TEMA 4
LENGUAJES PARA EL ALMACENAMIENTO Y TRANSMISIÓN DE INFORMACIÓN

1. Tipos de lenguajes:
   1.1 De marcas: XML (eXtended Markup Language).
   1.2 De listas: JSON (JavaScript Object Notation).
2. XML: estructura y sintaxis. Etiquetas.
3. Herramientas de edición.
4. Elaboración de documentos XML bien formados y válidos (DTD y XSD):
   4.1 Definición de tipo de documento (DTD, Document Type Definition).
   4.2 Esquema XML (XSD, Xml Schema Definition).
5 Diseño: CSS o XSL
6 Utilización de espacios de nombres en XML.

1. TIPOS DE LENGUAJES.

XML = METALENGUAJE

1.1 De marcas: XML (eXtended Markup Language).

Una breve introducción al mundo XML que explica qué es este lenguaje y sus tecnologías relacionadas.

Qué es XML: XML es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Vamos a ver una introducción al mundo XML, es decir, al lenguaje así como a las tecnologías que trabajan con él, sus usos, ventajas y modos de llevar a cabo las tareas.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Así pues, el XML juega un papel importantísimo en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable, fácil. Además, XML permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

Vemos que XML no está sólo, sino que hay un mundo de tecnologías alrededor de él, de posibilidades, maneras más fáciles e interesantes de trabajar con los datos y, en definitiva, un avance a la hora de tratar la información, que es en realidad el objetivo de la informática en general. XML, o mejor dicho, el mundo XML no es un lenguaje, sino varios lenguajes, no es una sintaxis, sino varias y no es una manera totalmente nueva de trabajar, sino una manera más refinada que permitirá que todas las anteriores se puedan comunicar entre si sin problemas, ya que los datos cobran sentido.

XML es interesante en el mundo de Internet y el e-bussiness, ya que existen muchos sistemas distintos que tienen que comunicarse entre sí, pero como se ha podido imaginar, interesa por igual a todas las ramas de la informática y el tratamiento de datos, ya que permite muchos avances a la hora de trabajar con ellos.

Vamos a ver algunas características importantes de la tecnología que nos permitirán comprender mejor el mundo XML y cómo soluciona nuestros problemas a la hora de trabajar con los datos.

Historia del XML: XML proviene de un lenguaje diseñado por IBM allá por los años 70. El lenguaje inicial se llamó GML (General Markup Language) y surgió por la necesidad de almacenar grandes cantidades de información de temas diversos.
Este lenguaje gustó mucho al organismo de estandarización ISO, y por el 86 trabajaron para normalizar el lenguaje, creando el SGML, que no era más que el GML pero estándar (Standar en inglés). SGML es un lenguaje muy versátil y adaptable a un gran abanico de contextos.
Por el año 89, se empieza a desarrollar y expandir el lenguaje HTML en Internet. Este lenguaje fue adoptado rápidamente por la comunidad de Internet y varias organizaciones comerciales crearon sus propios visores de HTML y riñeron entre ellos para hacer el visor más avanzado, inventándose etiquetas según sus necesidades o intereses. Así que, con el fin de poner un poco de orden, desde el año 1996 el nuevo organismo de estandarización W3C establece sus reglas y etiquetas para que el lenguaje web sea un estándar.
El mismo W3C en el 98 comenzó el desarrollo de XML (Extended Markup Language) y su la adaptación del HTML a las normas de este nuevo lenguaje, dando lugar al lenguaje XHTML. XML aún sigue en pleno desarrollo y pretende solucionar las carencias del HTML en lo que se respecta al tratamiento de la información, como:

- No mezclar el contenido con los estilos que se le quieren aplicar.
- Permitir compartir información con todos los dispositivos, como pueden ser ordenadores o teléfonos móviles.
- Que la presentación en pantalla no dependa del visor que se utilice.

