**Universidad de San Carlos de Guatemala**
**Facultad de Ingeniería**
**Escuela de Ingeniería en Ciencias y Sistemas**

# LENGUAJE EXTENSIBLE DE ESTILO DE PÁGINAS

**ROSA ENOE ALBUREZ RUANO**

**Asesorada por Ing. Rodolfo Loukota Castellanos**

**Guatemala, febrero de 2004**

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# LENGUAJE EXTENSIBLE DE ESTILO DE PÁGINAS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

ROSA ENOE ALBUREZ RUANO
ASESORADA POR ING. RODOLFO LOUKOTA CASTELLANOS

AL CONFERÍRSELE EL TÍTULO DE
INGENIERA EN CIENCIAS Y SISTEMAS

**Guatemala, febrero de 2004**

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

**NÓMINA DE JUNTA DIRECTIVA**

| | |
|---|---|
| DECANO | Ing. Sydney Alexander Samuels Milson |
| VOCAL I | Ing. Murphy Olympo Paiz Recinos |
| VOCAL II | Lic. Amahán Sánchez Alvarez |
| VOCAL III | Ing. Julio David Galicia Celada |
| VOCAL IV | Br. Kenneth Issur Estrada Ruiz |
| VOCAL V | Br. Elisa Yazminda Vides Leiva |
| SECRETARIO | Ing. Pedro AntonioAguilar Polanco |

**TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO**

| | |
|---|---|
| DECANO | Ing. Herbert René Miranda Barrios |
| EXAMINADOR | Inga. Marlen Morales |
| EXAMINADOR | Ing. Byron López |
| EXAMINADOR | Ing. Iram Urrutia |
| SECRETARIA | Inga. Gilda Marina Castellanos de Illescas |

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

LENGUAJE EXTENSIBLE DE ESTILO DE PÁGINAS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas con fecha octubre de 2002.

Rosa Enoe Alburez Ruano

# DEDICATORIA

A DIOS:

Por ser mi amigo fiel, amarme y estar siempre a mi lado.

A MIS ABUELOS:

Augusto, Soledad, Efraín y Enoe, con mucho cariño y como una muestra de agradecimiento por el apoyo que me brindaron, pues su deseo fue siempre que llegara a ser una profesional.

A MIS PADRES:

Alvaro y Esthela, por ser los verdaderos merecedores de este triunfo, ya que gracias a su sacrificio, perseverancia y profundo amor permitieron que culminara mi carrera.

A MIS HERMANOS:

Augusto y David, porque siempre se que cuento con ellos en los momentos buenos y malos.

A MIS AMIGOS:

En especial a Marlene Mora, por su ayuda y apoyo incondicional.

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS

# GLOSARIO

**ASCII** Abreviación de *American Standard Code for Information Interchange*, un código de 7-bit, que sustituye las letras del alfabeto romano por cifras y otros caracteres informáticos. Los caracteres ASCII permiten comunicar con ordenadores, que utilizan un lenguaje especial llamado binario, formados por ceros y unos. Al escribir en el teclado, el ordenador interpreta cada letra escrita en lenguaje binario ASCII, para que puedan ser leídas, manipuladas, almacenadas o recuperadas. Los ficheros ASCII son denominados ficheros de texto.

**CDATA** Es una secuencia de caracteres tomados del conjunto de caracteres del documento y puede incluir entidades de caracteres. Los agentes de usuario deberían interpretar los valores de atributos como sigue:

- Reemplazar las entidades de caracteres con caracteres.
- Ignorar los avances de línea.
- Reemplazar el retorno de carro o tabulación con un espacio simple.
- Los agentes de usuario pueden ignorar el espacio en blanco inicial o final de valores de atributos.

**GML** Abreviación de *General Markup Language* fue uno de los primeros lenguajes de marcación que fue diseñado para componer estructuras de datos descriptivas, esto es, un metalenguaje, estructuras de datos que describen otras estructuras de datos. GML eventualmente se convirtió en GSML.

**HTML** El Lenguaje de marcación de hipertexto (*HyperText Markup Language*) es utilizado para la creación de documentos de hipertexto e hipermedia. Es el estándar usado en Internet.

**IBM** *International Business Machines Corporation,* es una empresa que se dedica al desarrollo, fabricación y comercialización de hardware, software y de todo tipo de tecnologías relacionadas con las tecnologías de la información. Además, presta servicios relacionados con dichas tecnologías como consultoría, integración de sistemas, outsourcing (externalización) de la gestión de sistemas informáticos, formación, mantenimiento, etc.

**Internet** Es una red de cómputo a nivel mundial que agrupa a distintos tipos de redes usando un mismo protocolo de comunicación. Los usuarios de Internet pueden compartir datos, recursos y servicios. Las computadoras que lo integran van desde modestos equipos personales, minicomputadoras, estaciones de trabajo, mainframes, hasta supercomputadoras. Internet no tiene una autoridad central, es descentralizada. Cada red mantiene su independencia y se une cooperativamente al resto, respetando una serie de normas de interconexión.

**JavaScript** JavaScript es un lenguaje scripting que permite hacer que las documentos HTML sean dinámicos, por ejemplo, haciendo que el relieve de un botón cambie al posicionar el cursor sobre éste.

**Metalenguaje** Es un lenguaje que permite definir lenguajes.

**Navegador de Internet** El web browser es el software especial que se utiliza para acceder a información de Internet y la información depositada en los Servidores de una Intranet. Permite al usuario hacer click en hipervínculos, de manera que proporciona al servidor información sobre dónde está depositada la página que quiere ver, y automáticamente mandarle y ver online documenos llamados páginas.

**SGML** El lenguaje generalizado de marcación ("Standard Generalized Markup Language") fue diseñado para permitir el intercambio de información entre distintas plataformas, soportes físicos, lógicos y diferentes sistemas de almacenamiento y presentación (bases de datos, edición electrónica, etc.), con independencia de su grado de complejidad. El material que constituye un documento se puede distribuir en diferentes archivos, tanto como sean necesarios: además pueden estar almacenados en distintos computadores.

**URI** El Identificador Universal de Recursos (*Universal Resouce Identifier*) es un indicador de la localización de un objeto en Internet.

**WML** *Wireless Markup Language* es el lenguaje que se utiliza para realizar páginas para cualquier elemento que utilice la tecnología WAP, como algunos teléfonos móviles.

**W3C** *World Wide Web Consortium* crea los estándares para Internet. La misión del W3C es llevar Internet a su máximo potencial**,** lo que se logra desarrollando tecnologías (especificaciones, directrices, software, y herramientas) que crearán un foro para la información, el comercio, la inspiración, el pensamiento independiente, y la comprensión colectiva.

# OBJETIVOS

- **General**

Describir uno de los lenguajes clave para trabajar con documentos XML, demostrando que con la trasnformacion de dichos documentos se puede realizar cualquier presentacion de información más eficazmente, y llegar a detalles que otros lenguajes no pueden realizar.

- **Específicos**

1. Describir los conceptos más importantes sobre XML.
2. Explicar qué es XSL y para qué fue creado.
3. Describir las especificaciones de toda la familia de tecnología XSL.
4. Mostrar la estructura común de transformación de datos.
5. Mostrar la transfomación de documentos XML a presentación para usuarios finales.

# RESUMEN

El Lenguaje Extensible de Marcas, (XML, *Extensible Markup Language* ) es un lenguaje para definir lenguajes. Los documentos XML están compuestos por unidades de almacenamiento llamados elementos, y en su sintaxis éstos se nombran mediante etiquetas. Las etiquetas codifican una descripción de la estructura de almacenamiento del documento y su estructura lógica, mientras que los elementos contienen en sí la información.

El Lenguaje Extensible de Estilo de Paginas, XSL, fue propuesto por el *World Wide Web Consortium* (W3C) para proveer de significado por medio de formato a los datos de XML. Para ello, se necesita de los siguientes componentes: un método para especificar qué parte de el documento se va a utilizar, un medio para describir que operación debería de realizarse para generar un documento; con esa parte de la información y una forma de incorporar instrucciones que indique como debe de presentarse dicha información.

El proceso de transformación de la informacion se hace en dos pasos: primero, a partir de un documento XML se crea un árbol fuente, con el cual se construye un árbol resultado. Segundo, se interpreta el árbol resultado para producir un formato que sirva para la presentación de la información en pantalla, en papel, en teléfono móvil, u otro tipo de medio.

XSL sirve para manipular información y sus especificaciones están basadas en XML, paralelamente surgieron dos frentes; XSLT y XSL-FO, con especificaciones propias cada uno; ambos pueden hacer el proceso completo de transformacion.

XSLT y su aliado XPath surgieron juntos, trabajando así: XSLT es una serie de plantillas de formato que junto a instrucciones de localización de XPath hacen que

el procesador de XSLT acople las plantillas a nodos de un documento de entrada de XML. XPath provee la navegación y búsqueda de fragmentos sobre un documento XML, mientras que XSLT se encarga del formato.

XSL-FO define un grupo de objetos para formato que especifican como la información de XML debe de ser presentada, lo cual permite especificar con detalle las caracteristicas de formato y la apariencia imposibles de obtener con XSLT. Su principal deficiencia es que aún no ha sido implementada una parte importante de sus especificaciones.

# INTRODUCCIÓN

Debido al desarrollo de Internet y lo importante que es para el intercambio y presentación de información surge el lenguaje XML, un lenguaje de marcas flexible y de fácil compresión, su importancia radica en que es tipo texto y es extensible, ya que no es un formato prefijado.

La principal deficiencia de XML es que la prestación de información por sí solo quedaba aón por debajo de otros lenguajes existentes, como HTML y CSS. Además cuando es necesario que solo una parte de la información sea presentada o el orden de ésta sea diferente, las especificaciones de XML no son suficientes para hacerlo. Por esta razón surge XSL, un lenguaje flexible y potente que aprovecha las ventajas de XML y que acopla plantillas para la presentación de la información de una forma mas fácil y rápida.

Las especificaciones de XSL se dividieron en dos ramas: Transformaciones de XSL (XSLT) y Formateo de Objetos de XSL (XSL-FO). Luego surgió una tercera rama llamada XPath, que es un lenguaje que identifica segmentos en un documento de XML.

El contenido de este trabajo de graduación expone sobre los propósitos y sintaxis de los tres lenguajes que forman parte de la familia de tecnología XSL. Presentando un marco teórico de XML y su evolución hasta llegar a XSL, luego se expone para qué sirve y como trabaja XPath. Ademas se presenta el procesamiento de datos de un documento por medio de XSLT, sus elementos y sus funciones; a continuación se presenta CSS el predecesor de XSL-FO introduciendose de esta manera las plantillas de información. Por ultimo, se define XSL-FO y la estructura de sus documentos. En cada uno de los temas se presentan ejemplos que hacen más fácil la comprension de la sintaxis y el funcionamiento.

