XML Schema

Departamento de Informática
Universidad de Oviedo

Lenguajes de Esquemas

Esquema = definición de estructura de un conjunto de documentos XML

Validar = Chequear que un documento sigue un esquema

- Principal Ventaja: Protección de errores
- Otras aplicaciones: Edición, compresión, etc.

DTDs = un ejemplo de esquemas (con varias limitaciones)

XML Schema = desarrollo posterior del W3c

Existen Otros:

- RELAX-NG, Schematron, etc.

Características de DTD's

- Especifican estructura del documento:
  - Elementos, atributos, anidamientos, etc.
- Integridad referencial mínima (ID, IDREF)
- Mecanismo sencillo de abstracción
  - Entidades ≈ Macros
- Inclusión de documentos externos
- Integrados en XML (Parte de la especificación)
- Sencillos de comprender (≈ Expresiones regulares)

Limitaciones de DTD's

- La Sintaxis ***no es XML*** (difíciles de manipular)
- No soportan ***Espacios de nombres***
- No permiten especificar ***tipos de datos*** (por ejemplo: enteros, flotantes, fechas, etc.
- No permiten especificar ***secuencias no ordenadas***

  ((e1,e2,e3)|(e1,e3,e2)|(e2,e1,e3)|...(e3,e2,e1))
- No hay soporte para declaraciones ***sensibles al contexto***: Los elementos se definen todos a nivel de documento, ejemplo, contenido con el mismo nombre cuya estructura cambia en diferentes contextos
- Soporte limitado para ***Referencias cruzadas,*** no es posible formar claves a partir de varios atributos o de elementos
- ***No son extensibles*** (una vez definido, no es posible añadir nuevos vocabularios a un DTD)

XML Schema
Objetivos de Diseño

- Sintaxis XML
- Soporte para Espacios de Nombres
- Mayor expresividad
  - Restricciones numéricas
  - Integridad dependientes del contexto
- Tipos de datos
  - Gran cantidad de tipos de datos predefinidos
  - Creación de tipos de datos por el usuario
- Extensibilidad
  - Inclusión/Redefinición de esquemas
  - Herencia de tipos de datos
- Soporte a Documentación

Ejemplo

alumnos.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.uniovi.es/alumnos"
           xmlns="http://www.uniovi.es/alumnos">
<xs:element name="alumnos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="alumno" minOccurs="1" maxOccurs="200"
               type="TipoAlumno"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="nacim" type="xs:gYear"/>
  </xs:sequence>
  <xs:attribute name="dni" type="xs:integer"/>
 </xs:complexType>
</xs:schema>
```

Elemento raíz schema y espacio de nombres determinado

Permite especificar rangos de inclusión

Permite especificar tipos

Estructura del Schema

El esquema está formado por:

- Elemento raíz: **schema** del espacio de nombres **http://www.w3.org/2001/XMLSchema**
- Atributo: **targetNamespace** indica el espacio de nombres que se está definiendo
- Subelementos:
  - Declaraciones globales de elementos y atributos
  - Definiciones de tipos de elementos y atributos
  - Anotaciones

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.uniovi.es/alumnos"
           xmlns="http://www.uniovi.es/alumnos">
   <xs:element name="alumnos">
     . . .
  </xs:element>
  . . .
</xs:schema>
```

Tipos Complejos vs Simples

Pueden declararse 2 tipos:

- Complejos: Pueden contener sub-elementos y atributos

Ejemplo de Tipo Complejo

```
<alumno dni="9873435">
  <nombre>Jose</nombre>
  <apellidos>Bueno</apellidos>
</alumno>
```

Simples

- Simples: No contienen sub-elementos ni atributos
  - Pueden aparecer dentro de elementos o en valores de atributos

Validación: esquemas e Instancias

- Un documento XML Schema define un conjunto de documentos con una determinada estructura
- Un documento XML puede validarse contra varios esquemas
  - Puede asociaciarse explícitamente mediante el atributo **schemaLocation**
    - Utiliza 2 cadenas, el espacio de nombres y la URL del documento
  - Si no se utiliza espacio de nombres, puede usarse **noNamespaceSchemaLocation**

alumnos.xml

```
<alumnos
  xmlns="http://www.uniovi.es/alumnos"
  xsi:SchemaLocation="http://www.uniovi.es/alumnos
                      alumnos.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
. . .
</alumnos>
```

