
Primeros pasos con XML y XSL

Ricardo Borillo Domenech

Table of Contents

1. Apartados principales	1
2. Introducción al lenguaje de marcas XML	2
3. Estructura de los documentos: DTDs	2
3.1. Asociar un DTD a un documento XML: Elemento raiz	2
3.2. Elementos padre/hijo en los documentos XML	3
3.3. Operador (*): Cero o muchas apariciones de un nodo	3
3.4. Operador (+): Una o muchas apariciones de un nodo	4
3.5. Operador (?): Cero o una aparición de un nodo (opcionalidad)	4
3.6. Combinación de los operadores +, * y ?	5
3.7. Operador (!): Aparición de uno u otro elemento	6
3.8. Intercalado de nodos y texto en un documento	7
3.9. Utilización de los atributos en los nodos	7
3.10. Tipos de atributos CDATA, NMTOKEN y NMTOKENS	8
3.11. Tipo de atributo ID	9
3.12. Tipo de atributo IDREF y IDREFS	10
3.13. Enumeraciones en los atributos de un nodo	10
3.14. Atributos requeridos, opcionales y valores por defecto	11
3.15. El elemento vacío	11
4. DTD Vs XML-Schema	12
4.1. Ejemplo de documento XML para matrícula	12
4.2. Ejemplo de DTD para matrícula	13
4.3. Ejemplo de XML-Schema para matrícula	13
4.4. XML-Schemas y RELAX NG	14
5. XSL: Hojas de estilo para la transformación de documentos XML	14
5.1. Patrones de transformación XSLT	15
5.2. Expresiones de acceso a nodos XML con Xpath	17
6. Diseño de hojas de estilos XSL	19
7. Transformación de ficheros XML con Apache Ant	24
8. Formatting Objects	25
Introducción	25
Implementaciones disponibles	26
Ventajas del uso de FO para la generación de documentos	26
Descripción del proceso de generación de PDFs	27
Documento XML que contendrá la información a mostrar	27
Estructura del documento XSL-FO resultado	27
Creación de una hoja de estilos XSL que transformará el documento XML en un fichero XSL-FO	31
Ejecutar la transformación con un parser XSL	32
Procesamiento de dicho fichero XSL-FO con Apache FOP	32
7. Herramientas para el trabajo con XML	32
8. Entornos de trabajo con XML	33

Abstract

Conceptos básicos para conocer las tecnologías que han nacido alrededor de XML y como aplicarlas.

1. Apartados principales

1. Introducción al lenguaje de marcas XML
2. Estructura de los documentos: DTDs
3. DTD Vs XML-Schemas
4. Proceso de transformación de los documentos XML
5. XSL: Hojas de estilo para la transformación de documentos XML
6. Diseño de hojas de estilos XSL
7. Herramientas para el trabajo con XML

2. Introducción al lenguaje de marcas XML

3. Estructura de los documentos: DTDs

3.1. Asociar un DTD a un documento XML: Elemento raíz

Un documento XML es válido si ha sido asociado a un documento de definición de tipos y si el documento cumple las restricciones expresadas en él. El documento de definición de tipos tiene que aparecer antes del primer elemento del documento. El nombre que sigue a DOCTYPE en el documento de definición de tipos debe ser el mismo que el nombre del elemento raíz.

Example 1.

Un documento puede contener únicamente el elemento raíz tutorial que contiene algún texto.

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

Un documento válido que contiene algún texto

```
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">  
<tutorial>Este es un documento XML</tutorial>
```

Este documento también es válido

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

3.2. Elementos padre/hijo en los documentos XML

Un tipo elemento puede contener otros elementos hijos. En este caso no podrá contener ningún texto sino tan solo elementos separados, opcionalmente, por espacios en blanco.

Example 2.

El elemento raíz XXX debe contener únicamente un elemento AAA seguido de otro elemento BBB. Los elementos AAA y BBB pueden contener texto pero no otros elementos.

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

Un documento válido que contiene algún texto

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>Comienzo</AAA>
  <BBB>Fin</BBB>
</XXX>
```

Este documento también es válido

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

3.3. Operador (*): Cero o muchas apariciones de un nodo

Si el nombre de un elemento en una DTD va seguido por un asterisco [*], este elemento puede aparecer ninguna, una o varias veces.

Example 3.

El elemento raíz XXX puede contener ninguno, uno o varios elementos AAA seguido de exactamente un elemento BBB. El elemento BBB tiene que estar siempre presente.

```
<!ELEMENT XXX (AAA* , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

Un documento válido

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

```
<XXX> <AAA/> <BBB/> </XXX>
```

Otro documento válido. El elemento AAA no es obligatorio

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

Más de un elemento AAA puede aparecer dentro del documento

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

3.4. Operador (+): Una o muchas apariciones de un nodo

Si el nombre de un elemento en una DTD está seguido por el caracter más [+], este elemento tiene que aparecer una o más veces .

Example 4.

El elemento raíz XXX debe contener uno o más elementos AAA seguidos de exactamente un elemento BBB. El elemento BBB tiene que estar siempre presente.

```
<!ELEMENT XXX (AAA+ , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

Un documento válido

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

Pueden aparecer varios elementos AAA en el documento

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

3.5. Operador (?): Cero o una aparición de un nodo (opcionalidad)

Si el nombre de un elemento en la DTD está seguido por un signo de interrogación [?], este elemento puede aparecer ninguna o una vez.

Example 5.

El elemento raíz XXX puede contener un elemento AAA seguido de exactamente un elemento BBB. El elemento BBB tiene que estar siempre presente:

```
<!ELEMENT XXX (AAA? , BBB)>
<!ELEMENT AAA (#PCDATA)><!ELEMENT BBB (#PCDATA)>
```

Un documento válido:

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

El elemento AAA no es obligatorio:

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

3.6. Combinación de los operadores +, * y ?

Este ejemplo usa una combinación de [+ * ?]

Example 6.

El elemento raíz XXX puede contener un elemento AAA seguido de uno o más elementos BBB. El elemento AAA puede contener un elemento CCC y varios elementos DDD. El elemento BBB tiene que contener, exactamente, un elemento CCC y un elemento DDD:

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC , DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Un documento válido:

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

Los elementos en AAA no son obligatorios:

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

El elemento AAA no puede ser omitido:

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

3.7. Operador (|): Aparición de uno u otro elemento

Con el caracter [|] se puede seleccionar uno de entre varios elementos.

Example 7.

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)>
```

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>
```

3.8. Intercalado de nodos y texto en un documento

El texto puede ser intercalado con elementos.

Example 8.

El elemento AAA puede contener o bien BBB o CCC. Por otro lado el elemento BBB puede contener cualquier combinación de texto y elementos CCC.:

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

Un documento válido que explora varias posibilidades:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC>Exactamente un elemento.</CCC>
  </AAA>
  <AAA>
    <BBB>
      <CCC/>
      <CCC/>
      <CCC/>
    </BBB>
  </AAA>
<BBB/>
<BBB>
  Esta es <CCC/> una combinacion <CCC/> de <CCC> elementos CCC </CCC> y texto <CCC/>.
</BBB>
<BBB>
  Sólo texto.
</BBB>
</XXX>
```

3.9. Utilización de los atributos en los nodos

Los atributos se usan para asociar pares nombre-valor con elementos. La especificación de atributos sólo puede aparecer dentro de la etiqueta de apertura y en los elementos vacíos. La declaración comienza con ATTLIST seguido del nombre del elemento al que pertenece el atributo y después le sigue la definición individual de cada atributo.

