


# Nuevas Tecnologías de la Programación




Universidad  
de Huelva

Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML

1. Antecedentes
2. Fundamentos
3. Esquemas: DTD's y Schema XML (XSD's)
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT
  - 4.3. Arquitectura DOM





# Nuevas Tecnologías de la Programación



Universidad  
de Huelva

Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML

- 1. Antecedentes**
2. Fundamentos
3. Esquemas: DTD's y Schema XML (XSD's)
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT
  - 4.3. Arquitectura DOM





1



- Es un conjunto de reglas y tecnologías, que permiten definir nuevos lenguajes basados en etiquetas, para la creación de documentos para el intercambio de información estructurada entre aplicaciones informáticas.

- NO, ya existían lenguajes y meta-lenguajes de marcas

- SGML (Standard Generalized Markup Language) **Metalenguaje**
- HTML (HyperText Markup Language) **Lenguaje**



## Fundamentos

- W3C
  - Desarrollado y Mantenido por World Wide Web Consortium W3C  
<http://www.w3.org/> - (oficina española en <http://www.w3c.es>)  
<http://www.w3.org/XML>
- Características
  - Meta-Lenguaje de marcas
  - Define la estructura y semántica; pero no el formato de presentación
- Algunas Aplicaciones
  - MathML (Mathematical Markup Language)
  - CML (Chemical Markup Language)
  - RSS (Really Simple Syndication)
  - WML (Wireless Markup Language)
- Desarrollo
  - Editores
  - Parsers
  - Navegadores y otras aplicaciones

## Ejemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Mi Biblioteca -->
<biblioteca>
  <libro cod="0001">
    <titulo>XML en 24 horas</titulo>
    <autor>
      <nombre>Jonh Palmer</nombre>
      <apellido> Brian </apellido>
    </autor>
    <estado prestado="no"/>
  </libro>
  <libro cod="0002">
    <titulo>XML en 48 horas</titulo>
    <autor>
      <nombre>Juan Ramón</nombre>
      <apellido>López</apellido>
      <apellido>Prats</apellido>
    </autor>
    <isbn>8-12345-09</isbn>
    <estado prestado="si"/>
  </libro>
</biblioteca>
```

Biblioteca

Libro

Libro

Documento XML, presenta

- Información estructurada
- con significado

Pero,

- No establece el formato de presentación

## Ejemplo

**Etiquetas**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Mi Biblioteca -->
<biblioteca>
  <libro cod="0001">
    <titulo>XML en 24 horas</titulo>
    <autor>
      <nombre>Jonh Palmer</nombre>
      <apellido>Brian</apellido>
    </autor>
    <estado prestado="no"/>
  </libro>
  <libro cod="0002">
    <titulo>XML en 48 horas</titulo>
    <autor>
      <nombre>Juan Ramón</nombre>
      <apellido>López</apellido>
      <apellido>Prats</apellido>
    </autor>
    <isbn>8-12345-09</isbn>
    <estado prestado="si"/>
  </libro>
</biblioteca>
```

**Atributos**

Documento XML, contiene:

- Etiquetas o marcas  
`<et>... </et>`  
`<et ... />`
- Atributos de etiquetas  
`<et at=".." >`
- Contenido  
`<et> contenido </et>`

## Fundamentos

- Validación de documentos
  1. Reglas básicas de la especificación XML (documentos bien formados)
    - Debe tener un único objeto raíz
    - Debe existir etiquetas de inicio y fin para cada elemento del lenguaje  
`<etiqueta> contenido_al_que_se_aplica </etiqueta>`  
Si la etiqueta no afecta a otro texto sería: `<etiqueta ... />`
    - Es sensible a mayúsculas y minúsculas
    - No se pueden intercalar etiquetas :  
`<libro><paginas>XML</libro>12</paginas>`
    - Los atributos de las etiquetas deben estar entrecomillados
    - Los comentarios van encerrados entre : `<!-- comentario -->`
  2. Estructura o esquema del lenguaje (documentos válidos)



- Validation of documents: Parsers
  - **Xerces2 Java Parser**  
<http://xerces.apache.org/xerces2-j/>
  - **Libxml2:**  
<http://xmlsoft.org/>
  - **Validome-Team**  
<http://www.validome.org/>
  - **Edinburgh University's Language Technology Group (Richard Tobin)**  
<http://www.hcrc.ed.ac.uk/~richard/xml-check.html>
  - **Brown University's Scholarly Technology Group**  
<http://www.stg.brown.edu/service/xmlvalid/>



## Esquemas

- Document Type Definitions (DTD)
  - Modelo tradicional para crear esquemas (importado de SGML)
  - Fácil uso para definir lenguajes simples
  - Limitado para poder expresar lenguajes completos
  - Requiere conocer un nuevo lenguaje (no XML)
- XML Schemas (XSD)
  - Método más potente e intuitivo que los DTD's
  - Utiliza XML para definir los esquemas

## Ejemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Mi Biblioteca -->
<biblioteca>
  <libro cod="0001">
    <titulo>XML en 24 horas</titulo>
    <autor>
      <nombre>Jonh Palmer</nombre>
      <apellido> Brian </apellido>
    </autor>
    <estado prestado="no"/>
  </libro>

