Tema 3. DTD.

Cabe hacer hincapié en que cuando un documento XML no contiene un DTD, cualquier etiqueta que aparezca en el mismo se considerará válida. De manera que el parser solo podrá comprobar que el documento está bien formado. La existencia del DTD permite asegurar que los documentos siguen las reglas del lenguaje.

Cuando se pretende usar XML en un entorno profesional, es imprescindible la especificación de un DTD que defina formalmente el lenguaje de etiquetado requerido. Este debe ser el primer paso antes de escribir ningún documento XML. Se puede observar una cierta similitud con la forma de trabajo con una base de datos. Donde el primer paso es la definición del esquema y posteriormente el resto de usuarios ya pueden trabajar contra este esquema.

En el tema anterior hemos visto como crear documentos XML bien formados. En este tema vamos a conseguir crear documentos bien formados y válidos. Por ejemplo, el primer ejercicio del tema anterior versaba sobre un documento XML que una empresa mandaba a un sitio web para publicitar ciertos artículos (coches). Imaginemos que muchas empresas mandan información a este sitio web y cada una de ellas implementa su propia versión de documento XML. EL sitio web debe
ser capaz de entender todos los documentos que le llegan y tratar la información que contienen. Esto es inviable. La solución óptima es que la empresa web anunciante crea un DTD donde dice cómo deben ser exactamente los documentos que va a recibir. Todas las empresas que quieran anunciar productos enviaran documentos XML validados por esta DTD (ya veremos como más adelante). De esta forma nos aseguramos que el sitio web va a recibir documentos XML válidos y con la misma estructura.

Objetivos.

- Conocer los elementos principales que contiene un DTD.
- Saber crear DTD para definir gramáticas sencillas.
- Saber crear documentos XML bien formados y válidos.
- Saber validar un XML contra su DTD asociado.

Contenido.

1. Declaración del DTD.
2. Declaración de tipos de elementos.
3. Declaración de tipos de atributos.
4. Declaración de Entidades.
5. La declaración del tipo de documento DOCTYPE.

1.- Declaración del DTD.

Como se ha explicado anteriormente, el DTD, que es de carácter opcional forma parte del prólogo del documento, apareciendo justo después de la declaración XML. EL DTD puede incluir directamente declaraciones de etiquetado y/o hacer referencia a una entidad externa, normalmente un archivo, que contiene declaraciones de etiquetado. Además la declaración debe especificar cuál es el elemento raíz del documento.

Ejemplo

```
<?xml versión=”1.0”?>
<[¡DOCTYPE elemento_raiz [
Las declaraciones internas van aquí
]>
<elemento_raiz>...</elemento_raiz>
```

Normalmente un DTD se utiliza para validar un gran número de documentos XML. La mayoría de las veces tiene poco sentido que el DTD esté incluido dentro del documento XML ya que se tendría que repetir en todos los documentos XML pertenecientes a un mismo lenguaje (DTD). Teniendo esto en cuenta se puede distinguir entre dos tipos de referencias externas:

- Un documento DTD aún no publicado. Se especifica con la palabra SYSTEM seguida de la URL con la ubicación del documento: <!DOCTYPE elemento_raiz SYSTEM “archivo_declaraciones.dtd”>.
- En caso de utilizar un DTD que ha sido publicado utilizaremos la palabra PUBLIC seguida por el identificador público asociado a este DTD. Sigue siendo necesario incluir la URL al fichero DTD que solo será utilizado en caso de fallar la localización del fichero. Usando el identificador público: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">. Que es la DTD pública utilizada para validar documentos XHTML estrictos.

Para continuar , establecemos un ejemplo en el que utilizar una DTD. Se quiere almacenar los mensajes de móviles que se envían a un servidor. Los datos que se guardan son los siguientes:

- Número de teléfono del usuario.
- Fecha de envío.
- Hora de envío.
- Contenido del mensaje.