# 1. CONCEPTOS GENERALES

Desde el surgimiento de la computación, se ha buscado una forma sencilla, segura y práctica de almacenar, mantener e intercambiar información. En 1969, IBM desarrolló el lenguaje GML, que sirvió como base para la creación del metalenguaje **Lenguaje Estándar de Marcas Generalizado, SGML** (por sus siglas en ingles *Standard Generalized Markup Language*) que fue reconocido. SGML proporciona un método consistente y preciso de aplicar etiquetas, para describir las partes que componen un documento; además permite el intercambio de documentos entre diferentes plataformas. Sin embargo, problemas que se atribuye a SGML son su excesiva dificultad de desarrollo e interpretación.

Debido al crecimiento de Internet y el uso de SGML se define el lenguaje **HTML** (*Hypertext Markup Language*), que es un lenguaje prefijado y rígido, por lo que empezó a ser necesario un lenguaje más poderoso y sencillo, que permitiera continuar con la idea de combinar etiquetas y datos en el mismo archivo, por eso en 1988 nace la primera versión del **Lenguaje Extensible de Marcas**, **XML** (*Extensible Markup Language*). XML es un lenguaje que separa los datos de la presentación, y debido a su versatilidad es muy usado y versátil; además, mantiene el formato de la descripción y la estructura de los datos, definiendo su propio vocabulario, el cual esta basado en texto y es independiente a la plataforma. La combinación de todas estas características hace que XML tenga un enorme campo de trabajo en Internet y en aplicaciones gerenciales ya que puede utilizarse como medio de intercambio de información entre dos organizaciones o aplicaciones. Una de las necesidades de las empresas es presentar su información de muchas formas y con diferentes restricciones; para esto, se necesita de un proceso de transformación, con el cual no cuenta XML, debido a esta realidad nace el **Lenguaje Extensible de Estilo de Páginas**, **XSL** (*Extensible Stylesheet Language*), que manipula documentos de XML.

## 1.1. Lenguaje Extensible de Marcas (XML)

El Lenguaje Extensible de Marcas, XML es un lenguaje para definir lenguajes. Los documentos XML están compuestos por unidades de almacenamiento llamados **elementos**, y en su sintaxis éstos se nombran mediante **etiquetas**. Las etiquetas codifican una descripción de la estructura de almacenamiento del documento y su estructura lógica, mientras que los elementos contienen en sí la información. XML proporciona un mecanismo para imponer restricciones al almacenamiento y a la estructura lógica de la información.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, y casi para cualquier cosa que se pueda pensar. Sin ir más lejos, algunos lenguajes definidos en XML, recorren áreas como química, física, matemática, dibujo, tratamiento del habla, y otras muchas.

XML constituye la capa más baja dentro del nivel de aplicación, sobre el que se puede montar cualquier estructura de tratamiento de documentos, hasta llegar a la capa de presentación.

### 1.1.1. Antecesores del XML y sus comparaciones

Los orígenes de XML se remontan a 1986, cuando nace el SGML que es el estándar internacional para la definición de la estructura y el contenido de diferentes tipos de documentos. Es decir, que es un metalenguaje que permite definir lenguajes para indicar la estructura y el contenido de nuestros documentos. SGML proporciona un modo consistente y preciso de usar etiquetas para describir las partes que componen un documento, además, permite el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se atribuye a SGML es su

excesiva dificultad de desarrollo; basta con pensar que la recomendación ocupa unas 400 páginas. La definición de la estructura y el contenido se realiza en una **Definición de Tipo de Documento, DTD** (*Document Type Definition*); en ella, se declaran los elementos que conformarán este tipo de documentos y cómo tienen que estar organizados para que el documento sea correcto. Para un documento SGML, la definición del DTD es obligatoria.

Por ejemplo, un DTD define cómo tienen que ser los documentos HTML. Por lo tanto, el HTML no es más que un tipo de documento SGML que se utiliza en Internet, mientras XML no es ningún tipo de documento SGML, sino que es una versión abreviada de éste, que es optimizada para su utilización en Internet. Esto significa que con XML es posible definir nuestros propios tipos de documentos y nuestras propias etiquetas, sin depender de un único e inflexible tipo de documento HTML.

HTML es un invento importante, debido a su sencillez, es el sistema más exitoso de la historia para la presentación de documentos. Gracias a HTML se ha podido acceder y publicar mucha información a través de Internet de una manera amigable, pero a su vez ha sido victima de su propio éxito, pues por desgracia no evoluciono de la mejor manera, porque sigue siendo igual de rígido e inflexible en la definición de su sintaxis.

El XML más que un HTML aumentado hay que considerarlo como un SGML reducido y optimizado para su utilización en Internet. Como escribió Richard Ligth, "XML ofrece el 80% de las ventajas del SGML con un 20% de su complejidad"[1], debido a que sus diseñadores intentaron dejar fuera aquellas partes que raramente se utilizan; de esto resulto que la especificación XML ocupa aproximadamente 30 páginas, frente a las 400 del SGML.

### 1.1.2. Objetivos de XML

XML se diseño para cumplir los siguientes objetivos:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben poderse elaborar fácilmente.
- La concisión en las marcas XML es de mínima importancia.

### 1.1.3. Sintaxis del XML

A continuación, se presenta un documento muy sencillo de XML, que se estudiará para ver las características del lenguaje:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ficha>
   <nombre>Mateo</nombre>
   <apellido>Flores</apellido>
   <dirección>6 Ave. 21-24 zona 4</dirección>
</ficha>
```

La primera línea indica que lo que la sigue es XML; aunque es opcional, es recomendable incluirla siempre, y puede tener varios atributos, algunos obligatorios y otros no:

- `version`: como su nombre lo indica la versión de XML es usada en el documento. Es obligatorio ponerlo, a menos que sea un documento externo a otro que ya lo incluía.

- `encoding`: indica la forma en que se ha codificado el documento. Por defecto es UTF-8, pero también existen el UTF-16, US-ASCII, ISO-8859-1, etc. No es obligatorio, salvo que sea un documento externo a otro principal.

- `standalone`: indica si el documento va acompañado de un DTD o no; en principio, no hay por qué ponerlo, ya que luego se indica el DTD, si se necesita.

En cuanto a sintaxis del documento, y antes de entrar en el estudio de las etiquetas, hay que resaltar algunos detalles importantes de conocer:

- Los documentos XML son sensibles a mayúsculas y minúsculas. Por eso `<FICHA>`, sería una etiqueta diferente a `<ficha>`.

- Todos los espacios y retornos de carro se toman en cuenta (dentro de las etiquetas y en los elementos).

- Los caracteres especiales reservados que forman parte de la sintaxis de XML: <, >, &, " y '. Al ser necesario representarlos en un texto, se deben sustituir por las entidades `&lt;`, `&gt;`, `&amp;`, `&quot;` y `&apos;` respectivamente.

- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y las comillas simples (').

Dentro del contenido del documento, hay etiquetas similares a las de HTML, que describe a los elementos y los elementos que son la información en sí.

Hay dos tipos de elementos: los vacíos y los no vacíos. Hay varias consideraciones importantes que se deben tener en cuenta al respecto:

- Toda etiqueta no vacía debe tener una etiqueta de cerrado: `<etiqueta>` debe estar seguida de `</etiqueta>`. Esto se hace para evitar la equivocación a la que habían llegado todos los navegadores HTML, de permitir que las etiquetas no se cerraran, lo que deja los elementos sujetos a posibles errores de interpretación.

- Los elementos vacíos son aquellos que no tienen contenido dentro del documento. Un ejemplo en HTML son las imágenes. La sintaxis correcta para estos elementos es `<etiqueta/>`.

- Todos los elementos deben estar perfectamente anidados. No es válido poner:

  `<ficha><nombre>`**`Mateo`**`</ficha></nombre>`

  lo correcto es:

  `<ficha><nombre>`**`Mateo`**`</nombre></ficha>.`

### 1.1.4. Entidades

Las primeras entidades que se presentaron fueron los caracteres especiales &, ", ', < y >, que deben escribirse entre textos, mediante las declaraciones: `&amp,` `&quot;` `&apos;,` `&lt;` y `&gt;`. Las entidades no sólo sirven para incluir caracteres especiales no ASCII; también se pueden usar para incluir cualquier documento u objeto externo.

Por ejemplo, se puede crear en un DTD o en el documento XML una entidad que haga referencia a un nombre largo:

```
<!ENTITY AEI "Asociación de Estudiantes de Ingeniería">
```

de este modo, cada vez que sea necesario que en nuestro documento aparezca el nombre "Asociación de Estudiantes de Ingeniería", bastará con escribir `&AEI;`. Los beneficios son claros, y además es muy sencillo.

El texto referenciado mediante la entidad puede ser de cualquier tamaño y contenido, si por ejemplo, debe incluir una porción de otro documento dentro de un documento de XML, se puede hacer la siguiente forma:

```
<!ENTITY midoc SYSTEM
http://www.usac.edu.gt/~ralburez/midoc.xml">
```

Con la palabra SYSTEM se indica al procesador que la referencia es externa, y a continuación entre comillas, se coloca la dirección del documento.

Para incluir entidades en DTDs se debe usar el carácter %:

```
<!ENTITY % uno "(uno | dos | tres)">
```

Y para expandirlo se debe que escribir:

```
<!ELEMENT numero (%uno;) #IMPLIED>
```

### 1.1.5. Documentos de XML

Todo documento XML posee una estructura física y una lógica. La primera está compuesta de unidades llamadas entidades, las cuales pueden hacer referencia a otras entidades para que éstas sean incluidas en el documento, iniciando con la entidad "raíz" o entidad documento. La estructura lógica está compuesta de **declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento**, los cuales se indican en el documento mediante marcas explícitas. Las estructuras física y lógica deben anidarse de manera correcta.

Hay dos **tipos de documentos** XML: **los válidos y los bien formados**.

Se dice que un **documento XML está bien formado** si, considerándolo como conjunto, encaja con las especificaciones XML de producción, lo que implica que:

- Contiene uno o más elementos.

- Hay exactamente un elemento llamado raíz o elemento documento, de manera que ninguna parte del mismo aparece en el contenido de ningún otro elemento.

- Cumple todas las restricciones que proporciona su especificación a través del DTD. Si no se utiliza DTD, el documento debe comenzar con un Declaración de Documento Único, SDD (*Standalone Document Declaration*), así:
  `<?XML version="1.0" standalone="yes"?>`

- Cada una de sus partes procesadas esta bien formada.

- Todas las etiquetas deben estar balanceadas: esto es, todos los elementos que contengan datos de tipo carácter deben tener etiquetas de principio y fin (no está permitida la omisión, excepto para los elementos vacíos, (véase a continuación).

- Todos los valores de los atributos deben ir entrecomillados (el carácter comilla simple o también llamado apóstrofe puede utilizarse si el valor contiene caracteres comillas dobles, y viceversa; si se necesitan ambos, se debe utilizar `&apos;` y `&quot)`

- Cualquier elemento VACÍO (por ejemplo aquellos que no tienen etiqueta final como `<IMG>`, `<HR>`, y `<BR>` y otros de HTML) deben terminar con '/>' o deben hacerlos no VACÍOS, añadiéndoles una etiqueta de fin;
  Ejemplo: `<BR>` se convertirá en `<BR/>` o en `<BR></BR>`.

- No debe haber etiquetas aisladas tales como < ó & en el texto, las cuales deben escribirse como `<` y `&` por ejemplo, la secuencia ]]> debe escribirse como `]]>`

- Los elementos deben anidar dentro de sí sus propiedades (no se deben sobreponer etiquetas);

- Los ficheros bien-formados sin-DTD pueden utilizar atributos en sus elementos, pero éstos deben ser todos del tipo CDATA (se explica más adelante), por defecto.

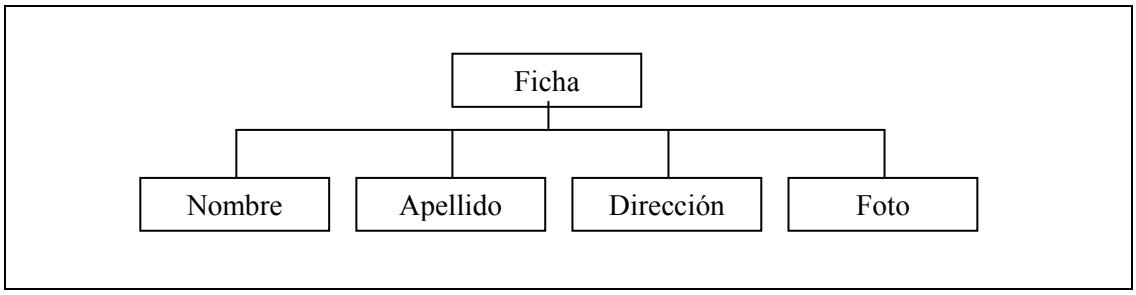
Es **documento válido**, si además de estar bien formado, deben seguir una estructura y una semántica determinada por un DTD, es decir, sus elementos; la estructura jerárquica y los atributos deben ajustarse a lo que el DTD dicte.

## 1.2. Definición de Tipos de Documento, (DTD)

EL DTD es la definición de la gramática del documento; en él se definen los elemento, la forma y los atributos que un documento XML puede incluir.

Un documento XML presenta una jerarquía muy determinada, definida en el DTD; si es un documento válido dicha jerarquía, se puede representar como un árbol de elementos. Existe un **elemento raíz**, que siempre debe ser único (sea nuestro documento válido o sólo bien formado) y su nombre se define con `<!DOCTYPE`; de él descienden las ramas de sus respectivos elementos descendientes o hijos. De este modo, la representación en forma de árbol de nuestro documento XML de ejemplo sería:

Fig 1. Árbol de documento XML




Se puede observar que es un documento muy sencillo, con una profundidad de 2 niveles nada más: el elemento raíz es ficha, y sus hijos son nombre, apellido, dirección, foto. Es obvio que cuanta más profundidad existe, mayor tiempo se tarda en procesar el árbol, pero la dificultad siempre será la misma gracias a que se usan, como en todas las estructuras de árbol, algoritmos recursivos para tratar los elementos.

El DTD, por ser la definición de esa jerarquía, describe precisamente la forma de ese árbol. La diferencia está en que el DTD define la forma del árbol de elementos, y un documento XML válido puede basarse en ella para estructurarse, aunque no tienen que tener en él todos los elementos, si el DTD no obliga a ello. Un documento XML bien formado sólo tendrá una estructura jerarquizada, sin ajustarse a ningún DTD concreto.

### 1.2.1. Formas de referenciar un DTD a un documento XML

- Incluir dentro del documento una referencia externa al DTD en forma de Identificador Universal de Recursos, URI (*Universal Resource Identifier*) y mediante la siguiente sintaxis:

```
<!DOCTYPE ficha SYSTEM  "http://www.usac.edu.gt/ralburez
/ficha.dtd">
```

La palabra `SYSTEM` indica que el DTD se obtendrá a partir de un elemento externo seguido de la dirección entre comillas.

- Incluir dentro del propio documento el DTD de este modo:

```
<?xml version="1.0"?
<!DOCTYPE ficha [
<!ELEMENT ficha (nombre+, apellido+, direccion+, foto?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST nombre sexo (masculino|femenino) #IMPLIED>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT dirección (#PCDATA)>
<!ELEMENT foto EMPTY>
]>
<ficha>
<nombre>Mateo</nombre>
<apellido>Flores</apellido>
<dirección>6 Ave. 21-24 zona 4</dirección>
</ficha>
```

La forma de incluir el DTD directamente es añadir a la declaración `<!DOCTYPE`, el nombre del tipo de documento, en vez de la dirección. El DTD debe de ir entre los símbolos '[' y ']' y todo lo que esté entre dentro de ellos se considera parte del DTD.

### 1.2.2. Definición de elementos

Esta definición es bastante intuitiva: después de la cláusula `<!ELEMENT` se incluye el nombre del elemento (el que se indicará en la etiqueta), luego seleccionan las diferentes características en función del elemento:

- Para un elemento no vacío entre paréntesis se indica el contenido del elemento, el cual puede ser: la lista de hijos o descendientes, separados por comas; o el tipo de contenido; el contenido mas común es `#PCDATA`, que indica datos de tipo texto.

- Si es un elemento vacío, se indica con la palabra `EMPTY`.

Los elementos descendientes deben ir seguidos de caracteres especiales: `'+'`, `'*'`, `'?'` o `'|'`, los cuales indican qué tipo de uso se permite hacer de esos elementos dentro del documento:

- `+`: uso obligatorio y múltiple; permite uno o más elementos de ese tipo dentro del elemento padre, pero como mínimo uno.

- `*`: opcional y múltiple; puede no haber ninguna, una o varias ocurrencia del elemento.

- `?`: opcional pero singular; puede no haber ninguna o como mucho una ocurrencia.

- `|`: equivale a un `OR`, es decir, da la opción de usar un solo elemento de entre los que forman la expresión.

De este modo, si por ejemplo un DTD tiene la siguiente declaración:

```
<!ELEMENT ficha (nombre+, apellido+, direccion*, foto?, telefono*|fax*)>
```

indica que el elemento `ficha` puede contener los siguientes elementos: un `nombre` y un `apellido` como mínimo, pero puede tener más de uno de ellos; opcionalmente puede incluir una o varias `direcciones`, aunque no es obligatorio; además puede contener una única `foto`; y al final, colocar, aunque no es obligatorio, uno o más `telefonos` o uno o más números de `fax`.

Para la definición de **atributos**, se usa la declaración `<!ATTLIST`, seguida de:

- El **nombre de elemento** del que se están declarando los atributos.

- El **nombre del atributo**.

- Los posibles **valores del atributo**, entre paréntesis y separados por el carácter |, que al igual que para los elementos, significa que el atributo puede tener uno y sólo uno de los valores incluidos entre paréntesis. O bien, si no hay valores definidos, se escribe `CDATA` para indicar que puede ser cualquier valor. También es posible indicar con la declaración `ID` que el valor alfanumérico que se le dé será único en el documento, y se podrá referenciar ese elemento a través de ese atributo y valor.

- De forma opcional y entre comillas, se puede colocar un valor por defecto del atributo.

- Por último, debe incluirse una de las siguientes opciones:
  - `#FIXED` si el valor de dicho atributo se debe mantener fijo a lo largo de todo el documento para todos los elementos del mismo tipo.
  - `#REQUIRED` si es obligatorio cada vez que se usa el elemento.
  - `#IMPLIED` si no es obligatorio.

Es importante destacar un aspecto para la optimización del diseño de DTDs: muchas veces es necesario decidir entre especificar atributos de elementos como elementos descendientes o como atributos en sí. Por ejemplo, una pieza de maquinaria con unas características determinadas puede definirse de dos formas diferentes:

```
<pieza>MiPieza
      <color>Rojo</color>
</pieza>
```

o bien considerar la opción:

```
<pieza color="Rojo">Mipieza</pieza>
```

Queda a discreción del diseñador el decidir entre ambas, pero hay que tener en cuenta que si se usa la primer forma, el procesador tiene que bajar al siguiente nivel del árbol de elementos para saber el `color` de `MiPieza`, mientras que en el

segundo caso se puede obtener haciendo referencia directa al atributo "color" del elemento. Por otro lado, el primer caso está más acorde con la filosofía de XML, en la que las etiquetas hacen referencia siempre a su contenido, sin necesidad de acudir a los atributos, que era lo que se hacía en HTML.

### 1.3. Espacios de nombre

Los espacios de nombre (*Namespaces*) le dan un calificativo especial a los nombres de una manera organizada, para evitar conflictos entre elementos con el mismo nombre; a través de ellos, se asegura que los nombres de los elementos no tengan conflictos y sean claros, pero no determinan cómo procesar dichos elementos. El *parser* debe de conocer el significado de los elementos y como procesarlos.

Cuando etiquetas iguales viene de varios lugares de Internet, pueden ser confundidas; utilizando espacios de nombre; dichas etiquetas pueden existir en el mismo documento XML, pero deben de estar direccionados desde dos esquemas diferentes, únicamente cualificados por semántica. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<Lista>
   <Item>
      <Posicion>0001</Posicion>
      <Descripcion>
         <Libro>Base de datos distribuidas</Libro>
         <Posicion>Estantería 8</Posicion>
      </Descripcion>
   </Item>
   <Item>
      <Posicion>0002</Posicion>
      <Descripcion>
         <Libro>Sistemas Operativos</Libro>
         <Posicion>Estantería 6</Posicion>
      </Descripcion>
   </Item>
```

En el cuadro anterior, se pueden notar cuatro elementos <Posicion> que significan diferentes cosas, lo que provoca confusión. Con los espacios de nombre, se pueden ver los documentos como capas, con la utilización de prefijos para diferenciar un documento de otro, como se ve en el siguiente cuadro.

```
<?xml version="1.0" encoding="UTF-8"?>
<Lista xmlns:desc=
   "http://www.usac.edu.gt/ralburez/descripcion">
   <Item>
      <Posicion>0001</Posicion>
      <desc:Descripcion>
         <desc:Libro>Base de datos distribuidas</desc:Libro>
         <desc:Posicion>Estantería 8</desc:Posicion>
      </desc:Descripcion>
   </Item>
   <Item>
      <Posicion>0002</Posicion>
      <desc:Descripcion>
         <desc:Libro>Sistemas Operativos</desc:Libro>
         <desc:Posicion>Estantería 6</desc:Posicion>
      </desc:Descripcion>
   </Item>
```

### 1.4. Que es el Lenguaje Extensible de Estilo de Páginas (XSL)

Es un lenguaje, cuyo propósito original es proveer de significado por medio de formato a los documentos de XML. Esto se hace en dos pasos, primero, se construye un árbol resultado partiendo de un árbol fuent,e y luego se interpreta el árbol resultado para producir un formato resultado, que sirva para la presentación en una pantalla, en papel, en un teléfono móvil, u otro tipo de medio.

