LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE INFORMACIÓN

UNIDAD 4:

Definición de esquemas y vocabularios en XML

INDICACIONES TAREA 4 : RELACIÓN CON LA TEORÍA

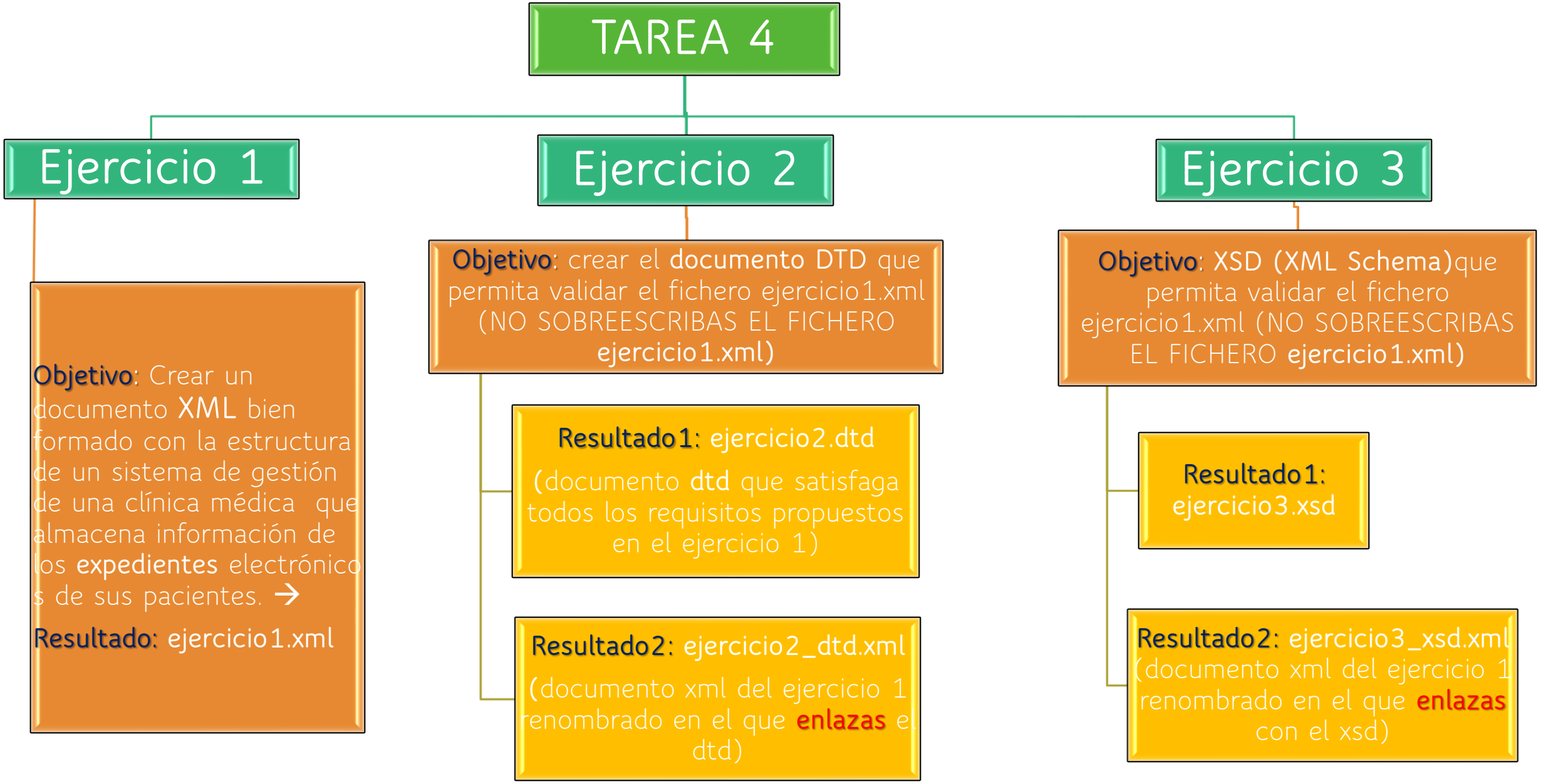
FPA FORMACIÓN PROFESIONAL ANDALUZA

Elena Fernández Chirino

DEPARTAMENTO DE INFORMÁTICA

http://www.iestrassierra.com

1.- INDICACIONES GENERALES TAREA 4



Ejercicio 1

¿Qué parte de la unidad se está trabajando es esta actividad?

