I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)

Índice

Contenidos oficiales del módulo.-

a) Reconocimiento de las características de los lenguajes de marcas:

Concepto de lenguaje de marcas.
Características comunes.
Identificación de ámbitos de aplicación.
Clasificación.
XML: estructura y sintaxis.
Etiquetas.
Herramientas de edición.
Elaboración de documentos XML bien formados.
Utilización de espacios de nombres en XML.

b) Utilización de lenguajes de marcas en entornos web:

HTML: estructura de una página web.
Identificación de etiquetas y atributos de HTML.
XHTML: diferencias sintácticas y estructurales con HTML.
Ventajas de XHTML sobre HTML.
Versiones de HTML y de XHTML.
Herramientas de diseño web.
Transmisión de información mediante lenguajes de marcas.
Hojas de estilo.

c) Aplicación de los lenguajes de marcas a la sindicación de contenidos:

Características de la sindicación de contenidos.
Ventajas.
Ámbitos de aplicación.
Estructura de los canales de contenidos.
Tecnologías de creación de canales de contenidos.
Validación.
Utilización de herramientas.
Directorios de canales de contenidos.
Agregación.

d) Definición de esquemas y vocabularios en XML:

Definición de la estructura de documentos XML.
Definición de la sintaxis de documentos XML.
Utilización de métodos de definición de documentos XML.
Creación de descripciones.
Asociación con documentos XML.
Validación.
Herramientas de creación y validación.
Documentación de especificaciones.

e) Conversión y adaptación de documentos XML:

Técnicas de transformación de documentos XML.
Formatos de salida.
Ámbitos de aplicación.
Descripción de la estructura y de la sintaxis.
Utilización de plantillas.
Utilización de herramientas de procesamiento.
Verificación del resultado.
Depuración.
Elaboración de documentación.

f) Almacenamiento de información:

Utilización de XML para almacenamiento de información.
Ámbitos de aplicación.
Sistemas de almacenamiento de información.
Inserción y extracción de información en XML.
Técnicas de búsqueda de información en documentos XML.
Manipulación de información en formato XML.
Lenguajes de consulta y manipulación.
Almacenamiento XML nativo.
Herramientas de tratamiento y almacenamiento de información en formato XML.

g) Sistemas de gestión empresarial:

Instalación.
Identificación de flujos de información.
Adaptación y configuración.
Integración de módulos.
Elaboración de informes.
Planificación de la seguridad.
Implantación y verificación de la seguridad.
Integración con aplicaciones ofimáticas.
Exportación de información.

TEMA 1.- Los lenguajes de marcas.

1.- Lenguajes de marcas.Historia.

Los lenguajes de programación son aquellos lenguajes artificiales que permiten escribir código para resolver problemas de una manera intuitiva. Ejemplos de lenguajes de programación de alto nivel son: Java, C, C++, Pascal, Delphi, ASP, Visual C, etc.

Estos lenguajes de programación tienen estructuras de código más o menos complejas como son los bucles (do-while, for, repeat) o los saltos condicionales (if, switch).

Frente a los lenguajes de programación surgieron otros lenguajes sencillos cuyo objetivo inicial era simplemente darle formato a un texto, mediante un conjunto de marcas de formato. Éstos últimos se llaman **lenguajes de marcas**, y realmente no podemos considerarlos lenguajes de programación puesto que no tienen la complejidad de éstos, sin embargo han ido evolucionando para ofrecer más prestaciones y se han convertido en la base para la creación de las páginas WEB.

Recuerda:

- Un lenguaje de marcas es una forma de codificar un documento que, junto con el texto, incorpora **etiquetas o marcas** que contienen información adicional acerca de la estructura del texto o su presentación.

En los años 60, IBM intentó resolver sus problemas asociados al tratamiento de documentos en diferentes plataformas a través de un lenguaje de marcas denominado **GML** (*Generalized markup Language* o Lenguaje de marcas generalizado).

GML libera al creador del documento de preocupaciones específicas de formato como pueden ser el tipo de letra, la alineación de los párrafos, el espaciado entre líneas, el uso de tablas, etc. GML fue una manera de estandarizar estos formatos de forma que un mismo documento se pudiera enviar a una impresora, mostrar en diferentes pantallas, etc.

Más tarde GML pasó a manos de ISO y se convirtió en **SGML** (ISO 8879), *Standart Generalized Markup Language*. Esta norma es la que se aplica desde entonces todos los lenguajes de marcas, cuyos ejemplos más conocidos son el HTML y el RTF.