Example 9.

Un atributo del tipo CDATA puede contener cualquier caracter si éste se atiene a las reglas de formación. Los atributos #REQUIRED deben estar siempre presentes, los #IMPLIED son opcionales:

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

Los atributos CDATA pueden contener cualquier caracter que se atenga a las reglas:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
<attributes aaa="#d1" bbb="*~*">
  Text
</attributes>
```

El orden de los atributos es indiferente:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
<attributes bbb="$25" aaa="13%">
  Texto
</attributes>
```

El atributo bbb puede omitirse ya que es #IMPLIED:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
<attributes aaa="#d1" />
```

3.10. Tipos de atributos CDATA, NMTOKEN y NMTOKENS

Un atributo del tipo CDATA puede contener cualquier caracter si éste se atiene a las reglas de formación. Si es del tipo NMTOKEN sólo puede contener letras, dígitos, punto [.], guión [-], subrayado [_] y dos puntos [:]. Los del tipo NMTOKENS pueden contener los mismos caracteres que NMTOKEN más espacios en blanco. Un espacio en blanco consiste en uno o más espacios, retornos de carro o tabuladores.

Example 10.

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>
```

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"/>
```

3.11. Tipo de atributo ID

El valor de un atributo de tipo ID puede contener sólo caracteres válidos en NMTOKEN y debe comenzar con una letra. Ningún tipo de elemento puede tener especificado más de un atributo de tipo ID. El valor de un atributo ID debe ser único entre todos los valores de atributos ID.

Example 11.

Los atributos id, code y X determinan de manera inequívoca su elemento:

```
<!ELEMENT XXX (AAA+ , BBB+ , CCC+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>
<!ATTLIST AAA
  id ID #REQUIRED>
<!ATTLIST BBB
  code ID #IMPLIED
  list NMTOKEN #IMPLIED>
<!ATTLIST CCC
  X ID #REQUIRED
  Y NMTOKEN #IMPLIED>
```

Todos los valores ID son únicos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA id="a1"/>
  <AAA id="a2"/>
  <AAA id="a3"/>
  <BBB code="QWQ-123-14-6" list="14:5"/>
  <CCC X="zero" Y="16" />
</XXX>
```

Los atributos list y Y son del tipo NMTOKEN no ID. Éstos pueden tener, por lo tanto, el mismo valor que los atributos ID o tener el mismo valor en varios elementos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA id="L12"/>
```

```
<BBB code="QW" list="L12" />
<CCC X="x-0" Y="QW" />
<CCC X="x-1" Y="QW" />
</XXX>
```

3.12. Tipo de atributo IDREF y IDREFS

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.

Example 12.

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>
```

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>
```

3.13. Enumeraciones en los atributos de un nodo

Se pueden definir los valores permitidos en un atributo en la DTD.

Example 13.

Esta DTD declara los valores exactos que son permitidos:

```
<!ELEMENT XXX (AAA+, BBB+)>
```

```
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA
  true ( yes | no ) #REQUIRED>
<!ATTLIST BBB
  month (1|2|3|4|5|6|7|8|9|10|11|12) #IMPLIED>
```

Todos los valores se dan en la DTD:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA true="yes"/>
  <AAA true="no"/>
  <AAA true="yes"/>
  <BBB month="8" />
  <BBB month="2" />
  <BBB month="12" />
</XXX>
```

3.14. Atributos requeridos, opcionales y valores por defecto

Si un atributo es opcional (#IMPLIED), puede definírsele un valor por defecto para cuando el atributo no se usa.

Example 14.

Ambos atributos son opcionales. Se dan sus valores por defecto.:

```
<!ELEMENT XXX (AAA+, BBB+)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ATTLIST AAA
  true ( yes | no ) "yes">
<!ATTLIST BBB
  month NMTOKEN "1">
```

Los valores de true son yes, no y yes. Los valores de month son 8, 2 y 1.:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA true="yes"/>
  <AAA true="no"/>
  <AAA/>
  <BBB month="8" />
  <BBB month="2" />
  <BBB/>
</XXX>
```

3.15. El elemento vacío

Un elemento puede ser definido EMPTY (vacío). En ese caso sólo puede contener atributos pero no texto.

Example 15.

Los elementos AAA pueden contener solamente atributos pero no texto:

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

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

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

4. DTD Vs XML-Schema

4.1. Ejemplo de documento XML para matrícula

Definición general del documento XML que representa la imagen de matrícula de un alumno que realiza la Automatrícula en la Universitat Jaume I. En este documento no diferenciamos si su estructura interna va a ser definida por un DTD o por un XML-Schema ...

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<matricula>
  <personal>
    <dni>52945813C</dni>
    <nombre>Ricardo Borillo Domenech</nombre>
    <titulacion>Enginyeria Informàtica (Pla 2001)</titulacion>
    <curso_academico>2002/2003</curso_academico>

    <domicilios>
      <domicilio tipo="familiar">
        <nombre>C/ Principal nº1</nombre>
      </domicilio>
      <domicilio tipo="habitual">
        <nombre>C/ Secundaria nº2</nombre>
      </domicilio>
    </domicilios>
  </personal>

  <pago>
    <tipo_matricula>Matrícula ordinària</tipo_matricula>
  </pago>
</matricula>
```

En el caso de que queramos definir la estructura del documento mediante un XML-Schema, deberemos sustituir la cabecera del documento de la forma siguiente:

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

```
<matricula xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocati
...
</matricula>
```

Si por el contrario, queremos seguir utilizando la estructura clásica de un DTD:

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

4.2. Ejemplo de DTD para matrícula

```
<!ENTITY matricula (personal, pago)>

<!ENTITY personal (dni, nombre, titulacion, curso_academico, domicilios)>
<!ENTITY dni (#PCDATA)>
<!ENTITY nombre (#PCDATA)>
<!ENTITY titulacion (#PCDATA)>
<!ENTITY curso_academico (#PCDATA)>
<!ENTITY domicilios (domicilio+)>
<!ENTITY domicilio (nombre)>
<!ATTLIST domicilio
    tipo (familiar|habitual) #REQUIRED>
<!ENTITY nombre (#PCDATA)>

<!ENTITY pago (tipo_matricula)>
<!ENTITY tipo_matricula (#PCDATA)>
```

4.3. Ejemplo de XML-Schema para matrícula

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xml:lang="ES">
  <xs:element name="matricula" type="tMatricula"/>