Validación: esquemas e instancias

alumnos.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.uniovi.es/alumnos"
           xmlns="http://www.uniovi.es/alumnos">
<xs:element name="alumnos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="alumno" minOccurs="1" maxOccurs="200"
               type="TipoAlumno"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexTy
  <xs:sequenc
    <xs:eleme
    <xs:eleme
    <xs:eleme
  </xs:sequen
  <xs:attribu
 </xs:complex
</xs:schema>
```

alumnos.xml

```
<alumnos
  xmlns="http://www.uniovi.es/alumnos"
  xsi:schemaLocation="http://www.uniovi.es/alumnos
                      alumnos.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
. . .
</alumnos>
```

Los espacios de nombres deben coincidir.
También puede usarse:
xsi:noNameSpaceLocation

Tipos Anónimos vs. con nombre

```
<xs:element name="alumno">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:element>
```

+ legible

```
…
<xs:element name="alumno" type="TipoPersona"/>
…
<xs:ComplexType name="TipoPersona">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:ComplexType>
```

+ Reutilizable

Otra posibilidad: Referencias

```
<xs:element name="alumno">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:element>
```

```
<xs:element name="alumnos">
<xs:sequence>
  <xs:element ref="alumno" />
</xs:sequence>
</xs:element>
```

Tipos complejos: Creación a partir de tipos simples

```
<xs:element name="precio">
       <xs:complexType>
         <xs:simpleContent>
           <xs:extension base="xs:decimal">
           <xs:attribute name="moneda" type="xs:string" />
         </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<precio moneda="euros">23.45</precio>
```

Tipos Complejos: Secuencia

Construcción básica mediante secuencia de elementos

```
<xs:element name="alumno">
  <xs:complexType>
  <xs:sequence>
    <xs:element name="nombre"    type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="nacim" type="xs:gYear"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="dni" type="xs:integer"/>
 </xs:complexType>
</xs:element>
```

```
<alumno dni="9399390">
 <nombre>Juan</nombre>
 <apellidos>García García</apellidos>
 <nacim>1985</nacim>
</alumno>
```

Tipos Complejos: Alternativa

choice: Representa alternativas

OJO: Es una o-exclusiva

```
<xs:complexType name="Transporte">
   <xs:choice>
    <xs:element name="coche" type="xs:string"/>
     <xs:element name="tren" type="xs:string"/>
     <xs:element name="avión" type="xs:string"/>
   </xs:choice>
 </xs:complexType>
```

```
<transporte>
<coche>Renault R23</coche>
</transporte>
```

Tipos Complejos: Contenido Mixto

El contenido Mixto permite mezclar texto con elementos

```
 <xs:complexType name="TCom" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="emph" type="xs:string"/>
  </xs:choice>
</xs:complexType>

<xs:element name="comentarios" type="TCom" />
```

```
<comentarios>
 Es un poco <emph>listillo</emph>
</comentarios>
```

Secuencias no ordenadas

all = Todos los elementos en cualquier orden

En DTDs requería enumerar las combinaciones:

(A,B,C)|(A,C,B)|...|(C,B,A)

```
<xs:complexType name="TipoLibro">
  <xs:all>
      <xs:element name="autor" type="xs:string"/>
      <xs:element name="título" type="xs:string"/>
  </xs:all>
</xs:complexType>
<xs:element name="libro" type="TipoLibro" />
```

```
<libro>
 <autor>Juanita la Loca</autor>
 <título>No estoy loca</título>
</libro>
```

```
<libro>
  <título>El kigote</título>
  <autor>Cerbantes</autor>
 </libro>
```

Agrupaciones

Es posible nombrar agrupaciones de elementos y de atributos para hacer referencias a ellas

```
<xs:group name="nombApell">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

```
<xs:complexType name="TipoAlumno">
 <xs:group ref="nombApell" />
 <xs:element name="carrera" type="xs:string"/>
</xs:complexType>
```