Estructura de un documento XML bien construido (Unidad 1)

PRÓLOGO: 2 partes

Declaración XML

- Declaración la versión
- Declaración de la codificación empleada para representar los caracteres
- Declaración de la autonomía del documento

```
<?xml version= “1.0” encoding=”iso-8859-1” standalone=”yes” ?>
```

Declaración tipo documento

```
<!DOCTYPE Nombre_tipo ...>
```

CUERPO, EJEMPLAR O RAÍZ:

Contiene los datos reales del documento. Está formado por elementos anidados.

espacios de nombres, o `namespaces`;

Ejercicio 1

Nuevo concepto: Espacio de nombres o NAMESPACES

ESPACIO DE NOMBRES

- Los **espacios de nombres en XML (XML namespaces)** se utilizan para evitar conflictos de nombres entre elementos y atributos en un documento XML. Permiten asignar un identificador único a cada elemento y atributo, incluso si tienen el mismo nombre, pero pertenecen a diferentes vocabularios o dominios.
- Para más información: Espacios de nombres en XML - Eniun

- **Por ejemplo**, varios documentos XML se pueden combinar entre sí, pudiendo en estos casos coincidir el nombre de algunos elementos. Dos documentos XML podrían contener un elemento llamado “carta”, pero con significados distintos.
- Espacios de nombres en XML | Tutorial de XML | Abrirllave.com

NOTA:

No es obligatorio usar namespaces en la Tarea 4, pero sí tenemos que aprender el concepto en esta unidad.

EJERCICIOS 2 Y 3

¿Todo documento XML bien formado es válido? NO

VALIDACIÓN DE UN DOCUMENTO XML

- Cuando un documento XML **cumple estrictamente las reglas generales de creación XML** se dice que **está bien formado**.
- Cuando además sigue las reglas de un documento de validación entonces se dice que **es válido**.

TÉCNICAS PARA VALIDAR UN DOCUMENTO XML:

Las técnicas más populares para validar documentos son:

- **DTD** (Document Type Definition) → Se deberá realizar en el ejercicio 2
- **XML** (Schema o esquemas XML) → Se deberá realizar en el ejercicio 3

EJERCICIO 2

Generar un documento DTD que valide el ejercicio 1

- ✓ 1º Crearemos el documento DTD: **ejercicio2.dtd**
- ✓ 2º Copiamos el ejercicio1.xml y lo renombraremos como: **ejercicio2_dtd.xml** y lo modificaremos para enlazarlo con **ejercicio2.dtd**
- ✓ Tipos de DTD:

  DTD incrustada: Es posible incluir la DTD en el mismo documento XML, en concreto en el prólogo, después de la declaración XML.

  DTD externa: Es posible separar ambos elementos, guardándolos en archivos diferentes. Así, se puede situar en un archivo aparte la DTD y después enlazarlo con el archivo XML mediante ***<!DOCTYPE nombre SYSTEM "archivo.dtd">***

En el ejercicio 2 lo haremos de esta forma

EJERCICIO 3

Generar un XML Schema (Extensión del archivo .xsd)

- ✓ 1º Crearemos el documento XSD: **ejercicio3.xsd**
- ✓ 2º Copiamos el ejercicio1.xml y lo renombraremos como: **ejercicio3_xsd.xml** y lo modificaremos para enlazarlo con **ejercicio3.xsd**
- ✓ Nota inicial: Los elementos y atributos XML que se utilizan para generar esquemas pertenecen al espacio de nombres XML Schema, con el URI: http://www.w3.org/2001/XMLSchema.
- ✓ Un documento XML Schema tiene como **elemento raíz**, ejemplar, un elemento *<xs:schema>*. Este elemento contendrá las declaraciones de todos los elementos y atributos que puedan aparecer en un documento XML asociado. Este elemento tendrá un elemento hijo que será el elemento raíz del documento XML, que declararemos con *<xs:element>*.
- ✓ Por tanto, nuestro documento .xsd deberá comenzar:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

