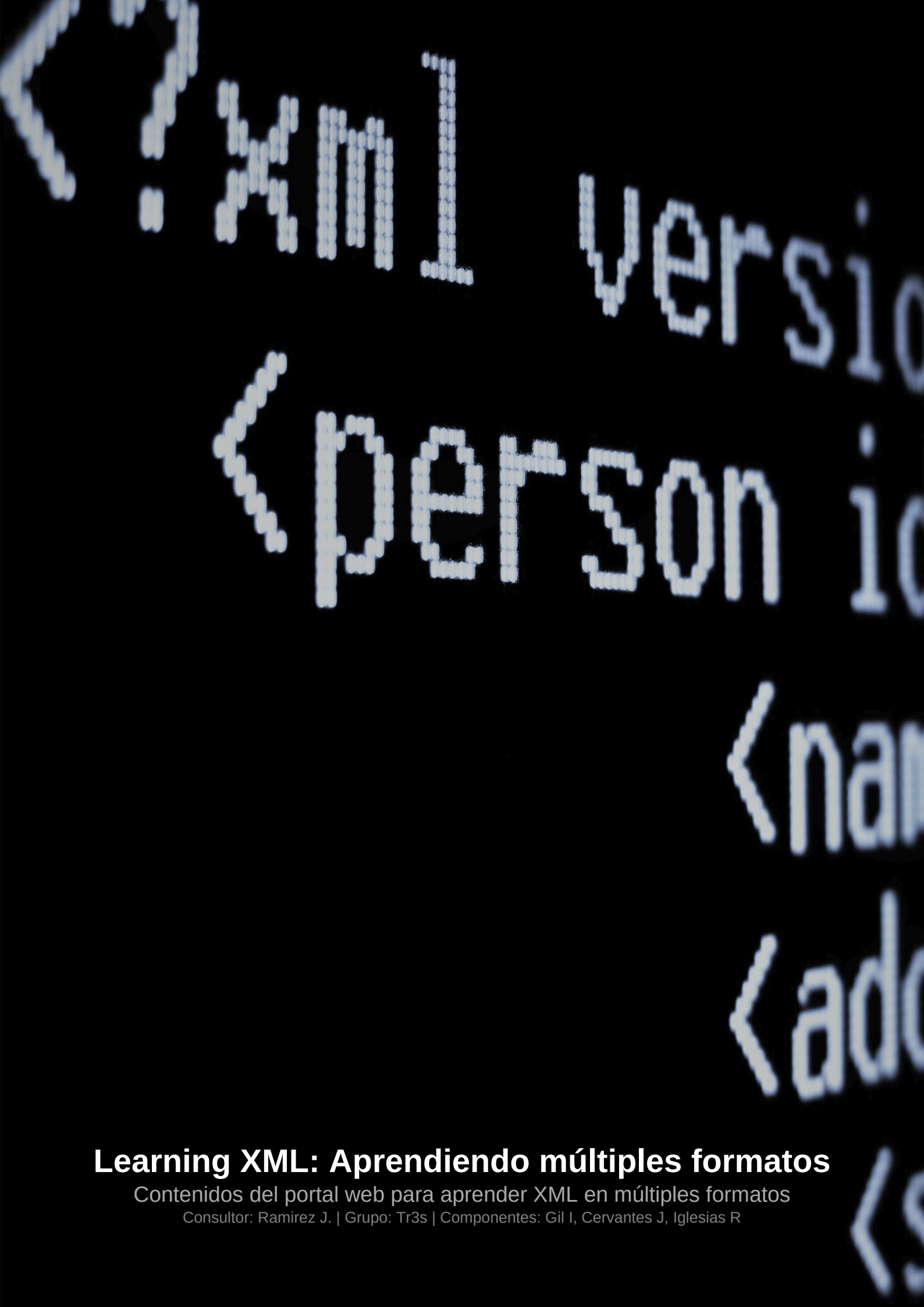


Learning XML: Aprendiendo múltiples formatos

Contenidos del portal web para aprender XML en múltiples formatos

Consultor: Ramirez J. | Grupo: Tr3s | Componentes: Gil I, Cervantes J, Iglesias R

INDICE

Proyecto: Learning XML

Proyecto: Learning XML

1. Lenguajes de marcas

1.a. ¿Qué son los lenguajes de marcas?

Son un método de codificación de documentos que además del texto incluyen etiquetas o marcas que contienen información adicional acerca de la estructura del texto, su contenido semántico o cualquier otra información lingüística o extralingüística. En definitiva se podría definir como el lenguaje en que se escribe el texto para que el ordenador pueda manipularlo.

La mayoría de los lenguajes de marcas son legibles, pues dichas anotaciones pueden distinguirse del texto. Por ejemplo encontramos las etiquetas de formato “<” y “>” en HTML, xHTML y XML y sabemos que lo que hay dentro de estos delimitadores es el lenguaje de marcado, y que el resto del texto es simplemente el texto anotado.

También consideramos importante destacar que los lenguajes de programación no son lo mismo que los lenguajes de marcas, ya que el lenguaje de marcado no tiene funciones aritméticas o variables.

Las características más relevantes de los lenguajes de marca son:

Utilizan texto plano: pueden ser editados con cualquier procesador de textos y son independientes de la plataforma.

Son compactos: las marcas se combinan con el contenido.

Tienen facilidad de procesamiento: Por lo general resultan lenguajes fáciles de comprender y aprender.

Presentan una gran flexibilidad: Se pueden usar en cualquier área.

Los lenguajes de marca se pueden clasificar en dos grandes grupos:

Lenguajes de presentación y lenguajes de procedimiento: Se utilizan para definir el formato y la presentación del texto. Por tanto, este tipo de marcado es útil para maquetar la presentación de un documento (por ejemplo la fuente del texto o la aparición de saltos de página) y el programa debe interpretar el código con el mismo orden en que aparece.

Lenguajes descriptivos o semánticos: Definen el significado de los fragmentos de texto mediante las marcas, sin detallar cómo serán

representados ni en qué orden. Los fragmentos se etiquetan tal como son y no tal como deben aparecer. Por ejemplo, definen el texto de un hipervínculo para que cuando el usuario pulse sobre este texto produzca la acción de llevarle a una página determinada.