Un punto fuerte de XML es el almacenamiento de información, pero carece de la presentación de la mismos; por esta razón, XML separa la información de la presentación, para que pueda ser reutilizada en múltiples presentaciones. Al utilizar XSL en documentos XML, se logra que la información se presente con un aspecto bonito y agradable; esto es soportado por los tradicionales navegadores de Internet, salidas de voz y dispositivos móviles.

### 1.4.1. Historia de XSL

XSL nació en 1997 suministrado por el **W3C** (*World Wide Web Consortium)*; la organización encargada de velar por los estándares de Internet, se basa en dos tecnologías antiguas **Lenguaje de Estilo de Documentos por Semánticas y Especificaciones, DSSSL** (*Document Style Semantics and Specification Language*) y **Página de Estilos de Cascada, CSS** (*Cascading Style Sheets*).

El DSSSL es el estándar para presentación de documentos SGML; es un lenguaje de programación completo y muy potente basado en lenguaje LISP. En vista de que XML es una versión reducida de SGML, resulta congruente que también exista una versión reducida equivalente para XML, llamada XSL.

La CSS es un procesador muy utilizado en el diseño profesional de páginas con HTML, trabaja con base en plantillas de descripción de formatos, para ser aplicadas al contenido de un documento. Las descripciones se archivan en documentos de texto sin formato con la extensión *.css, para su identificación. Sus ventajas son: las plantillas pueden ser enlazadas con uno o varios documentos, por lo que todo el trabajo realizado en una plantilla, puede ser aprovechado por múltiples documentos; cada vez que se modifica algún dato en cualquier plantilla, automáticamente quedan actualizadas las presentaciones de todos los documentos enlazados con ella y, por otro lado, permite controlar el formato de un documento hasta niveles de detalle imposibles de definir con las etiquetas de formato HTML.

XSL es un lenguaje basado en XML, que sirve para la manipulación de información. Las especificaciones de XSL son para formatear documentos XML, como despliegue en Internet; realizar este proceso implica dos pasos diferentes: primero, se transforma la estructura de la información a un nuevo formato de XML y luego se adiciona la información necesaria para que los datos aparezcan en la presentación en una forma específica. Debido a esto XSL se dividió en dos caminos XSLT y XSL-FO con especificaciones propias cada uno; ambos son capaces de hacer todo el proceso de transformación.

XSL-FO define un grupo de objetos para formateo que especifican como la información de XML debe de ser presentada. Tiene un concepto similar a la CSS pero más poderoso.

XSLT y su aliado XPath surgieron juntos, y trabaja así: XSLT es una serie de plantillas que, junto a instrucciones de localización de XPath, hacen que el procesador de XSLT acople las plantillas a nodos de un documento de entrada de XML. XPath provee la navegación y búsqueda sobre un documento XML.

XSLT es usado para transformar datos de XML de una estructura a otra. Esta transformación puede ser usada para:

- Crear un nuevo contenido
- Convertir un árbol de XML a otro que puede o no tener formato XML
- Agregar un nuevo contenido
- Extraer una porción del contenido
- Llevar a cabo trabajo computacional, tal como realizar búsquedas
- Generar un documento para esquema

Además XSLT extiende su uso para trasformar datos en un formato apropiado para la presentación de:

- Despliegue en navegadores de Internet, que incluye salidas de HTML, JavaScript, WML, etc.
- Preparación de material para buscadores no tradicionales como salidas de audio.
- Incorpora gráficas o transforma la información para representaciones gráficas.

### 1.4.2. Hojas de estilo de XSL

El procesador de XSLT esta formado por plantillas e instrucciones de XPath. Por cada plantilla, el procesador busca en el documento de entrada, hasta encontrar un nodo que se iguale a la estructura especificada en la plantilla, entonces aplica la

plantilla a los datos encontrados. De esta forma, genera la salida de cada parte de la plantilla. El resultado de aplicar plantillas a un documento de entrada en XML es otro documento en XML llamado documento resultado o de salida. Note que la salida también puede ser un documento en HTML o un archivo plano.

### 1.4.3. Modelo de procesamiento de XSL

El modelo de procesamiento tiene dos métodos: el **declarativo o funcional,** en el cual el procesador responde al encontrar elementos de XML, y el **imperativo,** en el cual el procesador introduce una ejecución o una serie de operaciones explicitas. Ejemplos de programación funcional son *Lisp, Goffer*, mientras que de programación imperativa son *C, Pascal y Java*.

Por ejemplo, se desea transformar los elementos contenidos en un documento dentro un fragmento de HTML, listo para desplegarse en un buscador.

Documento de Entrada

```
<Descrip_Trabajo>
   <Puesto>Administrador de Base de Datos</Puesto>
   <Empresa>Facultad de Ingeniería, USAC</Empresa>
   <Localizacion>
      <Ciudad>Guatemala</Ciudad>
      <Direccion>Edif.T3, Ciudad Universitaria</Direccion>
   </Localizacion>
   <Salario moneda='Quetz.'>10000</Salario>
   <Descripcion>Mantenimiento de la base de datos,
         generar copias de seguridad </Descripcion>
</Descrip_Trabajo>
```

Documento de Salida

```
<h2>Vacante</h2>
<h3>Administrador de Base de Datos</h3>
<h3>Facultad de Ingenieria, USAC</h3>
<p>Edif. T3, Ciudad Universitaria, Guatemala</p>
<p>Q.10000</p>
<p>Mantenimiento de la base de datos, generar copias de seguridad
</p>
```

Utilizando el método funcional una descripción de la transformación entre los dos documentos debería hacer lo siguiente:

- Cuando el elemento <Descrip_Trabajo> es encontrado, genera un elemento <h2> que contiene el texto 'Vacante' y entonces el proceso busca la información contenida dentro de <Descrip_Trabajo>.

- Cuando un elemento <Puesto> es encontrado, genera el texto contenido dentro de elementos <h3>.

- Cuando un elemento <Empresa> es encontrado, genera el texto contenido dentro de elementos <h3>.

- Cuando un elemento <Localizacion> es encontrado, inicia un párrafo, a cada elemento encontrado le aplica reglas para generar texto y finaliza el párrafo.

- Cuando un elemento <Salario> es encontrado, genera un párrafo conteniendo el símbolo de la moneda y el texto dentro del elemento.

- Cuando un elemento <Descripcion> es encontrado, genera un párrafo con el contenido del elemento.

Utilizando el método imperativo, la secuencia seria la siguiente:

- Por cada elemento <Descrip_Trabajo> encontrado en el documento de entrada hacer lo siguiente:

    - Primero genera un elemento <h2> conteniendo el texto 'Vacante'

    - Genera un elemento <h3> conteniendo el texto del elemento <Puesto> que esta por debajo del elemento <Descrip_Trabajo>

- Genera un elemento <h3> que contiene el texto del elemento <Empresa>, que está por debajo del elemento <Descrip_Trabajo>

- Genera un párrafo que contiene el texto de <Direccion> que esta por debajo del elemento <Localizacion>, que a su vez esta por debajo del elemento <Descrip_Trabajo> y el texto de <Ciudad> el cual esta por debajo del elemento <Localización>, que a su vez esta por debajo del elemento <Descrip_Trabajo>.

- Genera un párrafo conteniendo el símbolo de la moneda contenida en el atributo Moneda del elemento <Salario> que está por debajo del elemento <Descrip_Trabajo>, seguido del contenido del elemento <Salario>

- Genera un párrafo conteniendo el texto de <Descripcion>, que está por debajo del elemento <Descrip_Trabajo>

El método declarativo o funcional es mas usado cuando se procesan datos que pueden tener diferentes formatos, o contener cualquier número de especificaciones de elementos, debido a que no es necesario conocer la estructura del documento y las especificaciones con que cuenta, mientras que el método imperativo es más usado para programas lógicos o de secuencias.

Es fácil de ver con este ejemplo que el método declarativo o funcional es más reutilizable. Si en el ejemplo se debe agregar el teléfono dentro de la Localidad únicamente se agrega la plantilla con esa información. Si una parte de la información es borrada actualmente, no es necesario hacer nada. Mientras que en el método imperativo si se borra un trozo del archivo de entrada, el programa generará error y se interrumpirá, porque, en vez de decir "Si encuentra XXX haga esto" está diciendo "usted debe encontrar XXX, entonces haga esto con lo encontrado".

## 1.5. Herramientas para XSL

Como ya se mencionó, la transformación de XSL es ejecutada por el Procesador de XSL. El procesador de XSLT para correr hojas de estilo XSLT contra un documento de entrada, y el procesador XSL-FO para tomar documentos XSL-FO e interpretarlos para el despliegue.

Los procesadores XSLT son ampliamente disponibles, incluyendo Internet Explorer 5 tiene un procesador XSLT llamado MSXML (www.microsoft.com), Netscape 6 cuenta con su procesador de ingeniería Mozilla llamado TransforMiiX (www.netscape.com), el Xalan-C++ y Xalan-Java de Apache (www.apache.org), Lotus XSL de IBM (www.alphaworks.ibm.com), Saxon de Michael Kay (saxon.sourceforge.net) y el XT de James Clark (blnz.com/tx/). Los procesadores más populares tienden a basarse en C++ o Java. El parser MSXML3 de Microsoft, que incluye un procesador de XSLT puede utilizarse como un COM de interface con Windows. Muchos de los procesadores XSLT tiene fuentes abiertos, y la mayoría son implementados con los últimos estándares de W3C.