Como veremos en los siguientes apartados y temas, XML es un lenguaje basado en **marcas**, que son elementos del leguaje que permiten delimitar partes del documento para darles entidad propia e indicar cuáles son sus significados. Por tanto, las marcas o etiquetas, son palabras clave que pueden estar predefinidas en el lenguaje (por ejemplo HTML) o pueden ser elaboradas por el propio diseñador web (por ejemplo XML).

1.2 De listas: JSON (JavaScript Object Notation).

JSON (JavaScript Object Notation), es un formato ligero para el intercambio de datos. Se trata de un subconjunto del lenguaje JavaScript que no requiere el uso de XML y permite almacenar información de forma organizada y de fácil acceso.

Ahora pues, no se trata de etiquetas, sino de objetos que se crean a través de la sintaxis de Javascript. Los objetos se pueden crear y agrupar formando arrays o listas de objetos del mismo tipo. Veamos un ejemplo, para crear un objeto libro:

```
var libro = {
   "titulo" : "El ingenioso hidalgo don Quijote de La Mancha",
   "autor" : "Miguel de Cervantes Saavedra",
   "año" : "1605"
};
```

2. XML: ESTRUCTURA Y SINTAXIS.

En general se dice que el XML es una simplificación del original SGML, esto es porque en realidad las normas que tiene son muy simples. Se escribe en un documento de texto ASCII, igual que el HTML y en la cabecera del documento indicamos que se trata de un tipo de documento XML y su versión de la siguiente forma:

<?xml version="1.0"?>

Usamos etiquetas como las de HTML, las etiquetas que nosotros elijamos, de ahí nombre del lenguaje XML, lenguaje de etiquetas extendido. Las etiquetas se escriben anidadas, unas dentro de otras.

<ETIQ1>...<ETIQ2>...</ETIQ2>...</ETIQ1>

Cada nueva etiqueta puede tener atributos. Le podemos diseñar a cada una los atributos que queramos. De la misma forma que ocurre en HTML con los atributos establecidos.

<ETIQ atributo1="valor1" atributo2="valor2"...>

Los comentarios de XML se escriben igual que los de HTML.

<!-- Comentario -->

Y esto es todo lo que es el lenguaje XML en sí, aunque tenemos que tener en cuenta que el XML tiene muchos otros lenguajes y tecnologías trabajando alrededor de él. Sin embargo, no cabe duda que la sintaxis XML es realmente reducida y sencilla.

Para definir qué etiquetas y atributos debemos utilizar al escribir en XML tenemos que fijarnos en la manera de guardar la información de una forma estructurada y ordenada. Por ejemplo, si deseamos guardar la información relacionada con una película en un documento XML podríamos utilizar un esquema con las siguientes etiquetas:

```
<?xml version="1.0"?>
<libro nombre=" El ingenioso hidalgo don Quijote de La Mancha " anio="1605">
        <autor>
                <nombre>"Miguel de Cervantes Saavedra"</nombre >
                <nacimiento>"Alcala de Henares"</nacimiento>
        </autor>
        <tipo nom='Libro de caballeria'/>
</libro >
```

Hemos inventado y usado las etiquetas que nos han sido necesarias y las hemos anidado de manera que adoptan una estructura jerárquica. Primero el LIBRO y dentro de él tenemos el AUTOR y el TIPO. A su vez, dentro del AUTOR tenemos tanto el NOMBRE como la etiqueta NACIMIENTO. Cada marca o etiqueta puede tener sus propios atributos, como también puede anidar otras etiquetas, como ocurre en HTML.