1.b. Ejemplos de lenguajes de marcas

GML (Generalized Markup Language):

Este lenguaje descriptivo se utiliza para definir el tipo de texto (si se trata de un párrafo, cabecera, lista, tabla...). El documento puede tener diferentes perfiles y puede cambiar de un tipo de perfil a otro. Por ejemplo, es posible dar formato a un documento para impresora láser simplemente especificando un perfil para el dispositivo en cuestión sin modificar el documento en sí.

SGML (Standard Generalized Markup Language):

El desarrollo de GML cambió al SGML que es un estándar para definir lenguajes de marcas generales para documentos descendiente del GML.

XML (Extensible Markup Language):

Fue inicialmente un simplificado desarrollo de SGML que ha recibido una gran aceptación. Más adelante explicaremos de qué se trata.

HTML (HyperText Markup Language):

Es un lenguaje de marcado de presentación ampliamente utilizado para definir el contenido de una página web (maquetación y estructuración). Al ser un estándar, HTML permite que cualquier web escrita en una determinada versión pueda ser interpretada de la misma forma por cualquier navegador web actualizado.

xHTML (eXtensible HyperText Markup Language):

Podríamos definirlo como HTML expresado como XML válido. Lo estudiaremos más detalladamente a lo largo del presente trabajo.

Hay muchos más lenguajes de marcado, como por ejemplo: RTF (lenguaje de presentación), TeX (lenguaje de procedimiento),YAML (lenguaje descriptivo).

2. Extensible Markup language (XML)

2.a. Definición de XML

El XML es un lenguaje de adaptación del SGML (Standard Generalized Markup), es un lenguaje que permite el etiquetado y organización de documentos. Así pues, XML no es un lenguaje como tal, sino más bien un sistema que permite definir los lenguajes. Como ejemplo de algunos lenguajes que el XML puede definir podemos encontrarnos con el SVG, o el XHTML.

El metalenguaje aparece como un standard que estructura el intercambio de información entre las diferentes plataformas. Las hojas de cálculo, los documentos de textos, o incluso las páginas web, son algunos de los campos donde se utiliza el lenguaje XML.

Las ventajas que derivan de la utilización del XML podrían ser como por ejemplo su extensibilidad, ya que permite añadir nuevas etiquetas tras el diseño del documento; su analizador es standard, no requiere de cambios para cada metalenguaje que se desea utilizar; y quizá como más importante o llamativo, es que facilita el análisis y el procesamiento de los documentos XML creados por terceros.

El XML presenta una serie de ventajas muy atractivas para los desarrolladores, especialmente porque permite relacionar aplicaciones de diferentes lenguajes e incluso plataformas. El XML busca una universalidad de los lenguajes de desarrollo.

Nos encontramos con lenguajes creados con XML, como ejemplos podríamos nombrar el XSL (Extensible Stylesheet Languaje) o el XLINK.

2.b. Utilidades de XML

Debemos partir de la idea que el XML no es un lenguaje de programación sino de marcado. Así pues, podemos decir que es un sistema que ofrece la posibilidad de estructurar y representar datos. Debemos observarlo como una herramienta. Actualmente es muy común que los programas incluyen archivos de configuración en XML. Podríamos citar aplicaciones de Microsoft del tipo .NET, o incluso en Apache.

Si tratáramos de desarrollar un programa con interfaz gráfica sería necesario organizar todas las imágenes de manera que vayan cargando a medida que se necesitaran. Aquí, el XML es de gran ayuda puesto que permite agruparlas, etiquetarlas, especificar su ubicación, relacionarlas con otros datos, todo a preferencia de los mismos diseñadores.

Pero el XML no sólo sirve como una herramienta de organización de recursos y configuración de programas. También cumple un papel muy importante, quizás su punto más fuerte o relevante. La posibilidad de comunicación con otras aplicaciones de diferentes plataformas y sin importar el origen de la información. Sirva como ejemplo que se podría tener un programa en ejecución en Windows con una base de datos de SQL Server, e inclusive tener en Linux uno con Oracle, compartiendo ambos datos gracias a una estructura en XML.

Con toda seguridad en los servicios web son componentes de la red que brindan la posibilidad de realizar una serie variada de operaciones. Gracias al XML, se pueden comunicar sin importar el tipo de plataforma, cualquiera puede hacer uso de sus ventajas.

Quizá habría que resaltar que es una herramienta de escasa complejidad, es fácil de usar e innegablemente útil.

2.c. Estructura de documentos XML

Estructura genérica XML:

```
<raíz>
<hijo1>
<subhijo1_1>
<subhijo1_1_1> ... </subhijo1_1_1>
<subhijo1_1_2> ... </subhijo1_1_2>
...
</subhijo1_1>
<subhijo1_2> ... </subhijo1_2>
...
</hijo1>
<hijo2> ... </hijo2>
...
</raíz>
```