XSLT es mucho más apoyado que XSL-FO; esto es debido a que XSL-FO aun esta en proceso de estandarización, sin embargo hay algunos procesadores que lo soportan como FOP de Apache (www.apache.org), XEP de RenderX (www.rendex.com) y Antenna House XSL Formatter (www.antennahouse.com), que tiene un formato apropiado para el despliegue por medio de Windows GDI.

## 1.6. Otras alternativas para transformar XML

XSL no es el único mecanismo para manipular XML. Debido a que XML es almacenado en archivos de texto planos es muy fácil que se produzcan programas que carguen la información a una estructura de árbol y la manipulen. Escribir un

programa para leer un documento de XML es difícil, pero es posible, haciendo un trabajo arduo. Debido a esto, hay dos estándares para leer documentos de XML, llamados **APIs** (*Application Programming Interfaces*) descritos a continuación:

- **DOM** (Document Object Model) Construye a partir del documento una representación de árbol en memoria y permite recuperar y manipular el árbol y sus nodos. La implementación de DOM provee un mecanismo para poder grabar el árbol de memoria aun disco; ese mecanismo puede depender del procesador.

- **SAX** (Simple API for XML parsing) Ofrece un programa para parsear documentos. Notifica los eventos encontrados como inicio de documento, encontrar un elemento, etc. Es responsabilidad del programador mantener la pista de alguna información necesaria para el proceso de almacenamiento. No todo el documento es mantenido en memoria, lo que es una ventaja respecto a DOM, ya que se puede procesar documentos más grandes.

# 2. XPATH: LENGUAJE DE RUTA DE XML

**XPath** es la clave para definir que plantilla es o no aplicada a un nodo del documento XML fuente. XPath fue diseñado para ser usado por XSLT y por medio de localizar fragmentos de árbol dentro de un documento XML.

## 2.1. Introducción a XPath

XPath es la notación que hace posible la transformación con XSLT a una aplicación específica, con XPath se selecciona quá partes del documento fuente son apropiadas para la aplicación de una plantilla, la cual produce el árbol resultado deseado.

### 2.1.1. Para que sirve XPath

El enfoque inicial del uso de XLST-XPath es producir páginas para Internet. Por ejemplo en una intranet, se puede desplegar toda la información de una base de datos utilizando simplemente HTML, pero si una página similar es destinada para Internet, puede ser necesario presentar solo información no confidencial.

XPath permite hacer los cambios necesarios al documento utilizando una de las siguientes opciones: se puede seleccionar para despliegue sólo los elementos no confidenciales del documento fuente o bien filtrar información con base en los atributos de los elementos; también es posible filtrar la información con base en un determinado criterio de datos, como presentar sólo la información que parte de una

fecha específica. Por último, XSLT-XPath se utiliza en el intercambio de información.

Los propósitos para los cuales se pueden utilizar las expresiones de XPath en XSLT son:

- Seleccionar nodos para procesamiento
- Especificar condiciones para diferentes formas de procesar un nodo
- Generación de texto que será insertado en el árbol resultado.

Las expresiones de XPath permiten enfocar la parte seleccionada del documento origen, que provee la información adecuada para crear el documento de salida o árbol resultado. El tipo más común e importante de expresión de XPath es la ruta de localización.

### 2.1.2. Cómo trabaja XPath

El propósito de XPath es dirigir al procesador de XSLT en la navegación dentro de los nodos del árbol del documento fuente o **árbol fuente**, los cuales pueden requerir de una procesamiento para producir los nodos del **árbol resultado**.

Una forma simple de entender los conceptos básicos XPath es compararlo con alguien, al preguntar sobre una dirección o un edificio en particular. Se empieza dando el punto de inicio; se le dice que camine en una dirección específica, que cruce en determinado sentido; cuando esté en cierto punto o haya caminado cierta cantidad de metros, se da determinada característica del destino. Por ejemplo, de aquí camine dos cuadras hacia el norte, cruce hacia la derecha donde esta una farmacia de color azul y la tercera casa es la que busca. Esencialmente XPath trabaja de la misma forma, guiando a XSLT a través del árbol de nodos del documento fuente de XML.

Siguiendo con el ejemplo, el grupo de direcciones de calles es la **ruta de localización**, el punto de inicio es el **contexto**, cada referencia como punto de partida hacia otro lugar es un **nodo de contexto**, los puntos cardinales son **ejes** que incluyen **eje paterno, eje antecesor, eje hijo, eje descendiente***,* en total XPath cuenta con 13 ejes. Cuando se mencionan características como el ejemplo farmacia de color azul para XPath, eso es un **atributo.**

XPath tiene la capacidad de guiar a través de características relativas al lugar, que seria utilizar el **contexto** o guiar por características del destino, un **atributo**.

Utilizando el nodo de contexto la ruta de localización tiene los siguientes **pasos de localización**:

- Un eje
- Una prueba de nodo
- Un predicado que es opcional.

Un ejemplo de ruta de localización sería

```
child::capitulo[child::titulo]
```

Aquí el *eje* es `child`*,* seguido de doble coma como separador y el *nodo de prueba* es `capitulo`. La parte final que está dentro de los corchetes es el *predicado* significa que sólo los nodos de elementos `<capitulo>` que tienen como hijo el nodo elemento `<titulo>` serán incluidos en el *nodo seleccionado.* En este ejemplo, se incluye un único paso de localización, pero una ruta de localización puede tener varios pasos de localización, que deben de ir seguidos por el caracter "/".

Existen dos rutas de localización de XPath: **Ruta de localización relativa y la ruta de localización absoluta**. La ruta de localización absoluta siempre debe de partir de *nodo raíz,* ya que él es su nodo contexto. Una ruta de localización consiste

en una secuencia de uno o más pasos de localización separados por "/", los pasos en una ruta de localización se componen de izquierda a derecha y cada paso selecciona un conjunto de nodos relativos a un nodo contextual. Una secuencia inicial de pasos se une al paso siguiente de la siguiente forma. La secuencia inicial de pasos selecciona un conjunto de nodos relativos a un nodo de contexto; cada nodo de este conjunto se usa como nodo de contexto para el siguiente paso; los distintos conjuntos de nodos identificados por ese paso se unen obteniendo un nodo simple, nodos múltiples o posiblemente vacío.

### 2.1.3. Modelo de datos

XPath opera sobre un documento XML considerándolo un árbol y modelándolo como tal; este modelo es solamente conceptual y no impone ninguna implementación en particular.

El árbol contiene nodos, de los cuales hay siete tipos:

- Nodos raíz
- Nodos elemento
- Nodos atributo
- Nodos instrucción de procesamiento
- Nodos comentario
- Nodos texto
- Nodos espacio de nombres (*"namespace")*

Para cada tipo de nodo, hay una forma de determinar un **valor de cadena**. Para algunos tipos de nodo, el valor de cadena es parte del nodo; para otros, se calcula a partir del valor de cadena de sus nodos descendientes.

Existe el **orden de documento** que esta definida sobre todos los nodos del documento. El orden de documento depende de la precedencia de sus etiquetas en el

documento XML. El **nodo raíz** es el primer nodo. Los **nodos elemento** aparecen antes de sus hijos. Los **nodos atributo** y los **nodos espacio de nombre** de un elemento aparecen antes que los hijos del elemento. Los nodos espacio de nombre aparecen por definición antes que los nodos atributo.

Todos los nodos, salvo el nodo raíz tienen exactamente un **nodo padre**. Un nodo raíz o un nodo elemento es el padre de cada uno de sus **nodos hijo**. Los **nodos descendiente** de un nodo son los nodos hijo del nodo y los nodos descendiente de los nodos hijo del nodo. Los nodos raíz y los nodos elemento tienen una lista ordenada de nodos hijo. Dos nodos nunca comparten un nodo hijo.

#### 2.1.3.1. Nodo raíz

El nodo raíz es la raíz del árbol y solo existe uno dentro del árbol. El nodo del elemento de documento es hijo del nodo raíz. El nodo raíz tiene también como hijos los nodos instrucción de procesamiento y comentario, correspondientes a las instrucciones de procesamiento y otros comentarios.

El valor de cadena del nodo raíz es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo raíz en orden de documento.

El nodo raíz no tiene nombre expandido.

#### 2.1.3.2. Nodos elemento

Hay un nodo elemento por cada elemento en el documento. Los nodos elemento tienen un nombre expandido calculado expandiendo el **Calificativo del Nombre, QName** (*Qualified Name*) del elemento especificado en la etiqueta de acuerdo con la recomendación de espacios de nombre de XML.

Los hijos de un nodo elemento son los nodos elemento, comentario, instrucción de procesamiento y nodos texto.

El valor de cadena de un nodo elemento es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo elemento en orden de documento.

#### 2.1.3.3. Nodos atributo

Cada nodo elemento tiene asociado un conjunto de nodos atributo; el elemento es el padre de cada uno de esos nodos atributo; sin embargo, los nodos atributo no son hijos de su elemento padre.

El valor de cadena de un nodo atributo es una cadena de longitud cero.

#### 2.1.3.4. Nodos espacio de nombre

Cada elemento tiene un conjunto asociado de nodos espacio de nombre, uno para cada uno de los distintos prefijos de espacio de nombre con efecto sobre el elemento (incluyendo el prefijo xml, que está implícitamente declarado según la Recomendación de espacios de nombre de XML) y uno para el espacio de nombre por defecto. El elemento es el padre de cada uno de los nodos espacio de nombre; sin embargo, los nodos espacio de nombre no son hijos de su elemento padre.

Esto significa que los elementos tendrán un nodo espacio de nombre si:

- Para cada atributo del elemento cuyo nombre empiece con xmlns:.
- Para cada atributo de un elemento ancestro, cuyo nombre empiece con xmlns: salvo que el propio elemento o un ancestro más cercano redeclaren el prefijo.

- Para atributos xmlns, si el elemento o alguno de sus ancestros tienen dicho atributo y el valor del atributo en el más cercano de los elementos que lo tienen es no vacío.

#### 2.1.3.5. Nodos instrucción de procesamiento

Hay un nodo instrucción de procesamiento para cada instrucción de procesamiento, salvo para aquellas que aparezcan dentro de la declaración de tipo de documento.

Las instrucciones de procesamiento tienen un nombre expandido: la parte local es el destinatario de la instrucción de procesamiento; el espacio de nombre es nulo. El valor de cadena de un nodo instrucción de procesamiento es la parte de la instrucción de procesamiento, que sigue al destinatario y todo el espacio en blanco. No incluye la terminación ?>. La declaración XML no es una instrucción de procesamiento, por lo cual no hay nodo instrucción de procesamiento correspondiente a ella.

