacia él”.

Sintaxis de las URL

http://www.acl.lanl.gov/URI/archive/uri-archive.index.html

- Especificador de esquemas de identificación seguido por una cadena cuyo formato depende del esquema de identificación.

1. PrePrefijo:
   - Toda URL debe comenzar por URL:

2. Esquema:
   - "Nombre_Esquema": ("//" indica presencia del protocolo)
   - Los esquemas que hacen referencia a un protocolo Internet suelen tener la misma estrucutra.

Sintáxis de las URL

- Nombre de usuario (opcional)
  - “Nombre_usuario:password(opt)@”
- Nombre de dominio Internet.
  - Como alternativa, el número de IP.
- Número de puerto.
  - Existe un valor por defecto.
- Path:
  - Como se comunica el cliente con el servidor, incluyendo información paraser procesada por éste.

Sintáxis de las URL

- Ejemplos de protocolos:
  - http
  - ftp
  - Gopher
  - mailto
  - News
  - rlogin
  - telnet, etc.

URL, URI, URN

- La Web es un espacio de información.
  - HTML es un formato de presentación.
  - HTTP es un protocolo de comunicación.

- Una URI es un punto en el espacio.

- URI asegura la univocidad de nombres:
  - Independiente de la tectología usada.
  - Independiente del protocolo de acceso.
  - Identificación de recursos en Internet.

- URI = Uniform Resource Identifiers.

URL, URI, URN

- Dos URI son idénticas si lo son carácter a carácter:

  www.pagina.com <-> www.Pagina.com

- URL = URI + protocolo de acceso.
  - Muchas URI son URL: son direcciones.

- URN = Uniform Resource Name
  - Identificador independiente de la localización.

    urn:ISBN:0-7897-2242-9
  - PURL: Persistent URL (www.purl.org).

Necesidad de los espacios de nombres

- Solución a los conflictos de nombres (extensible).
  - Un mismo nombre puede tener significados diferentes en contextos diferentes .
- Espacio universal de nombres para XML:
  - Poder utilizar los mismos nombres (etiquetas) en diferentes dominios de problema.
- Propuesta: asociar un prefijo con referencia global (y única) a cada elemento.
  - Nombre_global=prefijo+nombre_local.
- Un prefijo para cada contexto.

Espacios de nombres: Definición

- Hay que declarar los prefijos para poderlos usar.
- La declaración asocia una URI con cada prefijo.
- Los “namespaces” dependen del mecanismo de registro de URI’s.

```
<doc    xmlns:en1="http://www.pagina.com/javalab/1.0"
        xmlns:en2="http://www.pagina2.com/mmlab/1.5"
        xmlns="http://www.midominio.org">
  <en1:libro> ... <en1:libro/>
  <libro> ... <libro>
  <en2:libro> ... <en2:libro/>
<doc/>
```

Espacios de nombres: ámbito y validación

- Si definimos el espacio de nombres en el elemento raiz del documento,éste afecta a todos los elementos del documento.
- Si lo definimos en un elemento cualquiera, sólo afecta a él mismo y a todos sus elementos hijo.
- Se pueden cualificar atributos, pero no tiene casi utilidad.
- Con el modelo de DTD no se pueden validar elementos que utilicen espacios de nombres si no se declaran tal cual.
- Con otros modelos de esquema XML (XML-Schema) SI.

Problemas de los DTD

- Fueron concebidos para sistemas de publicación (SGML): contenidos textuales.
- No tienen tipado de elementos (declaraciones globales).
- Las cardinalidades son: 0, 1, infinito.
- Sintaxis especial y poco clara (no es XML).
- En XML imponen un orden (no hay '&').
- XML, por su orientación Web, tiene necesidades nuevas respecto a SGML.
- No permiten la reutilización sencilla de código.
- No pueden validar espacios de nombres.

XML Schema aporta ...

- Utiliza XML como sintaxis.
- Soporta “Tipado de Datos”.
- Ofrece características de Orientación a Objetos:
  - Tipos de Datos.
  - Herencia y reutilización de tipos.
  - Etc.
- Da mayor control sobre la creación de documentos XML.
- Permite validar XML de formas diferentes.
- Casi todos procesadores XML lo soportan.
- Cada vez más editores lo soportan.