  <libro cod="0002">
    <titulo>XML en 48 horas</titulo>
    <autor>
      <nombre>Juan Ramón</nombre>
      <apellido>López</apellido>
      <apellido>Prats</apellido>
    </autor>
    <isbn>8-12345-09</isbn>
    <estado prestado="si"/>
  </libro>
</biblioteca>
```

### DTD

añadir la siguiente etiqueta

<!DOCTYPE ....

### XML Schema

añadir el atributo xsi al elemento raíz

<biblioteca ... xsi:... >

## Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML

1. Antecedentes
2. Fundamentos
3. Esquemas: **DTD's** y Schema XML (XSD's)
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT
  - 4.3. Arquitectura DOM



## Document Type Definitions (DTD)

- Definición de tipos de documentos
  - Aunque se pueden crear documentos XML sin haber diseñado formalmente su sintaxis, no es aconsejable seguir esta filosofía. Lo ideal es establecer la sintaxis del lenguaje mediante un DTD, ya que esto permitirá la validación, de los documentos XML que creemos de ese lenguaje, de forma automática.
  - El DTD o sintaxis del lenguaje puede ser:
    - (1) Incluido en el propio documento
    - (2) Un fichero externo (.dtd)
    - (3) Público

```
<?xml version="1.0" ... >
<!DOCTYPE biblioteca [
    aquí la sintaxis ] >
...
(1)
```

```
<?xml version="1.0" ...>
<!DOCTYPE biblioteca SYSTEM
    "biblioteca.dtd" >
...
(2)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE biblioteca PUBLIC "identificador public"
    "dirección URL" >
...
(3)
```

## Document Type Definitions (DTD)

- Definición de tipos de documentos

Ejemplo de documento XML Público: Documento XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE biblioteca PUBLIC "identificador public"
    "dirección URL" >
...
```

```
?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
...
```

Ejemplo documento XHTML

## Document Type Definitions (DTD)

- Definición de tipos de documentos

Creación de un DTD para un lenguaje XML

- Componentes de un lenguaje XML (DTD)
  - Elementos: Cada una de las partes del documento XML.  
p.e. libro, título, autor, etc  
**<!ELEMENT libro (autor+, título, isbn?, estado)>**
  - Atributos: Propiedades de los elementos.  
p.e. código libro, prestado de estado  
**<!ATTLIST estado prestado (Si | No) "No">**
  - Entidades: Permiten establecer acciones especiales en un documento XML.  
p.e. caracteres especiales &lt; &acutote;  
**<!ENTITY dtd "Document Type Definitions">**
  - Notaciones: Determinan como procesar un tipo de datos binarios (no XML).  
**<!NOTATION gif SYSTEM "C:/windows/viewer.exe">**



## Document Type Definitions (DTD)

### Ejemplo de DTD

```
<!ELEMENT biblioteca (libro+)>
<!ELEMENT libro (autor+, titulo, isbn?, estado)>
<!ELEMENT autor (nombre, apellido+) >
<!ELEMENT titulo ( #PCDATA ) >
<!ELEMENT isbn ( #PCDATA ) >
<!ELEMENT estado EMPTY >
<!ELEMENT nombre ( #PCDATA ) >
<!ELEMENT apellido ( #PCDATA ) >
<ATTLIST libro cod ID #REQUIRED
             relacionados IDREFS #IMPLIED>
<ATTLIST estado prestado (Si | No) "No">
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Mi Biblioteca -->

<biblioteca>
  <libro cod="0001">
    <titulo>XML en 24 horas</titulo>
    <autor>
      <nombre>Jonh Palmer</nombre>
      <apellido> Brian </apellido>
    </autor>
    <estado prestado="no"/>
  </libro>

  <libro cod="0002">
    <titulo>XML en 48 horas</titulo>
    <autor>
      <nombre>Juan Ramón</nombre>
      <apellido>López</apellido>
      <apellido>Prats</apellido>
    </autor>
    <isbn>8-12345-09</isbn>
    <estado prestado="sí"/>
  </libro>
</biblioteca>
```

## Document Type Definitions (DTD)

### • Elementos

<!ELEMENT *nombre* ( *modelo\_contenido* )>

donde *modelo\_contenido* establece la composición de un elemento:

- Lista de elementos que lo compone separados por comas. Estos pueden incluir:
  - + . Incluye una o más ocurrencias de ese elemento
  - \* . Incluye 0 o más ocurrencias del elemento
  - ? . El elemento es opcional
- #PCDATA. Indica que el elemento esta compuesto por datos
- EMPTY. Indica que el elemento no contiene nada.
- ANY. Indica que contiene cualquier valor.
- | : Alternativa de contenido

Ejemplos:

```
<!ELEMENT a (b, (c | d)*, e+, f?) >
<!ELEMENT libro (autor+, titulo, isbn?, estado)>
<!ELEMENT autor (nombre, apellido+) >
<!ELEMENT titulo ( #PCDATA ) >
<!ELEMENT estado EMPTY >
```

## Document Type Definitions (DTD)

### Atributos

`<!ATTLIST nombre_elem n_atr1 tipo_atr1 "val_atr1" ... >`

*tipo\_atr*. Puede tomar los valores:

CDATA. El atributo es una cadena de caracteres cualquiera.