Estructura documento XML con prólogo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nota>
<para>Pedro</para>
<de>Laura</de>
<titulo>Recordatorio</titulo>
<contenido>El jueves es mi cumpleaños</contenido>
</nota>
```

Los documentos XML tienen una estructura general que está formada en dos partes:

-Prólogo, aunque opcional, contiene una secuencia de instrucciones de procesamiento y/o declaración del tipo de documento. Se puede dividir en dos partes:

Declaración XML. Establece la versión de XML, el tipo de codificación y si es un documento autónomo.

En la declaración XML nos encontramos que es una instrucción de procesamiento especial, y que cumple diferentes funciones:

-Marca el documento como texto XML

-Incluye la declaración de la versión de XML utilizada en el documento

-Aporta información sobre la codificación empleada para representar los caracteres

-Indica si el documento es autónomo o no.

Si observamos este ejemplo:

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

Nos encontramos con que nos indica que es un documento XML, y que la versión XML es la 1.0.

Con la información “encoding=ISO-8859-1” nos informa que el lenguaje es el asociado a los países de la Europa Occidental, permite acentos y es el usado en castellano. Cómo información general hay que indicar que por defecto el encoding sería “UTF-8”, pero no admite caracteres ni acentos usados en el castellano.

Por último, nos encontramos con la información standalone=”yes”, que indica que el documento es autónomo y que tiene toda la información para su interpretación. Cabe recordar que nos indica si es autónomo o no, así pues las únicas posibilidades es un “yes” o un “no”.

Declaración de tipo de documento. Establece el tipo de documento que es.

Se escribe en el prólogo y tiene un formato especial, distinto de las marcas y de las instrucciones de procesamiento. Provee una serie de mecanismos que aportan funcionalidad a XML. Hace posible definir una serie de restricciones adicionales que deben cumplir los documentos. También incorpora la

posibilidad de utilizar ciertas herramientas que facilitan al usuario XML algunas tareas. Todas estas adicionales se engloban bajo lo que se denomina un tipo. Su declaración no siempre es necesaria.

El prólogo añade información sobre el documento, concretamente, declara que el documento es un documento XML e incluye información sobre la versión utilizada. Además puede incluir información sobre el tipo de codificación de caracteres utilizado en el documento, si es autónomo o no, y el tipo al que se ajusta el documento.

Pese a que el prólogo es opcional, es recomendable su incursión, ya que facilita un proceso fiable de la información contenida.

-Cuerpo, es el contenido de información del documento, está organizado como un árbol único de elementos marcados.

El cuerpo es la parte más importante y que contiene toda la información del documento. El cuerpo de los documentos XML tiene una estructura de árbol, en la que siempre existe un elemento principal, dentro del cual se encuentran el resto de los elementos. Todos los elementos de un documento XML pueden a su vez contener sub-elementos

También cabe destacar que el standard permite la inclusión de un epílogo al final del documento, que puede contener instrucciones de procesamiento.

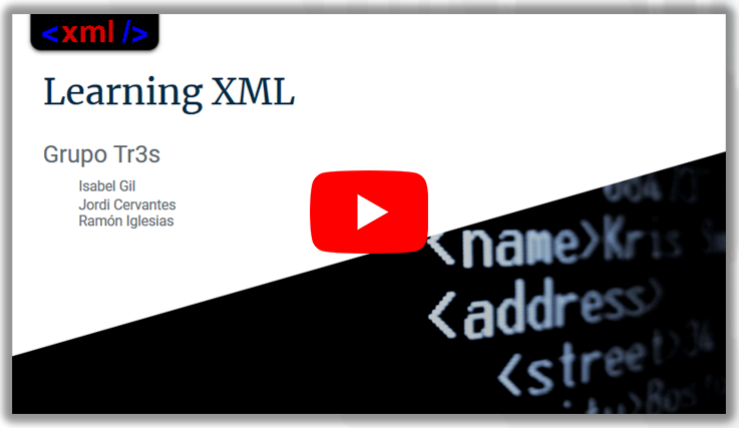
Las instrucciones de procesamiento se utilizan para enviar información a las aplicaciones que van a procesar el documento XML. Las instrucciones de procesamiento pueden aparecer en varios lugares del documento. Por ejemplo entre el prólogo y el cuerpo, dentro del cuerpo o en el epílogo.

Para distinguir entre el cuerpo del documento y el documento completo se usa:

Document entity (Document root); todo el documento

Document element; al cuerpo o a cualquiera de sus partes

Para finalizar la sección, también dispones de un video resumen a continuación:



Grupo Tr3s | Link: https://www.youtube.com/embed/VFbKqtO79TI

2.c.i ¿Que es DTD?

Una DTD es un documento que define la estructura de un documento XML: los elementos, los atributos, las entidades, las notaciones,...que pueden aparecer, el orden y el número de veces que pueden aparecer, cuales pueden ser hijos de cuales,... El procesador XML utiliza la DTD para verificar si un documento es válido, siendo válido dicho documento si cumple con las reglas del DTD, lo cual explicaremos más concretamente en el apartado “d”.

La DTD que debe utilizar el procesador XML para validar el documento XML se indica mediante la etiqueta DOCTYPE. La DTD puede estar incluida en el propio documento, ser un documento externo o combinarse ambas. La clara desventaja de hacerlo en el propio documento es que estaremos definiendo reglas que sólo se aplicarán a ese documento.

-La DTD incluida en el propio documento se definiría de la siguiente forma:

<!DOCTYPE nombre [

.... declaraciones …

]>

-La DTD puede estar en un documento externo y, si sólo quisiéramos que fuera utilizada por una aplicación, su sintaxis sería:

<!DOCTYPE nombre SYSTEM "uri">

-Se podría combinar una DTD externa con una DTD interna con la siguiente sintaxis:

<!DOCTYPE nombre SYSTEM "uri" [

... declaraciones …

]>

-La DTD puede estar en un documento externo, pero si va a ser utilizada por varias aplicaciones, esta sería su sintaxis:

<!DOCTYPE nombre PUBLIC "fpi" "uri">

-Se podría combinar una DTD externa con una DTD interna, como por ejemplo:

<!DOCTYPE nombre PUBLIC "fpi" "uri" [

... declaraciones ...

]>

Para todos lo ejemplos, debemos remarcar tres consideraciones:

“nombre” es el nombre del tipo de documento XML, que deberá coincidir con el nombre del elemento raíz del documento XML

“uri” es el camino hasta la DTD (absoluto o relativo)

“fpi” es un identificador público formal (Formal Public Identifier)

2.c.ii Ejemplos de gramáticas DTD

Debemos partir de que el Unicode es el alfabeto de los documentos XML. El uso de Unicode es un requisito impuesto a XML para la codificación de los caracteres de sus textos. De este modo se asegura la universalidad buscada para la web.

Gracias al Unicode podemos usar cualquier carácter, a excepción de los que tienen algún significado especial para el lenguaje, por ejemplo:

&(utilizado para dar un texto que reemplaza a una entidad)

< y > (usadas para las etiquetas)

“ (reservadas para delimitar valores de atributos)

2.c.iii Principales elementos de una gramática DTD

Hay que recalcar que nos encontramos con una aportación procedente de SGML incorporada a XML. También se encargan de ello los Esquemas XML.

Una DTD es una colección de declaraciones de elementos (ELEMENT), atributos (ATTLIST), entidades (ENTITY) y notaciones (NOTATION); a partir de las cuales se describe la validez de un documento. Pasamos a relatarlos:

Aquí en nombre se referencia al documento XML junto con la “uri” donde localizarlo:

<!DOCTYPE nombre [

…

]>

Los elementos de DTD son bloques primarios de todo documento y se declaran así:

<!ELEMENT nombre (modelo de contenido)>