  <xs:complexType name="tMatricula">
    <xs:sequence>
      <xs:element name="personal" type="tPersonal"/>
      <xs:element name="pago" type="tPago"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="tPersonal">
    <xs:all>
      <xs:element name="dni" type="xs:string"/>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="titulacion" type="xs:string"/>
      <xs:element name="curso_academico" type="xs:string"/>
      <xs:element name="domicilios" type="tDomicilio"/>
    </xs:all>
  </xs:complexType>

  <xs:complexType name="tPago">
    <xs:all>
      <xs:element name="tipo_matricula" type="xs:string"/>
    </xs:all>
  </xs:complexType>
```

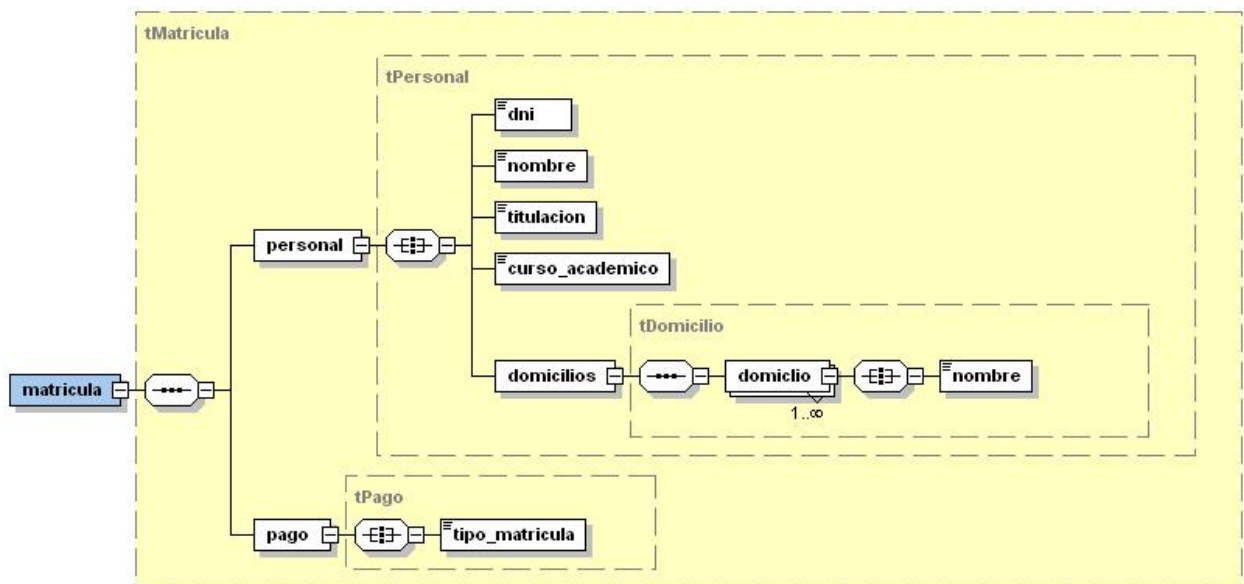
```

<xs:complexType name="tDomicilio">
  <xs:sequence>
    <xs:element name="domiclio" maxOccurs="unbounded">
      <xs:complexType>
        <xs:all>
          <xs:element name="nombre" type="xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

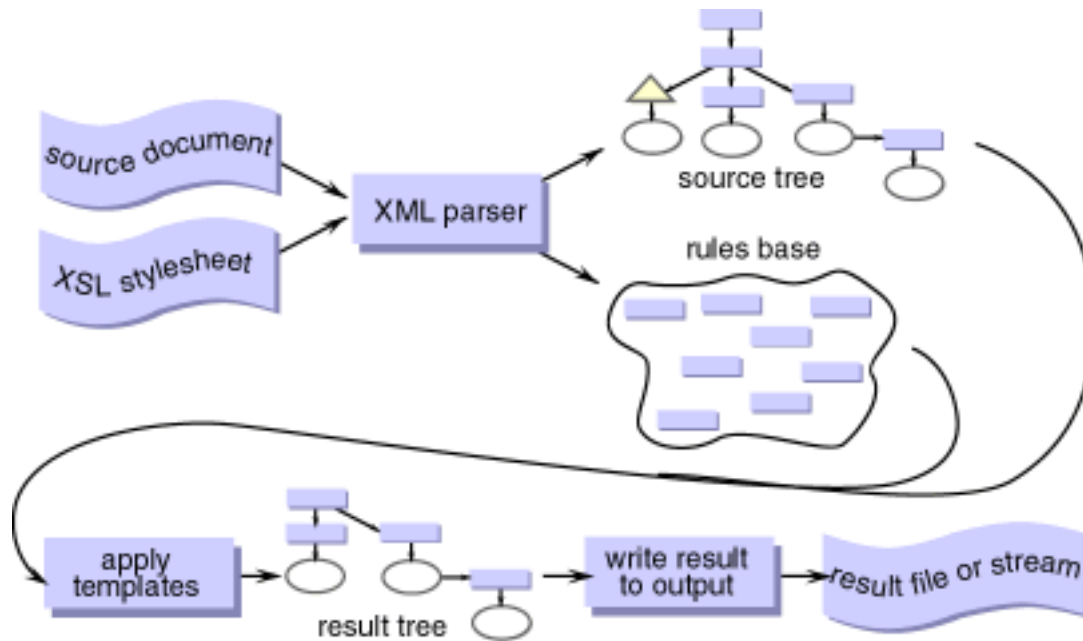
4.4. XML-Schemas y RELAX NG

Al igual que surgió XML Schema como una mejora necesaria sobre la idea en la que se fundamentaban las DTDs, han surgido distintas propuestas al margen de los Schemas, de estas ideas no vamos a quedar con una, RELAX NG, que es, a su vez, la fusión de otras dos iniciativas TREX y RELAX que para conseguir un mayor avance han decidido aunar fuerzas. Básicamente, RELAX NG maneja documentos XML que representan esquemas e instancias a través de un modelo abstracto de datos, esto que suena un poco oscuro viene a querer expresar que, para RELAX NG, un documento XML es la representación de un elemento y que a su vez un elemento está formado por una serie de "partes" como son: un nombre, un contexto, un conjunto de atributos y una secuencia ordenada de cero o más hijos. Y así con cada una de estas partes. Aunque la especificación completa se puede encontrar en OASIS7, vamos a mostrar cuales son las principales diferencias, no tanto conceptuales si no de uso, de RELAX NG con respecto a XML Schema. En primer lugar es importante resaltar que RELAX NG, y sus precursores RELAX y TREX, son intentos de simplificar y/o potenciar la utilidad de las DTDs en general y en particular de XML Schema. También es importante indicar que esta iniciativa no es oficial de W3.ORG, si bien está impulsándose dentro de un nutrido grupo de gente lideradas por James Clark, MURATA Makoto, y aún no está reconocido por el estándar ni tan siquiera como recomendación. En segundo lugar, podemos decir, sin mucho margen de error, que la principal mejora de RELAX NG respecto a XML Schema es la mayor simplicidad en manejo y aprendizaje que requiere. Como consecuencia de esta simplificación renunciamos al uso de los `<simpleType>` y `<complexType>` a favor del uso de, únicamente, elementos y ciertas etiquetas especiales para marcar número requerido de apariciones, tipo de dato de un elemento, etc.