EJERCICIO 2: DTD

Los DTDs describen la estructura de los documentos XML mediante declaraciones

Tipos de declaraciones DTD

- Declaraciones de **entidades:** permite **definir atajos para caracteres especiales o textos largos que se utilizan frecuentemente**.
  - **Generales**
    - Internas
    - Externas
  - **De parámetro o paramétrica**
    - Internas
    - Externas
- Declaraciones de **notaciones**. Cuando se incluyen ficheros binarios en un fichero, se indica a la aplicación cómo ha de hacerse cargo de ellos utilizando notaciones. La sintaxis para declarar notaciones es: `<!NOTATION nombre SYSTEM aplicacion>` → EJEMPLO:
- Declaraciones de **elementos**.
  - Terminales
  - No terminales
- Declaraciones de **atributos**.

LAS MÁS UTILIZADAS

EJERCICIO 2: DTD. Declaración de entidades generales internas

Predefinidas en el lenguaje

- **<:** Se corresponde con el signo menor que, <.
- **>:** Hace referencia al signo mayor que, >.
- **":** Son las comillas rectas dobles, ".
- **':** Es el apóstrofe o comilla simple, '.
- **&:** Es el et o ampersand, &.

Definidas propias

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE libros [
3     <!ELEMENT libros (libro)+>
4     <!ELEMENT libro (#PCDATA)>
5
6     <!ENTITY autor "Cervantes">
7     <!ENTITY editorial "Alfaguara">
8   ]>
9
10 <libros>
11     <libro>Don Quijote de la Mancha fue escrito por &autor;</libro>
12     <libro>SIDI fue escrito por Arturo Pérez-Reverte y publicado por &editorial;</libro>
13     <libro>Tiempos recios fue escrito por Mario Vargas LLosa y publicado por &editorial;</libro>
14 </libros>
```

<!ENTITY nombre_entidad "valor_entidad">

Se hace referencia a la entidad usando &nombre_entidad

Al abrir el fichero en un navegador, se vería:

```
−<libros>
   <libro>Don Quijote de la Mancha fue escrito por Cervantes</libro>
  −<libro>
     SIDI fue escrito por Arturo Pérez-Reverte y publicado por Alfaguara
   </libro>
  −<libro>
     Tiempos recios fue escrito por Mario Vargas LLosa y publicado por Alfaguara
   </libro>
 </libros>
```

EJERCICIO 2: DTD. Declaración de entidades generales externas

---

- La declaración de una entidad externa **cuando sólo va a ser utilizada por una única aplicación** es:

```
<!ENTITY nombre_entidad SYSTEM "http://localhost/docsxml/fichero_entidad.xml">
```

- La declaración de una entidad externa **cuando va a ser utilizada por varias aplicaciones** es:

```
<!ENTITY nombre_entidad PUBLIC "identificador público formal" "camino hasta la DTD (uri)">
```

EJERCICIO 2: DTD. Declaración de entidades de parámetros

Permiten dar nombres a partes de un DTD y hacer referencia a ellas a lo largo del mismo. Son especialmente útiles cuando varios elementos del DTD comparten listas de atributos o especificaciones de contenidos. Se denotan por % entidad;

Internas

```
<!ENTITY % direccion "calle, numero?, ciudad, cp"> y se puede usar como sigue:
<!ELEMENT almacen (%direccion;, web)>
<!ELEMENT oficina (%direccion;, movil)>
<!ELEMENT central (%direccion;, telefono)>
<!ELEMENT tienda (%direccion;, fax)>
```

Externas

Permite incluir en un DTD elementos externos, lo que se aplica en dividir la definición DTD en varios documentos.

```
<!ENTITY %persona SYSTEM "persona.dtd">
```

EJERCICIO 2: DTD. Declaración de elemento terminal

La declaración de elementos en una DTD (Definición de Tipo de Documento) en XML se realiza utilizando la sintaxis ***<!ELEMENT nombre_del_elemento tipo_de_contenido >.*** Aquí ***tipo_de_contenido*** especifica el contenido permitido en el elemento.

Elemento terminal: Aquellos que no contienen más elementos →***tipo_de_contenido***

- **#PCDATA**: Indica que los datos son analizados en busca de etiquetas, resultando que el elemento no puede contener elementos, es decir solo puede contener datos de tipo carácter exceptuando los siguientes: <, [, &,], >.