1.- Se empieza con “ELEMENT”, seguido de un nombre identificativo (hay que recordar que se distingue entre mayúsculas y minúsculas), y entre paréntesis se especificaría el contenido que está permitido, que será un contenido concreto y debe existir una etiqueta con ese nombre. Por ello se conocen como especificación o modelo de contenido. También puede contener más elementos creando una posible cadena de subelementos entre sí.

Los modelos de contenido que puede tener un elemento son:

#PCDATA: “Parser Character Data”alerta de que no sólo se pueden indicar nombres de elementos como contenido concreto, sino que el elemento podrá incluir texto. No obstante, este texto debe cumplir las reglas XML, como no contener símbolos prohibidos.

<!ELEMENT cosa (#PCDATA)>

“cosa” debe contener un tipo de dato analizable.

Como ejemplo podríamos indicar que sólo pueda tener un elemento:

<!ELEMENT clase (profesor)>

Otro ejemplo, podría ser:

<!ELEMENT clase (profesor, aula)>

Aquí especifica que el elemento clase debe contener un elemento profesor seguido de un aula. En este ejemplo estamos hablando de una secuencia, que consiste en una serie de elementos separados por una coma.

EMPTY: El elemento no tiene contenido, se llama elemento vacío:

<!ELEMENT salto-de-pagina EMPTY>

Otro ejemplo de elemento vacío es:

<salto-de-pagina></salto-de-pagina>

o bien:

<salto-de-pagina />

ANY: Indica que el elemento puede tener cualquier contenido.

<!ELEMENT clase ANY>

MIXED: Indica que el elemento puede tener caracteres de tipo dato o una mezcla de caracteres y subelementos, pero al contrario que ANY sus contenidos deben estar debidamente especificados:

<!ELEMENT postre (helado | pastel)>

El símbolo “|” lo utilizaremos para establecer una elección, donde se podrá elegir entre las opciones que hay a cada lado del símbolo.

Símbolo

Descripción

Ninguno

El elemento aparece sólo una vez

Signo más (+)

El elemento aparece una o más veces

Asterisco (*)

El elemento es opcional y puede aparecer cualquier número de veces

Signo de interrogación (?)

El elemento es opcional y puede aparecer sólo una vez

2.- Una declaración de atributo permite añadir información sencilla y desestructurada a los elementos de un documento, puesto que puede existir más de un atributo por elemento. Se utiliza una lista para ello en la llamada declaración de lista de atributos ATTLIST:

<!ATTLIST elemento nombre del atributo TYPE Palabra clave>

Un ejemplo podría ser:

<!ELEMENT mensaje (de, a, texto)>

<!ATTLIST mensaje prioridad (normal | urgente) normal>

Nos encontramos con tres palabras clave: #REQUIRED, significa que no tiene valor por defecto, por lo que es obligatorio especificar este atributo. Por otro lado, si el atributo es opcional se usa la palabra clave #IMPLIED:

<!ATTLIST IMG URL CDATA #REQUIRED>

<!ATTLIST IMG ALT CDATA #IMPLIED>

Expresa que el atributo “URL” es obligatorio, mientras que el atributo “ALT” es opcional.

La tercera palabra clave #FIXED especificaría que el valor del atributo es constante y no puede cambiar a lo largo del documento:

<!ATTLIST código postal #FIXED "08030">

Nos indica que el código 08030 es el único que se utilizará.

Los posibles atributos son:

-Cadenas CDATA; datos de caracteres que pueden tomar como valor cualquier secuencia o cadena de caracteres a excepción de los símbolos con significado especial:

-ID, indica que el atributo tiene un nombre definido y que con su uso se determina el valor aceptable para cada instancia del elemento al que se aplica.

Suponiendo un ejemplo de una empresa al que queremos relacionar, a cada empleado se le asigna un número de seguridad social, y que dos empleados no pueden tener el mismo:

<!ELEMENT empleado (#PCDATA)>

<!ATTLIST empleado nss ID #REQUIRED>

-IDREF; mientras ID identifica a un elemento, IDREF apunta a un elemento con un atributo ID. Ejemplo de un sistema de hipervínculos en un documento:

<!ELEMENT enlace EMPTY>

<!ATTLIST enlace destino IDREF #REQUIRED>

-Enumeraciones; corresponde a atributos que sólo pueden contener un valor de entre un número reducido de opciones. Se limita a los valores que aparecen entre paréntesis y separados por el símbol “|”.

<!ATTLIST telefono lugar (oficina | movil | particular) "oficina">

-NMTOKEN (autenticaciones) parecidos a las cadenas CDATA, aunque imponen restricciones sobre los valores de los atributos. Solo aceptan caracteres válidos para nombrar cosas.

<!ATTLIST pais población NMTOKEN #REQUIRED>

3.- ENTITY, permite especificar un atributo cuyo valor sea una entidad. Se refiere a un objeto usado para guardar información y por ello necesariamente cada documento tiene al menos la entidad del propio documento. Su razón de ser es doble, permite guardar un contenido que puede usarse varias veces y poder descomponer un documento grande en subconjuntos más manejables.

<!ENTITY derechos "Copyright 2019">

-Entidad externa: su contenido no está dentro de la propia DTD, sino en cualquier otro sitio del sistema.

<!ENTITY intro SYSTEM "Contenidos XML">

-Entidad paramétrica: permite agrupar datos dentro de la DTD para poderlos escribir de forma abreviada, se distinguen por el uso de un nombre encabezado por “%”

<!ENTITY %nombre "valor">

Como ejemplo:

<!ATTLIST calcetin %comun>

el analizador interpretaría:

<!ATTLIST calcetin

talla (pequeña|media|grande)" ´media´

color "(rojo|azul|negro|blanco)" ¨blanco¨

precio CDATA #REQUIRED">

4.- NOTATION. Tiene como significado notas o anotación. De esta forma nos indica que su idea es proporcionar información adicional. Su sintaxis sería:

<!NOTATION nombre SYSTEM "información de notación">

La información puede ser una simple palabra clave, un URL, o cualquier otro tipo de descripción.

Se distinguen su uso de atributos y entidades añadiendo NOTATION a ATTLIST, lo que permite especificar en el momento de declarar el atributo que su valor se ajusta a una anotación dada:

<!ATTLIST fecha NOTATION (ISO-DATE| EUROPEAN-DATE) #REQUIRED>

Para su uso dentro de una entidad se incorpora en la entidad externa la palabra clave NDATA.

Un ejemplo sería, incluir en la DTD un logo del que se tienen dos versiones, una GIF y otra JPEG.

<!NOTATION gif SYSTEM "gif">

<!NOTATION jpeg SYSTEM "jpeg">

A continuación se incluiría en las entidades, con las anotaciones “gif” o “jpeg”

<!ENTITY logo-gif SYSTEM "imagenes/compañia-logo.gif" NDATA gif>

Las DTD tiene la posibilidad de incluir o ignorar declaraciones, las palabras claves serían INCLUDE, para incluir; e IGNORE para excluir.

<![INCLUDE [

<!ELEMENT nombre (#PCDATA)>

]]>

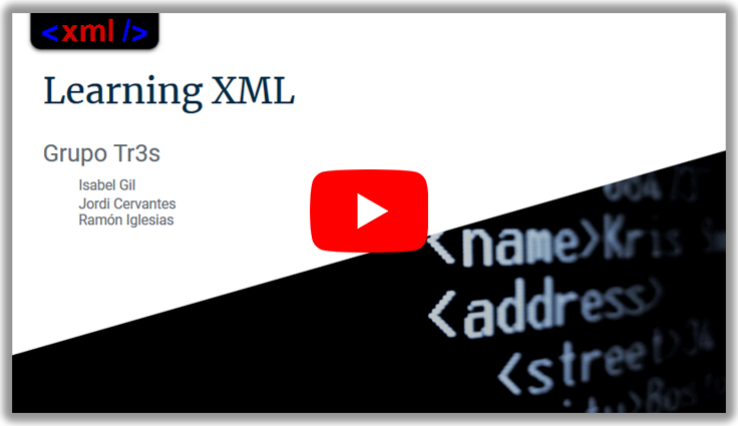
<![IGNORE [

<!ELEMENT mensaje (#PCDATA)>

]]>

A su vez, al estar precedidas del carácter ”%” son entidades paramétricas, y por tanto sólo se podrían usar dentro de la DTD que ya se han declarado. Así pues, de la forma %aceptar y %rechazar, representan INCLUDE e IGNORE respectivamente.

Para finalizar la sección, también dispones de un video resumen a continuación:



Grupo Tr3s | Link: https://www.youtube.com/embed/aqxCtEnYMPU

2.d. Validación documentos XML

Para poder entender la validación antes repasaremos en qué consiste un documento XML bien formado.