5. XSL: Hojas de estilo para la transformación de

documentos XML



XSL es un lenguaje creado para dar estilo a los documentos XML a los cuales se aplica. Así, XSL es una especificación que se compone de partes o recomendaciones:

- Xpath. Lenguaje que permite escribir expresiones para la búsqueda de nodos dentro del árbol XML.
- XSLT. Reglas o patrones para la transformación del contenido de los nodos XML sobre los cuales se consigue una correspondencia. Si lo que queremos es generar páginas HTML a partir de documentos XML, podemos complementar/sustituir XSL por CSS.
- Formatting Objects. Actualmente, constituye un lenguaje de especificación de estilo en base al cual nos es posible la generación de PDFs.

especificación que se compone de partes o recomendaciones:

5.1. Patrones de transformación XSLT

1. Definición de las cabeceras de la página XSL

Example 16.

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

2. El patrón o template: Elemento básico de la páginas de estilo

Para cada nodo del árbol XML tenemos la posibilidad de definir un patrón de estilos que marcará como se va a presentar este nodo en el documento final. Por ejemplo, si tenemos el siguiente documento XML:

Example 17.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<test1>
  <titulo>Prueba de patrones XSL</titulo>
  <descripcion>Cuerpo del documento</descripcion>
</test1>
```

... y esta hoja de estilos:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="test1">
    <html>
      <head>
        <title><xsl:apply-templates select="titulo" mode="head"/></title>
      </head>

      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="titulo" mode="head">
    <xsl:value-of select="text()"/>
  </xsl:template>

  <xsl:template match="titulo">
    <h1><xsl:value-of select="text()"/></h1>
  </xsl:template>

  <xsl:template match="descripcion">
    <h3><xsl:value-of select="text()"/></h3>
  </xsl:template>
</xsl:stylesheet>
```

... el resultado de la transformación será el siguiente:

```
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<title>Prueba de patrones XSL</title>
</head>
<body>
  <h1>Prueba de patrones XSL</h1>
  <h3>Cuerpo del documento</h3>
</body>
</html>
```

En este ejemplo podemos apreciar de varias instrucciones XSL:

- Uso de la instrucción "xsl:template" para definir nuevos templates con los que van a ir identificándose los nodos del documento XML. En los templates, podemos utilizar el atributo modificador "mode" con el fin de poder llamar a un mismo template desde dos contextos distintos en los que queremos dos resultados distintos. Es por esto que procesar el nodo "title" desde la cabecera o desde el cuerpo del documento HTML, va a tener una representación distinta.
- Uso de sentencias de procesamiento de templates como "xsl:apply-templates". Esta instrucción lanza la búsqueda de templates a aplicar a partir del punto en el que nos encontremos del árbol XML. Si ya hemos procesado el nodo titulo con un template 'xsl:template match="titulo"' y desde aquí queremos procesar la descripción, no podremos utilizar directamente un "xsl:apply-templates" ya que este nodo se encuentra en otro punto de la jerarquía. A esta instrucción podemos aplicarle el atributo modificador "select", para restringir los templates que se aplican a los de un tipo determinado.

5.2. Expresiones de acceso a nodos XML con Xpath

Ejemplos de expresiones que podemos aplicar dentro del modificador "select" de un a instrucción "xsl:apply-templates":

- Acceso a todos los nodos "titulo":

```
<xsl:apply-templates select="titulo"/>
```

- Acceso a todos los nodos "titulo" que tengan como padre a "test1":

```
<xsl:apply-templates select="test1/titulo"/>
```

- Acceso al nodo raíz del documento XML:

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

- Acceso a todos los nodos "titulo" que tengan como antecesor a "test1":

```
<xsl:apply-templates select="test1//titulo"/>
```

- Acceso al primero de los nodos "titulo" que tengan como padre a "test1":

```
<xsl:apply-templates select="test1/titulo[1]"/>
```

- Acceso al último de los nodos "titulo" que tengan como padre a "test1":

```
<xsl:apply-templates select="test1/titulo[position()=last()]" />
```

- Acceso a los nodos "titulo" que sean pares y que tengan como padre a "test1":

```
<xsl:apply-templates select="test1/titulo[position() mod 2 = 0]"/>
```

- Acceso a todos los nodos "titulo" en cualquier parte del documento:

```
<xsl:apply-templates select="//titulo"/>
```

- Acceso a todos los nodos "titulo" en cualquier parte del documento, a partir del contexto actual:

```
<xsl:apply-templates select="../titulo"/>
```

- Acceso a todos los nodos "titulo" que tengan como "abuelo" a "test1":

```
<xsl:apply-templates select="test1/*/titulo"/>
```

- Acceso a todos los nodos "titulo" que tengan un atributo "id":

```
<xsl:apply-templates select="titulo[@id]"/>
```

- Acceso a todos los nodos "titulo" que NO tengan un atributo "id":

```
<xsl:apply-templates select="titulo[not(@id)]"/>
```

- Acceso a todos los nodos "titulo" que tengan un atributo "id" con valor "XXX":

```
<xsl:apply-templates select="titulo[@id='XXX']"/>
```

- Acceso a todos los nodos "test1" que tengan un hijo "titulo" con valor "XXX":

```
<xsl:apply-templates select="test1[titulo='XXX']"/>
```