```
<!ELEMENT ejemplo (#PCDATA)>
```

XML asociado **correcto**:

```
<ejemplo />
o
<ejemplo>Esto es un ejemplo</ejemplo>
```

- **EMPTY**: Indica que el elemento no es contenedor, es vacío, es decir, que no puede tener contenido.

```
<!ELEMENT ejemplo EMPTY>
```

XML asociado **correcto**:

```
<ejemplo></ejemplo> ó <ejemplo />
```

- **ANY**: Permite que el contenido del elemento sea cualquier cosa (texto y otros elementos). No es recomendable usar este tipo de elemento.

EJERCICIO 2: DTD. Declaración de elemento NO terminal

Estos son los elementos que están formados por otros elementos. Para definirlos utilizamos referencias a los grupos que los componen tal y como muestra el ejemplo: ***<!ELEMENT A (B, C)>.*** En este caso se ha definido un elemento ***A*** que está formado por un elemento ***B*** seguido de un elemento ***C***..

Operadores: Además, también hay operadores que nos permiten definir la cardinalidad de un elemento:

- **Operador opción, ?**: Indica que el elemento no es obligatorio.

```
<!ELEMENT telefono (trabajo?, casa )>
```
En el elemento telefono el subelemento trabajo es opcional

- **Operador uno-o-más, +**: Define un componente presente al menos una vez.

```
<!ELEMENT provincia (nombre, (cp, ciudad)+ )>
```
El elemento provincia, está formado por el nombre y otro grupo, que puede aparecer una o varias veces

- **Operador cero-o-mas, *** : Define un componente presente cero, una o varias veces.

```
<!ELEMENT provincia (nombre, (cp, ciudad)* )>
```
El elemento provinicoa, está formado por el nombre y otro grupo, que puede no aparecer o aparecer varias veces

- **Operador de elección, |**: Cuando se utiliza sustituyendo las comas en la declaración de grupos indica que para formar el documento XML hay que elegir entre los elementos separados por este operador.

```
<!ELEMENT provincia (nombre, (cp | ciudad) )>
```
El elemento provincia, está formado por el nombre y el CP o por el nombre y la ciudad

EJERCICIO 2: DTD. Declaración de listas

Mediante las listas vamos a declarar los atributos asociados a un elemento:

```
<! ATTLIST nombre_elemento_del_atributo nombre_atributo tipo_del_atributo "Valor_por_defecto">
```

Si un elemento tiene más de un atributo:

```
<!  ATTLIST  nombre_elemento_del_atributo1  nombre_atributo1  tipo_del_atributo1  "Valor_por_defecto"
                                          nombre_atributo2   tipo_del_atributo2   "Valor_por_defecto"
                  ..........................................................................>
```

Tipos de atributos:

- **Enumeración**: El atributo solo puede tomar uno de los valores determinados dentro de un paréntesis y separados por el operador |.