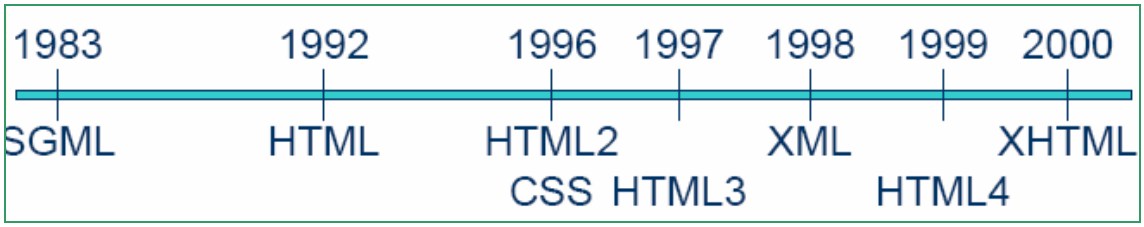
Conviene repetir que los lenguajes de marcas no son equivalentes a los lenguajes de programación aunque se definan igualmente como "lenguajes". Son sistemas de descripción de información, normalmente documentos, que si se ajustan a SGML, se pueden controlar desde cualquier editor de texto ASCII.

Las **marcas o etiquetas** más utilizadas suelen describirse por textos descriptivos encerrados entre signos de "menor" (<) y "mayor" (>), siendo lo más usual que existan una marca de principio y otra de final.

Ejemplo de la estructura básica de un documento HTML (página web):

```
<HTML>
  <HEAD>
     <TITLE>Mi primera página</TITLE>
  </HEAD>
  <BODY>
     Esta es la primera página web
  </BODY>
</HTML>
```

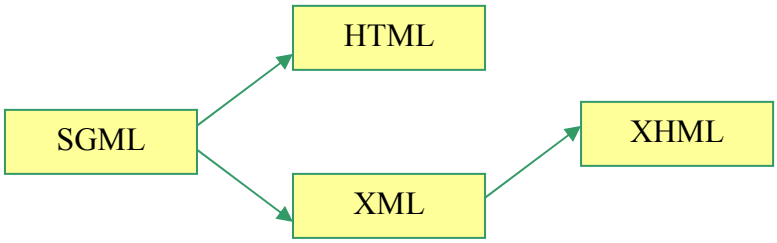
La evolución histórica de los lenguajes de marcas más utilizados es la siguiente:



En esta módulo iremos conociendo estos lenguajes en el siguiente orden: primero analizaremos el HTML, lenguaje que todavía en la actualidad es la base de escritura de casi todas las páginas WEB. Luego veremos el XHTML que es una evolución del HTML que incorpora las ventajas de XML. Y por último nos centraremos

en el XML y todo el abanico de posibilidades asociadas a él, que están revolucionando el intercambio de datos entre aplicaciones, y que es el futuro inmediato de Internet.

No obstante no podemos perder de vista que históricamente la evolución de estos lenguajes sigue el siguiente diagrama:



2.- Otros ejemplos de lenguajes de marcas.

SGML (Standard Generalized Markup Language)

Es un ejemplo de lenguaje genérico que apareció con el identificador 8879 como norma ISO (International Organization for Standardization). Lo creó la comunidad de editores e imprentas, ya que consideraban que era de vital importancia contar con una manera normalizada de transmitir los documentos en un formato adecuado para los procesos de edición e impresión.

SGML es apropiado para describir texto altamente estructurado, aunque también se pueden incluir en los documentos otros elementos, como por ejemplo diagramas y gráficos, independientemente de su formato de codificación.

SGML contiene las reglas para crear una infinita variedad de lenguajes de marcado, pero no describe el formato de los documentos marcados. Esto significa que SGML sólo aporta reglas para definir nuestros propios lenguajes de marcas, es decir, es un **metalenguaje**. Esto hace posible que, mediante la utilización de una definición de tipo de documento (denominada DTD Document Type Definition), se pueda especificar la estructura lógica de una clase de escrito.

Resumiendo SGML no es un lenguaje en sí, sino una manera de crear otros lenguajes. La dificultad de su uso hizo que dejara de utilizarse, pero es el padre de otros lenguajes que utilizaremos nosotros como HTML y XML.

HyTime (Hypermedia/Time-Based Structuring Language)

HyTime es una extensión de SGML que especifica un conjunto de conceptos básicos con los que se puede definir la estructura lógica de documentos de hipertexto y multimedia.

HyTime normaliza aquellos mecanismos que se refieren a la localización de porciones de documentos hipermedia y sus componentes multimedia de información, incluyendo enlaces, alineamiento en el espacio y sincronización en el tiempo, es decir,

proporciona una manera homogénea de enlazar a un documento cualquier tipo de elemento, en cualquier parte y en cualquier instante.

Sirve como base de intercambio de informaciones hipermedia, independientemente de la aplicación que las haya creado, y se ocupa de normalizar la estructura del documento y la identificación de los objetos de información que lo conforman.

Debido a la dificultad en su uso, este lenguaje no ha sido utilizado de una forma masiva, aunque muchas de sus ideas se encuentran en las normas que en la actualidad se están imponiendo.

SMIL (Synchronized Multimedia Integration Language)

SMIL es un lenguaje basado en XML para la definición de aplicaciones multimedia interactivas, de manera que un autor puede describir el comportamiento temporal de presentaciones multimedia, asociar enlaces a contenidos de cualquier tipo (por ejemplo, vídeos, sonidos, programas, etc.) y describir la presentación en la pantalla.