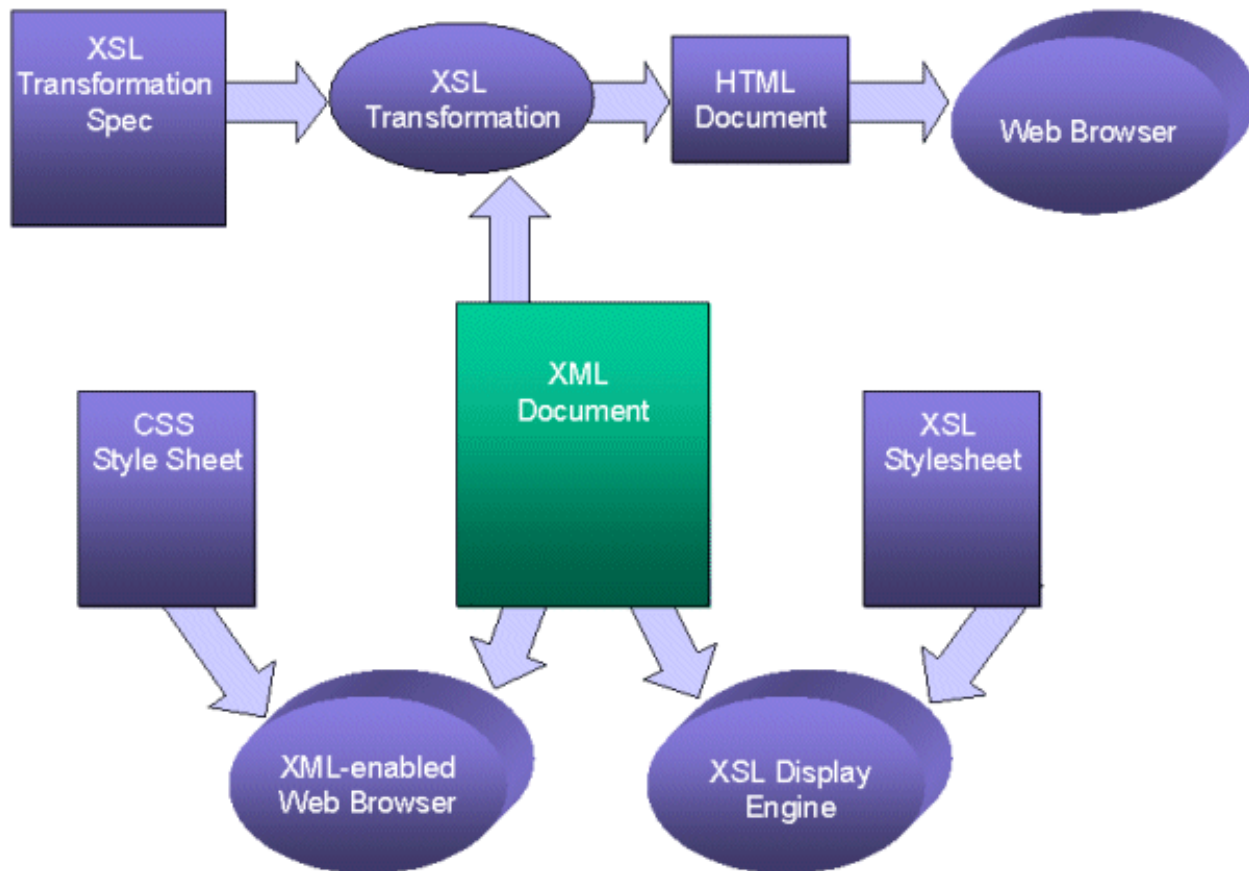
- Acceso a todos los nodos "test1" que tengan un hijo "titulo" con valor "XXX", normalizando la búsqueda, es decir, eliminado espacios en blanco al inicio y final del nodo:

```
<xsl:apply-templates select="test1[normalize-space(titulo)='XXX']"/>
```

- Acceso a todos los nodos "titulo" o "descripcion":

```
<xsl:apply-templates select="titulo|descripcion"/>
```

6. Diseño de hojas de estilos XSL



1. Selección de valores individuales.

Example 18.

Para la selección de valores individuales, extraídos de los nodos del árbol XML, debemos utilizar la instrucción "xsl:value-of", con ella podemos hacer referencia tanto a nodos, como variables, cálculos matemáticos, etc. Para el acceso a los nodos, debemos utilizar las expresiones Xpath previamente descritas:

```

<xsl:value-of select="matricula/personal/dni" />
<xsl:value-of select="$valor" />
<xsl:value-of select="3+2" />
  
```

2. Templates y el modificador "mode" (Ya espuesto en el apartado 5.1, punto 2).

3. *Templates con nombre* . Este tipo de templates no se corresponden con ningún nodo del documento XML y sólo podemos invocarlos haciendo referencia a ellos directamente. Su sentencia de definición ya no incluye el atributo "match", sino que este se sustituye por "name". Podemos aumentar las funcionalidades de este tipo de templates pasándole parámetros. El siguiente ejemplo hace uso de la definición de templates por nombre y paso de parámetros:

Example 19.

Documento XML que usaremos como base:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<test2>
  <titulo>Prueba de patrones XSL</titulo>
  <descripcion>Cuerpo del documento</descripcion>
</test2>
```

Documento XSL que aplica los templates por nombre:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="test2">
    <html>
      <head>
        <title><xsl:apply-templates select="titulo" mode="head"/></title>
      </head>
      <body>
        <xsl:apply-templates/>
        <xsl:call-template name="print">
          <xsl:with-param name="in"><xsl:value-of select="/test2/titulo"/></xsl:with-param>
        </xsl:call-template>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="titulo" mode="head">
    <xsl:value-of select="text()"/>
  </xsl:template>
  <xsl:template match="titulo">
    <h1><xsl:value-of select="text()"/></h1>
  </xsl:template>
  <xsl:template match="descripcion">
    <h3><xsl:value-of select="text()"/></h3>
  </xsl:template>
  <xsl:template name="print">
    <xsl:param name="in">Nothing</xsl:param>
    <i><xsl:value-of select="$in"/></i>
  </xsl:template>
</xsl:stylesheet>
```

Resultado de la transformación:

```
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<title>Prueba de patrones XSL</title>
```

```
</head>
<body>
  <h1>Prueba de patrones XSL</h1>
  <h3>Cuerpo del documento</h3>
  <i>Prueba de patrones XSL</i>
</body>
</html>
```

4. *Procesamiento procedural, otro enfoque para aplicar templates.*

Example 20.

```
<xsl:for-each select="row">
  <xsl:for-each select="col">
    <xsl:apply-templates select="cell"/>
  </xsl:for-each>
</xsl:for-each>
```

5. *Procesamiento condicional.*

Example 21.

```
<xsl:if test="@atributo='x'">
  <h1>Este tipo de condicional no tiene ELSE</h1>
</xsl:if>
```

6. *Procesamiento condicional para varias opciones/comprobaciones.*

Example 22.

```
<xsl:choose>
  <xsl:when test="$variable=1"><h3>Valor uno</h3></xsl:when>
  <xsl:when test="$variable=2"><h3>Valor dos</h3></xsl:when>
  <xsl:otherwise>
    <h1>Variable con valor erroneo</h1>
  </xsl:otherwise>
</xsl:choose>
```

7. *Reportar errores desde la hoja de estilos.*

Example 23.

```
<xsl:message>
  <xsl:text>Este es el mensaje de error !!</xsl:text>
  <xsl:value-of select="@atributo"/>
</xsl:message>
```

8. Creación de nuevos elementos dinámicamente.

Example 24.

```
<xsl:variable name="test">a</xsl:variable>
<xsl:element name="{ $test }">
  <xsl:attribute name="href">http://www.si.uji.es</xsl:attribute>
  Servei d'informàtica
</xsl:element>
```

9. Ordenación de elementos en XSLT

Example 25.

```
<xsl:template match="personal">
  <xsl:apply-templates>
    <xsl:sort select="dni" data-type="string" order="descending"/>
    <xsl:sort select="curso_academico" data-type="number" order="ascending"/>
  </xsl:apply-templates>
</xsl:template>
```

10. Expresiones matemáticas

Example 26.