Un ejempo de documento XML que guarde estos mensajes SMS podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BDsms
[
<!ELEMENT BDsms (sms*)>
```

```
<!ELEMENT sms (telefono,fecha,hora,mensaje)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
]>
<BDsms>
<sms>
  <telefono> 212456389</telefono>
  <fecha>01/04/2015</fecha>
  <hora>20:50</hora>
  <mensaje>Hola María</mensaje>
</sms>
<sms>
  <telefono> 212458562</telefono>
  <fecha>01/05/2015</fecha>
  <hora>18:50</hora>
  <mensaje>Estoy bien</mensaje>
</sms>
<sms>
  <telefono> 212956387</telefono>
  <fecha>22/03/2015</fecha>
  <hora>04:30</hora>
  <mensaje>Llegaré tarde</mensaje>
</sms>
</BDsms>
```

2.- Declaración de tipos de elementos.

Durante el proceso de diseño se identifican los distintos tipos de elementos que forman parte del documento. En el DTD hay que incluir la declaración de cada uno de ellos.

PCData: su significado en ingles es Parsed Charater Data. Indica que entre la etiqueta de apertura y cierre de ese elemento se almacenarán caracteres como texto y serán analizados por un parser, pudiendo aparecer entidades y elementos, si es solo texto podría confundir al parser.

CDATA: su significado en inglés es Character Data. A efectos es practicamente igual que PCData pero el contenido de ese elemento, entre sus etiquetas de apertura y cierre, no se analizará por parser de análisis de entidades y elementos XML.

Los elementos se declaran de una de las dos siguientes maneras:

```
<!ELEMENT nombre tipo_contenido>
```

```
<!ELEMENT nombre tipo_contenido(nodos_hijos)>
```

El nombre del elemento debe ser un nombre XML válido. Solo podrá haber una declaración por elemento.

Podemos tener diferentes tipos de elementos clasificados según su contenido:

- vacíos.
- que solo contienen datos.
- que solo contienen elementos y mixtos.

Elementos vacíos.

Un ejemplo es la declaración
 de XHTML. Se declaran especificando la palabra EMPTY.

```
<!ELEMENT br EMPTY>
```

Por ejemplo:

DTD. Los elementos AAA pueden contener solamente atributos pero no texto (tutorial.dtd):

```
<!ELEMENT XXX (AAA+)>
<!ELEMENT AAA EMPTY>
<!ATTLIST AAA  true ( yes | no ) "yes">
```

Por ejemplo el siguiente fichero xml:

```
 <!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA true="yes"/>
  <AAA true="no"></AAA>
</XXX>
```

Ambas formas son válidas. En el segundo caso la etiqueta de cierre debe seguir inmediatamente al de apertura.

Otro ejemplo de fichero xml (con errores):

```
 <!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA true="yes"/>
  <AAA true="no"></AAA>
  <AAA>     </AAA>
  <AAA>Hola!</AAA>
</XXX>
```

El elemento AAA no puede contener ningún texto y el tag de apertura se tiene que cerrar inmediatamente.

Elementos que solo contienen datos.

En la declaración se especifica con **#PCDATA**.

```
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
```

Por ejemplo:

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

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

Elementos que solo contienen elementos.

Se distinguen dos tipos de relación entre los elementos hijos. Secuencia o alternativa.

Secuencia: Imaginemos un elemento mensaje que contiene elementos hijos: remitente, destinatario, asunto y cuerpo. La declaración del elemento mensaje sería:

```
<!ELEMENT mensaje (remitente, destinatario, asunto, cuerpo)>
```

En nuestro ejemplo del mensaje podemos ver que el elemnto sms está compuesto por telefono, fecha, hora y mensaje. Esta es una relación padre-hijos. Imaginemos que una vez definida la DTD, nos damos cuenta que, por cada SMS recibido se pueden almacenar varios mensajes. En un primer momento se puede pensar que para salir del paso la solución sería esta:

```
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (telefono,fecha,hora,mensaje)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```

```
<!ELEMENT mensaje2 (#PCDATA)>
```

Pero esta no es una buena solución, estamos particularizando en lugar de llegar a una generalidad, además al modificar el DTD podemos obligar a cambiar el XML. La solución es modificar la ocurrencia de los elementos.

Además de especificar qué elementos hijos puede contener el elemento y en qué orden, se puede establecer cuántas veces aparece cada uno de ellos mediante un carácter que, basándose en las expresiones regulares, indique el factor de repetición:

El carácter '*****': el elemento o grupo de elementos puede repetirse 0 o más veces.

<!ELEMENT sms (telefono,fecha,hora,mensaje*)>

El carácter '**?**': el elemento o grupo de elementos puede aparecer 0 o 1 veces.

<!ELEMENT sms (telefono,fecha,hora,mensaje?)>

El carácter '**+**': el elemento o grupo de elementos puede repetirse 1 o más veces.

<!ELEMENT sms (telefono,fecha,hora,mensaje+)>

Alternativa: Cuando el elemento contiene uno y solo uno de los elementos hijos especificados.