Modelo de datos

- XML-Schema ⇔ DDL – ODL (para BD).
  - Un documento XML como instancia de BD.
  - BD relacional: modelo relacional de tablas.
  - BDOO: modelo de datos OO.
  - Documento XML: Modelo de datos jerárquico.

- Definición de:
  - Estructuras.
  - Restricciones de integridad.
  - TAD's.

Validación de esquema.

- Definir “clases” de documentos XML.
- Instancia: documento XML que cumple con una definición de “clase”.
- Comprobación de semántica del documento XML respecto del modelo de datos.
- Diferencia con modelos OO: no hay definición de dinámica (métodos).

Propuestas para esquemas XML

- XML-Data (Microsoft).
- DCD (Document Content Description).
  - Versión simplificada de XML-Data.
- DDML (Document Definition Markup Lang.).
  - Desarrolado por la XML-Dev mailing list.
- SOX (Schema for O-O XML).
- XML-Schema (W3C) !!!

La Recomendación XML-Schema del W3C

- Aprobada el :
- La documentación consta de tres partes:
  - XML Schema Part 0: Primer (http://www.w3.org/TR/xmlschema-0/) : es un documento introductorio (no muy teórico) y con múltiples ejemplos.
  - XML Schema Part 1: Structures (http://www.w3.org/TR/xmlschema-1/) : documento que describe los mecanismos de construcción de las estructuras de datos. También define las reglas de validación.
  - XML Schema Part 2: Datatypes (http://www.w3.org/TR/xmlschema-2/) : documento que define los Tipos de Datos primitivos predefinidos.

Esquemas para esquemas

- Un documento XML-Schema es un modelo (meta-documento) que define un tipo de documentos XML.

- Un documento XML-Schema es un documento XML

  → debe haber un modelo para dicho documento

  = meta-modelo (meta-meta-documento):

  XML Schema para XML Schemas (xsd.xsd)

- También existe un DTD para XML Schemas (xsd.dtd).

Un esquema básico

- Un XML-Schema es un documento XML.
  - Por convenio llevan extensión ".xsd"

- Comienza con la declaración de documento XML.

- Utiliza la declaración del espacio de nombres de XML-Schema para sus meta-elementos.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2000/10/XMLSchema">
  ...
</xs:schema>
```

Comentarios en Schema

XML-Schema ofrece dos formas de insertar comentarios. Siempre dentro del elemento xs:annotation

- xs:documentation :es la forma de introducir documentación de código.

- xs:appinfo :está orientada a pasar información extra a las aplicaciones que hagan uso del Schema.

Pueden incluirse en cualquier parte del XML_Schema.

Comentarios en Schema (2)

- Mejoran la legibilidad (humana) del código.

```
<xs:annotation>
      <xs:documentation>
            texto
      </xs:documentation>
</xs:annotation>
```

- También se pueden utilizar los comentarios de XML.

```
<!- Comentario  -->
```

- Es preferible utilizar el tipo de comentarios propios de XML-Schema.
  - Se pueden estructurar.
  - Se pueden procesar como un documento XML.

Declaración Global o Local

- En un DTD todos los identificadores de elementos son globales (no se pueden repetir).
  - Siempre que aparezca en el documento XML tiene la misma definición.

- En XML Schema el contexto de uso puede influir en la semántica.
  - Un mismo identificador de Tipo de Dato o de Elemento puede aparecer con definiciones diferentes en contextos diferentes.
  - La visibilidad (ámbito) de una declaración depende del contexto en el que se realiza.

Tipos simples y complejos

El Tipo de Datos (TD) de los elementos puede ser:

- Elementos de *tipo simple*: sólo pueden contener tipos simples (texto), ni siquiera atributos.
- Elementos de *tipo complejo*: pueden contener elementos hijos y atributos.
- Los atributos son considerados como elementos simples, pues sólo pueden contener texto.

- Ambos TD pueden ser definidos con nombre (y reutilizables) o anónimos (definición interna, y no reutilizables).

Definición de Tipos de Datos simples

- Existen predefinidos (en la Recomendación W3C) un conjunto de tipos simples divididos en dos grupos (la diferencia es un poco arbitraria):
  - Tipos Primitivos.
  - Tipos Derivados: definidos en función de los primitivos.

- Un usuario puede definir nuevos tipos simples propios por derivación de los existentes.

- Declaración de elemento con tipo simple:

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

<xs:element name="peso" type="xs:integer"/>
```

Tipos simples (2)

- La lista completa de los tipos simples predefinidos está en:
  - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

- Los tipos simples predefinidos deben comenzar siempre con el prefijo "xs:" correspondiente al namespace de XML-Schema.