Los documentos XML bien formados son aquellos que se ocupan de que las reglas de XML sean respetadas y por tanto garantizan que no se darán incoherencias al utilizar el lenguaje. No obstante, y aunque estén bien formados, tales documentos no tienen por qué ser coherentes.

Cuando hablamos de coherencia nos referimos a que el documento no podrá presentar errores (debe ser semánticamente correcto), mientras que cuando decimos que está bien formado nos referimos a la sintaxis, a que la secuencia en sí sea correcta.

Con la validación (mediante un analizador de validación), por tanto, nos aseguraremos de que nuestros documentos cumpla unas reglas más concretas y podremos determinar protocolos para los documentos, estableciendo lenguajes propios de marcado que garanticen una coherencia.

Algunos de los principales lenguajes de validación que se utilizan para decretar las reglas de construcción de documentos XML son: DTD, XML Schema, Relax NG y Schematron.

Hoy en día el lenguaje de validación que es considerado el más coherente por los programadores es XML Schema, pero todavía no tiene una implantación del 100%, pues se considera algo complejo y presenta ciertas desventajas. No obstante DTD es el más veterano y utilizado por su baja complejidad, su compatibilidad con mayor software y porque permite definir entidades, entre otras tantas ventajas.

Por tanto podríamos generalizar diciendo que un documento puede ser correcto a dos niveles: bien formado cumpliendo las reglas de XML y válido cumpliendo las reglas establecidas en DTD.

2.d.i Herramientas de validación de documentos XML.

Para validar un documento con su DTD podemos utilizar diferentes métodos:

Descargar de internet un complemento (en Internet Explorer es “Internet Explorer Tools for Validating XML and Viewing XSLT Output”). Este complemento es muy sencillo de usar: una vez instalado simplemente abrimos el documento en el navegador y damos click derecho, posteriormente seleccionamos “Validate XML” del menú desplegable y nos saldrá una ventana emergente donde nos informarán de si dicho documento es válido o no.

Utilizar un servicio online como por ejemplo ·”Validome” utilizando la URL de la página donde tenemos el documento o introduciendo el documento directamente.

Editores de texto como Notepad++ instalando un plugin.

Utilizando un software específico que deberemos instalar en nuestro ordenador, como por ejemplo “Editix”.

2.e. Transformación de estructuras XML

2.e.i ¿Que es el lenguaje de transformación XSLT?

XSLT (eXtensible Stylesheet Language Transformations) es un lenguaje de programación declarativa. Básicamente permite, a partir de un documento XML y una hoja de estilo XSLT; reestructurar radicalmente un documento XML original generando un nuevo documento XML o incluso convertirlo a nuevos formatos como XHTML, HTML, JSON, CSV, PostScript o PDF. (Aunque los dos últimos necesitan de XSL-FO para la presentación).

Por tanto, tal y como se ilustra en la siguiente figura, para realizar una transformación es necesario como mínimo un documento XML (donde encontraremos los contenidos) un documento XSLT (donde encontraremos las instrucciones para transformar ese XML en un nuevo archivo) y un procesador XSLT (software que se encargará de interpretar las instrucciones del documento XSLT en relación a los contenidos del archivo XML original para crear el nuevo documento).

Conviene prestar especial atención a que este diseño permite separar contenido (dentro del XML) de la presentación (definida por el XSLT). Esto ha favorecido que en la práctica, uno de los entornos donde se emplee más éste lenguaje sea en el desarrollo web, ya que permite adaptar un mismo contenido descrito en XML a diferentes pantallas y navegadores empleando XSLT.

Como se ha comentado antes XSLT es un lenguaje declarativo. Por tanto, las instrucciones contenidas en el archivo XSLT no es una secuencia de instrucciones (como por ejemplo la secuencia de instrucciones que encontramos en el lenguaje imperativo C) sino una colección de reglas de plantilla (template rules) que se aplican en los elementos que seleccionemos del XML original. (selección que haremos empleando XPath)

Por ejemplo, analizando aisladamente la siguiente template rule:

<xsl: for-each select="Elemento/s o Expresión">

<!-- Aquí el tratamiento que recibirán los elementos-->

</xsl: for-each>

Por un lado tendríamos la instrucción en sí. En este caso una template rule “for-each” que está emulando un bucle para cada elemento que recibirá, y por el otro lado, un “select” que utilizando el lenguaje XPath permitirá concretar qué elementos o nodos del XML original serán tratados por la template rule.

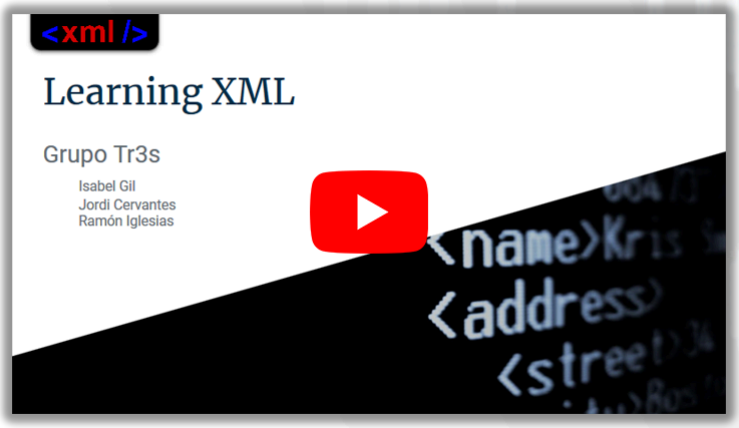
De hecho, una concepción válida para trabajar con XSLT sería pensar en template rules y grupos de nodos en lo que se aplicarán estas reglas (el grupo será el que seleccionemos con la expresión XPath en la opción select o previamente con la template rule <xsl: template match=”condicionesExpresadasConXPath”>)

Por último, señalar que el lenguaje XSLT desarrollado y normalizado por el W3C. Éste, a través de sus grupos de trabajo, ha ido incorporando nuevas funciones y características que se publican en las sucesivas versiones etiquetadas como “recomendadas” (Actualmente, la última versión recomendada por el W3C es la XSLT 3.0, presentada el 8 de Junio del 2017)

Sin embargo, son varios los procesadores XSLT que no han avanzado más allá de la versión XSLT 1.0 (no tan solo navegadores como Chrome y Firefox sino en IDEs como el entorno .NET de Microsoft, donde se encuentra solicitudes de implementación de XSLT 3.0, incluido XQuery 3.0 y XPath 3.0, desde Abril del 2016)

En consecuencia, aunque las sucesivas versiones han ido incorporando funcionalidades interesantes, que pueden facilitar el trabajo con XSLT, al empezar nuevos proyectos conviene plantear previamente qué IDE vamos a emplear y qué necesidades tiene nuestro proyecto, ya que podríamos tener problemas de compatibilidad empleando las últimas versiones de XSLT y XPath.

Para finalizar la sección, también dispones de un video resumen a continuación:



Grupo Tr3s | Link: https://www.youtube.com/embed/aAeAk-iapec

2.e.ii Ejemplos de transformaciones XSLT

Documento XML original

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Primero declaramos versión y codificación del doc. XML

<?xml-stylesheet type="text/xsl" href"direcciónDeLaHojaXSLT" ?>

Opcionalmente vinculamos el documento XSLT que emplearemos para la transformación, aunque hay métodos más avanzados cómo utilizar PHP para que un mismo documento XML puede tener varias hojas XSLT y por tanto diferentes transformaciones.
-->

<personas>
<estudiante>
<apellido>Gil</apellido>
<nombre>Maria Isabel</nombre>
<desarrolla>¿Qué son los lenguajes de marcas?</desarrolla>
</estudiante>
<estudiante>
<apellido>Cervantes</apellido>
<nombre>Jordi</nombre>
<desarrolla>Definición de XML</desarrolla>
</estudiante>
<estudiante>
<apellido>Iglesias</apellido>
<nombre>Ramón</nombre>
<desarrolla>¿Qué es el lenguaje de transformación XSLT?</desarrolla>
```