NMTOKEN (o NMTOKENS). El atributo toma una palabra (o varias) como valor.

ID. El valor debe ser un nombre clave (no repetido).

IDREF (o IDREFS). El valor (o valores) indica su relación con el ID de otro elemento.

Lista de nombres. El atributo toma uno de los valores.

*val\_atr*. Puede tomar los valores

#REQUIRED. El atributo debe tener un valor específico.

#IMPLIED. El valor del atributo no es requerido.

"valor". Valor por defecto para el atributo.

#FIXED "value". Establece el valor que debe tomar el atributo.

Ejemplos: `<!ATTLIST libro cod ID #REQUIRED relac IDREFS #IMPLIED>`  
`<!ATTLIST estado prestado (Si | No) "No">`

## Document Type Definitions (DTD)

### Ejemplo

```
<!-- fichero biblioteca.dtd -->

<!ELEMENT biblioteca (libro+)>
<!ELEMENT libro (autor+, titulo, isbn?, estado)>
<!ELEMENT autor (nombre, apellido+)>
<!ELEMENT titulo ( #PCDATA )>
<!ELEMENT isbn ( #PCDATA )>
<!ELEMENT estado EMPTY>
<!ELEMENT nombre ( #PCDATA )>
<!ELEMENT apellido ( #PCDATA )>
<!ATTLIST libro cod ID #REQUIRED
relacionados IDREFS #IMPLIED>
<!ATTLIST estado prestado (Si | No) "No">
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE biblioteca SYSTEM
"biblioteca.dtd">

<biblioteca>
...
<libro cod="0002" relac="0001">
<titulo>XML en 48 horas</titulo>
<autor>
<nombre>Juan Ramón</nombre>
<apellido>López</apellido>
<apellido>Prats</apellidos>
</autor>
<isbn>8-12345-09</isbn>
<estado prestado="Si"/>
</libro>
...
</biblioteca>
```

## Document Type Definitions (DTD)

Ejercicio 1. Crear el DTD para este documento XML

<pre>&lt;buzon&gt;   &lt;mensaje prioridad="urgente"&gt;     &lt;de&gt;Juan&lt;/de&gt;     &lt;a&gt;Pepe&lt;/a&gt;     &lt;texto idioma="es"&gt;       Estimado amigo: ....     &lt;/texto&gt;   &lt;/mensaje&gt;   &lt;mensaje&gt;     &lt;de&gt;Joss&lt;/de&gt;     &lt;a&gt;Pepe&lt;/a&gt;     &lt;texto idioma="en"&gt;       Dear colleague: ....     &lt;/texto&gt;   &lt;/mensaje&gt;   ... &lt;/buzon&gt;</pre>	<pre>&lt;!ELEMENT buzon (mensaje+)&gt; &lt;!ELEMENT mensaje (de, a, texto)&gt; &lt;!ATTLIST mensaje prioridad (normal   urgente) "normal"&gt; &lt;!ELEMENT de (#PCDATA)&gt; &lt;!ELEMENT a (#PCDATA)&gt; &lt;!ELEMENT texto (#PCDATA)&gt; &lt;!ATTLIST texto idioma CDATA #REQUIRED&gt;</pre>
---	---

## Document Type Definitions (DTD)

Ejercicio 2. Crear documento XML para el siguiente DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT agenda (persona+)>
<!ELEMENT persona (nombre, email*, telefono, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT relacion EMPTY>
<!ATTLIST relacion amigo-de IDREFS #IMPLIED>
```

Conociendo que los contactos son

Juan Ruiz. juan@empresa.com juan@particular.net 555001002 amigo de Eva  
Eva Gómez (email desconocido) 555002003 (amigo no conocido)  
Manuel López manuel@empresa.net 555444000 amigos Eva y Juan

## Document Type Definitions (DTD)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE agenda SYSTEM "agenda.dtd">
<agenda>
  <persona ID="juan" sexo="hombre">
    <nombre>Juan Ruiz</nombre>
    <email>juan@empresa.com</email>
    <email>juan@particular.net</email>
    <telefono>555001002</telefono>
    <relación amigo-de="eva" />
  </persona>

  <persona ID="eva" sexo="mujer">
    <nombre>Eva </nombre>
    <telefono>555002003</telefono>
  </persona>

  <persona ID="manu" sexo="hombre">
    <nombre>Manuel López</nombre>
    <email>manuel@empresa.net</email>
    <telefono>555444000</telefono>
    <relación amigo-de="eva juan" />
  </persona>
</agenda>
```

## Document Type Definitions (DTD)

### Ejercicio 3. Crear un DTD para un recetario de cocina

Cada receta tiene:


- Un atributo obligatorio que puede tomar los valores: 1plato, 2plato, postre
- Un nombre (texto)
- Varios ingredientes
- Varios pasos para su elaboración


Cada ingrediente contiene:

- Atributo con la cantidad
- Atributo con el nombre

Cada paso contiene:

- Un atributo identificador que establece el orden
- Un atributo que referencia otros pasos del cual depende éste
- Texto con la explicación







Universidad  
de Huelva


Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML

1. Antecedentes
2. Fundamentos
3. Esquemas: DTD's y **Schema XML (XSD's)**
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT
  - 4.3. Arquitectura DOM




## XML Schemas

- Tecnología alternativa a los DTDs para especificar la sintaxis de un lenguaje XML.
- Utilizan la sintaxis propia de XML (son documentos XML)
- Ventajas:
  - Permiten especificar tipos de datos: numéricos, fechas...
  - Fáciles de aprender (se usa también XML)
  - Procesables igual que los documentos XML

Universidad de Huelva

Dpto. Ing. Electrónica, Sist. Informáticos y Automática

Curso 2006/2007



## Qué encontramos en un esquema XML

- Un esquema XML define la estructura válida para un tipo de documento XML (al igual que las DTD), es decir:
  - Los elementos que pueden aparecer en el documento
  - Los atributos que pueden utilizarse junto a cada elemento
  - Cómo se pueden anidar los elementos (padres e hijos)
  - El orden en el que deben aparecer los elementos hijos de un mismo padre
  - El número permitido de elementos hijos
  - Si un elemento puede ser vacío o no
  - Tipos de datos para elementos y atributos
  - Valores por defecto y fijos para elementos y atributos




## XML Schemas

- La propuesta inicial de Microsoft dio lugar a los llamados “esquemas XDR”
- Posteriormente, el W3C diseñó un modelo de esquemas que es la propuesta oficial y la que debemos conocer (llamados “esquemas XSD”)
- XSD se publicó como una recomendación el 31 de marzo del 2001 (se considera oficial desde mayo)
- XSD es más complejo que otras alternativas anteriores, pero supuso un importante paso hacia adelante en la estandarización de XML




## Otras ventajas de XML Schemas

- Mayor precisión en la definición de tipos de datos mediante formatos y facetas
- Por ejemplo, la fecha:  
`<date type="date">1999-03-11</date>`  
¿es el 11 de marzo o el 3 de noviembre?
- Los esquemas se definen como documentos XML, en un documento aparte con extensión .XSD
- En los documentos XML que se basen en ese esquema, incluiremos una referencia al archivo .XSD




## Asociar DTD a documentos XML

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.us.com/dtd/note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>
  Don't forget me this weekend!
</body>
</note>
```



## Asociar Schema's a documentos XML


```
<?xml version="1.0"?>
<note xmlns="http://www.us.com"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
        "http://www.us.com/schema/note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>
    Don't forget me this weekend!
  </body>
</note>
```



## Ejemplo esquema W3C

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="note">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="to" type="xsd:string"/>
        <xsd:element name="from" type="xsd:string"/>
        <xsd:element name="heading" type="xsd:string"/>
        <xsd:element name="body" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```






## Esquemas XML – elemento schema

- Los elementos utilizados en la creación de un esquema “proceden” del espacio de nombres: <http://www.w3.org/2001/XMLSchema>
- El elemento *schema* es el elemento raíz del documento en el que se define el esquema:

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



## Esquemas XML – elementos “simples”

- Un elemento simple es un elemento que sólo puede contener texto (cualquier tipo de dato), pero no a otros elementos ni atributos
- Para definir un elemento simple, utilizamos la sintaxis:

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

- Ejemplos:

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

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

```
<xsd:element name="fecNac" type="xs:date"/>
```

## Esquemas XML – elementos “simples”, tipos de datos

- Los tipos de datos más utilizados son:
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time
- Un elemento simple puede tener un valor por defecto y un valor “fijo”
- Esto se indica mediante los atributos default y fixed

```
<xsd:element name="color" type="xsd:string" default="red"/>
```

## Esquemas XML – atributos (1)

- Los atributos se deben declarar de forma similar a los “elementos simples”
- Si un elemento tiene atributos, el elemento se deberá declarar como un elemento “complejo”
- Un atributo se declara de la siguiente forma:

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

**Ejemplo:**

```
<xsd:attribute name="idioma" type="xs:string"/>
```
- Los atributos tienen un tipo de dato: xsd:string, xsd:decimal, xsd:integer, xsd:boolean, xsd:date, xsd:time



## Esquemas XML – atributos (2)

- Los atributos pueden tener valores por defecto y valores fijos:  
`<xsd:attribute name="idioma" type="xsd:string" default="ES"/>`
- Por defecto, los atributos son opcionales.
- Para indicar que un atributo debe ser obligatorio, se debe añadir a su declaración en el esquema el atributo “use”


`<xsd:attribute name="lang" type="xsd:string" use="required"/>`

- El atributo “use” puede tomar el valor “optional” si el atributo no es obligatorio (opción por defecto)




## Esquemas XML – facetas

- Las facetas o restricciones permiten restringir el valor que se puede dar a un elemento o atributo XML
- Mediante restricciones podemos indicar que un valor debe estar comprendido en un rango determinado, debe ser un valor de una lista de valores “cerrada”, o debe ser mayor o menor que otro valor...
- Tipos de facetas:
  - Valor comprendido en un rango
  - El valor está restringido a un conjunto de valores posibles
  - Restringir el valor de un elemento a una serie de caracteres
  - Longitud de los valores de los elementos...



## Esquemas XML – facetas (ej. 1)

```
<xsd:element name="age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```



## Esquemas XML – facetas (ej. 2)

```
<xsd:element name="car">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Audi"/>
      <xsd:enumeration value="Golf"/>
      <xsd:enumeration value="BMW"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

## Esquemas XML – facetas (ej. 2, alt.)