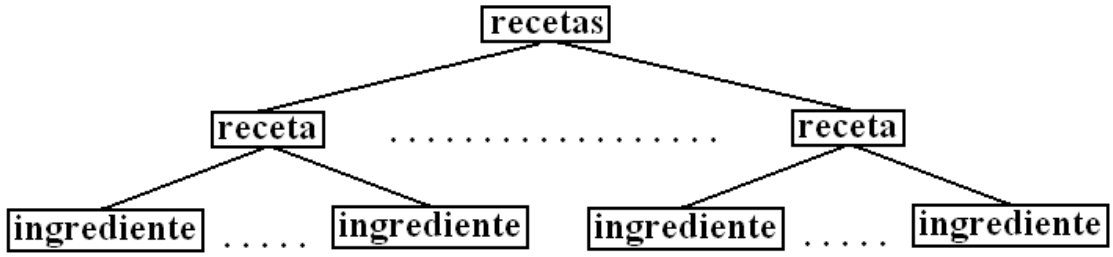
Es relativamente frecuente que antes de adentrarnos al mundo de XML, ya conozcamos el lenguaje HTML. Como ya hemos dicho muchas veces, HTML se preocupa por formatear datos y para ello son las etiquetas que tiene el lenguaje, para formatear la información que se desea mostrar. Sin embargo XML se preocupa por estructurar la información que pretende almacenar. La estructura, la marca la lógica propia de la información. El desarrollo del HTML se debió principalmente a la competencia entre los distintos navegadores y empresas del mercado. Querían ser los mejores e inventaban etiquetas nuevas que a la larga entraban a formar parte del estándar del W3C, como la etiqueta <FRAME>. A partir del año 1999, el W3C empezó a desarrollar con rigor el lenguaje XML, de forma diferente a lo que hicieron las empresas con intereses particulares. Procesar la información en HTML es inviable, por estar mezclada con los

estilos y las etiquetas que formatean la información. En XML se puede procesar la información con mucha facilidad, porque todo está ordenado de una manera lógica, así mismo el formateo de la información para que se pueda entender bien por el usuario es viable a través de un pequeño procesamiento, a través de hojas de estilos o similares.

Ejemplo de XML:
recetas.xml

```
<?xml version=”1.0”?>
<!DOCTYPE recetas SYSTEM "recetas.dtd">
<recetas>
        <receta nombre="Paella" precio="10">
                <ingrediente> Arroz </ingrediente>
                <ingrediente> Sepia </ingrediente>
                <ingrediente> Gambas </ingrediente>
        </receta>
        <receta nombre="Tortilla" precio="6">
                <ingrediente> Huevos </ingrediente>
                <ingrediente> Patata </ingrediente>
        </receta>
</recetas>
```

Como vemos las marcas tienen un significado propio de la aplicación. El documento **recetas.dtd** lo analizamos un poco más adelante, se trata de la “Declaración de Tipo de Documento” (DTD).
Es importante observar la distribución jerárquica en forma de árbol del leguaje:



XML se basa en la utilización de **elementos:**
• Un **elemento** está formado por:
– Una **etiqueta** inicial (nombre entre signos < y >): <etiqueta>
– La etiqueta inicial puede contener **atributos**: <etiqueta atributo=“valor”>
– El elemento debe acabar con una etiqueta final con el mismo nombre </etiqueta>
– El contenido del elemento es todo lo que hay entre la etiqueta inicial y la final:
• El contenido puede estar formado por otros elementos:

```
<receta nombre=“Tortilla” precio=“6”>
        <ingrediente> Patata </ingrediente>
        . . .
</receta>
```

• En caso de un elemento con atributos pero vacío usamos la sintaxis: <etiqueta />
<receta />
• En caso de un elemento con atributos y contenido usamos la sintaxis:
<ingrediente nombre=“Condimento”>sal</ingrediente>

3. HERRAMIENTAS DE EDICIÓN.

Al igual que ocurre con la edición de código HTML, XHTML, CSS, etc., la edición de código XML se puede realizar con el editor de textos más sencillo que podemos encontrar en nuestro sistema operativo (**Notepad** para Windows; **vim, gedit, pico,….** para Linux**; TextEdit** para Mac). Podemos construir documentos XML (instancias de documentos XML, DTD o esquemas XML, hojas, XSLT, etc...), puesto que tales documentos son elaborados en formato de texto llano. Una vez que tenemos los conocimientos básicos del lenguaje podemos trabajar con una herramienta de edición específica que hará más fácil la composición o modificación de cualquier documento XML, y sobre todo, permitirán en todo momento verificar que el documento XML se ajusta a las especificaciones XML, o, dicho de otra forma, validar el documento XML. Pero es importante insistir, en que en primera instancia, todo buen diseñador de código ha de elaborar los documentos en texto llano, lo que le permitirá familiarizarse con el lenguaje y tener el control del código en todo momento, aunque posteriormente haga uso de herramientas que agilicen el diseño del mismo.