```
<!ATTLIST fecha dia_semana (lunes|martes|miércoles|jueves|viernes|sábado|domingo) #REQUIRED>
```

- **CDATA**: Se utiliza cuando el atributo es una cadena de texto.

DTD:
```
<!ATTLIST ejemplo color CDATA #REQUIRED>
```

XML asociado:
```
<ejemplo color="verde" />
```

- **ID**: Permite declarar que el valor del atributo (no el nombre) debe ser único y no se puede repetir en otros elementos o atributos.

DTD:
```
<!ATTLIST libro codigo ID #REQUIRED>
```

XML asociado:
```
<libro codigo="Q1">El Quijote</libro>
```

- **IDREF**: Permite hacer referencias a identificadores. En este caso, el valor del atributo ha de corresponder con el de un identificador de un elemento existente en el documento.
- **NMTOKEN**: Permite determinar que el valor de un atributo ha de ser una sola palabra compuesta por los caracteres permitidos por XML, es decir letras, números y los caracteres “:”, “_”, “-” o “.”.

EJERCICIO 2: DTD. Declaración de listas

Tipos de atributos → Como declarar si un atributo es obligatorio o no

- **#IMPLIED**, determina que el atributo sobre el que se aplica es opcional.

DTD:
```
<!ATTLIST ejemplo color CDATA #IMPLIED>
```

XML asociado válido:
```
<ejemplo color="verde" />
<ejemplo color="" />
<ejemplo/>
```

- **#REQUIRED**, determina que el atributo tiene carácter obligatorio.

DTD:
```
<!ATTLIST ejemplo color CDATA #REQUIRED>
```

XML asociado válido:
```
<ejemplo color="" />
<ejemplo color="verde" />
```

XML asociado no válido:
```
<ejemplo />
```

- **#FIXED**, permite definir un valor fijo para un atributo, independientemente de que ese atributo se defina explícitamente en una instancia del elemento en el documento XML.

DTD:
```
<!ATTLIST ejemplo color CDATA #FIXED "verde">
```

XML asociado NO válido:
```
<ejemplo color="rojo" />
```

XML asociado válido:
```
<ejemplo color="verde" />
```

- **Literal**, (valor por defecto) asigna por defecto a un atributo el valor dado por una cadena entre comillas, pero puede tomar otro valor

DTD:
```
<!ATTLIST ejemplo colo (rojo|verde|amarillo) "verde">
```

XML asociado:
```
<ejemplo color="verde" />
```

EJERCICIO 2: DTD. EJEMPLO RESUELTO

La empresa **Reggio** tiene establecimientos por toda Italia, pero su sede central está en Cesena, al igual que el almacén donde se distribuye a todos los demás establecimientos. Esta empresa distribuye alpiste para pájaros, así como otros artículos ornitológicos.

Cada **establecimiento** tiene una **tienda**, así como un **almacén**, que pueden o no estar en la misma ubicación. Cuando se hace un pedido a la fábrica por parte de los establecimientos, éstas reciben los artículos en su almacén, y la documentación (albarán y pago) se remite a la tienda.

Cada **pedido** debe tener los datos siguientes:

- **Datos del establecimiento** que realiza el pedido (Nombre, dirección para envío y dirección almacén (si es la misma, sólo aparecerá una).
- **Código de pedido** (Cadena de caracteres formada por: Código establecimiento (1 letra y 2 números), número pedido (4 números), un guión y Año (dos números), por ejemplo: E180021-17
- **Nombre del empleado** que realiza el pedido.
- **Fecha** de pedido.
- **Tipo de envío** (cuyos valores son: Normal o Urgente)

Respecto a los **artículos del pedido**, se guardarán los siguientes datos:

- **Código** del artículo (formado por tres letras y 3 números, por ejemplo: ZZZ134.
- Número de **unidades**.
- **Precio por unidad**.
- **Observaciones**.

Herramienta para editar y validar documentos en el lenguaje XML→XML Copy Editor (Gratuito)

EJERCICIO 2: DTD. EJEMPLO RESUELTO