```
<xsd:element name="car" type="carType"/>

<xsd:simpleType name="carType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Audi"/>
    <xsd:enumeration value="Golf"/>
    <xsd:enumeration value="BMW"/>
  </xsd:restriction>
</xsd:simpleType>
```

## Esquemas XML – facetas (ej. 3)

```
<xsd:element name="letter">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-z]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

En este ejemplo, el elemento "letter" debe tomar como valor una letra minúscula (sólo 1)

## Esquemas XML – facetas (ej. 4)

```
<xsd:element name="initials">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

En este ejemplo, el elemento “initials” debe tomar como valor 3 letras mayúsculas o minúscula (sólo 3)

## Esquemas XML – facetas (ej. 5)

```
<xsd:element name="choice">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[xyz]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

En este ejemplo, el elemento “choice” debe tomar como valor una de estas letras:  
x, y o z



## Esquemas XML – facetas (ej. 6)

```
<xsd:element name="prodid">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```



## Esquemas XML – facetas (ej. 7)

```
<xsd:element name="letter">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([a-z])*/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

## Esquemas XML – facetas (ej. 8)

```
<xsd:element name="password">
  <xsd:simpleType>
    <xsd:restriction base="xs:string">
      <xsd:pattern value="[a-zA-Z0-9]{8}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

En este ejemplo, el valor del campo “password” debe ser 8 caracteres

## Esquemas XML – facetas (ej. 9)

```
<xsd:element name="password">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```


Los elementos length, minLength y maxLength permiten indicar el número exacto, mínimo y máximo de caracteres que puede tener un valor de un elemento.





## Elementos para restricciones

enumeration	Establece una lista de valores "aceptados"
fractionDigits	Número de cifras decimales
length	Número de caracteres obligatorios
maxExclusive y maxInclusive	Valor máximo de un rango
minExclusive y minInclusive	Valor mínimo en un rango
maxLength y minLength	Número máximo y mínimo de caracteres permitidos
pattern	Define una secuencia de caracteres permitida
totalDigits	Número exacto de dígitos permitidos
whiteSpace	Indica cómo se deben de tratar los espacios en blanco



## Elementos complejos

- Son elementos que contienen a otros elementos hijos, o que tienen atributos
- Se suelen dividir en 4 tipos:
  - Elementos vacíos
  - Elementos no vacíos con atributos
  - Elementos con elementos hijos
  - Elementos con elementos hijos y con "texto" o valor propio (como el contenido mixto de las DTD)

## Elementos complejos

- Ejemplos:

```
<product pid="1345"/>
```

```
<food type="dessert">Ice cream</food>
```

```
<description>Sucedió el <date>03.03.99</date> .... </description>
```

```
<employee>
```

```
  <firstname>John</firstname>
```

```
  <lastname>Smith</lastname>
```

```
</employee>
```

## Declarar elementos complejos

- Para definir elementos complejos se utiliza la siguiente sintaxis:

```
<xsd:element name="employee">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="firstname" type="xsd:string"/>  
      <xsd:element name="lastname" type="xsd:string"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

## Declarar elementos complejos

- Podemos usar otra sintaxis para reutilizar la “definición” de los elementos hijos en varios elementos:

```
<xsd:element name="employee" type="personinfo"/>
<xsd:element name="student" type="personinfo"/>
```

```
<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

- En la declaración de elementos complejos, es posible utilizar un mecanismo de “herencia” para reutilizar o extender elementos definidos con anterioridad (ver la siguiente página)

## Declarar elementos complejos

```
<xsd:element name="employee" type="fullpersoninfo"/>

<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="fullpersoninfo">
  <xsd:complexContent>
    <xsd:extension base="personinfo">
      <xsd:sequence>
        <xsd:element name="address" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

## Declarar elementos complejos

- Para declarar un elemento vacío con atributos, se utilizará la siguiente sintaxis:

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:attribute name="prodid" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

- <product prodid="1345" />

## Declarar elementos complejos

- Para declarar un elemento no vacío con atributos, y sin elementos hijos, se utilizará la siguiente sintaxis:

```
<xsd:element name="shoesize">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:integer">
        <xsd:attribute name="country" type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```



## Declarar elementos complejos

- Para declarar un elemento con contenido “mixto”, basta con añadir un atributo “mixed” al elemento `xsd:complexType`:

```
<xsd:element name="letter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="orderid" type="xsd:positiveInteger"/>
      <xsd:element name="shipdate" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



## Declarar elementos complejos

- La declaración anterior permitiría un texto como el siguiente:

```
<letter>Estimado cliente: <name>Juan Perez</name>. Su pedido número
<orderid>1032</orderid> se enviará el día <shipdate>2001-07-13</shipdate>. </letter>
```



## Declarar elementos complejos: Indicadores

- En los ejemplos anteriores hemos utilizado el elemento `xsd:sequence` como elemento hijo del elemento `xsd:complexType`
- `xsd:sequence` indica que los elementos anidados en él deben aparecer en un orden determinado
- Los esquemas XML nos ofrecen otras alternativas, además de `xsd:sequence`, para indicar cómo se deben tratar los elementos que aparecen anidados en un elemento complejo
- Las opciones o “indicadores” son: `xsd:all` y `xsd:choice`



## Declarar elementos complejos: Indicador `xsd:all`

- `xsd:all` indica que los elementos que contiene pueden aparecer en cualquier orden, pero como máximo sólo una vez

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

## Declarar elementos complejos: Indicador xsd:choice

- El indicador xsd:choice indica que puede aparecer sólo uno de los elementos que contiene

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

## Declarar elementos complejos: maxOccurs y minOccurs

- Existen indicadores para establecer el número máximo y mínimo de veces que puede aparecer un elemento hijo de un elemento complejo  
El atributo maxOccurs puede tomar el valor "unbounded", que indica que no existe ningún límite

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="full_name" type="xsd:string"/>
      <xsd:element name="child_name" type="xsd:string" maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

## El modelo de contenido: any

- En esquemas XML también contamos con un modelo de contenido ANY, que permite incluir elementos no declarados inicialmente en el esquema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
      <xsd:any minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

## El modelo de contenido: anyAttribute

- También contamos con un elemento que permite extender el número de atributos de un elemento:

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:element>
```



## Práctica 1

- Diseñar un esquema XML para crear documentos para control de préstamos de libros en una biblioteca
- En cada documento se indicarán:
  - El nombre y apellidos del bibliotecario
  - Fecha del préstamo y de devolución
  - Datos del lector (id, nombre, apellidos, teléfono y dirección) La dirección se dividirá en tipo de calle (que puede ser calle, avenida o plaza), nombre calle, número, piso y letra, c.p., localidad y provincia
  - Un máximo de tres ejemplares en préstamo. Para cada uno de ellos: el número de registro, título, autor(es)
  - El préstamo tendrá un atributo numérico que servirá como identificador

## Nuevas Tecnologías de la Programación



Universidad de Huelva

### Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML


1. Antecedentes
2. Fundamentos
3. Esquemas: DTD's y Schema XML (XSD's)
4. Otras tecnologías XML


### 4.1. XQuery (VER TUTORIAL EN LA WEB)

### 4.2. XSLT

### 4.3. Arquitectura DOM









Universidad  
de Huelva


Objetivos

- Conocer XML
- Entender los fundamentos de XML
- Definir el DTD de un lenguaje basado en XML
- Definir Schema's (XSD) de un lenguaje XML
- Presentar XQuery para consultar documento XML
- Presentar XSLT para transformar documentos para un navegador
- Introducir la arquitectura DOM.

## Tema 2. XML

1. Antecedentes
2. Fundamentos
3. Esquemas: DTD's y Schema XML (XSD's)
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT**
  - 4.3. Arquitectura DOM



## XSLT – Transformación de XML a XHTML

- XSLT es un lenguaje de transformación de documentos XML, que junto con XPath (rutas al documento XML) y XSL-FO (lenguaje de formato para XML) conforman XSL (eXtensible Stylesheet Language) o lenguajes de estilo para XML.
- XSLT (XSL Transformations) es el más importante de los componentes del estándar XSL. Su objetivo es transformar documentos XML es otro tipo de documentos. Nuestro interés está en transformar XML a XHTML.
- XSLT utiliza XPath para recorrer el documento XML.
- Para la transformación del documento se crea un fichero de hojas de estilos (.xsl) y el documento XML que se desea transformar se le incorpora este fichero de estilos.

Documento XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Mi Biblioteca -->
<biblioteca>
<libro cod="0001" >
...
```

Documento XML con XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="fichero.xsl"?>
<!-- Mi Biblioteca -->
<biblioteca>
<libro cod="0001" >
```

Universidad de Huelva
Dpto. Ing. Electrónica, Sist. Informáticos y Automática
Curso 2006/2007

## XSLT – Transformación de XML a XHTML

- Fichero **.xsl**

- Estructura

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  ...
</xsl:stylesheet>
```

- Existen dos formas de transformar el documento XML a XHTML

- Iterativa

- Se crea un *template* desde el punto del documento desde donde se desea iniciar la transformación
    - Se aplican etiquetas que permiten recorrer los elementos del documento

- Basada en *templates*.

- Se crean *templates* para cada elemento que se desea presentar.
    - El *template* dará formato al contenido del documento XML o indicará los *templates* de los que depende.

## XSLT – Transformación de XML a XHTML

- Contenido fichero **.xsl**

- `<xsl:template match="XPath_EltoXML"> ... </xsl:template>`

- `<xsl:value-of select="XPath_EltoXML" />`

- `<xsl:for-each select="EltoXML"> ... </xsl:for-each>`

- `<xsl:sort select="EltoXML" />`

- `<xsl:if test="EltoXM op valor"> ... </xsl:if>`, donde *op* =, !=, &lt; ó &gt;;

- `<xsl:choose>`

- `<xsl:when test="EltoXM op valor"> ... </xsl:when>`

- ...

- `<xsl:otherwise> .... </xsl:otherwise>`

- `</xsl:choose>`

- `<xsl:apply-templates [select="XPath_XML"] />`

## XSLT – Transformación de XML a XHTML

... <!-- Cabecera suprimida -->