Por ejemplo:

<!ELEMENT persona (física | jurídica)>

Ejemplo

<!ELEMENT elem (a, (b|c)*, d+, e?>

De acuerdo a esta declaración podemos encontrarnos con los siguientes documentos válidos:

<elem><a><d></d><elem>
<elem><a><d></d><d></d><e></e><elem>
<elem><a><d></d><elem>
<elem><a><c></c><c></c><d></d><elem>

Elementos mixtos.

Como ya vimos en el tema anterior este caso no suele utilizarse en XML. Se puede especificar qué elementos hijos podrán aparecer pero sin ningún control sobre su frecuencia o si forman parte de una secuencia de alternativa. La declaración sería:

<!ELEMENT elem (#PCDATA|a|b|c)*>

El formato de esta declaración es muy rígido: Siempre en primer lugar PCDATA, una lista alternativa, no se puede aplicar caracteres de repetición a los elementos hijos y debe especificarse obligatoriamente el carácter de repetición '*' a todo el grupo.

3.- Declaración de tipos de atributos.

Al igual que ocurre con los elementos, cada uno de los distintos atributos identificados en la fase de diseño debe declararse en la DTD. Una única declaración permite definir una lista de atributos asociados a un elemento. La sintaxis es la siguiente:

> **<!ATTLIST elemento nombre_atributo tipo_atributo valor_por_defecto>**
>
> **nombre_elemento:** es el elemento al que se le quiere añadir el atributo.
>
> **nombre_atributo**: es el nombre del atributo que se quiere añadir.
>
> **tipo_atributo**: existen muchos tipos de atributos (CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES)

Puede haber múltiples definiciones de listas de atributos para un mismo elemento. Pero si se declara varias veces el mismo atributo solo prevalece el primero.

3.1.- Atributos CDATA y NMTOKEN.

> El tipo de atributos ***CDATA*** consiste en una cadena de caracteres. Esta cadena puede incluir cualquier carácter a excepción de los caracteres especiales, incluidos los espacios en blanco. Si pretendemos limitar el tipo de caracteres que pueden aparecer como valor en el atributo, debemos utilizar el tipo ***NMTOKEN (letras, dígitos, guión "-", subrayado "_", punto "." y dos puntos":") es lo que se lama nombres válidos XML***. Es decir solo permite que aparezcan los mismos caracteres que utilizamos para definir elementos y atributos. Existe también la posibilidad de utilizar el tipo ***NMTOKENS***, esto indica que el atributo contendrá una lista de cadenas de tipo NMTOKEN, incluyendo el espacio en blanco " ", tabuladores o retornos de carro.
>
> > **<!ATTLIST coche color CDATA>**. La propiedad color puede tomar cualquier valor.
> >
> > **<!ATTLIST coche color NMTOKEN>**. La propiedad color puede tomar solo valores que contengan letras, dígitos, puntos, guiones y subrayados. Deben comenzar por letra y no pueden contener espacios en blanco.
> >
> > **<!ATTLIST coche color NMTOKENS>**. La propiedad color será una lista de NMTOKENS. Por ejemplo <coche color=”blanco negro gris”>

Por ejemplo:

DTD. Los atributos bbb y ccc siempre tienen que estar presentes, el atributo aaa es opcional:

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes
    aaa CDATA #IMPLIED
    bbb NMTOKEN #REQUIRED
    ccc NMTOKENS #REQUIRED>
```

Documento válido.

Todos los atributos obligatorios están presentes y sus valores son del tipo correcto:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">

<attributes aaa="#d1" bbb="a1:12" ccc=" 3.4 div    -4"/>
```

Todos los atributos obligatorios están presentes y sus valores son del tipo correcto:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">

<attributes bbb="a1:12" ccc="3.4 div  -4"/>
```

Documento con errores.

El caracter # no está permitido en los atributos de tipos NMTOKEN y NMTOKENS:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">

<attributes aaa="#d1" bbb="#d1" ccc="#d1"/>
```

El caracter de espacio en blanco " " no está permitido en los atributos de tipo NMTOKEN:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
<attributes bbb="A B C" ccc="A    B    C"/>
```

3.2.- Atributos enumerados.

Se usan cuando el valor del atributo está restringido a un conjunto de valores. En la declaración de usa el carácter **'|'** para separar los valores.