</estudiante>

</personas>

Documento XSLT para crear un nuevo documento HTML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Declaramos versión y codificación del documento XML,
aunque sea un documento XSLT sigue siendo un documento XML-->

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!--
Declaramos versión XSL que utilizaremos y el espacio de nombres
"xsl". Todas los elementos precedidos por "xsl:" se tratarán
como template rules definidas en http://www.w3.org/1999/XSL/Transform
-->
<xsl:template match="/">
<!--
Utilizando xsl:template match="ExpresiónXPath" permite seleccionar
qué grupo de elementos trataremos del XML original
-->
<html>
<head>
<title>Reparto del desarrollo</title>
</head>
<body>
<h2>Responsable sección y puntos a desarrollar</h2>
<table border="2" style="text-align:center" width="100%">
<tr style="color:white" bgcolor="#000FF">
```

<th>Encargado</th>

<th>Tema</th>

</tr>

<xsl:for-each select="personas/estudiante">

<!-- template rule tipo bucle "para".

Para cada elemento seleccionado por la expresión XPath

contenida en el "select" se aplicará lo contenido por esta

etiqueta hasta el cierre de la misma-->

<tr>

<td>

<xsl:value-of select="apellido"/>

<!-- Introduce el valor del elemento "apellido"-->

<xsl:text>, </xsl:text>

<!-- Introduce un valor textual ", "-->

<xsl:value-of select="nombre"/>

<!-- Introduce el valor del elemento "nombre"-->

</td>

<td>

<xsl:value-of select="desarrolla"/>

<!-- Introduce el valor del elemento "desarrolla"-->

</td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

<!-- Todas las etiquetas "xsl:" también deben cerrarse -->

Documento HTML resultante