```
<xsl:template match="/">
<html>
<body>
<h2>Mi Biblioteca Personal</h2>
<table border="1">
<tr>
<th align="left">Cod.Libro</th>
<th align="left">Titulo</th>
<th align="left">Autor</th>
<th align="center">Prestado</th>
</tr>

<xsl:for-each select="biblioteca/libro">
<tr>
<td><xsl:value-of select="@cod"/></td>
<td><xsl:value-of select="titulo"/></td>
<td>
<xsl:for-each select="autor/apellido">
<xsl:value-of select="."/>
</xsl:for-each>
,<xsl:value-of select="autor/nombre"/>
</td>
```

... <!-- continuación -->

```
<td>
<xsl:for-each select="estado">
<xsl:choose>
<xsl:when test="@prestado='si'"> Uff!
</xsl:when>
<xsl:when test="@prestado='no'"> Bien
</xsl:when>
</xsl:choose>
</xsl:for-each>

<xsl:for-each select="isbn">
(isbn)
</xsl:for-each>
```

### Mi Biblioteca Personal

Cod.Libro	Titulo	Autor	Prestado
0001	XML en 24 horas	Brian , Jonh Palmer	Bien
0002	XML en 48 horas	López, Prats, Juan Ramón	Uff (isbn)

</xsl:stylesheet>

Universidad de Huelva

Dpto. Ing. Electrónica, Sist. Informáticos y Automática

Curso 2006/2007

## XSLT – Transformación de XML a XHTML

- Entrás las marcas de inicio y de fin del elemento raíz xsl:stylesheet, se escribirán las reglas de transformación propiamente dichas
- Cada regla se definirá mediante un elemento xsl:template
- La regla indica qué instancias de los elementos del documento XML se van a transformar.
- La regla también indicará cómo se deben transformar cada una de ellas

Universidad de Huelva

Dpto. Ing. Electrónica, Sist. Informáticos y Automática

Curso 2006/2007

## XSLT – Transformación de XML a XHTML

```
<xsl:template match="//nombre">
  <h2>
    <xsl:value-of select="." />
  </h2>
</xsl:template>
```

- La regla se aplicará a todas las instancias del elemento nombre. Esto se indica mediante el atributo **match** que acompaña al elemento **xsl:template**.
- Entre las etiquetas de inicio y de fin del elemento **xsl:template** se escribe la transformación que se debe realizar...
- es decir, **qué texto y qué marcas se escribirán en el documento resultado de la transformación**, cada vez que se encuentre una instancia del elemento **nombre** en el documento origen.
- Con **<xsl:value-of...>**, se recupera y escribe en el documento resultado el valor del elemento que está siendo procesado.

## XSLT – Transformación de XML a XHTML

- Ejemplo

```
<?xml version="1.0"?>
<ciudades>
  <ciudad>
    <nombre>Madrid</nombre>
    <habitantes>3500000</habitantes>
  </ciudad>
  <ciudad>
    <nombre>Málaga</nombre>
    <habitantes>800000</habitantes>
  </ciudad>
  <ciudad>
    <nombre>Toledo</nombre>
    <habitantes>50000</habitantes>
  </ciudad>
</ciudades>
```

## XSLT – Transformación de XML a XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo XSLT</title>
      </head>
      <body>
        <xsl:apply-templates select="nombre" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="//nombre">
    <h2>
      <xsl:value-of select="." />
    </h2>
  </xsl:template>
</xsl:stylesheet>
```

## XSLT – Transformación de XML a XHTML

- Funcionamiento
  - La regla `<xsl:template match="/">` se ejecuta cuando se encuentra el elemento raíz del documento XML
  - Dentro de esta regla, podemos incluir llamadas a otras reglas definidas en la hoja de estilo, mediante el elemento:  
`<xsl:apply-templates select="..." />`
  - El atributo `select` tomará como valor el nombre del elemento asociado a la regla que queremos “disparar”
  - Esto nos ofrece un control real sobre el “orden” de ejecución de las reglas



## XSLT – Transformación de XML a XHTML

- El resultado de la transformación del ejemplo:

```
<html>
  <head>
    <title>Ejemplo XSLT</title>
  </head>
  <body>
    <h2>Madrid</h2>
    <h2>Málaga</h2>
    <h2>Toledo</h2>
  </body>
</html>
```