- Los tipos simples de usuario se pueden definir globales, o bien localmente dentro de un tipo complejo.

Tipos “Fecha” y “Tiempo”

- Para representar cantidad de tiempo transcurrido:
  xs:timeDuration → formato: PnYnMnDTnHnMnS
- Instante de tiempo:
  xs:time → hh:mm:ss.sss
- Otros tipos de este grupo:
  xs:timeInstant xs:date
  xs:month xs:year xs:century
  xs:recurringDate xs:recurringDay

<xs:element name=“gestación” type=“xs:timeDuration>

<gestación>P9M15D</gestación>

Tipos numéricos

xs:decimal

xs:integer

xs:positiveInteger

xs:negativeInteger

xs:nonPositiveInteger

xs:nonNegativeInteger

xs:float

xs:double

Derivación de Tipos Simples

Existen tres mecanismos de derivación:

- Por **restricción**: se restringe el rango de valores del tipo de datos.
- Por **enumeración** (lista): el rango se describe explícitamente como la lista de valores válidos.
- Por **unión** : mediante unión de dos o más tipos de datos.
  - El mecanismo de **extensión** sólo se aplica a los tipos de datos complejos.

Derivación por restricción: mediante patrones

Son expresiones regulares basadas en la sintaxis Perl.

. = cualquier carácter.

\d = un dígito \D = un no-dígito

\s = cualquier carácter blanco (nueva-línea, tab, retorno de carro).

\S = cualquier carácter no-blanco.

* = cardinalidad 0..N ? = opcionalidad

+ = cardinalidad 1..N [abc] = sólo uno del grupo

[0-9] = rango de valores

(xy) = "y" conjunción | = operador "or" lógico

X{5} = 5 ocurrencias de x x{5,} = al menos 5 ocurrencias de x

X{5,8} = mínimo 5 y máximo 8 ocurrencias de x

(xyz){2} = dos ocurrencias de xyz

Derivación mediante patrones: ejemplo

```
<xs:simpleType name="TipoCódigoPostal">
  <xs:restriction base="xs:string">
        <xs:pattern value="\d{5}(-\d{4})?"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="códigoPostal" type="TipoCódigoPostal">
                    ----------------------------------------
<códigoPostal>12345-9876</códigoPostal>
<códigoPostal>31415</códigoPostal>
```

Tipos Simples Anónimos

- Pueden definirse tipos internamente sin nombre.
  - No se pueden usar en la definición de otro elemento.

```
<xs:element name="códigoPostal">
  <xs:simpleType>
  <xs:restriction base="xs:string">
        <xs:pattern value="\d{5}(-\d{4})?"/>
  </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricción por rangos

```
<xs:simpleType name="fechaCreación">
  <xs:restriction base="xs:date">
        <xs:nimInclusive value="2001:09:27"/>
  </xs:restriction>
</xs:simpleType>
```

xs:nimExclusive

xs:maxInclusive

xs:maxExclusive

Otros tipos de restricciones

- Limitar la longitud de una cadena:

  <xs:length value="x">
  <xs:maxLength value="m">
  <xs:minLength value="n">

- Limitar el número de digitos de una cifra:

  <xs:precision value="n"> : máximo número de dígitos totales.

  <xs:scale value="m"> : máximo numero de dígitos a la derecha del punto decimal.

Restricción por enumeración

En la instancia, los valores deben estar separados por “blanco”.

```
<xs:simpleType name="Paises">
  <xs:restriction base="xs:string">
        <xs:enumeration value="España"/>
        <xs:enumeration value="Francia"/>
        <xs:enumeration value="Portugal"/>
        <xs:enumeration value="Italia"/>
  </xs:restriction>
</xs:simpleType>

<pais>España</pais>
```

Tipo Simple derivado “Lista”

- Las listas son siempre tipos derivados que permiten varios valores en el contenido de un elemento
- Los valores de la lista se separan mediante “blancos”.