SMIL (que se pronuncia smile –sonrisa–) no es una solución que intente competir con tecnologías existentes de representación multimedia (por ejemplo, Quicktime o Flash) sino que lo que pretende es integrar esas tecnologías de manera estándar para que puedan combinarse.

Existen numerosos visualizadores de documentos SMIL que permiten mostrar todas sus características. Algunos de los más conocidos son: RealPlayer, Quicktime Player (la versión gratuita sólo permite ver documentos muy sencillos) y GRINS Player.

Además, algunos navegadores Web, como Internet Explorer, permiten visualizar también estos documentos. Un documento SMIL se compone de una cabecera (HEAD) y un cuerpo (BODY). En la cabecera se definen tanto la metainformación del documento como información relativa a como deben aparecer en la pantalla los elementos de información. En el cuerpo del documento se incluirán los contenidos del documento así como las relaciones entre ellos. Con este lenguaje puedes "partir" la ventana en varias partes y proyectar en cada sección un elemento diferente.

WML (Wireless Markup Language)

Con el auge de las páginas web, pronto de comprendió que era necesario contar con un mecanismo que las hiciera visibles en los teléfonos móviles. Para ello fue necesario trabajar en un protocolo de comunicaciones adaptado a estos dispositivos. Fue así como surgió el protocolo WAP (Wireless Application Protocol) que permite el desarrollo de aplicaciones sobre dispositivos móviles a través de redes inalámbricas.

Una vez desarrollado este protocolo fue necesario crear un lenguaje de marcas adaptado. WML y WMLScript son los equivalentes dentro del mundo "inalámbrico" al HTML y al JavaScript dentro de las redes que usan el protocolo TCP/IP como hace Internet.

Los documentos WML pueden mostrarse en teléfonos móviles, pero también en cualquier otro dispositivo que contenga un micronavegador (es decir, un dispositivo que sepa interpretar WML y WMLScript), como puede ser una agenda personal (PDA) o una aplicación en nuestro ordenador personal.

El lenguaje WML es un lenguaje de descripción de páginas que permite definir la presentación de la información en el teléfono móvil, solicitar entradas del usuario y responder a ciertas interacciones del usuario con el móvil, como puede ser la pulsación de una tecla. Este lenguaje se basa en una DTD de XML por lo que todo documento WML es a su vez un documento XML.

3.- Características de los lenguajes de marcas.

1.- Uso de texto plano.
2.- Compactos.
3.- Fáciles de procesar.
4.- Flexibles.

1.- Uso del texto plano.

Los archivos de texto plano son aquellos que están compuestos únicamente por texto sin formato, sólo caracteres. Estos caracteres se pueden codificar de distintos modos dependiendo de la lengua usada. Algunos de los sistemas de codificación de caracteres más usados son: ASCII, ISO-8859-1, Latín-1, Unicode, etc...

Los lenguajes de marcas se escriben en archivos de texto plano, y como principal ventaja es que estos archivos pueden ser interpretados directamente. Esto es una ventaja evidente respecto a los sistemas de archivos binarios, que requieren siempre de un programa intermediario para trabajar con ellos que lo interprete.

Un documento escrito con lenguajes de marcado puede ser editado por un usuario con un sencillo editor de textos, sin perjuicio de que se puedan utilizar programas más sofisticados que faciliten el trabajo.

Al tratarse solamente de texto, los documentos son independientes de la plataforma, sistema operativo o programa con el que fueron creados. Esta fue una de las premisas de los creadores de GML en los años 60, para no añadir restricciones innecesarias al intercambio de información. Es una de las razones fundamentales de la gran aceptación que han tenido en el pasado y del excelente futuro que se les augura.

Nota: Microsoft Word es un editor de textos, pero no de texto plano, en principio guarda sus archivos con un formato propio. Para generar archivos de texto plano la opción más sencilla es usar el Notepad o Bloc de notas en entornos Windows (está en el menú accesorios).

2.- Compactos.

Las etiquetas o marcas se entremezclan con el propio contenido en un único archivo. Veamos un ejemplo en diferentes lenguajes de marcas:

Ejemplos	HTML	LaTeX	Wikitexto
Título	Título	\section{Título}	== Título ==
Lista	 - Punto 1 - Punto 2 - Punto 3 	\begin{itemize} \item Punto 1 \item Punto 2 \item Punto 3 \end{itemize}	* Punto 1 * Punto 2 * Punto 3
texto en **negrita**	texto	\bf{texto}	''' texto '''
texto en *cursiva*	texto	\it{texto}	'' texto ''

En estos ejemplos vemos cómo diferentes lenguajes de marcas usan sus propios conjuntos de marcas. Por ejemplo si queremos que una parte de un texto aparezca en **negrita** y estamos en HTML tendremos que marcarlo así:

Texto en negrita

Es decir, en HTML la marca es para la negrita.

Si por el contrario estamos usando LaTeX tendremos que poner la marca \bf, y en Wikitexto encerramos el texto en tres comillas simples.