## XSLT – Transformación de XML a XHTML

- El elemento `<xsl:value-of...>`
  - En el elemento `<xsl:value-of...>` se puede indicar que se quiere mostrar el valor del elemento que estamos procesando
  - También podemos indicar que queremos mostrar el valor de un elemento hijo, o descendiente, del elemento que se está procesando
  - En el ejemplo anterior, podríamos utilizar `xsl:value-of` para mostrar en el documento resultado de la transformación el título, código de registro o fecha de préstamo de cada libro...
  - Esto es posible porque en el atributo `select` podemos utilizar una “expresión XPATH”

## XSLT – Transformación de XML a XHTML

- Por ejemplo, para mostrar el valor del elemento titulo, que es un hijo del elemento ejemplar, podríamos utilizar la siguiente regla:

```
<xsl:template match="//ejemplar">
  <xsl:value-of select="./titulo" />
</xsl:template>
```

El valor del atributo select se puede leer de la siguiente forma: "dame el valor del elemento titulo que es hijo del elemento que estoy procesando". En este caso, cada uno de los elementos ejemplar

Esto se indica mediante ./

## XSLT – Transformación de XML a XHTML

- Ordenar la salida
  - Para ordenar los contenidos, se utiliza el elemento xsl:sort
  - xsl:sort es un elemento hijo de xsl:apply-templates
  - Acepta dos atributos:
    - select – que toma como valor el nombre del elemento que se va a utilizar como criterio de ordenación y
    - order – que indica si se debe utilizar un orden ascendente o descendente.

```
<xsl:apply-templates select="//ciudad">
  <xsl:sort select="ciudad" order="descending" />
</xsl:apply-templates>
```



## XSLT – Transformación de XML a XHTML

- Acceder a los atributos de los elementos
  - En XSLT podemos “filtrar” o indicar qué instancias de un elemento queremos procesar, tomando como criterio de selección el valor de los atributos que acompañan a los elementos
  - Para hacer esto, en un elemento `xsl:value-of`, podemos recuperar el valor de un atributo mediante la expresión `@nombreAtributo`, por ejemplo:

```
<xsl:template match="vuelo">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
    <td><xsl:value-of select="@origen" /></td>
    <td><xsl:value-of select="@destino" /></td>
    <td><xsl:value-of select="@hora" /></td>
  </tr>
</xsl:template>
```

## XSLT – Transformación de XML a XHTML

- Actuación condicional: if
  - Para indicar qué instancias de un elemento queremos procesar, o realizar una “ejecución condicional de código”, en XSLT disponemos del elemento `xsl:if`
  - `xsl:if` va acompañado de un atributo `test` que contiene una “condición”.
  - Si la condición se cumple para el elemento que se está procesando, la regla de ejecutará. Por ejemplo:

```
<xsl:if test="@destino='JFK'">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
    <td><xsl:value-of select="@origen" /></td>
    <td><xsl:value-of select="@destino" /></td>
    <td><xsl:value-of select="@hora" /></td>
  </tr>
</xsl:if>
```

## XSLT – Transformación de XML a XHTML

- Actuación condicional: choose, when y otherwise
  - Estos elementos “amplían” las posibilidades del elemento `xsl:if`
  - Permiten indicar qué transformación se debe realizar en el caso de que se cumpla una condición, y en el resto de casos
  - Se utilizan de forma conjunta. El elemento `xsl:choose` contendrá a uno o más elementos `xsl:when` y a un elemento `xsl:otherwise`.
  - El elemento `xsl:when` incluye un atributo `test` que tomará como valor la expresión que se evaluará. Si se cumple, se ejecutará el código escrito entre las etiquetas de inicio y de fin del elemento `xsl:when`.
  - El elemento `xsl:otherwise` contendrá el código que se ejecutará si no se cumplen las expresiones indicadas en los atributos `test` de los elementos `xsl:when`.

## XSLT – Transformación de XML a XHTML

```
<xsl:choose>
  <xsl:when test="expresión">
    ....
    ....
    ....
  </xsl:when>
  <xsl:when test="expresión2">
    ....
    ....
    ....
  </xsl:when>
  <xsl:otherwise>
    ....
    ....
  </xsl:otherwise>
</xsl:choose>
```

A 5x5 grid with black and orange squares. The pattern is as follows:


... <!-- continuación -->

```
<td>
  <xsl:for-each select="estado">
    <b>xsl:choose>
      <xsl:when test="@prestado='si'"> Uff!
    </xsl:when>
      <xsl:when test="@prestado='no'"> Bien
    </xsl:when>
  </xsl:choose>
</xsl:for-each>

  <xsl:for-each select="isbn">
    (isbn)
  </xsl:for-each>

```

Mi Biblioteca Personal

Cod Libro	Título	Autor	Prestado
0001	XML en 24 horas	Brian , Jonh Palmer	Bien
0002	XML en 48 horas	López, Prats, Juan Ramón	Uff! (isbn)

&lt;/xsl:stylesheet&gt;

Curso 2006/2007

Universidad  
de Huelva

## Tema 2. XML

1. Antecedentes
2. Fundamentos
3. Esquemas: DTD's y Schema XML (XSD's)
4. Otras tecnologías XML
  - 4.1. XQuery
  - 4.2. XSLT