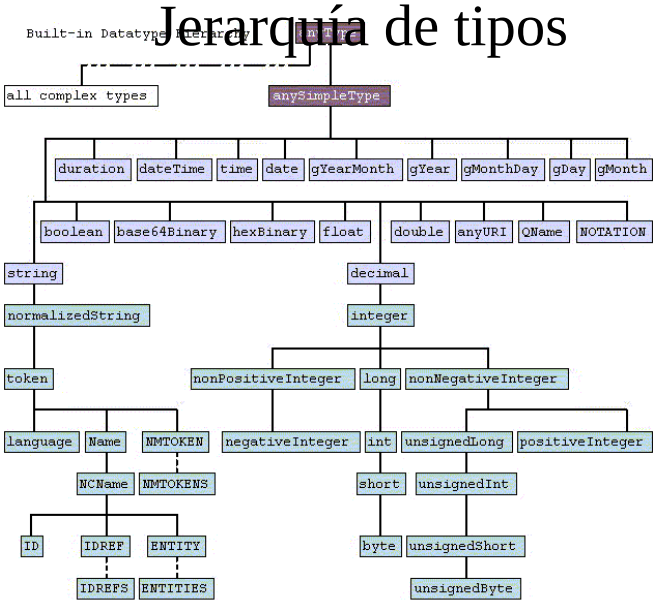
Tipos Simples

- Los tipos simples no pueden contener elementos o atributos
- Pueden ser:
  - Predefinidos o *built-in* (Definidos en la especificación)
    - Primitivos
    - Derivados
  - Definidos por el usuario
    - Restringiendo facetas de tipos predefinidos

Tipos simples Primitivos

- string
- boolean
- number, float, double
- duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth
- hexBinary, base64Binary
- anyURI
- QName = Nombre cualificado con espacio de nombres
- NOTATION = Notación binaria (similar a DTD)

Jerarquía de tipos



Creación de nuevos tipos simples

Facetas

Los nuevos tipos se construyen mediante restricción de facetas:

- *length, minlength, maxlength:* Longitud del tipo de datos
- *pattern:* Restricciones sobre valores mediante expresiones regulares
- *enumeration:* Restringe a una determinada enumeración de valores
- *whitespace:* Define política de tratamiento de espacios (preserve/replace, collapse)
- *(max/min)(in/ex)clusive:* Límites superiores/inferiores del tipo de datos
- *totaldigits, fractionDigits:* número de dígitos totales y decimales

Enumeraciones y Restricciones

Enumeración

```
<xs:simpleType name="TipoCarrera">
<xs:restriction base="xs:token">
 <xs:enumeration value="Gestión"/>
 <xs:enumeration value="Sistemas"/>
</xs:restriction>
</xs:simpleType>
```

Restricciones sobre valores

```
<xs:simpleType name="mes">
<xs:restriction base="xs:integer">
  <xs:minInclusive value="1" />
  <xs:maxInclusive value="31" />
</xs:restriction>
</xs:simpleType>
```

Listas

```
<xs:simpleType name="ListaComponentes">
 <xs:list itemType="TipoComponente"/>
</xs:simpleType>

<xs:simpleType name="Tipoomponente">
<xs:restriction base="xs:nonNegativeInteger">
   <xs:maxInclusive value="255" />
</xs:restriction>
</xs:simpleType>
```

Se pueden aplicar las facetas: length, maxLength, minLength, enumeration

```
<xs:simpleType name="ColorRGB">
   <xs:restriction base="ListaComponentes">
     <xs:length value="3" />
   </xs:restriction>
 </xs:simpleType>
```

```
<color>255 255 0</color>
```

Uniones

```
<xs:simpleType name="TipoNota">
<xs:union>
  <xs:simpleType>
    <xs:restriction base="xs:float">
      <xs:maxInclusive value="10" />
      <xs:minInclusive value="0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="No presentado" />
    </xs:restriction>
  </xs:simpleType>
</xs:union>
</xs:simpleType>

<xs:element name="nota" type="TipoNota" />
```

```
<nota> 5.75 </nota>
```

```
<nota> No presentado </nota>
```

Expresiones regulares

Ejemplos de expresiones regulares

```
<xs:simpleType name="NIF">
 <xs:restriction base="xs:token">
   <xs:pattern value="\d{7,8}[A-Z]" />
 </xs:restriction>
</xs:simpleType>

<xs:element name="nif" type="NIF" />
```

```
<nif>9394173J</nif>
```

```
<nif>11079845M</nif>
```

Expresión	Posibles valores
Elemento \d	Elemento 2
a*b	b, ab, aab, aaab, …
[xyz]b	xb, yb, zb
a?b	b, ab
a+b	ab, aab, aaab, …
[a-c]x	ax, bx, cx

