XML

Esquemas

Aplicaciones Distribuidas

Curso 2025/26

DIS
Departamento de
Informática y Sistemas

UNIVERSIDAD
DE MURCIA

DTD

- DTD *Document Type Definition*
- Permite la **definición de un lenguaje de marcado**.
- Especifica la estructura del documento:
  - Los tipos de elementos aceptables y sus atributos.
  - Orden de declaración de los elementos.

- Los documentos que se ajustan a un DTD (en general a un esquema) se denominan **documentos válidos**.
- Un documento válido es a su vez bien formado, pero no a la inversa.

DTD - Ejemplos

- Ejemplo 1:

```
<!ELEMENT poema (titulo,verso*)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT verso (#PCDATA)>
<!ATTLIST poema fecha CDATA #REQUIRED
      lugar CDATA #IMPLIED>
```

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

<!DOCTYPE poema SYSTEM "poema.dtd">

<poema fecha="Abril de 1915" lugar="Granada">
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>siente junto a la alborada</verso>
<verso>el dolor de sus amores</verso>
<verso>y el sueño de las distancias. </verso>
</poema>
```

DTD - Ejemplos

- Ejemplo 2:

```
<!ELEMENT inventor (#PCDATA)>
<!ELEMENT ingrediente EMPTY>
<!ELEMENT pizza (ingrediente*, inventor?, precio)>
...

<!ATTLIST pizza
        nombre CDATA #REQUIRED
        oregano (si|no) "si">
<!ATTLIST ingrediente
        nombre CDATA #REQUIRED
        calorias CDATA #IMPLIED>
<!ATTLIST precio
        moneda (euros|dolares) #REQUIRED
        valor CDATA #REQUIRED>
```

DTD - Ejemplos

- Ejemplo 2:

```
<?xml version="1.0" ?>
<!DOCTYPE pizza SYSTEM "pizza.dtd">

<pizza nombre="4 estaciones">
<ingrediente nombre="Jamón" />
<precio moneda="euros" valor="7" />
</pizza>
```

DTD – Propiedades de modelado

- Especifican estructura del documento: elementos, atributos, anidamientos, etc.
- Mecanismo sencillo de abstracción.
- Entidades: macros, inclusión de documentos.
- Integrado en la especificación XML.

DTD – Limitaciones

- **Expresividad limitada**:
  - Limitaciones de tipos. Por ejemplo, indicar que un elemento es un número o una fecha.

- Extensibilidad limitada:
  - No se define un mecanismo de extensibilidad.
  - Para añadir nuevas declaraciones es necesario editar el DTD.

XML Schema

- **Motivación** (igual que DTD):
  - Definir la estructura de los documentos XML.

- Un XML Schema permite especificar:
  - Elementos y atributos
  - Modelo de contenido para la declaración de elementos.
  - **Tipos** para elementos y atributos

- Alternativa a los DTD basada en una sintaxis XML.

- Adoptado como recomendación por el W3C en 2001.

XML Schema

- Ventajas respecto a DTD:
  - Mayor riqueza expresiva (>40 tipos de datos primitivos)
  - Restricciones más precisas sobre los documentos XML, tanto en estructura como en tipos de datos.
  - Permite tipos de datos definidos por el usuario.
  - Modular: permite referenciar múltiples esquemas
  - Extensible: creación de tipos derivados
  - Incluye claves primarias e integridad referencial
  - …

Cabecera de un esquema

- **Declaración de un esquema**:
  - Alias **xs**: declaraciones de XML Schema

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

<xs:element name="acta" ...
```

Cabecera documento XML

- **Declaración de un documento** (instancia):
  - Alias **`xsi`**: espacio de nombres para declaraciones propias de XML (tipos, localización de esquema, etc.)
  - **`xsi:noNamespaceSchemaLocation`**: enlaza el documento con la localización de su esquema.