```
<!ATTLIST coche color (blanco | negro | gris)>
```

De esta forma la propiedad color solo puede tomar los valores “blanco”, “negro” o “gris”. Cualquier otro valor hará que la validación del documento XML falle.

Por ejemplo:

DTD. El elemento raíz XXX debe contener un elemento AAA seguido de un elemento BBB. El elemento AAA tiene que contener un elemento CCC seguido de un elemento DDD. El elemento BBB tiene que contener bien un elemento CCC o bien un elemento DDD:

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (CCC , DDD)>
<!ELEMENT BBB (CCC | DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Documento válido.

Un documento válido:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <CCC/>
  </BBB>
</XXX>
```

Otro documento válido:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <DDD/>
  </BBB>
</XXX>
```

Documento con errores.

El elemento BBB puede contener un elemento CCC o bien DDD pero no ambos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <CCC/> <DDD/>
  </BBB>
</XXX>
```

El elemento BBB puede contener un elemento CCC o DDD pero no ambos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <DDD/> <CCC/>
  </BBB>
</XXX>
```

3.3.- Atributos ID, IDREF e IDREFS.

Es frecuente que algunos elementos tengan algún valor que los identifica de forma ***unívoca***. Cuando un elemento contiene una propiedad de este tipo hay que asegurarse que esta no se repite en otro elemento. Incluso con elementos diferentes.

- **ID**: es un identificador que permite identificar al elemento de manera única en todo el documento XML.
- **IDREF**: es un identificafor de otro elemento del propio documento XML.
- **IDREFS**: es una lista de identificadores a otros elementos.

La sintaxis es la siguiente:

```
<!ATTLIST coche matricula ID>
```

De esta forma nos aseguramos que en todo el documento XML no habrá otro elemento con un identificador igual. El valor del atributo identificador debe seguir las mismas reglas que los nombres de atributos y elementos.

Además, como es un identificador único permite que otros elementos puedan hacer referencia a él. Para ello pueden utilizar el tipo **IDREF**. Por tanto los valores que puede tomar un atributo IDREF son el conjunto de identificadores que están declarados en el documento. También podemos utilizar el tipo **IDREFS** que no es más que extender la definición de IDREF a una lista de valores. Es decir, el valor de un atributo IDREF tiene que corresponder con el valor de algún atributo ID del documento. El valor del atributo IDREFS puede contener varias referencias a elementos con atributos ID separados por espacios en blanco.

Por ejemplo, dado el siguiente DTD.

- Los atributos id y mark determinan inequívocamente su elemento.
- Los atributos ref hacen referencia a estos elementos

```
<!ELEMENT XXX (AAA+ , BBB+, CCC+, DDD+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
<!ATTLIST AAA mark ID #REQUIRED>
<!ATTLIST BBB id ID #REQUIRED>
<!ATTLIST CCC ref IDREF #REQUIRED>
<!ATTLIST DDD ref IDREFS #REQUIRED>
```

Documento válido.

Todos los valores ID son únicos y todos los valores IDREF e IDREFS apuntan a elementos con IDs relevantes:

```
 <!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA mark="a1"/>
  <AAA mark="a2"/>
  <AAA mark="a3"/>
```

```
    <BBB id="b001" />
    <CCC ref="a3" />
    <DDD ref="a1 b001 a2" />
  </XXX>
```

Documento con errores·

No hay atributos ID con valor a3 ni b001:

```
 <!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA mark="a1"/>
  <AAA mark="a2"/>
  <BBB id="b01" />
  <CCC ref="a3" />
  <DDD ref="a1 b001 a2" />
</XXX>
```

El atributo ref en el elemento CCC es de tipo IDREF. Solo puede contener una referencia:

```
 <!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA mark="a1"/>
  <AAA mark="a2"/>
  <AAA mark="a3"/>
  <BBB id="b001" />
  <CCC ref="a1 b001 a2" />
  <DDD ref="a1 b001 a2" />
</XXX>
```

Valores por defecto para los atributos.

Además del nombre y el tipo del atributo también se puede especificar cómo debe comportarse el parser ante la presencia o ausencia de cierto atributo en un elemento del documento. Existen cuatro posibles alternativas:

- **#REQUIRED**. El atributo es de carácter obligatorio.
- **#IMPLIED**. El atributo es opcional.
- **#FIXED**. El atributo tiene un valor fijo declarado en el DTD.
- **Valor por defecto**. Si el atributo no está declarado toma este valor.

Algunos casos especiales son por ejemplo los atributos de tipo ID que no pueden ser opcionales o obligatorios pero no pueden tomar valores por defecto ni ser fijos. Si no se define ninguna de estas alternativas el atributo será por defecto opcional.

Ejemplos:

```
<!ATTLIST coche matricula ID #REQUIRED>
<!ATTLIST coche color CDATA #IMPLIED>
<!ATTLIST coche color CDATDA “gris”>
```

<!ATTLIST coche marca FIXED “Ford”>

Actividad

Se quiere crear una biblioteca de libros en un documento XML. Para ello se necesita crear una DTD que almacene los siguientes campos de un libro:

- Código de libro.
- Título.
- Editorial.
- Edición.
- ISBN.
- Número de páginas.
- Autor.

4.- Declaración de Entidades.

En general entidad se refiere a un objeto usado para guardar información y por ello necesariamente cada documento tiene el menos la entidad del propio documento. Permite guardar contenido que puede ser utilizado muchas veces y poder descomponer un documento grande en subconjuntos más manejables.

Una ENTIDAD ofrece una entrada abreviada que se puede situar en el documento XML. El nombre abreviado es lo que nosotros incluimos en el parámetro de nombre. Las ENTIDADES resultan muy útiles para repetir información o bloques grandes de texto que pueden estar guardados en archivos separados. El nombre abreviado va seguido de un punto y coma (;) en el documento XML (&abbName;)

Entidad Interna. Es la más sencilla. Consiste en abreviaturas definidas en el DTD. Por ejemplo:**<!ENTITY derechos “Copyright 2002”>**. Al definir esta entidad, en el documento XML podemos utilizarla escribiendo ‘&derechos;’. El parser cambiará la entidad por el valor asignado.

Entidad externa. El contenido no está dentro del DTD sino en cualquier otro sitio del sistema. Se hace referencia a su contenido mediante una URI precedida de la palabra SYSTEM o PUBLIC según proceda. La sintaxis es

<!ENTITY nombre SYSTEM “URI”>

Por ejemplo: **<!ENTITY intro SYSTEM http://ww.miservidor.com/intro.xml>**

Estas entidades externas permiten descomponer grandes archivos en unidades más pequeñas.

5.- La declaración del tipo de documento DOCTYPE.

Una declaración DOCTYPE es obligatoria si el documento se va a procesar en un entorno de validación. Para ser válida, la declaración DOCTYPE debe identificar una DTD que corresponda a la estructura del documento. Los analizadores no validadores aceptarán documentos sin declaraciones DOCTYPE, para este tipo de analizadores la declaración es opcional:

- Solo puede aparecer una vez dentro del documento.
- Debe seguir a la declaración XML si existe y preceder a cualquier elemento o contenido de datos.
- Entre la declaración XML y DOCTYPE solo se pueden insertar comentarios.
- Todo documento que deba ser validado debe contener una declaración DOCTYPE.