XML ha tenido un gran desarrollo y podemos encontrar productos específicamente diseñados para trabajar en este entorno, tanto comerciales como gratuitos. Esto no quiere decir que editores tradicionales como FrontPage, Dreamweaver, etc., no den soporte al diseño de XML en sus aspectos más básicos.

Actualmente, la mayoría de los editores de código XML permiten, al menos, crear documentos XML bien formados (tanto en modo texto como en algún modo gráfico), validarlos frente a una DTD o un esquema XML, crear estas DTD o esquemas XML, y construir hojas XSLT para la transformación de documentos XML en documentos de otros lenguajes o formatos, llegando, en el caso de los más avanzados, a trabajar con otras tecnologías XML como XSL-FO o XQuery.

Editores y parsers (analizadores) XML:

- **Xerces** (http://xerces.apache.org/) de Apache (open source) es uno de los más completos parsers de XML e incluye soporte de XML Schema.
- **XMLSpy** (http://www.altova.com/) de Altova (comercial; uno de los mejores; evaluación de 30 días)
- **MSXML Parser** (http://msdn.microsoft.com/en-us/data/bb190600.aspx) de Microsoft
- **DocZilla** (http://www.citec.com/)de CiTEC (basado en Mozilla)
- **Stylus Studio XML** (http://www.stylusstudio.com/)(Plataforma IDE)
- **Expat XML Parser** (http://expat.sourceforge.net/)
- **XMLPad** de WMHelp (http://www.wmhelp.com/xmlpad3.htm) (gratuito)
- **XML Copy Editor** (http://xml-copy-editor.sourceforge.net/) (gratuito; licencia GNU)
- **EditiX XML Editor** (http://www.editix.com/) (comercial; versión *Lite* gratuita, con funcionalidad reducida, en http://free.editix.com/)
- **oXygen XML Editor** (http://www.oxygenxml.com/) (comercial)
- **XMetaL** (http://na.justsystems.com/) (comercial)
- **Stylus Studio** (http://www.stylusstudio.com/) (comercial)
- Liquid **XML Studio** (http://www.liquid-technologies.com/xml-studio.aspx) (comercial; evaluación de 30 días)
- XMLmind **XML Editor** (http://www.xmlmind.com/xmleditor/) (comercial; versión *Personal Edition* gratuita, con funcionalidad reducida, en http://www.xmlmind.com/xmleditor/persoedition.html)
- **XMLwriter** (http://xmlwriter.net/) (comercial)

4. ELABORACIÓN DE DOCUMENTOS XML BIEN FORMADOS Y VÁLIDOS (DTD Y XSD).

Un documento XML puede contener muchos tipos de información. Podemos definir muchos lenguajes escritos en XML para cualquier colectivo de usuarios y dentro de un contexto determinado. Por ejemplo:

- Si lo utiliza el colectivo de cocineros, podríamos crear un lenguaje en XML específico para almacenar recetas de cocina. Este lenguaje se podría llamar RecetasML.
- Si estamos escribiendo aplicaciones para móviles podremos utilizar un lenguaje para aplicaciones inalámbricas (Wireless), que se llame WML.
- Si los distribuidores de música utilizan XML podrán crear sus propios lenguajes para guardar la información de las canciones. Este lenguaje se podría llamar MusicaML.