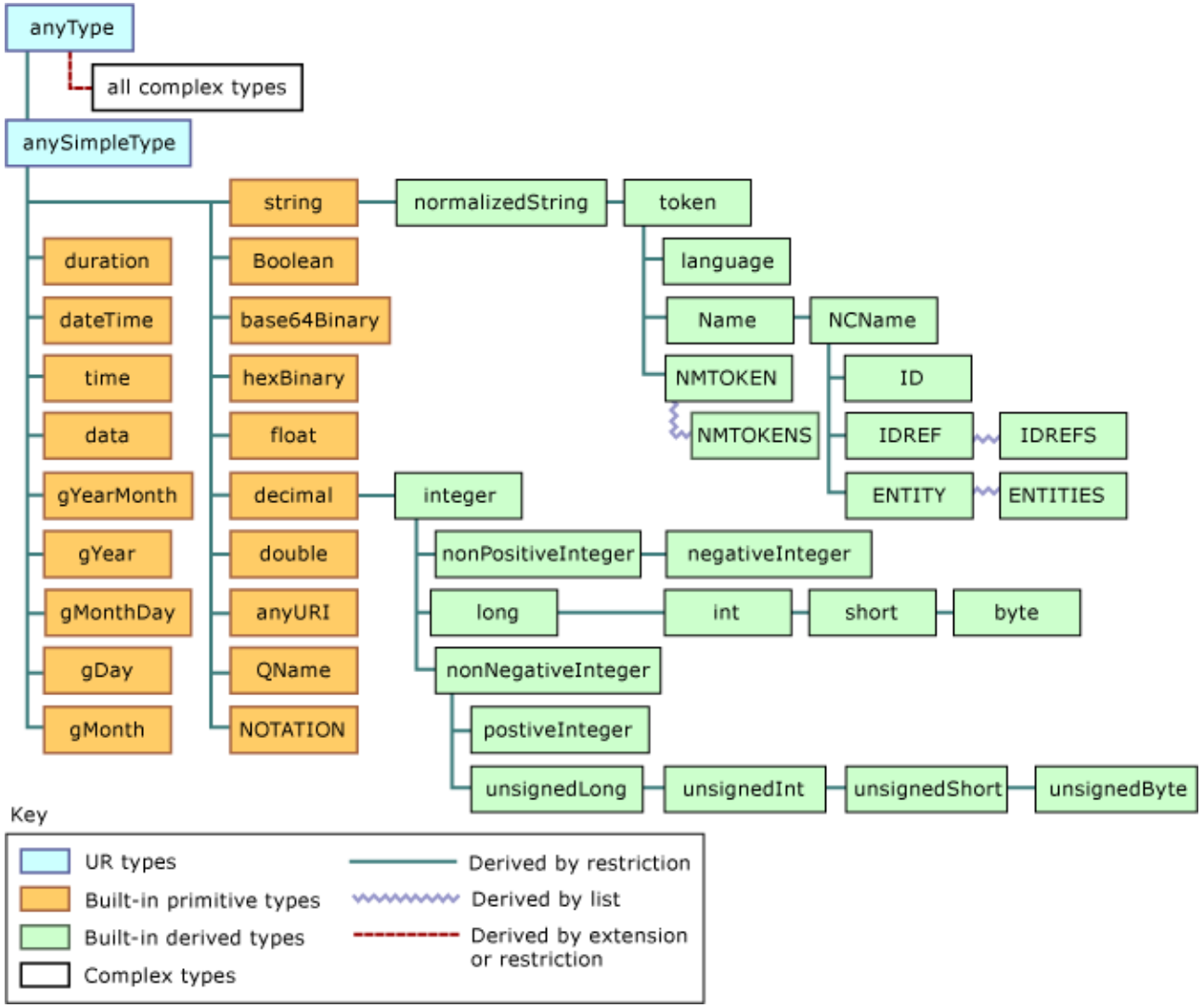
```
<acta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="acta.xsd">
…

</acta>
```

Tipos de datos

- Clasificación de tipos de datos que podemos utilizar en un XML Schema:
  - **Primitivos** y predefinidos.
  - **Tipos simples** (`simpleType`) → contienen sólo texto
    - Definidos mediante restricciones (*facetas*), enumeración o listas de valores de tipos primitivos u otros tipos simples.
  - **Tipos complejos** (`complexType`) → **sólo para elementos**
    - `simpleContent`: con atributos y texto.
    - `complexContent`: con **elementos**, atributos y texto.