```
<--Declaración de elemento no terminal pedido formado por los subelementos (establecimiento, empleado,..., artíiculos  -->
<!ELEMENT pedido (establecimiento,empleado,fecha_pedido,articulos)>
        <!ATTLIST pedido cod_pedido CDATA #REQUIRED>
    <!ATTLIST pedido tipo (normal|urgente) #REQUIRED>
    <!ELEMENT establecimiento (nomb_estab,direccion_envio,direccion_almacen?)>
        <!ELEMENT nomb_estab (#PCDATA)>
        <!ELEMENT direccion_envio (via,numero,localidad,provincia,cp)>
        <!ELEMENT direccion_almacen (via,numero,localidad,provincia,cp)>
            <!ELEMENT via (#PCDATA)>
            <!ELEMENT numero (#PCDATA)>
            <!ELEMENT localidad (#PCDATA)>
            <!ELEMENT provincia (#PCDATA)>
            <!ELEMENT cp (#PCDATA)>
    <!ELEMENT empleado (#PCDATA)>
    <!ELEMENT fecha_pedido (#PCDATA)>
        <!ELEMENT articulos (articulo+)>
    <!ELEMENT articulo (unidades,precio,observaciones?)>
                <!ATTLIST articulo cod_articulo CDATA #REQUIRED>
        <!ELEMENT unidades (#PCDATA)>
        <!ELEMENT precio (#PCDATA)>
                <!ATTLIST precio moneda CDATA #REQUIRED>
        <!ELEMENT observaciones (#PCDATA)>
```

Declaramos una lista en el elemento pedido, llamado, cod_pedido de tipo cadena obligatorio

Declaración del elemento terminal via que solo puede tener datos de tipo carácter

Elemento no terminal articulos que está formado por artículo que puede aparecer más de una vez

tarea04_01.dtd

Vamos a crear el XML y enlazamos este DTD

EJERCICIO 2: DTD. EJEMPLO RESUELTO

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pedido SYSTEM "tarea04_01.dtd">
<pedido cod_pedido="E180021-17" tipo="urgente">
        <establecimiento>
            <nomb_estab>Manitoba</nomb_estab>
            <direccion_envio>
                <via>Vittorio</via>
                <numero>1</numero>
                <localidad>Cesena</localidad>
                <provincia>Cesena</provincia>
                <cp>00400</cp>
            </direccion_envio>
                        <direccion_almacen>
                <via>Vittorio</via>
                <numero>5</numero>
                <localidad>Cesena</localidad>
                <provincia>Cesena</provincia>
                <cp>00400</cp>
            </direccion_almacen>
        </establecimiento>
        <empleado>Roberto Rossini</empleado>
        <fecha_pedido>2017-05-01</fecha_pedido>
                <articulos>
                        <articulo cod articulo="ZZZ134">
                    .....
                        </articulo>
                </articulos>
</pedido>
```

La solución completa está disponible en la plataforma Moodle:

RETROALIMENTACIÓN COMPLETA

EJERCICIO 3

Generar un XML Schema (Extensión del archivo .xsd)

- ✓ 1º Crearemos el documento XSD: **ejercicio3.xsd**
- ✓ 2º Copiamos 1jercicio1.xml y lo renombraremos el archivo XML creado en el ejercicio 1 como: **ejercicio3_xsd.xml** y lo modificaremos para enlazarlo con **ejercicio3.xsd**
- ✓ Nota inicial: Los elementos y atributos XML que se utilizan para generar esquemas pertenecen al espacio de nombres XML Schema, con el URI: http://www.w3.org/2001/XMLSchema.
- ✓ Un documento XML Schema tiene como **elemento raíz**, ejemplar, un elemento *<xs:schema>*. Este elemento contendrá las declaraciones de todos los elementos y atributos que puedan aparecer en un documento XML asociado. Este elemento tendrá un elemento hijo que será el elemento raíz del documento XML, que declararemos con *<xs:element>*.
- ✓ Por tanto, nuestro documento .xsd deberá comenzar:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

EJERCICIO 3: Tipos de elementos en XML Schema

Los **elementos** se utilizan para especificar las etiquetas válidas en un documento XML. Todos los elementos que se vayan a utilizar en el ejemplar XML tienen que estar declarados en el esquema. son **simples** (no puede contener otros elementos o atributos) o **complejos**:

Tipo simple:

```
<xsd:element name="nombreElemento"
ref="elementoReferenciado"
type="tipoDato"
minOccurs="valor"
maxOccurs="valor"
fixed="valor"
default="valor"/>
```

Valores máx y mín por defecto. Son opcionales

También opcionales y especifican un valor fijo o un valor por defecto rspectivamente