```
<!DOCTYPE elemento_raíz origen ubicación [subconjunto interno]>
```

Ejemplo de un documento XML con una declaración DTD interna:

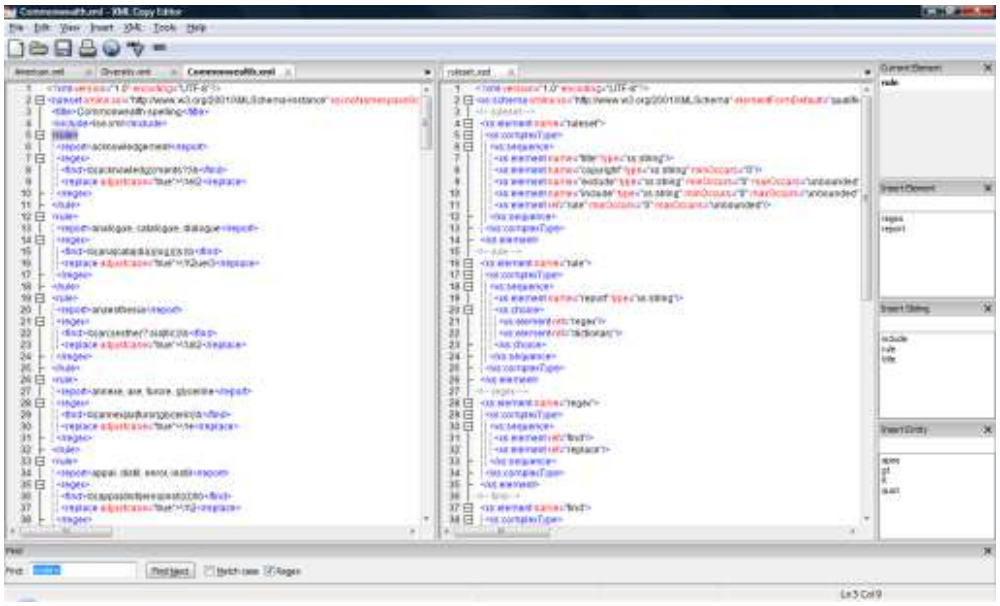
```
<?xml version="1.0"?>
<!DOCTYPE mensaje [
<!ELEMENT mensaje (para,de,titulo,cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
]>
<mensaje>
  <para>Jose</para>
  <de>Maria</de>
  <titulo>Recordatorio</titulo>
  <cuerpo>Recuerda que el sábado iremos al cine</cuerpo>
</mensaje>
```

Cuando este documento sea analizado por un parser primero comprobará que está bien formado, luego comprobará que los elementos y atributos corresponden con la definición dada en el DTD.

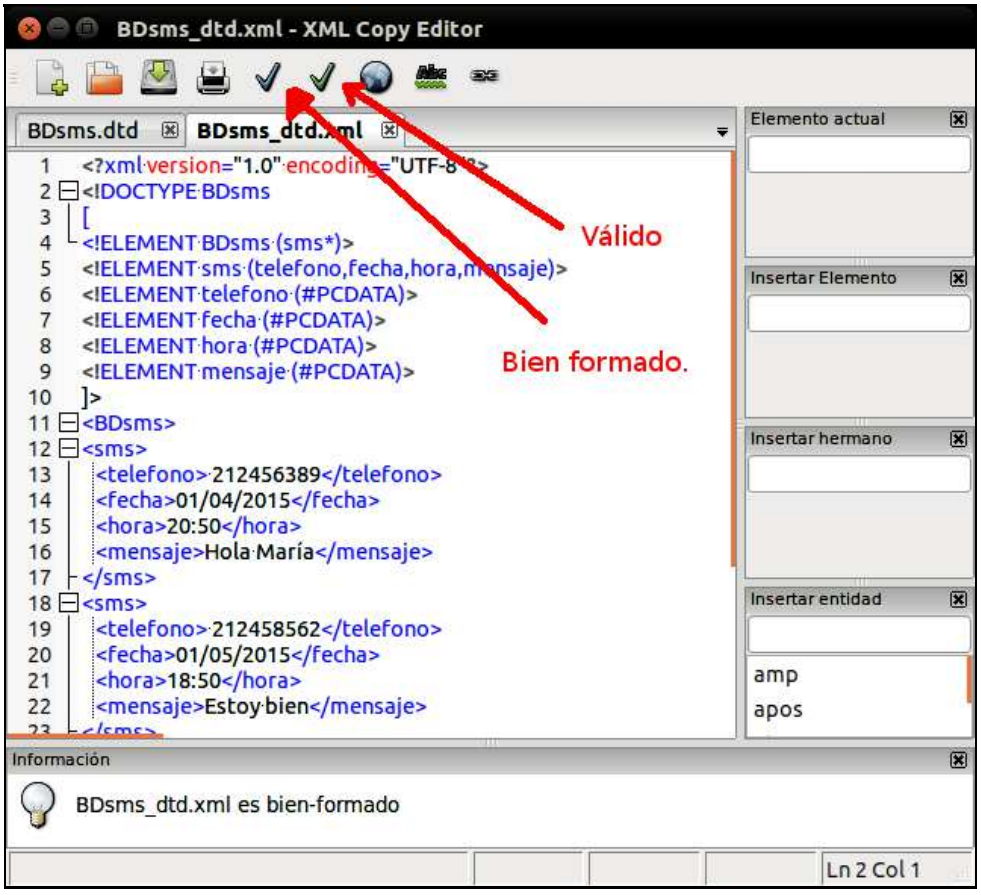
Actividad

Para comprobar la validación vamos a guardar el contenido del documento anterior en un documento llamado ‘msgdtd.xml’. Utilizaremos la herramienta **XMLCopyEditor.**

URL: http://xml-copy-editor.sourceforge.net/



Una vez hemos instalado la aplicación y hemos guardado nuestro fichero tenemos dos botones, uno para comprobar si el fichero xml está bien formado y otro para comprobar si es válido:

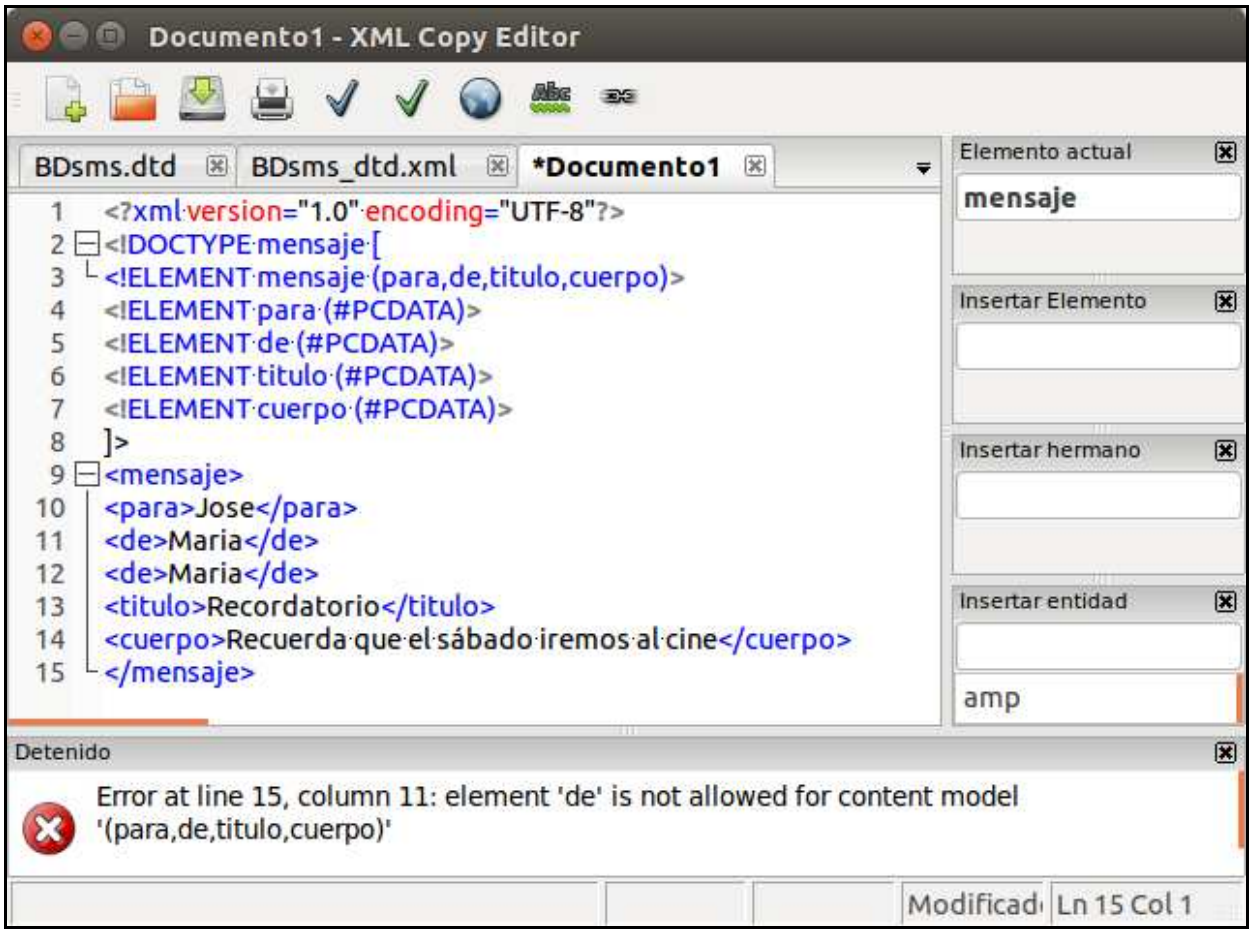


En la parte inferior "Información" no dirá el estado del fichero, y si hay errores. Por ejemplo, imaginemos que por error introducimos dos etiquetas <de> en el documento:

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