Éste es el concepto de **marca o etiqueta**. Afectan a la presentación visual (negrita, cursiva, etc), o a la estructuración del texto (listas, tablas, etc).

3.- Fáciles de procesar:

Las organizaciones de estándares han venido desarrollando lenguajes especializados para los tipos de documentos de comunidades o industrias concretas. Uno de los primeros fue el CALS, utilizado por las fuerzas armadas de EE.UU. para sus manuales técnicos. Otras industrias con necesidad de gran cantidad de documentación, como las de aeronáutica, telecomunicaciones, automoción o hardware, ha elaborado lenguajes adaptados a sus necesidades.

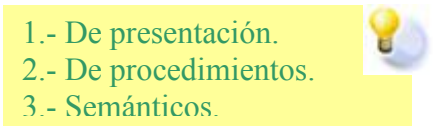
Esto ha conducido a que sus manuales se editen únicamente en versión electrónica, y después se obtenga a partir de ésta las versiones impresas, en línea o en CD. Un ejemplo notable fue el caso de Sun Microsystems, empresa que optó por escribir la documentación de sus productos en SGML, ahorrando costes considerables. El responsable de aquella decisión fue Jon Bosak, que más tarde fundaría el comité del XML.

4.- Flexibles.

Aunque originalmente los lenguajes de marcas se idearon para documentos de texto, se han empezado a utilizar en áreas como gráficos vectoriales, servicios web, sindicación web o interfaces de usuario. Estas nuevas aplicaciones aprovechan la sencillez y potencia del lenguaje XML. Esto ha permitido que se pueda combinar varios lenguajes de marcas diferentes en un único archivo, como en el caso de XHTML+SMIL o de XHTML+MathML+SVG.

4.- Clasificación de los lenguajes de marcas.

4.1.- Según la naturaleza de las marcas:



1.- Lenguajes de presentación:

Son aquellos que están especialmente diseñados para indicar formatos del texto, es decir, la forma o apariencia que adquirirá el texto (la presentación). Por lo tanto sirven para ver los textos adecuadamente pero no resultan de utilidad para procesar de forma automática la información que contienen.

El marcado de presentación resulta más fácil de elaborar, sobre todo para cantidades pequeñas de información. Sin embargo resulta complicado de mantener o modificar, por lo que su uso se ha ido reduciendo en proyectos grandes en favor de otros tipos de marcado más estructurados.

Se puede tratar de averiguar la estructura de un documento de esta clase buscando pistas en el texto. Por ejemplo, el título puede ir precedido de varios saltos de línea, y estar ubicado centrado en la página. Varios programas pueden deducir la estructura del texto basándose en esta clase de datos, aunque el resultado suele ser bastante imperfecto, por lo que no se utilizan para analizar información, sólo para presentarla.

Ejemplos: Rich Text Format (RTF), S 1000D, TeX, troff, HTML…

2.- Lenguajes de procedimientos:

Aunque también suelen incoporar marcas para la presentación su objetivo general es indicar los procedimientos que deberá realizar el software de representación. El marcado de procedimientos está enfocado hacia la presentación del texto, sin embargo, también es visible para el usuario que edita el texto. El programa que representa el documento debe interpretar el código en el mismo orden en que aparece.

Por ejemplo, para formatear un título, debe haber una serie de directivas inmediatamente antes del texto en cuestión, indicándole al software instrucciones tales como centrar, aumentar el tamaño de la fuente, o cambiar a negrita. Inmediatamente después del título deberá haber etiquetas inversas que reviertan estos efectos.

Ejemplos: nroff, troff, TeX.

Este tipo de marcado se ha usado extensamente en aplicaciones de edición profesional, manipulados por tipógrafos calificados, ya que puede llegar a ser extremadamente complejo.

3.- Lenguajes semánticos:

No se encargan de la presentación, sino de la estructura de la información almacenada. Es decir, describen las diferentes partes en las que se estructura el documento pero sin especificar cómo deben representarse.

El marcado descriptivo o semántico utiliza etiquetas para describir los fragmentos de texto, pero sin especificar cómo deben ser representados, o en qué orden.

Los lenguajes expresamente diseñados para generar marcado descriptivo son el SGML y el XML.

Las etiquetas pueden utilizarse para añadir al contenido cualquier clase de metadatos. Por ejemplo, el estándar Atom, un lenguaje de gestión, proporciona un método para marcar la hora "actualizada", que es el dato facilitado por el editor de cuándo ha sido modificada por última vez cierta información. El estándar no especifica cómo se debe representar, o siquiera si se debe representar. El software puede emplear este dato de múltiples maneras, incluyendo algunas no previstas por los diseñadores del estándar.

Una de las virtudes del marcado descriptivo es su flexibilidad: los fragmentos de texto se etiquetan tal como son, y no tal como deben aparecer. Estos fragmentos pueden utilizarse para más usos de los previstos inicialmente. Por ejemplo, los hiperenlaces fueron diseñados en un principio para que un usuario que lee el texto los pulse. Sin embargo, los buscadores los emplean para localizar nuevas páginas con información relacionada, o para evaluar la popularidad de determinado sitio web.