En realidad, se pueden crear tantos lenguajes, a partir del las reglas de sintaxis XML, como los que se nos pudiesen ocurrir en cualquier contexto de la vida. Pero para especificar cada uno de esos lenguajes que podemos crear a partir de XML, se utilizan unas reglas sintácticas propias de XML.
XML no es un lenguaje en sí, sino un "**metalenguaje**", es decir, un lenguaje especial ya que se usa para crear otros lenguajes. Es como las cadenas de fabricación automatizadas, formadas por máquinas o robots que a su vez fabrican de forma automática otras máquinas, como coches, frigoríficos, televisiones, etc. XML indica pues la sintaxis y qué etiquetas podemos o debemos encontrarnos en los documentos, en qué orden, dentro de qué otras, etc. además de especificar los atributos que pueden o deben tener cada una de las etiquetas.
Hay dos metalenguajes con los que definir los lenguajes que podemos obtener a partir de XML y que analizamos a continuación:

- **DTD**
- **XML Schema o XSD.**

4.1 Definición de tipo de documento (DTD, Document Type Definition).

La *DTD* (**Definition Type Document o Definición de Tipo de Documento**) tiene una sintaxis especial. Conocemos la sintaxis de XML, porque se parece mucho a la usada en XHTML. La sintaxis DTD también es relativamente sencilla, aunque un poco extraña si nunca antes hemos visto un documento similar. Siguiendo el ejemplo inicial de las recetas, la DTD tendría esta forma:
recetas.dtd

```
<!ELEMENT recetas (receta*)>
<!ELEMENT receta (ingrediente*)>
<!ELEMENT ingrediente (#PCDATA)>
<!ATTLIST receta precio CDATA #REQUIRED>
<!ATTLIST receta nombre CDATA #REQUIRED>
```

En el documento DTD del ejemplo, se definen los elementos (!ELEMENT) que integran el documento XML. En la primera línea se indica que el elemento principal es RECETAS, del que dependen cero o más (+) elementos RECETA. La segunda línea sirve para especificar que el elemento RECETA, a su vez, contiene cero o más

elementos de tipo INGREDIENTE. Detrás aparece el elemento INGREDIENTE que almacenará un dato. Una vez definidos los elementos, pasamos a especificar los atributos (si los tienen) e cada uno de ellos, el elemento RECETA tiene dos atributos: PRECIO que será de tipo cadena y será obligatorio y NOMBRE que será de tipo cadena también y obligatorio.

Consultar el archivo **ANEXO_DTD.doc** para ver las opciones de sintaxis que podemos usar en los documentos DTD.

El formato será más complejo cuantas más especificaciones tengamos que hacer de cada uno de los elementos. Aunque aún no nos centramos en todas las opciones posibles sobre elementos y atributos, modifiquemos el DTD del ejemplo anterior con propiedades más específicas sin entrar en detalles:

recetas.dtd

```
<!ELEMENT receta (ingrediente*, persona?)>
<!ELEMENT servicio (domicilio | restaurante) >
<!ELEMENT ingrediente EMPTY>
<!ELEMENT persona  (#PCDATA)>
<!ELEMENT domicilio (#PCDATA)>
<!ELEMENT restaurante(#PCDATA)>
<!ATTLIST receta nombre CDATA #REQUIRED>
<!ATTLIST ingrediente nombre CDATA #REQUIRED calorías CDATA #IMPLIED>
<!ATTLIST receta precio (euros | dólares) #REQUIRED>
<!ATTLIST persona código ID #REQUIRED>
```

Ahora el elemento RECETA, además podrá contener o no otro atributo que es PERSONA para especificar la persona que elaboró la receta, aparece un elemento más SERVICIO que tendrá un valor opcional DOMICILIO o RESTAURANTE. Etc.

Cuando creamos un documento XML y lo enviamos con un DTD, el documento se reconoce como "**XML válido**". En este caso, un intérprete de XML podría comparar los datos entrantes con las normas definidas en el DTD para comprobar que los datos se han estructurado correctamente. Los datos enviados sin un DTD se reconocen como "**bien formados**". En XML no existen DTDs predefinidos, por lo que es labor del diseñador especificar su propia DTD para cada tipo de documento XML. En la especificación de XML se describe la forma de definir DTDs particularizadas para documentos XML, que pueden ser internas (cuando van incluidas junto al código XML) o externas (si se encuentran en un archivo propio).