#### 2.1.3.6. Nodos comentario

Hay un nodo comentario para cada comentario, salvo para aquellos que aparezcan dentro de la declaración de tipo de documento.

El valor de cadena de los comentarios es el contenido del comentario sin incluir el inicio ‘<!–‘ ni la terminación ‘–->’.

Los nodos comentario no tienen nombre expandido.

#### 2.1.3.7. Nodos texto

Los datos tipo caracter se agrupan en nodos texto, que contiene todos los datos tipo caracter que sea posible, por tanto: un nodo texto nunca tiene un hermano inmediatamente anterior o siguiente que sea nodo texto. El valor de cadena de los nodos texto son los datos tipo carácter, del cual poseen al menos uno.

Los caracteres dentro de comentarios, instrucciones de procesamiento y valores de atributos no producen nodos texto. El espacio en blanco fuera del elemento de documento tampoco produce nodos de texto.

Por último, los nodos de texto no tienen nombre expandido .

### 2.1.4. Tipo de expresiones

Una expresión es la construcción de sintaxis más general de XPath, y de las expresiones; la ruta de localización es él tipo más importante.

Una expresión puede producir uno de los siguientes tipos:

- Conjunto de nodos
- Booleano
- Número flotante
- Texto *("String")* o cadena de caracteres

La evaluación de una expresión consiste en cinco pasos:

- Un nodo o nodo de contexto
- Dos números positivos para contener la posición y el tamaño del contexto
- Un conjunto de asignación de variables
- Una biblioteca de funciones
- Un conjunto de declaraciones de espacio de nombre aplicables a la expresión.

### 2.1.5. Ejes

Los ejes con los que cuenta XPath para moverse son:

- El eje ***child*** contiene los hijos del nodo contextual

- El eje ***descendant*** contiene los descendientes del nodo contextual; un descendiente es un hijo o el hijo de un hijo, etc.; de este modo, el eje *descendant* nunca contiene nodos atributo o nodos espacio de nombre

- El eje ***parent*** contiene el padre del nodo contextual, si lo hay.

- El eje ***ancestor*** contiene los ancestros del nodo contextual; los ancestros del nodo contextual consisten en el padre y el padre del padre, etc.; así, el eje *ancestor* siempre incluirá al nodo raíz, salvo que el nodo contextual sea el nodo raíz

- El eje ***following-sibling*** contiene todos los siguientes hermanos del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombre, el eje *following-sibling* es vacío

- El eje ***preceding-sibling*** contiene todos los hermanos precedentes del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombre, el eje *preceding-sibling* es vacío

- El eje ***following*** contiene todos los nodos del mismo documento que el nodo contextual que están después de este según el orden del documento; excluye los ancestros, así como los nodos nodos atributo y nodos espacio de nombre.

- El eje ***preceding*** contiene todos los nodos del mismo documento que el nodo contextual, que están antes de éste, según el orden del documento; excluye los ancestros, así como los nodos atributo y nodos espacio de nombre

- El eje ***attribute*** contiene los atributos del nodo contextual; el eje estará vacío a no ser que el nodo contextual sea un elemento

- El eje ***namespace*** contiene los nodos espacio de nombre del nodo contextual; el eje estará vacío, a no ser que el nodo contextual sea un elemento

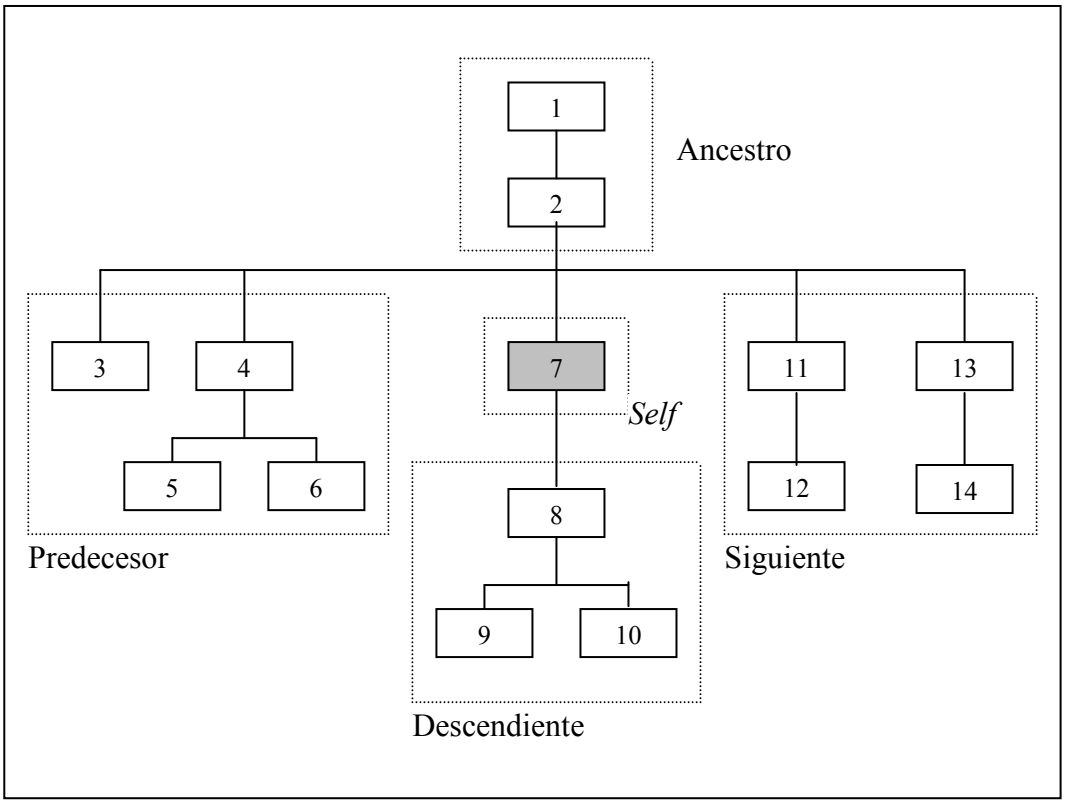
- El eje ***self*** contiene el propio nodo contextual

- El eje ***descendant-or-self*** contiene el nodo contextual y sus descendientes

- El eje ***ancestor-or-self*** contiene el nodo contextual y sus ancestros; así, el eje *ancestor-or-self* siempre incluirá el nodo raíz

A continuación, se presenta un ejemplo donde con la utilización de ejes se puede ir en cualquier dirección, a través el documento; **7** es el nodo contextual.

Fig 2. Ejemplo de Ejes de XPath



### 2.1.6. Pruebas de nodos

Cada eje tiene un **tipo principal de nodo**. Si un eje puede contener elementos, entonces el tipo principal de nodo es elemento; esto se puede resumir en:

- Para el eje ***attribute***, el tipo de nodo principal es atributo.
- Para el eje ***namespace***, el tipo de nodo principal es espacio de nombre.
- Para todos los demás ejes, el tipo de nodo principal es elemento.

Una prueba de nodo es verdadera, sólo si el tipo de nodo solicitado es el tipo principal del nodo. Por ejemplo `child::titulo` selecciona los elementos `titulo` hijos del nodo contextual, si el nodo contextual no tiene ningún hijo `titulo` seleccionará un conjunto vacío de nodos. `attibute::cantidad` selecciona el atributo `cantidad` del nodo contextual, si el nodo contextual no tiene atributo `cantidad` seleccionará un conjunto vacío de nodos.

Una prueba de nodo `*` es verdadera para cualquier nodo del tipo principal de nodo. Por ejemplo `child::*` selecciona todos los hijos del nodo contextual y `attribute::*` selecciona todos los atributos del nodo contextual.

La prueba de nodo `text()` es verdadera para cualquier nodo de texto. Por ejemplo `child::text()` seleccionará los nodos de texto hijos del nodo contextual. De la misma manera, `comment()` es verdadera para cualquier nodo comentario y la prueba de nodo `proccesing-instruction()` es verdadera para cualquier instrucción de procesamiento.

Una prueba de nodo `node()` es verdadera para cualquier nodo sin importar el tipo que sea.

### 2.1.7. Predicado

El predicado filtra un conjunto de nodos en base a un eje, para así producir un nuevo conjunto de nodos. Para cada nodo del conjunto de nodos a filtrar, se evalúa la expresión del predicado utilizando ese nodo como el nodo contextual y solo si la evaluación resulta verdadera el nodo se incluye en nuevo conjunto de nodos.

Los ejes están orientados hacia delante y hacia atrás. Un eje sólo puede contener el nodo contextual y nodos que están a continuación del nodo contextual, lo que también es llamado ir hacia delante, lo que se consigue utilizando los ejes descendientes y siguientes. O bien un nodo sólo puede contener el nodo contextual y los nodos que están antes del nodo contextual, lo que también es llamado ir hacia atrás; esto se consigue utilizando los ejes ancestros y anteriores. El eje `self` aparecerá en cualquiera de los dos, por contener el nodo contextual.

El predicado también se trabaja en relación a la posición del nodo con respecto al eje, es importante mencionar que la primera posición es la 1, ejemplos de posiciones son:

```
following-simbling::*[position()=1]
```

El nodo de prueba * seleccionará todo el nodo del hermano inmediatamente siguiente al nodo del contexto.

```
preceding-simbling::*[position()=1]
```

El nodo de prueba * seleccionará todo el nodo del hermano inmediatamente anterior al nodo del contexto, o sea el primer nodo encontrado, si el documento se recorre en orden inverso.

### 2.1.8. Funciones

Por facilidad, se agruparán las funciones, según el tipo dato que utilicen.

**Funciones de conjunto de nodos**

- número **count** ( nodo de contexto )
  Devuelve el número de nodos en el conjunto de nodos argumento
  `count(child::*[@nombre])` retorna el número de elementos hijos del nodo de contexto que tiene un atributo nombre.

- nodo de contexto **id** ( objeto )
  Selecciona elementos mediante su identificador único. El resultado es un conjunto de nodos que contienen los elementos en el documento que tengan el identificador único.
  `id("prueba")` selecciona el elemento con identificador único prueba.
  `id("prueba")/child::titulo[position() = 5]` selecciona el quinto elemento hijo de titulo con identificador único prueba.

- número **last** ()
  Retorna el número índice del último nodo correspondiente al nodo contextual.
  `child::*[position() = last() - 1]` Selecciona el penúltimo elemento hijo del nodo contextual.