El marcado descriptivo también simplifica la tarea de reformatear un texto, debido a que la información del formato está separada del propio contenido. Por ejemplo, un fragmento indicado como cursiva (texto), puede emplearse para marcar énfasis o bien para señalar palabras en otro idioma. Esta ambigüedad, presente en el marcado de presentación y en el procedimental, no puede soslayarse más que con una tediosa revisión a mano. Sin embargo, si ambos casos se hubieran diferenciado descriptivamente con etiquetas distintas, podrían representarse de manera diferente sin esfuerzo.

El marcado descriptivo está evolucionando hacia el marcado genérico. Los nuevos sistemas de marcado descriptivo estructuran los documentos en árbol, con la posibilidad de añadir referencias cruzadas. Esto permite tratarlos como bases de datos, en las que el propio almacenamiento tiene en cuenta la estructura.

Ejemplos: XML, SGML, ASN.1, EBML, YAML.

Ejemplo de filosofía de XML:

```
<persona>
    <nombre>Javier</nombre>
    <apellido>González</apellido>
```

```
    <apellido>Márquez</apellido>
    <edad>34</edad>
  </persona>
```

Podemos ver cómo se estructura una persona, pero no tenemos información de cómo mostrar sus datos en la pantalla.

4.2.- Según el uso que se da al lenguaje:

1.- Elaboración de documentos electrónicos:
- RTF
- TeX
- Wikitexto
- DocBook

2.- Uso en Internet (páginas web, canales de noticias, etc):
- HTML, XHTML
- RDF
- RSS

3.- Especializados en ámbitos:
- MathML
- VoiceXML
- SVG
- MusicXML

5.- VRML: un ejemplo específico para la creación 3D.

Antes de entrar en el contenido específico de este módulo y dado que estamos terminando este primer tema de carácter teórico quería presentarles un lenguaje de marcas muy especializado. Se trata del VRML, un lenguaje diseñado para la creación de mundos virtuales en Internet.

Los archivos, que recuerda que son de texto plano, tienen que tener la extensión .vrml (VRML significa Lenguaje de Marcado de Realidad Virtual). Por otra parte para que estos archivos puedan verse en un navegador debemos haber instalado en él un intérprete de VRML. Te adjunto uno gratutito: el cliente VRML de CORTONA.

Explicar este lenguaje es una tarea que lleva bastante tiempo y está fuera del alcance de esta materia, pero no obstante vamos a hacer un ejemplo muy sencillo, crear formas simples.

El fichero de VRML debe comenzar siempre con la cabecera, y después irá continuado con las marcas de los **nodos**. En este tema colocamos sólo los nodos más básicos del lenguaje, que permite desde dibujar objetos simples hasta colocar cámaras, iluminación, diseñar espacios 3D, crear avatares, etc.

Ejemplos de sintaxis del lenguaje:

* **Cabecera**:
#VRML V2.0 utf8
después de la cabecera pueden aparecer: nodes, prototipos, routes.

* **Modelo de colores**: es RGB, y cada valor un número entre 0 y 1.
0.0 0.0 0.0 = Negro 0.0 0.0 1.0 = Azul 1.0 1.0 1.0 = Blanco

* **Unidades**: las unidades de distancia se asume que son metros y los ángulos radianes, aunque esto no es más que una convención.

* Nodo **Shape**: en los nodos shape (forma) están los objetos visibles de VRML. Este nodo tiene dos campos, appearance (apariencia) y geometry (geometría). En el campo de apariencia se definen aspectos como el color, la textura, etc, que será aplicada a la geometría. Y la geometría indica qué forma será dibujada.

```
Sintaxis:   Shape {
                appearance NULL
                geometry NULL
                }
```

El campo apariencia es opcional, y puede no aparecer. El campo geometría puede ser uno de los nodos siguientes:

- Box
- Cone
- Cylinder
- ElevationGrid
- Extrusion
- IndexedFaceSet
- IndexedLineSet
- PointSet
- Sphere
- Text

Veremos primero los cuatro que son primitivas, esto es, formas simples: Box, Cone, Cylinder, Sphere. El resto son formas geométricas que aglutinan conjuntos de puntos, caras, etc.

* **Box**: define un paralelepípedo rectangular donde especificamos la anchura, la altura y la profundidad de la caja. Estos datos van en el único campo opcional que es size.

```
Box {
        size 2.0 2.0 2.0
        }
```

El centro de la caja está en (0,0,0) del sistema de coordenadas local.

* **Sphere**: define una esfera. El campo radius es opcional.

```
Sphere {
        radius 1.0
        }
```

* **Cone**: esta primitiva contiene cuatro campos. BottomRadius y Height definen las propiedades geométricas del cono, los valores de estos parámetros deben ser mayores que cero para que el cono sea visible. Los campos side y bottom especifican qué partes del cono se van a mostrar. Si side es False no se muestra la cara y sólo aparecerá un círculo que es la base del cono. Los cuatro parámetros son opcionales.