Resumiendo:

Documentos bien formados (Well-formed XML)

- Cumplen reglas de sintaxis de XML, (estructuras anidadas, etc.).
- No tienen por qué cumplir una estructura predefinida.
- Los ficheros bien-formados sin-DTD pueden utilizar atributos en sus elementos, pero éstos deben ser todos del tipo CDATA, por defecto. El tipo CDATA (character DATA) usa caracteres.

Documentos válidos (Valid XML)

- Tienen que ser bien formados.
- Cumplen una estructura predefinida (DTD).
- El parser es el encargado de comprobar la validez del documento.

Otro sencillo ejemplo de documento DTD sería éste relacionado con libros:

libros.dtd

```
<!ELEMENT libros (libro)+>
<!ELEMENT libro (titulo, autor, precio)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
```

En la DTD del ejemplo, se definen los elementos (!ELEMENT) que integran el documento XML. En la primera línea se indica que el elemento principal es LIBROS, del que dependen uno o más (+) elementos LIBRO. La segunda línea sirve para especificar que el elemento LIBRO contiene tres elementos (titulo, autor, precio) que se deben marcar en dicho orden. Las restantes líneas aclaran que los elementos titulo, autor y precio, contienen valores de cadenas de texto.

Otro sencillo ejemplo de documento XML:

publicaciones.xml

```
<?xml version=”1.0”?>
<obra>
        <titulo>El quijote</titulo>
        <escritor> Miguel de Cervantes</escritor>
        <editor> Mc Graw Hill</editor>
        <paginas> 2000 </paginas>
        <año> 2010 </año>
</obra>
```

Vamos a realizar un ejemplo más completo. En primer lugar, necesitamos el código XML en el que tratamos datos de personas. En la primera línea además de indicar que se trata de un documento XML declaramos el tipo de codificación:
encoding="ISO-8859-1" ,encoding="UTF-8" ,encoding="Shift-JIS" (japonés), etc.
Por otro lado, la declaración *independiente* indica si un documento se basa en información de una fuente externa, como una definición de tipo de documento externo (DTD), para su contenido. Si la declaración *independiente* tiene un valor de "yes", por ejemplo, <?xml version="1.0" standalone="yes"?>, el analizador informará de un error si el documento hace referencia a una DTD o externa o entidades externas.
Omitir una declaración independiente produce el mismo resultado que incluir una declaración independiente de "no". El analizador XML aceptará recursos externos, si hay alguno, sin informar de ningún error.

personas.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE personas SYSTEM "personas.dtd">
<persona>
        <nombre>John Smith</nombre>
        <nacimiento dia='212' mes='enero' anio='1969'/>
```

```
        <direccion>
                <calle>Avda. Universidad, 30</calle>
                <poblacion>Leganes</poblacion>
                <provincia>Madrid</provincia>
                <cpostal>28905</cpostal>
        </direccion>
        <varon/>
</persona>
```

A continuación debemos especificar el documento DTD con las normas sintácticas que deben cumplir los elementos del documento XML:

personas.dtd