Tipos primitivos



Elementos simples

- Un **elemento simple** tan solo puede contener texto.
  - No puede contener otros elementos ni atributos.
- Su tipo puede ser primitivo o un *tipo simple* (se explica más adelante).

```
<producto>silla</producto>
<id>31</id>
<fecha>2023-03-19</fecha>
```

```
<xs:element name="producto" type="xs:string" />
<xs:element name="id" type="xs:integer" />
<xs:element name="fecha" type="xs:date" />
```

Tipos simples

- **<simpleType>**
  - Tipos definidos a partir de **restricciones** sobre tipos primitivos u otros tipos simples.

- Ejemplo:
  - El tipo “TipoNota” representa los valores enteros de 0 a 10, ambos incluidos.

```
<xs:simpleType name="TipoNota">
<xs:restriction base="xs:integer">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="10" />
</xs:restriction>
</xs:simpleType>
```

Tipos simples

- Un tipo simple puede ser utilizado como tipo en un **elemento simple** **o** **atributo**.
- Pueden tener una **declaración anónima**.
- Ejemplo:
  - El elemento “edad” permite valores enteros en el rango de 0 a 120, ambos incluidos.

```
<xs:element name="edad">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0" />
      <xs:maxInclusive value="120" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Tipos simples

- Las restricciones que se establecen sobre un tipo se denominan **facetas**.
- Ejemplos:
  - Sobre cadenas, patrón de expresión regular:

```
<xs:pattern value="[0-9]+" />
```

  - Sobre cadenas, longitud:

```
<xs:length value="8" />
```

Tipos simples

- Un tipo simple también se puede definir por **enumeración.**

- Ejemplo:
  - El elemento “color” es de tipo string y toma valores de la enumeración “rojo”, “verde” y “azul”.

```
<xs:element name="color">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="rojo" />
      <xs:enumeration value="verde" />
      <xs:enumeration value="azul"  />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Elementos complejos

- Son elementos que pueden tener atributos y contener otros elementos.

- `simpleContent`: con atributos y texto
  - **Caso especial** → **Vacíos** (sin texto).

- `complexContent`: con elementos, atributos y texto.
  - **Caso especial** → **Mixtos**. Permiten mezclar elementos y texto

Elementos complejos

- **simpleContent**: extiende un tipo predefinido o un tipo simple

- Ejemplo:
  - Declara el elemento “talla” con contenido de tipo string y que incluye el atributo “pais”

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

Elementos complejos

- Otro ejemplo:

```
<xs:element name="altura">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="unidad" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<altura unidad="cm">190</altura>
```

- **Importante**: cuando el contenido es simple (`simpleContent`), tan solo puede contener atributos y texto

Elementos complejos

- Todos los **atributos** son de tipo primitivo o simple.
- **Modificadores** aplicables:
  - Valor por defecto (`default`):

```
<xs:attribute name="idioma" type="xs:string" default="ES" />
```

  - Obligatorio (`use="required"`) u opcional (`use="optional"`):

```
<xs:attribute name="idioma" type="xs:string" use="required" />
```

Elementos complejos

- **complexContent**: puede tener atributos y contener elementos.

- La estructura de los elementos que contiene (**modelo de contenido**) puede ser de tres tipos:
  - Secuencia (**`sequence`**)
  - Cualquier orden (**`all`**)
  - Alternativa (**`choice`**)

- **Nota**: la etiqueta `<complexContent>` no se declara al especificar un modelo de contenido (es implícita).

Elementos complejos

- Modelo de contenido, **secuencias (`sequence`)**: establecen el orden de declaración de los elementos que contiene.

- Ejemplo:
  - Declara el elemento “persona” que contiene los elementos “nombre” y “apellidos”, en ese orden.

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

Elementos complejos

- **Nota**: al igual que los tipos simples, los tipos complejos también se pueden declarar como tipos nombrados.
  - Útil para ser reutilizados por varios elementos.