Sintaxis: Cone {

```
                bottomRadius 1.0
                height 2.0
                side TRUE
                bottom TRUE
                }
```

* **Cylinder**: un cilindro donde indicamos el radio, la altura, si mostramos la cara, la tapa superior y la tapa inferior.

Sintaxis: Cylinder {

```
                Radius 1.0
                Height 2.0
                Side TRUE
                Bottom TRUE
                Top TRUE
                }
```

* **PointSet**: especifica un conjunto de puntos en 3D, dentro de nuestro sistema local de coordenadas, con sus colores asociados.

Sintaxis: PointSet {

```
                Color NULL
                Coord NULL
                }
```

Ejemplo:

```
#VRML V2.0 utf8
Shape {
  appearance    Appearance {
    material        Material {    }
    texture         NULL
    textureTransform        NULL
  }
  geometry
    PointSet {
      coord         Coordinate {
                      point  [ -0.5 0.5 0,
                               -0.5 -0.5 0,
                                0.5 -0.5 0,
                                0.5 0.5 0,
                                0 0 1 ]
                    }
      color Color { color [ 1.0 1.0 1.0,
                            1.0 1.0 0,
                            1.0 1.0 0,
                            1.0 1.0 1.0
                            1.0 0.0 1.0] }
          }
}
```

* **IndexedLineSet**: especifica un conjunto de poli-líneas en el sistema local de coordenadas. Tiene 5 campos.

El primero es *coord* que es un campo para coordenadas (conjuntos de coordenadas 3D).

El segundo campo es *coordIndex*, que especifica una lista de índices de coordenadas definiendo las líneas a dibujar. Para separar los índices de una línea se usa un espacio en blanco, para separar conjuntos de índices de dos líneas adyacentes se usa el marcador -1. (un -1 indica que la línea actual ha terminado y empieza otra línea).

El tercer campo es *color*. Es una lista de colores para las líneas. Si se ha especificado Material el color por defecto es el color emisivo de este nodo. Tanto emisivo como background son por defecto negro, luego si no especificamos un color o cambiamos el fondo no podremos ver las líneas.

El cuarto campo es *colorIndex*, que sirve al mismo propósito que coorindex pero para los colores.

El quinto campo es *colorPerVertex*, es un campo booleano que indica cómo se aplican los colores. Si es TRUE, los colores se aplican a cada vértice, de forma que las líneas empiezan en un color (del primer vértice) y terminan en el segundo (del último vértice), aplicándose un gradiente. OJO: en algunos navegadores no se aplica este gradiente sino se calcula para la línea el color medio entre los colores de los vértices. Si es FALSE, los colores se aplican a cada polilínea. Debe haberse indicado tantos in colorIndex como líneas hay. Si no hay coorIndex entonces los colores se aplican en el orden presentado en el campo Color.

Ejemplo:

```
#VRML V2.0 utf8
Shape{
        appearance Appearance {
                material Material {emissiveColor 1 1 1 }}
        geometry IndexedLineSet {
                coord Coordinate {
                        point [ 0 0 0,
                                1 1 0,
                               -1 0 0,
                               -1 -1 0,
                               1 -1 0 ] }
                coordIndex [0 2 3 4 1 0 ]
                color Color { color [1 0 0,
                                     0 1 0,
                                     0 0 1]}
                colorIndex [0 1 2 1 2 0]
                colorPerVertex TRUE
        }
}
```

* **IndexedFaceSet**: especifica un conjunto de caras planas en el sistema local de coordenadas.

El campo *coord* especifica un conjunto de coordenadas en 3D.

El campo *coordIndex* especifica una lista de índices definiendo las caras que se van a dibujar. Si queremos terminar el marcador de una cara usaremos -1, que significa que termina la cara en curso y comenzaremos a continuación con otra cara.

Nota: Las caras están definidas por polilíneas cerradas, luego no necesitamos definir el primer punto dos veces. Veamos el ejemplo para definir un cuadrado.

Coord Coordinate { point[000, 100, 110, 010] }

Aquí hemos definido las cuatro esquinas de un cuadrado. Ahora usaremos coorIndex para dibujarlo. coordIndex [0 1 2 3] Esto significa unir el punto 0 de la lista con el 1, luego con el 2 y luego con el 3. Al final cerramos la cara conectando el 3 con el 0.

El campo *color* define un node Color. Lo que indica es una lista de colores que luego aplicaremos utilizando colorIndex. Si se ha definido Material y en esta cláusula indicamos emissive color, éste será el color por defecto, puesto que el campo color es opcional. Ojo: si no se indica color y el background por defecto es el negro puede que no veamos las caras dibujadas.

El campo *colorIndex* sirve para el mismo propósito que coordIndex pero considerando los colores. Si no usamos colorIndex entonces en su defecto se usará coordIndex.

El campo *colorPerVertex* es un campo booleano utilizado para indicar cómo definimos los colores. Se usa de la misma forma que en el nodo IndexedLineSet.