```
<xs:simpleType name="listaFechas">
  <xs:list base="xs:date"/>
</xs:simpleType>

<xs:element name="listaCumpleaños" type="listaFechas">

<listaCumpleaños>
  1975-03-25 1999-05-03 2001-09-27
</listaCumpleaños>
```

Tipo Simple derivado “Unión”

- Se componen de al menos dos tipos de datos alternativos.
- El orden en que se definen los TD en la unión es significativo: los datos se intentan validar en ese orden.

```
<simpleType name="NumRomano">
  <restriction base="xs:string">
        <enumeration value="I"/>
        <enumeration value="II"/>
        <enumeration value="III"/>
  </restriction>
</simpleType>
```

Ejemplo de Unión

```
<simpleType name="CapituloRom">
  <list itemType="NumRomano"/>
</simpletType>
<simpleType name="CapituloDec">
  <list itemType="xs:integer"/>
</simpletType>
<simpleType name="union.Capitulos">
  <union memberTypes="CapituloRom CapituloDec"/>
</simpletType>

<numerosCap>1 2 3 4 5</numerosCap>
<numerosCap>I II III</numerosCap>
```

Definición de tipos complejos

- Cuando un elemento puede contener otros elementos hijos o atributos.

- Existen seis clases de tipos complejos según su modelo de contenido:
  - Elementos y atributos.
  - Vacíos: sólo contienen atributos.
  - Texto y atributos.
  - Mixtos: texto, elementos y atributos.
  - Sólo Texto.
  - ANY.

Modelo de Contenido con elementos y atributos

- Pueden contener Elementos y Atributos, pero no texto.

```
<xs:complexType name="TipoAnimal">
  <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="género" type="tipoGénero"/>  <xs:element name="especie" type="tipoEspecie"/>
  </xs:sequence>
  <xs:attribute name="código" type="xs:string"/>
</xs:complexType>
```

- Los elementos deben aparecer en orden.
- Puede contener otros grupos anidados.
- Es el equivalente a la (,) de los DTD.

Declaración de Elementos

- La declaración de un elemento consiste en asociar un nombre de elemento con un Tipo de Datos.

- El TD se puede declarar de dos formas:
  - Incluido: anidado dentro de la declaración del elemento.
  - Referenciado: mediante una referencia explícita a un TD declarado con nombre en otro ámbito.

- Si no se especifica ningún TD, se asume un TD por defecto: **ur-type (anyType)**.
  - Puede contener cualquier combinación de texto, elementos y atributos.

Declaración de Elementos: Ejemplo

```
<xs:element name="animal">
  <xs:complexType>
  <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="peso" type="xs:integer"/>
        <xs:element name="género" type="tipoGénero"/>      <xs:element name="especie"
type="tipoEspecie"/>
  </xs:sequence>
  <xs:attribute name="código" type="xs:string"/>
</xs:complexType>
</xs:element>
```

Declaración Global frente a Local de Elementos

- Declaración Global:
  - Deben ser hijos del elemento raiz del Schema: xs:schema
  - Cualquier otro Tipo Complejo puede hacer uso de esa declaración mediante una **referencia** (reutilización de código).
- Declaración Local:
  - Están anidadas dentro de la estructura del Schema en el interior de alguna otra declaración.

Referencia a elementos global: Ejemplo

Estos elementos pueden ser reutilizados mediante referencia desde cualquier definición de tipo complejo.

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

<xs:complexType name="autor">
  <xs:sequence>
        <xs:element ref="nombre">
  ...
</xs:complexType>
```

Control de la cardinalidad

- Hay dos atributos que se pueden incluir en cualquier elemento para indicar un rango en su cardinalidad
  - `minOccurs` : número mínimo de ocurrencias permitidas de un elemento (valor por defecto = 1).
  - `maxOccurs` : número máximo de ocurrencias permitidas de un elemento (valor por defecto = 1).

```
<xs:element name="autor" type="xs:string"
            minOccurs="1" maxOccurs="5"/>
```

- `maxOccurs` tiene predefinido un valor constante `unbounded` para indicar un número ilimitado de veces.
- Estos dos atributos no se pueden declarar en elementos globales, sólo en declaraciones locales.