- texto **local-name** ( nodo de contexto )
  Retorna la parte local del nombre del primer nodo del documento; el nodo de contexto es opcional
  La parte local del elemento `<xsl:value-of>` es `value-of`

- `texto` **`name`** `( nodo de contexto )`

  Retorna la parte local del nombre del primer nodo del documento; es un nombre total que presentará el prefijo del espacio de nombre; el nodo de contexto es opcional. Es igual que `local-name` excepto para nodos elementos y nodos atributo.

- `texto` **`namespace-uri`** `( nodo de contexto )`

  Devuelve el nombre expandido del conjunto de nodos argumento; el nodo de contexto es opcional, y si este es nulo se tomá por defecto el nodo contextual como único miembro del conjunto. Devolverá datos solo para nodos elemento y nodos atributo.

  `namespace-uri(@href)` devolverá `'http://www.wc3.org/xslt'`.

- `número` **`position`** `()`

  Retorna la posición del actual nodo de contexto según el nodo de prueba.

  `position()` retorna 1 para el primer nodo de contexto en el nodo de prueba.

**Funciones de cadena de caracteres**

- `texto` **`concat`** `( texto, texto, texto * )`

  Devuelve la concatenación de sus elementos.

  `concat ( 'con', 'c', 'at')` devuelve `'concat'`

- `booleano` **`contains`** `( texto, texto )`

  Devuelve verdadero si la primera cadena de caracteres contiene la segunda cadena de caracter y falso en otro caso. Si el segundo argumento es vacío entonces devuelve verdadero.

  `contains ( 'contains', 'ain' )` devuelve verdadero.

- texto **normalize-space** ( texto )

  Devuelve una cadena de caracteres similar a la cadena de caracteres del parámetro, pero sin los espacios en blanco que tenga al inicio o al final. Si es nulo, utiliza el nodo contextual.

  `normalize-space (' cadena de ')` devuelve 'cadena de'.

- texto **string** ( nodo de prueba )

  Convierte el nodo prueba en una cadena de caracteres. Si se trata de número, devuelve los valores '0', 'infinity', '-infinity', o '1234'. Si es booleano devuelve 'false' o 'true'. Si el argumento es nulo, se utiliza el nodo contextual.

- número **string-length**( texto )

  Devuelve la cantidad de caracteres que tiene la cadena de caracteres. Si es nulo, utiliza el nodo contextual.

  `string-length ( 'longitud' )` devuelve ocho.

- texto **substring** ( texto, texto, texto )

  Devuelve la subcadena de la primera cadena de caracteres del argumento que empieza en la posición del segundo elemento y tiene una longitud del tercer elemento. La posición del primer caracter es el 1. Si en algún número se le ponen valores no enteros, entonces utilizará redondeo para su uso.

  `substring ('1999/04/01',3,5)` devuelve '99/04'

- texto **substring-after** ( texto, texto )

  Devuelve la subcadena de la primera cadena de caracteres del argumento que esta después de la primera aparición de la segunda cadena de caracteres del argumento, o vacío si la primera cadena de caracteres no esta en la segunda cadena de caracteres. Si el segundo argumento es una cadena de caracteres vacía, el resultado será una cadena de caracteres vacía.

  `substring-after('1999/04/01', '/')` devuelve '04/01'

- `texto` **`substring-before`** `( texto, texto )`

  Devuelve la subcadena de la primera cadena de caracteres del argumento que está antes de la primera aparición de la segunda cadena de caracteres del argumento, o vacío si la primera cadena de caracteres no esta en la segunda cadena de caracteres. Si el segundo argumento es una cadena de caracteres vacía, el resultado será una cadena de caracteres vacía.

  `substring-before('1999/04/01', '/')` devuelve `'1999'`

- `texto` **`traslate`** `( texto, texto, texto )`

  Devuelve la cadena de caracteres del primer argumento con las apariciones de caracteres del segundo argumento substituidas por los caracteres en las posiciones correspondientes del tercer parámetro.

  `traslate ('abcdefg', 'aceg', 'ACE' )` devuelve `'AbCdEf'.`

#### 2.1.1.1 Funciones booleanas

- `booleano` **`boolean`** `( objeto )`

  Si el objeto es un número, devuelve verdadero cuando no es cero o nulo.

  Si se trata de un conjunto, es verdadero si no esta vacío

  Si es una cadena de caracteres, es verdadero si tiene una longitud mayor de cero.

  `boolean ('abcdefg')` devuelve verdadero

- `booleano` **`false`** `()`

  Devuelve falso.

- `booleano` **`lang`** `( texto )`

  Devuelve verdadero si en el nodo contextual hay una definición `xml:lang` que contenga el valor del parámetro como inicio de su valor.

  `lang ('ab')` es verdadero para

  ```
  <nombre xml:lang= "ab"/>
  ```

```
<nombre xml:lang= ”ab”><decrip/></nombre>
<nombre xml:lang= ”Abuelo”/>
```

- `booleano` **`not`** `( booleano )`

  Devuelve verdadero, si el parámetro es falso y falso en otro caso.

- `booleano` **`true`** `()`

  Devuelve verdadero.

#### 2.1.1.2 Funciones numéricas

- `número` **`ceiling`** `( número )`

  Devuelve el entero mayor al `número` del parámetro. Si el valor es nulo, devuelve nulo.

  `ceiling(2.9)` devuelve 3 y `ceiling(-1.6)` devuelve -1

- `número` **`floor`** `( número )`

  Devuelve el entero menor al `número` del parámetro. Si el valor es nulo, devuelve nulo.

  `floor(2.9)` devuelve 2 y `floor(-1.6)` devuelve -2

- `número` **`number`** `( objeto )`

  Si el objeto es booleano, devuelve uno si es verdadero y cero si es falso.

  Si el objeto es una cadena de caracteres, la convierte en un número o en nulo.

  Si es un conjunto de nodos, se le aplica la función `texto` y luego trabaja como si fuera cadena de caracteres.

  Si se omite el parámetro, se asume que es el nodo contextual.

  `number (' -25.45 ')` retorna –25.45

- número **round** ( número )

  Devuelve el entero más próximo al número del parámetro; si el valor a redondear es 5, entonces devolverá el valor mayor. Si el valor es nulo devuelve nulo.

  round(2.9) devuelve 3, round(-1.6) da –2, y round(2.5) devuelve 3.

- número **sum** ( nodo de contexto )

  Devuelve la suma de todos los nodos del conjunto de nodos argumento.

## 2.2. Sintaxis

XPath tiene dos tipos de sintaxis para escribir expresiones: sintaxis abreviada y sintaxis sin abreviar. Todas las expresiones pueden ser escritas en sintaxis sin abreviar, pero no todas pueden ser expresadas en sintaxis abreviada; con ambas sintaxis se puede considerar la ruta de localización relativa o absoluta.

Para comprender de manera más fácil la sintaxis, se presenta un ejemplo que se tomará como base de la explicación.

```
<?xml version="1.0" ?>
<!—Archivo que sirve para utilizarlo en la comprensión de sintaxis
de XPath.-->
<libro>
   <titulo>XSL</titulo>
   <capitulos>
      <capitulo numero="1"titulo="Conceptos Generales" />
      <capitulo numero="2"titulo="XPath, Lenguaje de Ruta de XML">
         <seccion numero="1">
            <parrafo numero="1">Para que sirve XPath.</parrafo>
            <parrafo numero="2">Como trabaja XPath.</parrafo>
            <parrafo numero="3">Modelo de datos.</parrafo>
            <parrafo numero="4">Funciones.</parrafo>
         </seccion>
```

```
         <seccion numero="2">
            <parrafo numero="1">Sintaxis sin abreviar.</parrafo>
            <parrafo numero="2">Sintaxis abreviada.</parrafo>
         </seccion>
         <seccion numero="3" />
        </capitulo>
        <capitulo numero="3">XSLT</capitulo>
        <capitulo numero="4">XSL-FO</capitulo>
   </capitulos>
   <apendices>
   </apendices>
</libro>
```

### 2.2.1. Sintaxis sin abreviar

#### 2.2.1.1. Sintaxis sin abreviar con localización relativa

Se presenta la definición de la sintaxis sin abreviar, utilizando localización relativa, y se hará referencia al ejemplo anterior para una mejor comprensión.

El nodo de contexto actual está determinado por la ruta de localización relativa.

```
child::parrafo
```

Esta expresión selecciona los elementos hijos de <parrafo> del nodo del contexto. En el ejemplo, si el nodo del contexto es <libro> devuelve un valor vacío, porque el elemento <libro> no tiene hijos <parrafo>, pero si el contexto esta en <seccion> del capitulo 2 con atributo de número 1, entonces el resultado será de cuatro nodos de párrafo.

```
child::*
```

La expresión child::* selecciona todos los elementos hijos del nodo de contexto. Si el nodo de contexto tiene a su vez como hijo otro elemento nodo, estos

nodos no son incluidos en el nodo resultante, porque solo los hijos serán parte del resultado no todos los descendientes. Por ejemplo, si el nodo de contexto está en <libro> entonces el nodo resultado tendrá <titulo>, <capitulos> y <apendices>.

```
child::text()
```

Selecciona todos los nodos de texto que son hijos del nodo de contexto. Si el nodo de contexto fuera <libro> un conjunto vacío será devuelto, pero si el contexto esta en <seccion> 2 del <capitulo> 2, entonces el nodo resultante tendría el texto "Para que sirve XPath." como resultado.

```
child::node()
```

La expresión selecciona todos los nodos hijos del nodo de contexto, cualquiera que sea su tipo de nodo. Note que a diferencia de child::* selecciona sólo los elementos del conjunto de nodos, mientras que child::node() también retorna el nodo espacio de nombre y el nodo atributo.

```
attribute::numero
```

Retorna el nodo atributo con nombre numero bajo el contexto del nodo. Según nuestro ejemplo, si el nodo contexto fuera <parrafo>, <seccion> o <capitulo> devolverá un conjunto de nodos; si el contexto fuera otro, devuelve vacío.

```
attribute::*
```

Selecciona todos los atributos nodos del nodo de contexto. Si el nodo de contexto esta en <capitulo> del capítulo 1 entonces el conjunto de nodo va a tener dos nodos representando él numero y titulo relacionados al capítulo 1.

```
descendant::parrafo
```

Esta ruta de localización selecciona todos los elementos <parrafo> descendientes del nodo de contexto. Si el nodo de contexto estuviera en <libro> o <capitulos>, devolvería un conjunto de nodo conteniendo un total de seis nodos, uno por cada uno de los <parrafo> del documento. Si el nodo contexto estuviera en la primera <seccion> del Capitulo 2, el conjunto de nodo contiene cuatro nodos conteniendo nodos <parrafo>.

```
ancestor:capitulo
```

Es una ruta de localización que selecciona los elementos nodo <capitulo>, que son ancestros del nodo de contexto. Si por ejemplo en contexto estuviera en <capitulos> devolvería un nodo vacío. Pero si el contexto estuviera en <seccion> devuelve él <capitulo>, del cual es parte esta <seccion>

```
ancestor-or-self:capitulo
```

Esta ruta de localización selecciona los elementos nodo <capitulo>, que son ancestros del nodo de contexto, pero si el nodo contexto es <capitulo>, también debe de ser seleccionado. Sólo se diferencia de los ejemplos anteriores, si el nodo contextual es <capitulo>.

```
descendant-or-self::seccion
```

Ahora la ruta de localización selecciona los elementos nodo <seccion>, que son descendientes del nodo de contexto, pero si el nodo contexto es <seccion>, también debe de ser seleccionado. Si el nodo de contexto está en <capitulo> del capitulo 2, el nodo debe de regresar tres nodos <seccion>, ya que son tres hijos. Si el nodo contextual estuviera en <capitulo> devolvería los mismos tres nodos,; solo que ahora se trataría de tres nietos. Por otro lado, si el nodo contextual estuviera en <capitulo> del capitulo 3 devolvería vacío.

```
self::parrafo
```

Devuelve la ruta de localización si el nodo de contexto es <parrafo> y vacío en otro caso.