El campo *texCoord* especifica un nodo de tipo TextureCoordinate.

El campo *texCoordIndex* también se usa como coordIndex pero aplicado a las texturas para las caras.

Hay tres campos *normal* que tienen el mismo significado que los campos de color, pero aplicados a los nodos *NORMAL*. No los explicaremos por el momento.

Existe un campo llamado *ccw* . Una cara tiene dos lados, y a veces necesitaremos indicar cuál es el lado frontal y cual el trasero. Imaginemos que pintamos una cara en el plano XY. Si ponemos ccw a TRUE y la cara la definimos como "counterclockwise" entonces la cara frontal es la que estamos viendo. Si ccw es FALSE entonces ocurre lo contrario, la cara que vemos es la trasera.

El campo *solid* indica al navegador o visor si debería dibujar ambos lado de la cara o sólo el lado frontal. Por defecto se asume que solid=TRUE, y esto significa que la figura que creamos es sólida y por lo tanto no hay necesidad de dibujar las caras traseras.Sólo si SOLID=FALSE se dibujarán los dos lados de cada cara.

El campo *convex* especifica si las caras definidas por coordIndex son convexas o no. En principio VRML sólo puede dibujar caras convexas.Un ejemplo con las esferas. Si necesitamos dibujar caras cóncavas el visor dividirá esta cara cónvava en muchas caras pequeñas que sean convexas para simularlo, lo que consume un tiempo. Si estamos seguros de que todas las caras dibujadas son convexas pondremos convex=trae, de forma que le indicamos al visor que no se preocupe en perder tiempo dividiendo nuestras caras en caras menores. .

El campo creaseAngle especifica un ángulo umbral de unión de caras. Si dos caras adyacentes forman un ángulo mayor que el umbral veremos claramente dónde se unen las dos caras, puesto que el punto de unión se verá claramente. En caso contrario encontraremos un borde de unión más difuso.

```
IndexedFaceSet {
        coord NULL
        coordIndex [ ]
        color NULL
        colorIndex [ ]
        colorPerVertex TRUE
        normal NULL
        normalIndex [ ]
        normalPerVertex TRUE
        texCoord NULL
        texCoordIndex [ ]
        ccw TRUE
        convex TRUE
        solid TRUE
        creaseAngle 0.0
}
```

Ejemplo:

```
#VRML V2.0 utf8

Shape{
        appearance Appearance { material Material { }}
        geometry IndexedFaceSet {
                coord Coordinate {
                    point [-1 -1 0, 1 -1 0, 1 1 0, -1 1 0, 0 0 1 ] }
                coordIndex [0 1 4 -1, 1 2 4 -1, 2 3 4 -1, 3 0 4]
                color Color { color [1 0 0, 0 1 0, 0 0 1]}
                colorIndex [1 2 0 2  ]
                colorPerVertex FALSE
                creaseAngle 0
                solid TRUE
                ccw TRUE
                convex TRUE
        }
}
```

* **Extrusion**: la idea del nodo extrusion es contar con una herramienta potente de dibujo de figuras donde a diferencia de IndexedFaceSet no tengamos que definir todas las caras una a una. Para eso nos basamos en conceptos como CrossSection (Sección de corte) y Spine (dirección del movimiento).

Sintaxis:
```
Extrusion {
        beginCap TRUE/FALSE
        endCap TRUE/FALSE
        ccw TRUE/FALSE
        convex TRUE/FALSE
        creaseAngle 0
        crossSection [1 1, 1 -1, -1 -1, -1 0, 1 1]
        orientation 0 0 1 0
        scale 1 1
        solid TRUE
        spine [0 0 0, 0 1 0]
        }
```

crossSection: es una proyección 2D de la figura en el plano XZ (ojo!) Por ejemplo, la proyección 2D de un cubo es un cuadrado. La Proyección 2D de un Cono o Cilindro es un círculo. Para representarla pondremos en el eje X horizontal (positivo a la derecha, negativo a la izquierda) y el eje Z vertical (positivo hacia abajo, negativo hacia arriba).

Spine: define la ruta sobre la que la figura definida en crossSection viajará para crear nuestra figura 3D final. En el ejemplo del cubo el spine es la recta vertical en el plano Y, luego podemos definir dos puntos (0,-1,0) y (0,1,0) de forma que creamos el cubo de abajo a arriba.

¿Cómo dibuja extrusion? Primero traslada la crossSection al primer punto del spine, luego reorienta la crossSection (definida en el plano XZ) para que el eje Y coincida con la dirección obtenida de los dos primeros puntos del spine. Después mueve la crosssection de un punto al otro del spine, y continúa estre proceso hasta recorrer todos los puntos del spine.