Consideramos la existencia de dos nodos con valores numéricos a partir de nuestro contexto, cuyos nombres son "x" e "y":

- a. <xsl:value-of select="x+y"/>
- b. <xsl:value-of select="x-y"/>
- c. <xsl:value-of select="x*y"/>

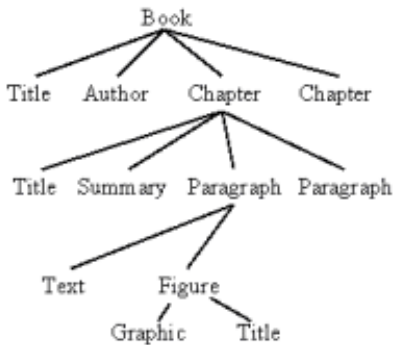
- d. `<xsl:value-of select="x div y"/>`
- e. `<xsl:value-of select="x mod y"/>`
- f. `<xsl:value-of select="sum(*)"/>`
- g. `<xsl:value-of select="floor(x)/>`
- h. `<xsl:value-of select="ceiling(x)/>`
- i. `<xsl:value-of select="round(x)/>`
- j. `<xsl:value-of select="count(*)"/>`
- k. `<xsl:value-of select="string-length(x)/>`

11. *Uso de variables*

Example 27.

```
<xsl:template match="pago">
  <xsl:variable name="test" select="/matricula/pago/tipo_matricula"/>
  <xsl:value-of select="$test"/>
</xsl:template>
```

XML Source Tree

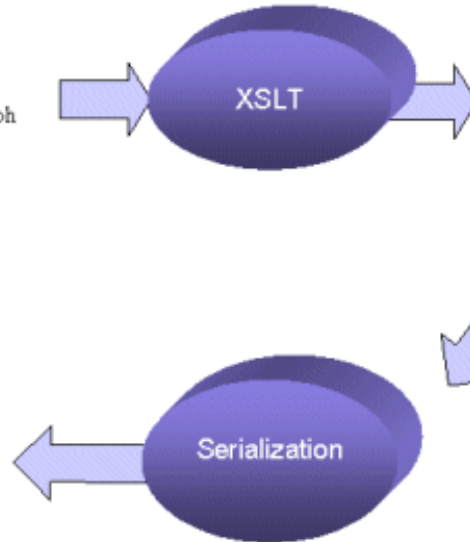
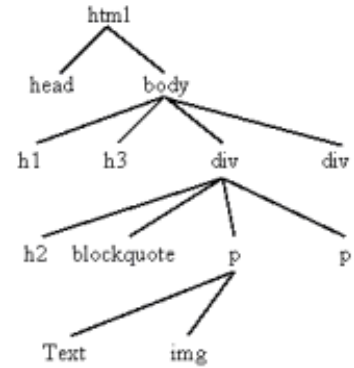


```

<html>
<head>...</head>
<body>
<h1></h1>
<h3></h3>
.....
</body>
</html>

```

XHTML Result Tree



7. Transformación de ficheros XML con Apache Ant

Ant es un sistema de compilación basada en Java. Su funcionalidad es similar a los clásicos Makefiles del mundo UNIX, aunque en lugar de poder extenderse mediante el uso de comandos del Shell, Ant puede extenderse en base a clases Java.

Ant ejecuta un conjunto de objetivos o targets descritos en un fichero XML que se llama por defecto "build.xml".

Todos los targets en Ant se ejecutan despues del target raiz (init).

En un fichero de compilacion de Ant podemos hacer que un target dependa de otros con el fin de que no se pueda realizar si todos los anteriores no se han completado.

Example 28.

Ejemplo de fichero Ant en el que tenemos el target raiz y un segundo target que depende del raiz.

```

<?xml version="1.0"?>
<project name="nombre_del_proyecto" default="target_por_defecto">

  <target name="init">
    <echo message="Esta es la tarea raiz !!" />
  </target>

  <target name="html" depends="">
    <echo message="Esta es la tarea html y requiere que la tarea raiz se haya completado primero !!" />
  </target>
</project>

```

Ant define una lista muy amplia de tareas que se pueden ejecutar dentro de un target, como por ejemplo:

- javac. Compila un fichero Java.
- java. Ejecuta un fichero Java.
- jar. Empaqueta un conjunto de recursos.
- mail. Envío de mails.
- ftp. Transmisión de ficheros por FTP.
- war. Empaqueta una aplicación J2EE.

Podemos encontrar un listado completo de todas las tareas soportadas por Ant en la dirección <http://ant.apache.org/manual/index.html>. Así, existen además una serie de tareas relacionadas con el procesamiento de ficheros XML:

- xmlproperty. Permite cargar un fichero XML como propiedades de Apache Ant.
- xslt. Transformación de documento XML mediante hojas de estilo XSL. Sólo podemos utilizar esta tarea si contamos con el .JAR del Xalan (procesador XSLT de Apache).
- xmlvalidate. Validación de documentos XML utilizando un interfaz SAX.

Example 29.

Ejemplo de fichero Ant en el que se transforma un documento XML aplicando una hoja de estilos XSL:

```
<?xml version="1.0"?>
<project name="nombre_del_proyecto" default="target_por_defecto">
  <target name="init" />

  <target name="html" depends="init">
    <echo message="Generando test.html ..." />
    <xslt in="test.xml" out="test.html" style="test.xsl">
      <outputproperty name="method" value="html" />
      <outputproperty name="standalone" value="yes" />
      <outputproperty name="encoding" value="ISO-8859-1" />
      <outputproperty name="indent" value="yes" />
    </xslt>
  </target>
</project>
```

8. Formatting Objects

Introducción

El estandar XSL, tal y como está definido actualmente, se divide en dos grandes partes:

- XSLT. Transformación de un documento de entrada XML en algún tipo de documento de salida, ya sea XML, HTML, PDF, etc.

- Formatting Objects. Se encargan de definir la visualización final del resultado de la transformación.

El único uso que se ha hecho en la actualidad de Formatting Objects, siempre ha estado enfocado a la generación de documentos de calidad destinados a la impresión. Este es el caso de los documentos PDF.

Implementaciones disponibles

Si queremos generar un PDF a partir de un documento en formato FO, podemos utilizar los siguientes conjuntos de utilidades, siendo las dos primeras de libre distribución:

- Apache FOP. API desarrollada por el proyecto Apache que permite la generación de los PDFs en línea de comandos o mediante llamadas al API.
- PassiveTex. Conjunto de macros LaTeX que nos permiten la transformación del documento XML a un documento en TeX sobre el que podremos generar un PS y posteriormente un PDF.
- XEP. Producto comercial de "RenderX" (<http://www.renderx.com>) escrito en Java.
- XSL Formatter. Producto comercial de "Antenna House" (<http://www.antennahouse.com>)
- Unicorn Formatting Objects. Producto comercial de "Unicorn Enterprises" (<http://www.unicorn-enterprises.com>) que además es sólo para Windows.

Ventajas del uso de FO para la generación de documentos

- Sencillo manejo de ciertas características de la generación de documentos como:
 - Páginación automática.
 - Definición de márgenes para el documento.
 - Definición de patrones distintos de presentación para cada una de las hojas.
 - Control "al milímetro" de la presentación de los elementos dentro del PDF.
 - Definición simplificada de cabeceras y pies de página.
 - Permite la inserción de diversos elementos gráficos como: imágenes, tablas, etc.
- Permite definir la presentación de cada elemento del documento en base a atributos muy similares, en la mayoría de los casos, a los atributos de una hoja de estilos CSS.

El API de Apache FOP permite una gran integración con otras APIs del proyecto Apache:

- FOP es una parte constituyente de Cocoon (Framework para la publicación de documentos XML a través de la web).
- FOP permite embeber gráficos SVG, renderizándolos a PNGs e insertando el resultado dentro del propio PDF.