```
<!DOCTYPE mensaje [
<!ELEMENT mensaje (para,de,titulo,cuerpo)>
<!ATTLIST mensaje prioridad (urgente | normal | baja) #REQUIRED>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
]>
<mensaje prioridad=”normal”>
<para>Jose</para>
<de>Maria</de>
<de>Maria</de>
<titulo>Recordatorio</titulo>
<cuerpo>Recuerda que el sábado iremos al cine</cuerpo>
</mensaje>
```

Si ahora procedemos a comprobar la validez del documento:



XMLCopyEditor nos mostrará el error, ya que en el DTD tenemos definido solo un elemento <de>.

La segunda manera de declarar el DTD con el que hay que validar un fichero XML, y la más utilizada, es separar el documento XML del DTD. Por tanto la declaración DTD irá en un fichero y el XML en otro fichero. Esto es muy habitual porque si hay que generar muchos documentos XML que se validan con un mismo DTD no es práctico incluir en todos los documentos XML la declaración DTD.

Imaginemos el ejemplo del tema anterior. Donde debemos mandar un documento XML con información sobre los coches que tenemos en oferta. Además de mandarlo nosotros es fácil imaginar que muchos concesionarios mandan periódicamente esta información al portal para que se actualice. Por tanto es lógico pensar que no tiene sentido mandar el DTD junto con todos los ficheros porque estamos repitiendo la misma información muchas veces. La segunda el DTD deberá ser común para todos los concesionarios de forma que el portal de ofertas entienda perfectamente cualquier XML que le mandan los concesionarios. Por tanto lo lógico es que sea el portal quien diseñe el DTD con la información que necesita y cómo la necesita. Los concesionarios crean los documentos XML con la estructura dada y los mandan al portal, el cual, los valida antes de tratarlos.

La sintaxis en este caso sería:

```
<!DOCTYPE oferta SYSTEM http://www.servidor.com/oferta.dtd>
```

De esta forma el que recibe el document XML sabe con qué DTD debe validarlo antes de procesarlo. Siguiendo con el ejemplo anterior de los mensajes vamos a separar del documento anterior la definición DTD del documento XML. Para ello creamos los dos ficheros siguientes:

mensaje.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
<mensaje>
  <para>Jose</para>
  <de>Maria</de>
  <titulo>Recordatorio</titulo>
  <cuerpo>Recuerda que el sábado iremos al cine</cuerpo>
</mensaje>
```

mensaje.dtd

```
<!ELEMENT mensaje (para,de,titulo,cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
```

Al primero le podemos llamar mensaje.xml y al segundo debemos llamarle mensaje.dtd que es el nombre utilizado en la declaración DOCTYPE. Es importante que los dos ficheros estén en el mismo directorio porque en la declaración DOCTYPE hemos escrito: ... SYSTEM “mensaje.dtd”. En caso de estar en diferente ubicación habría que especificarla ...SYSTEM “../DTDS/mensaje.dtd”.

6.- Validación de documentos XML.

La validación de los documentos XML es una serie de comprobaciones que permiten saber si el documento XML está bien formado y si se ajusta a una estructura previamente definida (ya sea DTD o esquemas). Cuando se habla de documento bien formado se quiere decir que se siguen las normas y reglas básicas que se establecen en todos los documentos XML. Cuando se habla de un documento válido, quiere decir que además de estar bien formado, su estructura es acorde con las normas que se dictan en la definición del tipo de documento o esquema asociado.

En resumen, ¿por qué son necesarios los procesos de validación?

- Porque nos permite asegurar que en una ransferencia de información XML entre un emisor y un receptor, ambos interpretarán su contenido de igual manera.
- Porque nos permite asegurar que el documento contiene toda la información declarada como obligatoria.
- Porque los datos vendrán en un formato conocido y correcto.

Documentos bien formados.

Todo documento que quiera serlo, debe cumplir una serie de normas básicas:

- Al principio del documento se debe declarar una cabecera (en formato etiqueta) que indique lo siguiente:
  - Versión del docuemnto XML.
  - Tipo de codificación utilizada. UTF-8, ISO-8859-1, etc.
  - Si esta basado en un documento de estructura externo (DTD).
- Todas las entidades, elementos y atributos deben tener una sintaxis correcta.
  - XML distingue entre mayúsculas y minúsculas.
  - Los nombres de los elementos pueden ser alfanuméricos, pero comenzar con una letra.
  - Elemento que se abra <etiq>, deberá cerrarse </etiq>
  - Los valores que se indiquen en los atributos deberán ir entre comillas dobles o simples. <etiq atributo="1">.
  - Si hay elemento vacío EMPTY, deberá autocerrarse. <etiq/>.
- En la estructura del documento XML, se encadenarán los elementos en un formato jerárquico, en el que solamente habrá un elemento raíz.
- Si se usan entidades definidas por el usuario dentro del documento XML, en la DTD deberán estar declaradas.