```
<html>
<head>
<title>Reparto del desarrollo</title>
</head>
<body>
<h2>Responsable sección y puntos a desarrollar</h2>
<table border="2" style="text-align:center" width="100%">
<tr style="color:white" bgcolor="#000FF">
<th>Encargado</th>
<th>Tema</th>
</tr>
<tr>
<td>Gil, Maria Isabel</td>
<td>¿Qué son los lenguajes de marcas?</td>
</tr>
<tr>
<td>Cervantes, Jordi</td>
<td>Definición de XML</td>
</tr>
<tr>
<td>Iglesias, Ramón</td>
<td>¿Qué es el lenguaje de transformación XSLT?</td>
</tr>
</table>
```

</body>

</html>

Vista del documento HTML en navegador Firefox 64.0.2 (64-bit)

2.e.iii Principales instrucciones XSLT.

Por un lado tendríamos las template rules. Algunas de las cuales son:

<xsl:template match="expressionXPath">

“template” Se utiliza para asociar los elementos de un documento XML que trataremos en el XSLT.

<xsl:value-of select="expressionXPath"/>

“value-of” Se utiliza para recoger el valor de un elemento concreto del grupo que estamos tratando en el XSLT.

<xsl:for-each select="expressionXPath">

“for-each” Se utiliza como un bucle “para”. Recogerá un grupo de elementos y hará lo contenido por la etiqueta a cada uno de ellos.

<xsl:sort select="expressionXPath"/>

“sort” Se utiliza dentro de la template rule “for-each”. Aplicará un orden de presentación a los elementos seleccionados.

<xsl:if test="expressionXPath">

“if” Es un condicional “si se cumple expresión entonces hacer tal...” “test” debe contener la expresión XPath que se evaluará dando un booleano true o false.

<xsl:choose>

<xsl:when test="expressionXPath">

</xsl:when>

<xsl:otherwise>

</xsl:otherwise>

</xsl:choose>

“choose”, “when” y “otherwise” siempre se utilizan conjuntamente.

Se utilizan para expresar múltiples condicionales.

“when” puede concatenarse con sucesivos “when” para evaluar diferentes expresiones.

Las expresiones XPath que se evalúen deben devolver un booleano true o false.

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

Indica el lugar donde se aplicarán las template rules.

Por otro lado las expresiones XPath. Algunas de las cuales son:

nombreDelNodo

Seleccionaríamos los elementos contenidos por el nodo indicado.

nodoPadre/nodoHijo

Seleccionaríamos los elementos contenidos dentro del nodo hijo indicado.

//nombreNodo

Seleccionaríamos todos los elementos con ese nombre, sin importar donde estén.

//@NombreAtributo

Seleccionaríamos los elementos que tengan un atributo con el nombre indicado, sin importar donde estén.

/nodoPadre/nodoHijo[N]

Selecciona dentro del nodoHijo el elemento número N.

/nodoPadre/nodoHijo[last()-N]

Selecciona dentro del nodoHijo el elemento número N pero empezando desde el último elemento.

/nodoPadre[nodoHijo>N]

Selecciona dentro del nodoPadre todos los nodoHijo que cumplan que su valor es mayor que N. Devuelve true si se cumple.

//nodoTal | //nodoCual

Selecciona todos los elementos nodoTal Y nodoCual.

nodoTal=N or nodoTal=M

Selecciona los elementos donde nodoTal sea igual a N o M. Devuelve true si se cumple.

2.e.iv Herramientas de transformación con XSLT

Para crear o editar documentos XSLT, al igual que con XML, no hace falta ningún software específico, se puede hacer con un editor de texto plano como el que incorpora Windows con su Bloc de Notas o con programas más avanzados como Notepad++. Aun así, no sería la opción más recomendable, ya que no tendríamos las sugerencias que nos pueda dar la DTD vinculada ni un depurador que nos diese información sobre posibles fallos y línea concreta donde estuviese fallando. Además, en el caso concreto de XSLT dependeremos necesariamente de un programa que se encargue de procesar la hoja XSLT, y aunque podríamos usar un navegador como Chrome, Firefox, Edge… seguiría siendo un sistema de trabajo demasiado limitado.

También existen programas específicos para trabajar con XML y XSLT. Un par de opciones serían eXcelon Stylus (licencia comercial) y XEmacs (licencia GNU) que cuentan con buenos depuradores y ambos permiten trabajar con XSLT/XPath v1.0 y XSLT/XPath v2.0.

Para los que prefieren no recurrir a pequeñas herramientas concretas, XSLT también es una opción que está presente en IDEs como Android Studio o Visual Studio .NET de Microsoft. De hecho Visual Studio ha sido la opción utilizada para preparar ejemplos del presente trabajo:

En este ejemplo podemos ver como Visual Studio permite trabajar con un documento XML, vincular una hoja XSLT (no de forma explícita dentro del XML, lo cual es muy importante si a un documento XML vas a vincular varias hojas XSLT) y ver la salida que genera.

Si nuestro objetivo pasa por migrar una base de datos, con las herramientas presentadas hasta aquí ya tendríamos suficientes recursos para llevar a cabo el trabajo. Pero si nuestro objetivo fuera publicar una web, donde tenemos el contenido en un documento XML (o varios) y varias hojas de estilo XSLT preparadas, si no queremos que la transformación la haga el cliente que se conecte a la web, sino el servidor, aún faltaría una última herramienta: el procesador XSLT del servidor web. Por ejemplo, en el caso de NGinx tendremos que habilitar su módulo XSLT en el servidor.

2.f. Ejemplos de lenguajes basados en XML

2.f.i xHTML

2.f.i.1 ¿Que és xHTML?

Definir qué es XHTML exactamente es algo complicado.

Antes de nada deberíamos saber que el XHTML (eXtensible HyperText Markup Languaje) es una reformulación de HTML en la síntesis XML. A su vez, es compatible con HTML.

Cuando se estandarizaron por primera vez XML y XHTML, ningún navegador los admitia. Así pues, para habilitar el uso al menos parcial de XHTML, el W3C (World Wide Web Consortium) ideó algo llamado “XHTML compatible con HTML”. Consistía en unas pautas para realizar documentos XHTML válidos, que podían ser validados casi como HTML.