Expresiones Regulares

[a-c]x	ax, bx, cx
[^0-9]x	Carácter ≠ dígito seguido de x
\Dx	Carácter ≠ dígito seguido de x
(pa){2}rucha	paparucha
.abc	Cualquier carácter (1) seguido de abc
(a|b)+x	ax, bx, aax, bbx, abx, bax,...
a{1,3}x	ax, aax, aaax
\n	Salto de línea
\p{Lu}	Letra mayúscula
\p{Sc}	Símbolo de moneda

Tipos Derivados por Extensión

Similar a las subclases de POO: Añadir elementos a un tipo base

```
<xs:complexType name="Figura" >
  <xs:attribute name="color" type="Color"/>
 </xs:complexType>

<xs:complexType name="Rectángulo">
 <xs:complexContent>
 <xs:extension base="Figura">
 <xs:attribute name="base" type="xs:float" />
 <xs:attribute name="altura" type="xs:float" />
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="Círculo">
...similar pero incluyendo el radio
</xs:complexType>
```

Tipos Derivados por Extensión

Los tipos derivados pueden utilizarse en los mismos sitios que la clase base

```
<xs:element name="figuras">
   <xs:complexType>
     <xs:sequence>
      <xs:element name="figura" type="Figura"
                  maxOccurs="unbounded" />
     </xs:sequence>
   </xs:complexType>
  </xs:element>
```

```
<figuras>
<figura base="23" altura="3" xsi:type="Rectángulo" />
<figura radio="3" xsi:type="Círculo" />
</figuras>
```

Es necesario especificar el tipo mediante **`xsi:type`**

Tipos Abstractos

Mediante abstract="true" se declara un tipo como abstracto.

Ese tipo no puede usarse directamente

```
<xs:complexType name="Figura" abstract="true">
  <xs:attribute name="color" type="Color"/>
 </xs:complexType>
```

También es posible limitar la derivación de tipos

final="restriction"

Declaración de Atributos

```
<xs:complexType name="Círculo">
 <xs:attribute name="radio"
               type="xs:float"
               use="required" />

 <xs:attribute name="color"
               type="Color"
               default="255 0 0"/>

 <xs:attribute name="tipo"
               type="xs:string"
               fixed="jpeg" />
</xs:complexType>
```

Por defecto los atributos son opcionales. Indicar que son obligatorios: **`use="required"`**

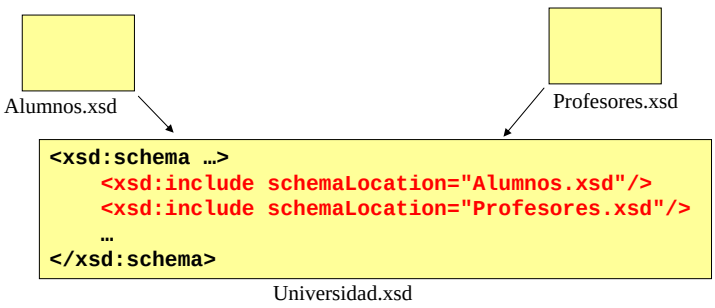
Valor por defecto de un atributo. Podría definirse otro valor.

Valor fijo de un atributo. Si no se define, se utiliza ése. Si se define, debe coincidir.

Inclusión de Esquemas

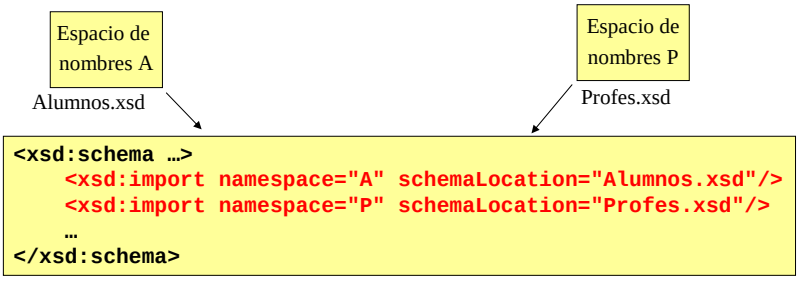
include permite incluir elementos de otros esquemas

- Los elementos deben estar en el mismo espacio de nombres
- Es como si se hubiesen tecleado todos en un mismo fichero



Importación de Esquemas

import permite incluir elementos de otros esquemas con distintos espacios de nombres