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

<xs:element name="persona" type="tipo_persona">
```

Elementos complejos

- Al declarar los elementos que forman parte del modelo de contenido, se puede establecer el **número de repeticiones**.
- Podemos establecer un valor mínimo de ocurrencias o repeticiones (**minOccurs**) y un valor máximo (**maxOccurs**).
  - Por defecto, el número de ocurrencias es 1.
- Para indicar sin límite de ocurrencias: **maxOccurs="unbounded"**

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="profesion" type="xs:string" maxOccurs="10" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Elementos complejos

- **Declaración de atributos**:
  - Si el elemento tuviera atributos, se declaran después del modelo de contenido
  - En el ejemplo, después de la secuencia.

```
<xs:element name="persona">
  <xs:complexType>
       <xs:sequence>
         ...
       </xs:sequence>

    <xs:attribute name="dni " type="xs:string" />

  </xs:complexType>
</xs:element>
```

Elementos complejos

- Modelo de contenido, **sin orden** (**`all`**):
  - Cada elemento contenido puede ocurrir como máximo una vez.
  - Se permite que sea opcional (minOccurs="0").

```
<xs:element name="persona">
  <xs:complexType>
    <xs:all>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="apellidos" type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

Elementos complejos

- Modelo de contenido, **alternativa** (**`choice`**):
  - De los elementos que contiene, sólo puede aparecer uno de ellos.
  - Es compatible con los modificadores de ocurrencias.

```
<xs:complexType name="calificacion">
  <xs:choice>
    <xs:element name="nota" type="xs:positiveInteger" />
    <xs:element name="valoracion" type="xs:string" />
  </xs:choice>
</xs:complexType>
```

Elementos complejos

- **Elementos vacíos**: se definen como un elemento complejo que solo puede tener atributos.

```
<producto id="1345" />
```

```
<xs:element name="producto">
  <xs:complexType>
    <xs:attribute name="id" type="xs:positiveInteger" />
  </xs:complexType>
</xs:element>
```

- Sin atributos: `<xs:complexType />`

Validación de documentos

- El análisis de un documento XML siempre comprueba si un documento tiene errores sintácticos. En tal caso, aborta el procesamiento.
- La **validación** se configura del siguiente modo:
  - Tipos de datos en paquete `javax.xml.validation`
    - Creación de un objeto que represente al esquema (**`Schema`**): construcción basada en una factoría.
    - Solicitar al `Schema` un validador (**`Validator`**).
    - … (continúa en la siguiente diapositiva)

Validación de documentos

- La **validación** se configura del siguiente modo (**continúa**):
  - …
  - Establecer en el validador un manejador de eventos de error utilizando el método **`setErrorHandler`**.

    → Acepta un objeto compatible con `DefaultHandler (SAX)`
  - Aplicar el método **`validate`**:
    - Recibe como parámetro una fuente de datos XML (interfaz **Source**): clases que la implementan

    → `DOMSource` (árbol DOM), `StreamSource` (fichero), `JAXBSource`.

Validación de documentos

```
SchemaFactory factoriaSchema =
       SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);

// Construye el esquema
Schema esquema = factoriaSchema.newSchema(new File(ficheroEsquema));

// Solicita al esquema la construcción de un validador
Validator validador = esquema.newValidator();

// Registra el manejador de eventos de error
validador.setErrorHandler(new DefaultHandler() {
   public void error(SAXParseException e) throws SAXException {
     System.out.println("Error: " + e.getMessage());
     // throw e; -> para detener la validación
   }
});

// Solicita la validación del fichero XML
validador.validate(new StreamSource(nombreFichero));
```

Validación de documentos

- Sobre el ejemplo de código anterior cabe destacar:
  - El manejador de eventos de error se ha declarado como una clase anónima por simplificación.
  - Sería mejor implementar una clase que registre los errores de validación en una lista.
  - En tal caso, al finalizar el método `validate` podremos saber si se han producido errores consultando la lista al manejador.

➔ **Importante**: el marco de validación presentado es general. Se puede aplicar también al framework JAXB.

Referencias

- Especificaciones:
  - XML: **http://www.w3.org/TR/REC-xml/**
  - XML Schema: **http://www.w3.org/TR/xmlschema-0/**

- Tutoriales:
  - **http://www.w3schools.com/xml/**