El XHTML es un lenguaje más robusto y aconsejable para la modelación de páginas web. Es un lenguaje que trata de recuperar la línea marcada por los estándares y, a su vez, prepararse para adaptarse a las nuevas necesidades y corrientes tecnológicas. En realidad el XHTML incorpora una nueva concepción, una nueva filosofía de modelación de las páginas web. Busca la creación de una web semántica.

Así pues, podríamos definir ahora el XHTML como una versión más estricta y limpia de HTML, que nace con el objetivo de reemplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML. (Definición de W3C)

2.f.i.2 Utilidades

Después de definir XHTML podemos afirmar que su objetivo es ser usado para la creación de páginas web. Su estandarización hace que sea un lenguaje más sencillo para la creación de éstas.

Pero no sólo es exclusivo para la creación de páginas web, también permite actuar sobre la estructura, imágenes, formularios, tablas básicas y soporte de objetos.

Con la continua evolución y avance de la tecnología, el lenguaje XHTML no se ha quedado atrás, sino que además ha evolucionado y ha permitido el diseño para teléfonos móviles, PDAs, o incluso Televisiones interactivas. Este lenguaje se ha llamado XHTML Basic.

Es importante destacar que, gracias a ello, las páginas creadas en XHTML Basic pueden ser renderizadas de manera diferente en navegadores web y en dispositivos móviles sin la necesidad de tener dos versiones diferentes de la misma página

2.f.ii RSS / Atom

2.f.ii.1 ¿Qué es RSS o Atom?

RSS (Really Simple Syndication) es un formato XML que gestiona y distribuye información y noticias web.

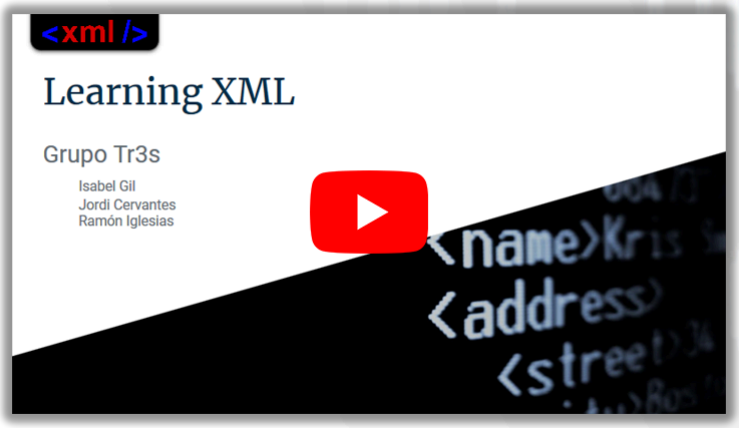
Esta herramienta nos permite recibir en nuestro ordenador o en una página web determinada información actualizada de nuestras páginas favoritas, sin que para ello tengamos que acceder a cada una.

Para ello necesitamos un lector RSS (programa que permiten a una persona darse de alta en las RSS de una determinada web) y que la página de la que queremos recibir dicho contenido disponga del servicio RSS.

La practicidad del RSS reside en que el usuario no necesita entrar a cada página seleccionada para mantenerse informado, sino que recibirá los artículos directamente, ahorrando tiempo, ganando un contenido que de otra forma seguramente no llegaría a consultar y teniendo las noticias aunadas en un mismo lugar.

Por otro lado Atom es un sublenguaje XML. No se trata de una versión de RSS, pero es un formato muy similar a éste con el que comparte un mismo objetivo: permitir la distribución de contenido y noticias web. Las mejoras que trataron de ofrecer frente a RSS en su creación fue que lograra que se extendiese más rápidamente y que pudiese contener mayor información y más compleja y consistente que un documento RSS.

A continuación adjuntamos un vídeo explicativo:



Grupo Tr3s | Link: https://www.youtube.com/embed/PiH_KBMVCHQ

2.f.ii.2 Utilidades

Como explicamos anteriormente las RSS facilitan el acceso a la información web actualizada con regularidad y permiten recuperar las noticias que sean de nuestro interés. Pero además otra ventaja es que están libres de SPAM, pues no requieren de un correo electrónico con el que registrarnos y la información que se recibe además de estar actualizada es totalmente gratuita.

Por otro lado Atom prácticamente nos ofrece lo mismo que RSS pero de forma mejorada como hemos explicado en el apartado anterior. No obstante, son muchos los usuarios que consideran que no es necesario ofrecer varios tipos de formato y siguen utilizando RSS, de hecho muchas páginas o blogs ofrecen un mismo canal para diferentes entradas independientemente del formato, pues les resulta más cómodo.

2.f.iii Otros lenguajes basados en XML

El W3C cuando definió XML lo concibió como una plataforma de referencia para la creación de lenguajes de marcado que pudieran resolver necesidades específicas. Es decir, XML es un metalenguaje que permite crear lenguajes basados en XML.

Teniendo eso en cuenta, podemos imaginar fácilmente que la familia de lenguajes basados en XML es muy grande.

No solo se compondría por extensiones del XML como XSL, SVG, XHTML… que están estandarizados por el W3C, sino por otros lenguajes conocidos como “Non W3C Grammars” (dialectos XML) desarrollados por terceros para resolver necesidades concretas.

Aquí podemos consultar un listado con varias de las extensiones XML más notables.

Destacando alguna podríamos señalar:

DocBook: Utilizando su propia DTD nos permite crear documentos en un formato neutro, independiente de la presentación. Este formato neutro recoge tanto el contenido como la estructura lógica, permitiendo ser publicado en varios formatos: HMTL, XHTML, EPub, PDF, hasta archivos “man” para documentar funcionalidades en la CLI de sistemas Linux.

CML: Acrónimo de Chemical Markup Languaje, permite expresar información molecular como; estructuras químicas de moléculas, reacciones químicas, datos de espectros y analíticos...

KML: Acrónimo de Keyhole Markup Language, permite mostrar información geográfica en aplicaciones como Google Earth. De hecho, fué adquirido por Google en 2004.

OpenDocument: Se trata de un estándar desarrollado por Sun Microsystems (desde el 2009 parte de Oracle) para el almacenamiento de hojas de cálculo, textos, gráficos y presentaciones en Open Office.

OpenXML: Estándar equivalente a OpenDocument, pero desarrollado por Microsoft para su suite ofimática.