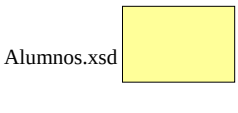



```
<xsd:schema …>
    <xsd:import namespace="A" schemaLocation="Alumnos.xsd"/>
    <xsd:import namespace="P" schemaLocation="Profes.xsd"/>
    …
</xsd:schema>
```

Universidad.xsd

Redefinición de Esquemas

redefine es similar a *include* pero permite modificar los elementos incluidos.




```
<xs:redefine schemaLocation="Alumnos.xsd">
  <xs:complexType name="TipoAlumno">
    <xs:complexContent>
      <xs:extension base="TipoAlumno">
         <xs:sequence>
           <xs:element name="nota" type="Nota" />
         </xs:sequence>
      </xs:extension>
     </xs:complexContent>
   </xs:complexType>
</xs:redefine>
```

AlumnosConNotas.xsd

Claves y Unicidad

- Los DTDs proporcionaban el atributo ID para marcar la unicidad (un valor ID era único en todo el documento)
- XML Schema tiene más posibilidades:
  - Indicar que un elemento es único (**unique**)
  - Definir atributos únicos
  - Definir combinaciones de elementos y atributos como únicos
  - Distinción entre unicidad y claves (**key)**
    - Clave = además de ser único, debe existir y no puede ser nulo.
  - Declarar el rango de un documento en el que algo es único

Claves y Unicidad

```
<xs:complexType name="Alumnos">
 <xs:sequence>
  <xs:element name="Alumno" type="TipoAlumno"/>
 </xs:sequence>
  <xs:key name=“ClaveDNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:key>
</xs:complexType>
```

Es necesario incluir el espacio de nombres (XPath)

La clave puede formarse para atributos y elementos

```
<xs:key name=“ClaveDNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:apells"/>
    <xs:field xpath="a:nombre"/>
</xs:key>
```

Una clave puede estar formada por varios elementos

Claves y Unicidad

```
<xs:complexType name="Alumnos">
 <xs:sequence>
  <xs:element name="Alumno" type="TipoAlumno"/>
 </xs:sequence>
  <xs:unique name="DNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:unique>
</xs:complexType>
```

Unique especifica que debe ser único, pero podría no existir

Referencias a Claves

keyref especifica que debe hacer referencia a una clave (Claves Externas)

```
<xs:element name="clase">
 <xs:sequence>
  <xs:element name="alumnos" ...
  <xs:element name="delegado" ...
 </xs:sequence>

<xs:key name="DNI">
 <xs:selector xpath="a:alumnos/a:alumno"/>
 <xs:field xpath="a:dni"/>
</xs:key>

<xs:keyref name="Delegado" refer="DNI">
  <xs:selector xpath="a:delegado"/>
  <xs:field xpath="a:dni"/>
</xs:keyref>
```

Valores Nulos

Indicar que un elemento puede ser nulo sin estar vacío.

- Vacío (Empty): Un elemento sin contenido
- Nulo (Nil): Un elemento que indica que no hay valor

```
<xsd:element name="Persona">
  <xsd:complexType>
  <xsd:element name="nombre" type="xsd:NMTOKEN"/>
  <xsd:element name="primerApell" type="xsd:NMTOKEN"/>
  <xsd:element name="segundoApell" type="xsd:NMTOKEN"
                                   nillable="true"/>
 </xsd:complexType>
</xsd:element>
```

```
<persona>
      <nombre>John</nombre>
      <primerApell>Smith</primerApell>
      <segundoApell xsi:nil="true"/>
</persona>
```

El segundo apellido puede ser un NMTOKEN o estar indefinido

Incluir cualquier contenido...

any indica cualquier contenido de un determinado espacio de nombres
anyAttribute cualquier atributo de un espacio de nombres

```
<xs:complexType name="Comentario">
 <xs:sequence>
  <xs:any namespace="http://www.w3.org/1999/xhtml"
          minOccurs="1"
          processContents="skip" />
 </xs:sequence>
 <xs:anyAttribute
      namespace="http://www.w3.org/1999/xhtml"/>
</xs:complexType>
```

También puede declararse ##any, ##local, ##other

```
<comentarios>
  <html:p>Es un
     <html:emph>Listillo</html:emph>
  </html:p>
</comentarios>
```

Otros valores
strict = obliga a validar
lax = valida si es posible

XML Schema Limitaciones

No soporta entidades. Mecanismo para crear macros

<!ENTITY &texto; "Esto texto se repite muchas veces" >

Es necesario seguir usando los DTDs ☹

Lenguaje de Restricciones limitado

Ejemplo: ¿Verificar valor total = suma de valores parciales?