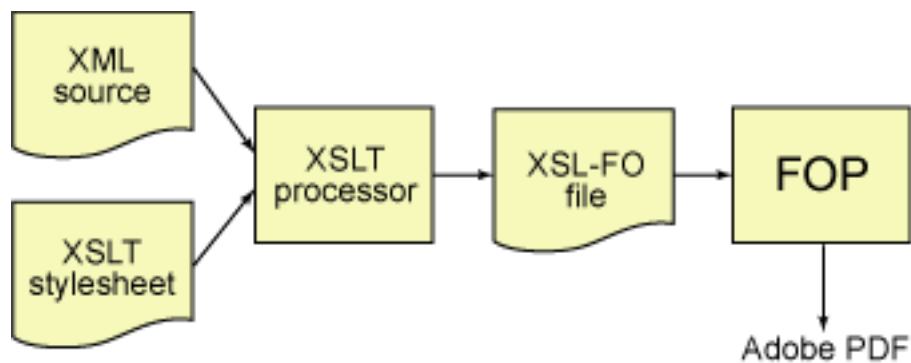
Con XSL-FO podemos acometer varios aspectos en la generación de documentos con una alta calidad de impresión:

- Definición del tamaño físico de la página que se creará (A4, Letter, etc).
- Control sobre propiedades de la página como los márgenes, cabeceras, pies, etc.
- Uso de elementos clásicos dentro del documento como son los párrafos, tablas, etc.
- Posibilidad de dar formato de presentación al texto generado, cambiando el tipo de fuente, su tamaño, su color y demás recursos gráficos.

Descripción del proceso de generación de PDFs

El proceso que cubriremos en esta guía (descrito en el siguiente gráfico) con el fin de obtener un PDF será el siguiente:

- Creación de un fichero XML que contendrá la información a mostrar en el PDF.
- Creación de una hoja de estilos XSL que transformará el documento XML en un fichero XSL-FO. El fichero contendrá las definiciones necesarias en lenguaje FO que detallaremos en próximos apartados, con el fin de generar el documento final.
- Ejecutar la transformación con un parser XSL, obteniendo como resultado, el fichero XSL-FO descrito anteriormente.
- Procesamiento de dicho fichero XSL-FO, con el procesador Apache FOP, generándose finalmente el fichero PDF.



Documento XML que contendrá la información a mostrar

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<listado>
  <cabecera>Listado de ventas</cabecera>
  <articulos>
    <articulo id="96537">Toalla de baño</articulo>
    <articulo id="38734">Cortina de plástico</articulo>
    <articulo id="76383">Servilleta de punto</articulo>
    <articulo id="09278">Papel de cocina</articulo>
  </articulos>
</listado>
```

Estructura del documento XSL-FO resultado

Antes de diseñar la hoja de estilos XSL, debemos saber que estructura tenemos que generar.

Un documento XSL-FO tiene una estructura que, a priori, parece muy complicada, pero no es así. La mayoría de los elementos que definimos en un documento XSL-FO son comunes a todos ellos, con lo que siempre podremos copiarlos de otro ya definido. Estos aspectos fijos son, por ejemplo, los márgenes, estructura de las páginas y otros aspectos de configuración de la estructura.

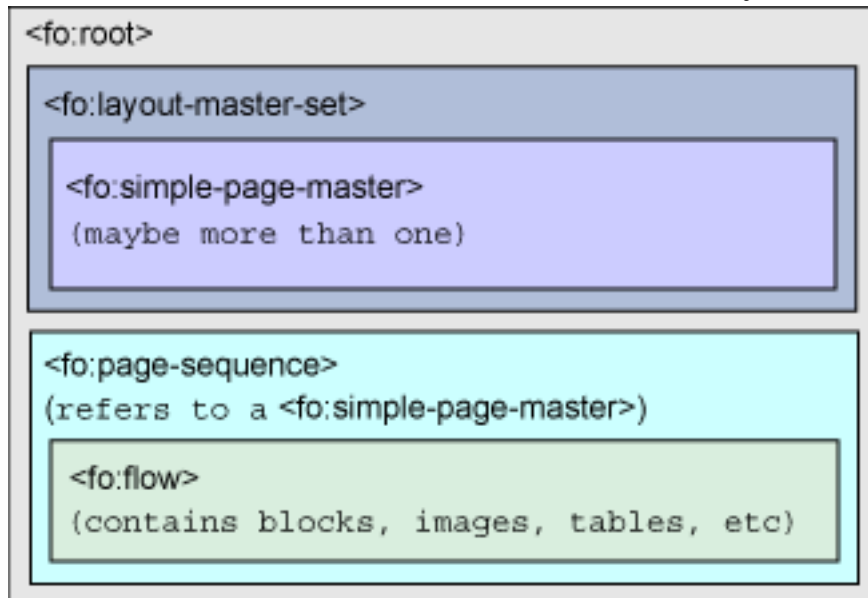
Ejemplo de esqueleto de un documento XSL-FO, cuya estructura se repetirá en la mayoría de los casos:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master margin-right="1cm" margin-left="1cm"
      margin-bottom="1cm" margin-top="1cm"
      page-width="21cm" page-height="29.7cm"
      master-name="first">
      <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="first">
    <fo:flow flow-name="xsl-region-body">
      <!-- Aquí va el contenido del documento -->
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Como podemos ver, no es tan complicado una vez sabemos que no hay necesidad de redefinirlo cada vez.

Entremos a ver con detalle cada elemento de la estructura con el fin de entenderlos mejor.



Elemento `<fo:root>`

Elemento que marca el inicio y el fin del documento XSL-FO. Nos apoyamos en este elemento para definir el espacio de nombres para FO.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  ...
</fo:root>
```

Elemento <fo:layout-master-set>

Engloba un conjunto de definiciones sobre la estructura o layout de las páginas, es decir, contiene uno o más <fo-simple-page-master>. Con esto conseguimos, por ejemplo, que en nuestro documento existan unas páginas con orientación vertical y otras apaisadas. También podemos así, definir distintos márgenes para las páginas del documento según si la página es par o impar.

En la mayoría de los casos, con una sola definición de página sería suficiente.

```
<fo:layout-master-set>
  ...
</fo:layout-master-set>
```

Elemento <fo:simple-page-master>

Se encarga de especificar los márgenes, altura y anchura de un página en concreto (para conseguir una orientación vertical o apaisada).

```
<fo:simple-page-master margin-right="1cm" margin-left="1cm"
  margin-bottom="1cm" margin-top="1cm"
  page-width="21cm" page-height="29.7cm"
  master-name="first">
  ...
</fo:simple-page-master>
```

Descripción de cada uno de los elementos constituyentes de este elemento:

- master-name

Como podemos crear multitud de definiciones de páginas, posteriormente debemos ser capaces de asociar cada página real con su definición de estructura. Lo haremos siempre a través del "master-name".