Tipo complejo

```
<!--Este es un ejemplo de XML Schema-->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!--Definición del elemento "libro"-->
  <xs:element name="libro">
    <xs:complexType>
      <!--El elemento "libro" tiene dos elementos hijos: "titulo" y "autor"-->
      <xs:sequence>
        <xs:element name="titulo" type="xsd:string"/>
        <xs:element name="autor" type="xsd:string"/>
      </xs:sequence>
      <!--El elemento "libro" tiene un atributo "isbn"-->
      <xs:attribute name="isbn" type="xsd:string"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

EJERCICIO 3: Tipos de elementos en XML Schema

Los **elementos** se utilizan para especificar las etiquetas válidas en un documento XML. Todos los elementos que se vayan a utilizar en el ejemplar XML tienen que estar declarados en el esquema. son **simples** o **complejos**:

-**Tipo simple:** No pueden contener otros elementos o atributos. Se pueden crear nuevos elementos "simpleType" a partir de uno ya existente, añadiendo condiciones a alguno de los tipos predefinidos en el XML Schema. Para ello se usa el elemento ***<xs:restriction>.***

- **Tipo complejo**: Pueden estar compuestos por otros elementos y/o atributos. Su contenido está definido entre las etiquetas de inicio y de cierre del elemento.

- **Esquema**, **xs:schema**, contiene la definición del esquema, es el elemento tipo complejo básico.
- **xs:complexType**, entre sus etiquetas de inicio y cierre se definen los elementos de tipo complejo. Formados por subelementos predefinidos en XML Schema como:
  - secuencias (***xs:sequence***),
  - alternativas (***xs:choice***),
  - secuencias no ordenadas (***xs:all***),
  - contenido mixto (definido dando valor true al atributo mixed del elemento ***<xs:complexType mixed="true">***),
  - y elemento vacío.

- **Referencia a otros elementos**: Además, en un documento xsd podemos definir elementos de forma global y luego hacer referencias a ellos desde otros elementos. Esto es muy útil si a lo largo de un documento se repiten determinados elementos.

EJERCICIO 3: Tipos de atributos en XML Schema

El elemento **"atributo"** permite definir los atributos de los elementos en el documento xsd para usarlos adecuadamente en el documento XML

Su sintaxis es la siguiente:

```
<xsd:attribute name="nombreAtributo"
ref="atributoReferenciado"
type="tipoAtributo"
use="valor"
fixed="valor"
default="valor"/>
```

Indica si es optional (valor por defecto), required o prohibited

Son opcionales e indican un valor fijo para el atributo o un valor por defecto, respectivamente

Ejemplo: crear el código xsd para el siguiente xml

xml

```
<libro codigo="001">
    <titulo>Don Quijote</titulo>
    <autor>Miguel de Cervantes</autor>
</libro>
```

Solución

```
<xs:element name="libro">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nombre"  type="xs:string"/>
            <xs:element name="autor" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="codigo" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

EJERCICIO 3: Tipos de datos en XML Schema

Son los distintos valores que puede tomar el **atributo type** cuando se declara un elemento o un atributo y representan el tipo de dato que tendrá el elemento o atributo asociado a ese **type** en el documento XML

- **string:** Se corresponde con una cadena de caracteres UNICODE.
- **boolean:** Representa valores lógicos, es decir que solo pueden tomar dos valores, true o false.
- **integer:** Número entero positivo o negativo.
- **positiveInteger:** Número entero positivo.
- **negativeInteger:** Número entero negativo.
- **decimal:** Número decimal.
- **dateTime:** Representa una fecha y hora absolutas.
- **duration:** Representa una duración de tiempo expresado en años, meses, días, horas, minutos segundos.
- **time:** Hora en el formato hh:mm:ss.
- **date:** Fecha en formato CCYY-MM-DD.
- **gYearMonth:** Representa un mes de un año determinado mediante el formato CCYY-MM.
- **gYear:** Indica un año gregoriano, el formato usado es CCYY.
- **gMothDay:** Representa un día de un mes mediante el formato --MM-DD.
- **gDay:** Indica el ordinal del día del mes mediante el formato - -DD.
- **gMonth:** Representa el mes mediante el formato - -MM.
- **anyURI:** Representa una URI.
- **language:** Representa los identificadores de lenguaje, sus valores están definidos en RFC 1766.
- **ID, IDREF, ENTITY, NOTATION, NMTOKEN:** Representan lo mismo que en los DTD's.

EJERCICIO 3: Facetas de los tipos de datos

Son los distintos valores que puede tomar el atributo type cuando se declara un elemento o un atributo y representan el tipo de dato que tendrá el elemento o atributo asociado a ese type en el documento XML

¿Cuáles son las restricciones que podemos aplicar sobre los valores de los datos de un elemento o atributo?

- Están definidas por las facetas o restricciones, que solo pueden aplicarse sobre tipos simples utilizando el elemento xs:restriction, que tiene como atributo "base" en el que se indica el tipo de dato a partir del cual se define la restricción.