Sensibilidad al contexto limitada

Por ejemplo: Especificar que el contenido depende del valor de un atributo

<transporte tipo="coche"> ...</transporte>

<transporte tipo="avión"> ...</transporte>

Tamaño de archivos XML Schema puede ser excesivo

Legibilidad de las especificaciones...XML no siempre es legible

Complejidad de la especificación:

Muchas situaciones/combinaciones excepcionales

Otras Técnicas

Relax NG

Schematron

Otras técnicas
Relax NG

- Relax NG. Desarrollado por OASIS a partir de TREX y RELAX
- Estandarizado por ISO como parte 2 del DSDL (Document Schema Definition Languages)
- Soporta mayor número de restricciones y gramáticas ambigüas
- Incluye una sintaxis abreviada (no XML)

Otras Técnicas
Relax NG

```
<element name="a:alumnos"
        xmlns="http://relaxng.org/ns/structure/1.0"
        xmlns:a="http://www.uniovi.es/alumnos"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <zeroOrMore>
   <element name="a:alumno">
    <attribute name="dni">
        <data type="nonNegativeInteger" />
    </attribute>
    <element name="a:nombre"><text/></element>
    <element name="a:apellidos"><text/></element>
    <element name="a:nota"><data type="double" /></element>
    <element name="a:email"><text/></element>
   </element>
  </zeroOrMore>
</element>
```

Otras Técnicas
Relax NG

Relax NG permite mayor expresividad

Ejemplo:

Un alumno debe tener un atributo DNI o un elemento DNI, pero no ambos

Otras Técnicas
Relax NG

```
<element name="a:alumnos"
        xmlns="http://relaxng.org/ns/structure/1.0"
        xmlns:a="http://www.uniovi.es/alumnos"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
 <zeroOrMore>
  <element name="a:alumno">
    <choice>
      <attribute name="dni"><data type="nonNegativeInteger" /></attribute>
      <element name="a:dni"><data type="nonNegativeInteger" /></element>
    </choice>
   <element name="a:nombre"><text/></element>
   <element name="a:apellidos"><text/></element>
   <element name="a:nota"><data type="double" /></element>
   <element name="a:email"><text/></element>
  </element>
 </zeroOrMore>
</element>
```

Otras Técnicas
Relax NG

Otro ejemplo:

Los alumnos de tipo=“delegado” solamente tienen nombre y apellidos.

Otras Técnicas
Relax NG

```
<element name="a:alumnos"
        xmlns="http://relaxng.org/ns/structure/1.0"
        xmlns:a="http://www.uniovi.es/alumnos"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
 <zeroOrMore>
 <choice>
   <element name="a:alumno">
     <attribute name="tipo"><value>delegado</value></attribute>
     <element name="a:nombre"><text/></element>
     <element name="a:apellidos"><text/></element>
   </element>
  <element name="a:alumno">
   <element name="a:nombre"><text/></element>
   <element name="a:apellidos"><text/></element>
   <element name="a:nota"><data type="double" /></element>
 </element>
 </choice>
 </zeroOrMore>
</element>
```

Otras Técnicas Schematron

- Schematron fue desarrollado por Rick Jelliffe
  - Estandarizado por ISO como parte 3 del DSDL (Document Schema Definition Languages)
  - Utiliza un modelo basado en reglas (en vez de gramáticas)
  - Asocia reglas de validación a expresiones XPath
  - Puede expresar restricciones arbitrarias

Otras Técnicas Schematron

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
   <sch:pattern name="Chequear DNIs">
      <sch:rule context="alumno">
         <sch:assert test="@dni &gt; 10000000 and @dni &lt; 99999999">
            El dni debe estar entre 10000000 y 99999999</sch:assert>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="Chequear nota">
      <sch:rule context="media">
         <sch:assert test="sum(//alumno/nota) div count(//alumno/nota) = .">
            No encaja la media
         </sch:assert>
      </sch:rule>
   </sch:pattern>
</sch:schema>
```

Otras Técnicas
Fundamentos teóricos

Taxonomy of XML Schema languages using Formal Language Theory, M. Murata, D. Lee, M. Mani, K. Kawaguchi, ACM Transactions on Internet Technology, 2005

- Relax NG ≈ Regular tree grammars
- XML Schema ≈ Simple type tree grammars
- DTDs ≈ local tree grammars

Ejercicios

Herramientas:

xmllint: Contiene la opción --schema que valida contra un esquema