```
<!ELEMENT persona (nombre, nacimiento?, direccion+, (varon|hembra))>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT nacimiento EMPTY>
<!ATTLIST nacimiento dia CDATA #REQUIRED mes CDATA #REQUIRED
    anio CDATA #REQUIRED>
<!ELEMENT direccion (calle,poblacion,provincia, cpostal)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT poblacion (#PCDATA)>
<!ELEMENT provincia (#PCDATA)>
<!ELEMENT cpostal (#PCDATA)>
<!ELEMENT varon EMPTY>
<!ELEMENT hembra EMPTY>
```

4.2 Esquema XML (**XSD**, **Xml Schema Definition**).

Para evitar el DTD, que tiene una sintaxis muy especial, como hemos visto, se intentó encontrar una manera de escribir parecida a XML y se elaboró otro lenguaje XML. Se definió entonces el lenguaje **XML Schema Definition** o **XSD,** y funciona bien, aunque puede llegar a ser un poco más complicado que especificarlo en DTD. Simplemente nos ahorramos el tener que aprender un nuevo lenguaje con su sintaxis particular.

5. DISEÑO: CSS O XSL

Cada documento XML que necesitemos presentar en pantalla con un determinado formato, necesita para su diseño una hoja de estilos o similar. También tenemos dos posibles lenguajes con los que formatear los textos de un documento XML para poder visualizarlo en pantalla:

- **CSS**, que ya conocemos.
- **XSL**, bastante más avanzada.

CSS (**Cascading Style Sheets** o hojas de estilo en cascada) no es nada nuevo, ya lo hemos usado con HTML y se creó en un intento de separar la forma del contenido en HTML. En XML también podemos utilizar CSS, y se utiliza de una manera muy similar a cómo se utilizan en HTML, por lo menos los atributos de estilo que podemos aplicar son los mismos y sus posibles valores también.

XSL (**XML Style Language**), es el segundo lenguaje con el que trabajar en XML para dar estilo al documento. Este lenguaje no se limita a definir qué estilo aplicar a cada elemento del documento XML, sino que además se pueden diseñar pequeñas instrucciones típicas de los lenguajes de programación y la salida no tiene porque ser un documento HTML, sino que además podría ser de otros tipos, cualquiera que podamos necesitar como un documento escrito en WML (para WAP, Wireless Application Protocol o protocolo de aplicaciones inalámbricas que es un estándar seguro que permite que los usuarios accedan a información de forma instantánea a través de dispositivos inalámbricos como PDAs, teléfonos móviles y teléfonos inteligentes o smartphones), un documento de texto plano u otro documento XML. XSL resulta mucho más potente que CSS y de hecho su uso y manejo, es mucho más adecuado en determinados contextos. Para que nos hagamos una idea de su potencialidad, consideremos la siguiente situación: partimos de un documento XML que queremos que se presente en múltiples dispositivos diferentes. Necesitaremos dar formato al documento XML usando XSL. En este supuesto tendríamos un solo documento XML y un documento XSL diferente, que aplicaría plantillas distintitas adaptadas a cada dispositivo en el que queramos presentar el documento: un navegador (Explorer, Firefox, etc.), un móvil (Ericson, Nokia, etc.). Si aparece un nuevo dispositivo en el mercado, por muy particular que sea, sólo necesitaremos crear un nuevo documento XSL, que aplicaría una plantilla diferente, para mostrar el documento XML en el nuevo dispositivo.

6. UTILIZACIÓN DE ESPACIOS DE NOMBRES EN XML.

Pensemos que en XML los DTD son elaborados por cada diseñador de código. Pueden existir miles y miles de DTD particulares y un documento acabaría siendo un caos de diferentes etiquetas procedentes de diferentes sitios. Pero el problema esencial es que se usarían etiquetas con el mismo nombre que, en realidad, significan cosas diferentes dentro del ámbito de un DTD particular (por ejemplo: banco, capital, gato, vela, etc, que pueden significar cosas diferentes). El concepto de espacios de nombres (**namespaces**) permite dividir el conjunto de todos los nombres posibles, de forma que se pueda definir a qué zona de ese espacio corresponde una etiqueta. De esta forma, etiquetas con el mismo nombre, pero definidas por dos autores diferentes, pueden diferenciarse en el espacio de nombres en el que han sido definidas. El espacio de nombres no es esencial en todos los documentos, pero resulta útil cuando se usan etiquetas procedentes de diferentes orígenes (por ejemplo, etiquetas nuevas dentro de un documento XML), o etiquetas que se quieren procesar de forma diferente.
La solución estaría en asociar a cada etiqueta un URI que indica a qué espacio de nombres pertenece la etiqueta usando el formato <namespace_URI:etiqueta>, por ejemplo:
[http:www.muebles.es]:banco
[http:www.edificios.com]:banco
Si consideramos que el elemento banco contiene a su vez otro elemento banco como un mueble de su mobiliario, quedaría de la siguiente forma:

```
<[http://www.edificios.es]:banco>
        <[http:www.muebles.com]:mesa> madera </[http:www.muebles.com]:mesa>
        <[http:www.muebles.com]:banco> marmol </[http:www.muebles.com]:banco>
</[http:www.edificios.es]:banco>
```

Es esencial saber que los URIs sólo se utilizan para que el nombre sea único, **no son**

enlaces, ni tienen que contener información, sin embargo, también los URIs sirven para acceder a recursos.

El espacio de nombres se abrevia ya que cuando existen varios en un documento sería muy larga su especificación, por ello se asocia un **alias** a los elementos de un espacio de nombres dentro de un ámbito usando xmlns:nombre del prefijo="……URI……" así:

```
……
xmlns:edificios="http://www.edificios.es"
xmlns:muebles="http://www.muebles.com">
…….

<edificios:banco>
        <muebles:mesa> madera  </muebles:mesa>
        <muebles:banco> mármol </muebles:banco>
        ……………
 </edificios:banco >
```

En caso de que no se especifique ningún prefijo, se puede también especificar qué espacio de nombres sigue, por defecto, el documento:

```
<mipersona xmlns='http://www.personas.org/mipersona'>
        <nombre>Francisca<nombre >
        <nacimiento dia="12" mes="octubre" anio="1931"/>
        <direccion>
                <calle>Avda. Universidad, 30</calle>
                <poblacion>Leganes</poblacion>
                <provincia>Madrid</provincia>
                <cpostal>28905</cpostal>
        </direccion>
        <varon/>
 </mipersona>
```

Resumiendo, el prefijo de un espacio de nombres es totalmente arbitrario; lo que define un espacio de nombres es, en realidad, el URI. (Universal Resource Identification).
En el ejemplo, declaramos el prefijo del espacio de nombres mediante el atributo **xmlns** (XML namespace). Por tanto, el espacio de nombres tiene que tener asignado un URI, que es básicamente algo que parece una dirección web, pero que no lo es, como ya se dijo. Lo único que se requiere de este URI es que sea único en el documento; además, es aconsejable que sea siempre el mismo cuando se use el mismo namespace, aunque no es estrictamente necesario, ni se puede comprobar. El que sea un URI significa, entre otras cosas, que si nos dirigimos a esa dirección no tiene porqué haber nada. Se trata simplemente de asignar un identificador único.
En el resto de los elementos se sigue usando el espacio de nombres. **Incluso se puede usar en los atributos**, si pertenecen al mismo espacio de nombres.
Un documento XML puede tener tantos espacios de nombres como se quieran declarar, y se pueden mezclar elementos de diferentes espacios de nombres, e incluso sin ningún espacio, tal como se hace en el siguiente ejemplo, en el que la dirección es un elemento perteneciente al elemento padre mipersona y definimos un espacio de nombres para cada uno de ellos. En este caso, hemos elegido el prefijo mp.:

```
<mp:mipersona xmlns:mp='http://www.personas.org/mipersona'
                    xmlns:direccion='http://www.personas.org/direccion'>
        <mp:nombre>Francisca</mp:nombre >
        <mp:nacimiento dia="12" mes="octubre" anio="1931"/>
        <mp:lugar>
                <direccion:calle>Avda. Universidad, 30</direccion:calle:calle>
                <direccion:poblacion>Leganes</direccion:poblacion>
                <direccion:provincia>Madrid</direccion:provincia>
                <direccion:cpostal>28905</direccion:cpostal>
        </mp:lugar>
        <mp:varon/>
 </mp:mipersona>
```

En este ejemplo, hemos declarado dos espacios de nombres, mp y direccion, y cada uno lo usamos para una cosa diferente.

Los espacios de nombres, conviene usarlos cuando no hay otro remedio, o cuando hay que combinar conjuntos de etiquetas XML procedentes de diferentes orígenes. En todo caso, en la documentación de un conjunto de etiquetas conviene especificar un espacio de nombres, para que se las pueda identificar fácilmente cuando aparezcan en un documento.
Los espacios de nombres en combinación de los DTDs, servirán para especificar qué diccionario de datos usar en cada momento.