Restricciones sobre el contenido en la declaración de Elementos

- Mediante el atributo default podemos dar un valor por defecto al elemento.

- El atributo fixed permite dar un valor fijo: el elemento debe ser vacío (y se comporta como default) o tener dicho valor.

```
<xs:element name="asistencia" type="xs:string" default="si"/>
```

- Estas restricciones no eran posibles para elementos en los DTD.

Definiciones agrupadas

- Permite fácil reutilización de código.

```
<xs:schema>

<xs:group name="características Animal">

  <xs:element name="nombre" type="xs:string"/>

  <xs:element name="peso" type="xs:integer"/>

  <xs:element name="género" type="tipoGénero"/>  <xs:element
  name="especie" type="tipoEspecie"/>

</xs:group>
```

- Son el equivalente a las las entidades paramétricas de los DTD.
- Para referenciar un grupo en un punto dado de una definición:

```
<xs:group ref="características Animal"/>
```

Declaración de Atributos

- Siempre aparecen dentro de la definición de un tipo complejo.
  - También puede haber declaraciones globales como con los elementos: son hijos del elemento xs:schema.
- Deben declararse al final de un componente.
- **NO** pueden contener **hijos**.
- Son siempre de **tipo simple**.
- Su declaración **no** impone un **orden de uso** (es desordenada).

Si no se les da un tipo, tienen por defecto el tipo **anySimpleType**:
representa cualquier cadena de caracteres XML válidos.

Cardinalidad y ocurrencia de Atributos

- No se puede especificar cardinalidad como en los elementos:
  - Sólo pueden aparecer una vez dentro de un elemento dado.
- Si no se especifica nada son opcionales.
- Otras posibilidades se declaran como valores del Atributo use :
  - required : el atributo es obligatorio.
  - optional : puede o no aparecer (estado por defecto).
  - prohibited : el atributo no puede aparecer en el elemento.

<attribute name="código" use="required"/>

Valores predefinidos para atributos

Existen dos opciones representadas por sendos atributos en la declaración del Atributo:

- default : valor por defecto.
  - Tanto si el elemento que lo contiene lo incluye o no el procesador de XML incluirá dicho atributo con el valor de este campo.

- fixed : valor fijo.
  - El atributo puede aparecer o no, pero si aparece debe contener sólo dicho valor.

Definición de grupos de atributos

```
<xs:attributeGroup name="característicasImagen">
  <xs:attribute name="localización"
                              type="xs:uri-reference"/>
  <xs:attibute name="alto" type="xs:string"/>
  <xs:attibute name="ancho" type="xs:string"/>
</xs:attributeGroup>
```

- Debe ser declarado globalmente.
- Un grupo de atributos puede tener referencias a otros grupos de atributos.
- Deben declararse al final de un componente.

Referencias a grupos de atributos

- Se realizan dentro de la definición de un tipo complejo.

```
<xs:attributeGroup ref="característicasImagen"/>
```

- Son análogos a las entidades paremetrizadas de los DTD.
- Sólo pueden referenciar a grupos de atributos definidos globalmente.
- Deben declararse al final de un componente.

Meta-elementos de composición: secuencia

```
<xs:complexType name="animal">
  <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="peso" type="xs:integer"/>
        <xs:element name="género" type="tipoGénero"/>      <xs:element name="especie" type="tipoEspecie"/>
  </xs:sequence>
  <xs:attribute name="código" type="xs:string"/>
</xs:complexType>
```

- Los elementos deben aparecer en orden.
- Puede contener otros grupos anidados.
- Es el equivalente a la (,) de los DTD.