Ejemplo del cubo:
```
shape {
        Geometry Extrusion {
                crossSection [-1 -1, -1 1, 1 1, 1 -1, -1 -1]
                spine [0 -1 0, 0 1 0}
        }}
```

beginCap y endCap: especifica si la figura resultante está abierta o cerrada en los lados del comienzo y del final. Es decir si los ponemos a false los dos tendremos nuestro cubo sin la tapa de abajo (beginCap=false) ni la de arriba(endCap = false). OJO: si **solid**=true

entonces sólo vemos las dos caras más cercanas a nosotros, pues esto indica que la figura es sólida y sólo se muestran las caras delanteras. Si ponemos SOLID=False vemos nuestro cubo sin tapas al completo.

Cómo se dibuja en spines de más de dos puntos. Vimos que en el caso de dos puntos se orienta la crossSection en la dirección formada por los dos puntos del spine. Este es el caso del cubo. Sin embargo si el spine tiene más puntos a partir del segundo y en cada uno de ellos la crossSection se orientará de forma que quede perpendicular a la tangente de la trayectoria del spine. Ver caso de una V con spine (3 5 0), (0 0 0), (-3 5 0).

Superficies de revolución: podemos crear superficies de revolución usando spines circulares. Ejemplo de spine circular:

```
spine [1 0 0,
       0.707 0 0.707,
       0 0 1,
       -0.707 0 0.707,
       -1 0 0,
       -0.707 0 -0.707
       0 0 -1,
       0.707 0 -0.707
       1 0 0]
```

Con este spine circular en el plano XY si indicamos un triángulo como crossSection en el plano XZ crearemos un cono. Ej: crossSection [-1 0, 0 0, -1 -2, -1 0]

Nota adicional: notar que las crossSection empiezan y acaban en el mismo punto. De esta forma declaramos figuras cerradas. Si no lo hacemos así deberíamos acabar con una figura con solid=false para ver bien el resultado.

Scale: además de lo que hemos comentado para cada punto del spine podemos escalar la crossSection a dibujar. Por ejemplo, estamos con la extrusión del cubo cuyo spine tiene dos puntos. El primero lo escalamos a (1 1) luego queda igual, pero el segundo lo escalamos a (0 0) ¿Cuál es el resultado? ➔ una pirámide. Ojo: los escalados se describen en 2D porque afectan a la crossSection. Ejemplo: scale [1 1, 0 0] debe tener tantos componentes como puntos el spine.

Orientation: funcionando de forma similar a scale sirve para dar una orientación a toda la figura, o bien una orientación por cada punto del spine. Las orientaciones llevan 4 valores cada una, tres definen el eje de rotación y uno más, el cuarto, define el ángulo de rotación. Ejemplo: orientation [0 1 0 0, 0 1 0 3.14]

Otros parámetros de extrusion: *ccw* especifica si los puntos de la crossSection tienen orden contador-de-reloj (counterclockwise) o reloj (clockwise) u orden desconocido (FALSE). Por otra parte *convex* especifica si la crossSection es convexa o no, si no lo fuera, esto es, es cóncava, el visor de VRML la dividiría en regiones más pequeñas convexas consumiendo un tiempo largo en esta tarea. Y para terminar el parámetro *creaseAngle* especifica un ángulo de umbral de forma que si dos caras adyacentes forman un ángulo mayor el borde de contacto entre las dos se verá claramente. En caso contrario este borde aparecerá difuminado.

* **ElevationGrid**: este nodo especifica una malla de puntos donde cada uno de los cuales tiene una altura definida. Se usa para representar terrenos, etc. Se construye en el plano XZ, comenzando en el origen y expandiendo en la dirección positiva de los ejes. Los parámetros principales son:

xDimension: el número de puntos en la dirección del eje X.
zDimension: el número de puntos en la dirección del eje Z.
xSpacing: la distancia entre dos puntos adyacentes del eje X.
zSpacing: la distancia entre dos puntos adyacentes del eje Z.

height: es una lista de valores que especifican la altura para cada punto del gris, ordenados de derecha a izquierda y de arriba abajo.
Color: define un nodo color. Es una lista de colores a aplicar a las caras. Es opcional.
colorPerVertex: booleano que funciona igual que en IndexedLineSet.

PRÁCTICA 1:
Crea un cono con radio de la base 75cm y altura 150 centímetros.

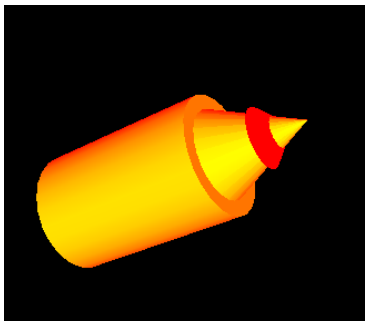
PRÁCTICA 2:
Agrega al cono del ejercicio anterior la apariencia por defecto (agrega las marcas appearance y material).

PRÁCTICA 3:
Modifica el fichero anterior para el cono sea de color verde.

PRÁCTICA 4:
El cono del ejercicio anterior aunque siga siendo verde debe emitir una luz roja.

PRÁCTICA 5:
Además del cono descrito debes colocar un cilindro 2 metros a su izquierda.

PRÁCTICA AVANZADA:
Crea la figura de la imagen:



I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)