- margin-top, margin-bottom, margin-left y margin-right

Definen el tamaño de los márgenes superior, inferior, izquierdo y derecho según la siguiente tabla de unidades:

Table 1. Tabla de unidades de definición

Unidad	Descripción
cm	Centímetros
mm	Milímetros
in	Pulgadas
pt	Puntos, siendo 72 puntos una pulgada
pc	Picas, siendo 12 puntos una pica y 6 picas una pulgada
px	Pixels (dependiente del dispositivo de visualización)
em	La longitud de la letra M mayúscula

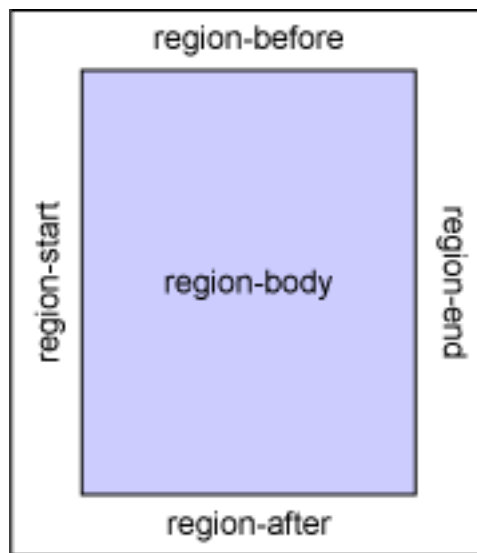
Definen el tamaño de los márgenes superior, inferior, izquierdo y derecho según la siguiente tabla de unidades:

- page-width y page-height

Tamaño físico de la página. En el ejemplo se utiliza 21x29.7, es decir, DIN-A4.

Elemento `<fo:region-before>`, `<fo:region-body>` y `<fo:region-after>`

- `region-body`. Dimensión del área principal en el centro de la página.
- `region-before`. Parte superior de la página, utilizada normalmente para la cabecera del documento.
- `region-after`. Parte inferior de la página, utilizada normalmente para el pie del documento.
- `region-start`. Parte izquierda del documento.
- `region-end`. Parte derecha del documento.



Elemento `<fo:page-sequence>`

Describe el conjunto de páginas con un formato o estructura determinado. Esta estructura de páginas viene referenciadas a través del atributo "master-reference". Este atributo es una referencia a un tag de tipo `fo:simple-page-master` cuyo "master-name" corresponde con el indicado.

```
<fo:page-sequence master-reference="first">
  .
  .
  .
</fo:page-sequence>
```

Elemento `<fo:flow>`

Define la región de la página en la que se insertará el contenido, como por ejemplo el "xsl-region-body".

```
<fo:flow flow-name="xsl-region-body">
  <!-- Aquí va el contenido del documento -->
</fo:flow>
```

Elementos de definición de contenido

Para terminar la definicion de nuestro ejemplo, vamos a ver dos de los elementos mas utilizados a la hora de definir contenido dentro de un documento FO:

- Definicion de bloques con <fo:block>. Define una seccion o parrafo de contenido al estilo de la etiqueta <p> de HTML. Este elemento siempre causa una salto de linea despues de su renderizado. En la propia etiqueta "block" pueden definirse ciertos estilos del texto, como son por ejemplo el tamaño o tipo de fuente.

```
<fo:block font-size="12pt" font-weight="bold">
.
.
.
</fo:block>
```

- Definicion de estilos para el texto contenido en un bloque <fo:inline>. Permite modificar el estilo del texto ya contenido dentro de un bloque, sobrescribiendo asi las definiciones que se habian hecho.

```
<fo:block font-size="12pt">
  Texto <fo:inline font-size="14pt" font-weight="bold">de</fo:inline> ejemplo.
</fo:block>
```

Creación de una hoja de estilos XSL que transformará el documento XML en un fichero XSL-FO

Con el fin de que la informacion que se genere en el documento XSL-FO de salida pueda ser dinamica, haremos que los datos se almacenen en un documento XML, transformandose en el documento de salida mediante el uso de una hoja de estilos XSL.

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

  <xsl:template match="listado">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master margin-right="1cm" margin-left="1cm" margin-bottom="1cm"
          margin-top="1cm" page-width="210mm" page-height="297mm" master-name="
          <fo:region-before extent="1cm"/>
          <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
          <fo:region-after extent="1cm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="first">
        <fo:flow flow-name="xsl-region-body">
          <fo:block>
            <xsl:value-of select="cabecera" />
          </fo:block>

          <fo:list-block space-before="0.5cm" space-after="0.5cm" font-size="12pt">
            <xsl:apply-templates select="articulos/articulo" />
          </fo:list-block>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <xsl:template match="articulo">
    <fo:list-item>
      <fo:list-item-label end-indent="label-end()">
        <fo:block text-align="start">
          <xsl:text>&#x2022;</xsl:text>
        </fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
```

```
<fo:block>
  <xsl:value-of select="text()"/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
</xsl:template>
</xsl:stylesheet>
```

Ejecutar la transformación con un parser XSL

Si vamos a utilizar un procesador en línea de comandos como puede ser el XSLTPROC, la instrucción a ejecutar sería la siguiente:

```
xsltproc sample.xsl sample.xml > sample.fo
```

Si vamos a utilizar Apache ANT. El fichero "build.xml" sería el siguiente:

```
<?xml version="1.0"?>
<project name="procesando-xml" default="fo">
  <target name="init" />

  <target name="fo" depends="init">
    <echo message="Generando sample.fo ..." />
    <xslt in="sample.xml" out="sample.fo" style="sample.xsl">
      <outputproperty name="method" value="xml"/>
      <outputproperty name="standalone" value="yes"/>
      <outputproperty name="encoding" value="ISO-8859-1"/>
      <outputproperty name="indent" value="yes"/>
    </xslt>
  </target>
</project>
```

La ejecución de esta definición nos permite tomar un fichero "sample.xml" como entrada, aplicarle una hoja de estilos xsl "sample.xsl" y obtener como salida un fichero "sample.fo" con el código FO correspondiente.

Procesamiento de dicho fichero XSL-FO con Apache FOP

Para obtener finalmente el fichero PDF que queremos como resultado, podemos ejecutar el siguiente script que viene con la distribución de Apache FOP:

```
/opt/fop-0.20.5rc2/fop.sh sample.fo -pdf sample.pdf
```

7. Herramientas para el trabajo con XML

- XML
 - Apache Xerces (Java)
 - 4Suite (Python)
 - Sablotron (C y PHP)
- XSL

- xlstproc (línea de comandos)
- Apache Xalan (Java)
- Jame's Clark XT (C y Java)
- 4Suite (Python)
- Sablotron (C y PHP)

- Formatting Objects
 - Apache FOP (Java)
 - Passive TeX (LaTeX - línea de comandos y conocido como PDF LaTeX)

8. Entornos de trabajo con XML

- Publicación de documentos XML aplicando plantillas XSL
 - Apache Cocoon (Java)
 - AxKit (Perl)

- Edición de documentos XML/XSL/Schemas/FO
 - XML Spy 4.4 (windows)
 - eXcelon Stylus (windows)
 - Emacs (modo PSGML/XSL)