- Las facetas se expresan como un elemento dentro de una restricción y se pueden combinar para lograr restringir más el valor del elemento. Son, entre otros:
  - enumeration: Restringe a un determinado conjunto de valores.
  - length, minlength, maxlentgh: Longitud del tipo de datos.
  - whitespace: Define el tratamiento de espacios (preserve/replace, collapse).
  - (max/min)(In/Ex)clusive: Límites superiores/inferiores del tipo de datos. Cuando son Inclusive el valor que se determine es parte del conjunto de valores válidos para el dato, mientras que cuando se utiliza Exclusive, el valor dado no pertenece al conjunto de valores válidos.
  - totalDigits, fractionDigits: número de dígitos totales y decimales de un número decimal.
  - pattern: Permite construir máscaras que han de cumplir los datos de un elemento. La siguiente tabla muestra algunos de los caracteres que tienen un significado especial para la generación de las máscaras.

EJERCICIO 3: Definición de tipos de datos simples

En los DTD se diferencia entre los elementos terminales y los no terminales ¿en este caso también? Sí, este lenguaje permite trabajar tanto con datos simples como con estructuras de datos complejos, es decir, compuestos por el anidamiento de otros datos simples o compuestos. En este apartado vamos a indicar las tres diferentes maneras que existen de extender un tipo de datos simple

En XML Schema, puedes aplicar restricciones, uniones y listas a los tipos de datos:

- **Restricción:** Se aplica una restricción sobre un tipo de datos XSD ya definido y se establece el rango de valores que puede tomar. Las restricciones son conocidas como facetas. Por ejemplo, puedes crear un elemento simple de nombre `edad` que representa la edad de un alumno de la ESO, por tanto, su rango está entre los 12 y los 18 años.

- **Unión**: Consiste en combinar dos o más tipos de datos diferentes en uno único. Por ejemplo, puedes crear un tipo de dato para las medidas de longitud que una el sistema internacional (cm, m y km) con el sistema inglés (pulgada, pie, yarda).

- **Lista:** Permite crear una lista de valores de un tipo simple determinado. Este tipo puede ser directo como **xs:string** o **xs:int** o definido por nosotros. La lista se separa por espacios en blanco. El elemento **xs:list** creado podrá tomar la posición de un tipo de dato simple. Puede ser útil para crear una especie de **xs:sequence** pero para tipos de datos simples.

EJERCICIO 3: Documentación del esquema

La documentación del esquema en XML Schema se realiza a través del elemento xs:annotation, que permite guardar información adicional. Este elemento puede contener una combinación de otros dos elementos:

- xs:documentation: Además de contener elementos del esquema XML, puede contener elementos XML bien estructurados. También permite determinar el idioma del documento mediante el atributo xml:lang.

- xs:appinfo: Se diferencia muy poco del elemento anterior, aunque lo que se pretendió inicialmente era que xs:documentation fuese legible para los usuarios y que xs:appinfo guardase información para los programas de software. También es usado para generar una ayuda contextual para cada elemento declarado en el esquema.

Estos elementos permiten incorporar información adicional al esquema, como quién es el autor, limitaciones de derechos de autor, utilidad del esquema, etc., de una manera que se preserva a través de las transformaciones del documento.

EJERCICIO 3: Ejemplo de documentación

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

    <xs:annotation>
        <xs:documentation xml:lang ="es-es">
            Materiales para formación e-Learning
            <modulo>Lenguajes de marcas y sistemas de gestión de información.<modulo>
            <fecha_creación> 2011<fecha_creacion>
            <autor> Nuky La Bruji</autor>
        </xs:documentation>
    </xs:annotation>

    <xs:element name="lmsgi"  type=xs:string>
        <xs:annotation>
            <xs:appinfo>
                <texto_de_ayuda>Se debe de introducir el nombre completo del tema</texto_de_ayuda>
            <xs:appinfo>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

Herramientas de creación y validación

- Notepad ++ (Gratuito)
- Editix XML Editor (Gratuito)
- XML Copy Editor (Gratuito)
- NetBeans (Gratuito)
- Visual Studio Code (Gratuito)

También tenemos herramientas de validación online, a continuación, algunas webs que nos validan un xml, dtd y schema:

- http://www.xmlvalidation.com
- http://www.utilities-online.info/xsdvalidation/