Meta-elementos de composición: selección

- Uno y sólo uno de los elementos definidos en el Compositor puede aparecer.

```
<xs:complexType name="direcciónPostal">
<xs:choice>
  <xs:element name="dirección" type="xs:string/>
  <xs:element name="dirCompleta" type ="dirCompleta"/>
</xs:choice>
</xs:complexType>
```

Meta-elementos de composición: Composición sin orden definido

- Los elementos definidos pueden aparecer en cualquier orden, pero no pueden aparecen con repetición.
- Un TC sólo puede tener un compositor all.

```
<xs:complexType name="animal">
<xs:all>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="peso" type="xs:integer"/>
  <xs:element name="género" type="tipoGénero"/>
  <xs:element name="especie" type="tipoEspecie"/>
</xs:all>
</xs:complexType>
```

Compositores: repeticiones y grupos

- Los compositores sequence y choice se pueden combinar y anidar como se quiera (salvo con all).
- A los meta-elementos de composición podemos añadirles atributos de cardinalidad (salvo para all):
  - minOccurs y maxOccurs.
- Podemos definir grupos de composición con nombre que luego sean referenciados dentro de un TC.
- Los grupos de composición pueden llavar los atributos de cardinalidad:
  - minOccurs y maxOccurs.

Modelo de Contenido Mixto

- Puede contener Elementos, Atributos y Texto.
- Se puede especificar el orden y la cardinalidad.

```
<xs:complexType name="parrafo" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="italica" type="xs:string"/>
   <xs:element name="negrita" type="xs:string"/>
   <xs:element name="subrayado" type="xs:string"/>
  </xs:choice>
</xs:complexType>

<xs:element name="texto" type="parrafo">
<texto>En un lugar de la <negrita>Mancha</negrita>de cuyo nombre ... </texto>
```

Modelo de Contenido Sólo-Texto

- Sólo puede contener texto y atributos, pero no elementos hijo.

```
<xs:element name =“Dirección”>
  <xs:complexType>
    <xs:simpleContent>
          <xs:extension base=“xs:string”>
             <xs:attribute name=“códigoPostal” type=”xs:integer”/>
          </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Modelo de Contenido Vacío

- El Elemento no puede tener contenido, pero puede tener Atributos.

```
<xs:element name =“imagen”>
  <xs:complexType>
        <xs:attribute name=“alto” type”xs:integer”/>
        <xs:attribute name=“ancho” type”xs:integer”/>
  </xs:complexType>
</xs:element>

<imagen alto=“25” ancho”50”/>
```

Modelo de Contenido genérico

- Corresponde al modelo ANY de los DTD:
  - xs:any = permite cualquier tipo de elementos.
  - xs:anyAttribute = permite cualquier atributo.

```
<xs:element name="XHTMLSection">
  <xs:complexType>
    <xs:sequence>
      <xs:any minOccurs="0" maxOccurs="unbounded"
                                processContents="lax"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

Valor Cero y valor Nulo

- Es necesario diferenciar entre valor de “cadena_vacía” y valor indefinido o nulo (nil, null).

```
<xs:element name=“Cantidad” type=“xs:integer” nillable=“true”/>
```

```
<Factura xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance>
  <Producto>
        <Cantidad xsi:nill=“true”></Cantidad>
   </Producto>
</Factura>
```

Herencia en la declaración de tipos complejos (1)

- Creación de tipos complejos basados en otros TC: el nuevo TC hereda las características del base y las extiende.
- Permite definir y reutilizar tipos complejos básicos: se puede definir una jerarquía de tipos complejos.

```
<xs:complexType name="DirecciónEspañola">
 <xs:complexContent>
  <xs:extension base="DirecciónBase">
   <xs:sequence>
        <xs:element name="Provincia" type="xs:string"/>
   </xs:sequence>
  </xs:complexContent>
</xs:complexType>
```

Herencia en la declaración de tipos complejos (2)

```
<xs:complexType name="datosPersona">
  <xs:complexContent>
  <xs:extension base="tipoBase">
  <xs:sequence>
        <xs:element name="domicilio" type="xs:string"/>
        <xs:element name="fechaNac" type="xs:date"/>
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="empleado" type="datosPersona"/>
```

Referencias

- XML
  - Aquí está todo: http://www.w3.org/XML/
  - XML-Schema: http://www.w3.org/XML/Schema
- XSL
  - XSLT: http://www.w3.org/TR/xslt
  - Xpath: http://www.w3.org/TR/xpath
  - exslt: http://www.jenitennison.com/xslt/exslt/functions/
- XML y Java
  - Sun XML: http://java.sun.com/xml/
  - Xerces, Xalan y más cosas: http://xml.apache.org/
- Tutoriales
  - De casi todo: http://www.w3schools.com/