```
xmllint --schema alumnos.xsd alumnos.xml
```

xsv (http://www.ltg.ed.ac.uk/~ht/xsv-status.html)

- Herramienta desarrollada en Python
- Funcionamiento a través de Web o en línea de comandos
- Salida en formato XML (difícil de leer al principio)

Diseño Vocabularios XML

Diseño Vocabularios XML

- Separación tradicional de dos mundos
  - Sistemas orientados a Datos
    - Información uniforme y fuertemente estructurada (ej. Tablas)
    - Mucha cantidad de información repetida
    - Objetivo: Procesamiento eficiente (Almacenes de datos)
  - Sistemas orientados a Documentación
    - Información poco uniforme y entrelazada (ej. Libros)
    - No existe un patrón uniforme
    - Objetivo: Comunicación, Presentación (Mensajes)
- Se podría añadir un tercer mundo:
  - Programación Orientada a Objetos
  - Propuestas para añadir capacidad de programación a documentos
- XML: Información semi-estructurada (Lugar intermedio)
  - Estructuras jerárquicas entrelazadas

Orientación del esquema
Datos vs. Documentos

```
<factura>
 <emisión>23-2-2006</emisión>
 <envío>3-3-2006</envío>
 <dirección>
   <nombre>Luis Caro</nombre>
   <calle>Uría</calle>
   <numero>2</numero>
   <ciudad>Gijón</ciudad>
   <cp>33021</cp>
 </dirección>
 <tlfno>985102012</tlfno>
</factura>
```

Orientado a Datos

```
<nota fecha="23-2-2006">
 <de>Juan Lopez</de>
 <para>Luis Caro</para>
 <asunto>Encargo</asunto>
 <msg>
  Necesito darte el encargo.
  Puedes enviarme un correo a
  <email>caro@gijon.com</email>
  o me llamas al
  <tlfno>985102012</tlfno>
 </msg>
</nota>
```

Orientado a Documentos

Diseño
Vocabularios XML

- Características a tener en cuenta
  - Tamaño de documentos
  - Facilidad de escritura
  - Facilidad de procesamiento
  - Flexibilidad (ej. HTML es muy flexible, Bases de Datos = menos)
  - Consistencia: Evitar características incoherentes
  - Nivel de abstracción: Buscar término medio en nivel de detalle
    - <fecha>10 Marzo 2003</fecha>
    - <fecha><día>10</dia><mes>Marzo</mes><año>2003</año></fecha>
- Patrones de diseño:
  - www.xmlpatterns.com

Ejemplo de Discusión

Representación de propiedades

```
<pizza
    nombre="Margarita"
    precio="6" />
```

¿Atributos o Elementos?

```
<pizza>
    <nombre>Margarita </nombre>
    <precio>6</precio>
</pizza>
```

Razones filosóficas:
- Atributos: valores asociados con objetos sin identidad propia (edad)
- Subelementos: valores con identidad propia (fecha-nacimiento)

Orígenes (SGML):
- Atributos: meta-información (información sobre el contenido)
- Subelementos: Contenido

Ejemplo de Discusión

Representación de propiedades

```
<pizza
    nombre="Margarita"
    precio="6" />
```

¿Atributos o Elementos?

```
<pizza>
    <nombre>Margarita </nombre>
    <precio>6</precio>
</pizza>
```

En los DTDs
- Pueden incluirse restricciones sobre su valor
  - Ej. valor "si" o "no"
- Pueden definirse valores por defecto
- Pueden validarse los valores ID e IDREF
- Pueden definirse restricciones sobre espacios en blanco (NMTOKENS)

Ocupan menos espacio
Más fáciles de procesar (SAX y DOM)
Acceso a entidades externas (datos binarios)

Soportan valores arbitrariamente complejos y repetidos
Establecen un orden
Soportan *atributos de atributos*
Mayor flexibilidad ante modificaciones

Diseño Vocabularios
En resumen...

...Aparición de una nueva torre de Babel...

Algunos Consejos:

- Estudiar dominio de la Aplicación (ver estándares ya definidos!!!)
- Considerar futuras ampliaciones (extensibilidad)
- Validar antes de que sea tarde
- Usar espacios de nombres
- etc. etc.

Fin