Hasta ahora se han mostrado sólo ejemplos simples, que contiene localización en un solo paso. Sin embargo, la ruta de localización de XPath puede contener más de un paso de localización.

```
child::capitulo/descendant::parrafo
```

La ruta de localización incluye dos pasos de localización. Selecciona el elemento <parrafo> descendiente de los elementos <capitulo> hijos del nodo de contexto.

```
child::*/child::parrafo
```

La ruta de localización incluye ahora múltiples pasos de localización. Selecciona todos los <parrafo> nietos del nodo de contexto. Si el nodo de contexto esta en <capitulos> devuelve un conjunto vacío. Pero si el nodo de contexto esta en <capitulo> de capitulo 2, el conjunto de nodo tendrá seis nodos <parrafo> hijos de <seccion> que a su vez es hijo del nodo de contexto.

A continuación, se presenta la ruta localización agregándole predicado, que es la utilización completa, que incluye eje, nodo de prueba y predicado. Es importante recordar que sólo el predicado puede ser opcional. El predicado es el código contenido dentro de los corchetes.

```
child::parrafo[position()=1]
```

Selecciona el primer elemento <parrafo> hijo del nodo contexto; esto por medio de la función position(). Si el nodo contexto está en el primer

elemento nodo <seccion> del el capitulo 2, el conjunto de nodo retornará el primer <parrafo> que es hijo del elemento <seccion>.

```
child::parrafo[position()=last()]
```

Selecciona él ultimo elemento <parrafo> hijo del nodo contexto; aquí se usa la función position() en combinación con la función last(), para conseguir la última posición del elemento hijo del nodo de contexto.

```
child::parrafo[position()=last()-1]
```

Selecciona el penúltimo elemento <parrafo> hijo del nodo contexto.

En los ejemplos anteriores, el predicado devolvía un solo nodo, pero también puede devolver múltiples nodos como se muestra a continuación.

```
child::parrafo[position()>1]
```

Selecciona todos los hijos <parrafo> del nodo contextual, sin incluir al primero. Para que no devuelva nulo debe de tener mas de un hijo <parrafo>. Si el nodo de contexto se encuentra en el primer elemento <seccion> del capítulo 2, entonces devolverá un conjunto de nodos de tres <parrafo>, ya que el primero no esta incluido.

```
following-sibling::capitulo[position()=1]
```

Selecciona el siguiente <capitulo> hermano del nodo de contexto. Si el contexto se encuentra en el capítulo 1, retornará el capítulo 2.

```
preceding-sibling::capitulo[position()=1]
```

Selecciona el anterior <capitulo> hermano del nodo de contexto. Si el contexto se encuentra en el capitulo 3, retornará el capitulo 2. Recuerde que la función position() es relativa al eje que se esta trabajando.

```
child::capitulos/child::capitulo[position()=2]/child::section[posi
tion()=2]
```

Esta ruta de localización envuelve tres pasos de localización, que contiene dos predicados. Aquí se puede notar como este tipo de localización puede ser muy largo. La ruta de localización selecciona el segundo elemento <seccion> hijo del segundo elemento <capitulo> hijo del elemento <capitulos>. Para nuestro ejemplo no será vacío, únicamente cuando el nodo contextual sea <libro>.

Dentro del predicando, también es posible usar filtros basados en el tipo o valor de los atributos.

```
child::capitulo[attribute::numero]
```

Esta ruta de localización selecciona todos los nodos elementos que representan al elemento <capitulo>, los cuales son hijos del nodo de contexto y también posee un atributo numero. Si el nodo de contexto está en el elemento <capitulos> devolverá los cuatro elementos <capitulo> que son hijos del nodo contexto; esto es porque todos los hijos contienen un atributo numero.

```
child::capitulo[attribute::numero][position()=3]
```

El uso de dos predicados consigue hacer una selección más precisa que la del ejemplo anterior. Si el nodo de contexto está en el elemento <capitulos> devolverá el elemento <capitulo> 3.

```
child::capitulo[attribute::titulo="XPath"]
```

Esta ruta de localización muestra el uso del predicado para seleccionar un elemento con un valor de atributo en particular. Si el nodo de contexto está en `<capitulos>` note que este tiene cuatro hijos `<capitulo>` y cada uno de ellos tiene un atributo `titulo`, pero solo uno de ellos tiene como valor de título `"XPath"`, por lo tanto, sólo este nodo será parte del conjunto de nodos resultado.

```
child::capitulo[position()=3][attribute::numero]
```

La nueva ruta de localización selecciona el `<capitulo>` hijo de nodo de contexto que esta en la posición 3, y además este tercer capítulo debe de tener un elemento `numero` para formar parte del conjunto de nodos resultado.

```
child::capitulo[child::titulo="Conceptos Generales"]
```

La ruta de localización selecciona los hijos `<capitulo>` del nodo de contexto que tengan uno o mas hijos cuyo `titulo` sea `Conceptos Generales`. Si el nodo contextual estuviera en `<capitulos>`, el cual tiene cuatro hijos `<capitulo>` pero como ninguno de ellos tiene como hijo `<titulo>` devuelve un conjunto vacío, para que devuelva nodos debe de cambiarse el predicado a la siguiente forma: `[attribute::titulo="Conceptos Generales"]`

Dentro del predicado, puede hacerse uso de los conectores lógicos `and` y `or` para hacer combinaciones de predicados.

```
child::*[self::capitulo or self::apendice][position()=last()]
```

Esta ruta de localización presenta la opción de que el primer predicado puede ser un hijo `<capitulo>` o un hijo `<apendice>`, cualquiera de ellos que aparezca al final del documento. De esta misma manera, pueden ser utilizados `and` y `or` para conseguir un conjunto de nodos resultado que contenga más de un nodo.

#### 2.2.1.2. Sintaxis sin abreviar con localización absoluta

Para la localización absoluta, el nodo de contexto siempre será el nodo raíz; por eso, este tipo de sintaxis es considerado un caso especial de localización relativa. Toda ruta de localización absoluta debe de iniciar con el caracter diagonal **"/"** que indica que es absoluto, y que selecciona el nodo raíz del documento, que es el padre del elemento del documento. Por ejemplo:

```
/child::*
```

Esta ruta de localización selecciona todos los nodos elementos que son hijos del nodo raíz. Para un documento de XML bien formado, únicamente retornará un valor, que será el elemento del documento, que para nuestro ejemplo será <libro>.

```
/descendant::parrafo
```

La ruta de localización selecciona todos los elementos <parrafo> del documento, o sea que selecciona todos los elementos <parrafo>, que son descendientes del nodo raíz. Para nuestro ejemplo, devolverá seis nodos.

```
/descendant::seccion/child::parrafo
```

Esta ruta de localización selecciona todos los elementos <parrafo> que son hijo de <seccion> descendientes del nodo raíz. Para el ejemplo devolverá seis nodos, que es lo mismo que anterior, porque todos los <parrafo> están dentro de <seccion>, pero si algún <parrafo> fuera hijo de <apendice> el resultado sería diferente, ya que este <parrafo> no aparecería en esta ruta de selección.

```
/descendant::capitulo[postion()=3]
```

La ruta de localización contiene el tercer nodo elemento <capitulo> de un documento. Para nuestro ejemplo, tendrá un atributo numero con valor "3".

### 2.2.2. Sintaxis abreviada

Anteriormente se presentó la forma más compleja de localización de ruta, que puede volverse muy larga y poco práctica; por eso, hay una sintaxis mas concisa para expresar lo mas comúnmente usado en las rutas de localización, llamada sintaxis abreviada. Es importante mencionar que hay rutas que solo pueden expresarse con la sintaxis sin abreviar.

#### 2.2.2.1. Sintaxis abreviada con localización relativa

Uno de los ejes mas comúnmente usados es el eje *child.* En la sintaxis abreviada, la palabra reservada `child::` puede ser omitida, porque por defecto utiliza el eje `child`. Por ejemplo:

```
parrafo
```

Selecciona los nodos elemento <`parrafo`> hijos del nodo de contexto. Esta sintaxis es equivalente a `child::parrafo`.

```
*
```

Ahora selecciona todos los nodos hijos del nodo de contexto, que es equivalente de `child::*`.

```
Text()
```

La ruta de localización selecciona todos los elementos texto hijos del nodo de contexto. `Text()` es equivalente a `child::Text()`.

```
.
```

Selecciona al nodo de contexto en sí. Su equivalente en sintaxis sin abreviar es `self::node()`.

```
..
```

Selecciona al padre del nodo de contexto. Su equivalente en sintaxis sin abreviar es `parent::node()`.

Tal como el eje *child* es usado por defecto en la abreviación, así el eje *attribute* es abreviado con el símbolo **@,** como se presenta en los siguientes ejemplos.

```
@numero
```

Selecciona el atributo `numero` de el nodo de contexto; su equivalente en sintaxis sin abreviar es `attribute::numero`.

```
@*
```

Esta ruta de localización selecciona todos lo atributos del nodo de contexto; esto es equivalente a `attribute::*`.

```
..@numero
```

Selecciona el atributo `numero`, padre del nodo de contexto. Su equivalente en sintaxis no abreviada es `parent::node()/attribute::numero`.

En cuanto al predicado, la función `position()` es la función por defecto, por ejemplo:

```
parrafo[1]
```

El predicado `[1]` implica por defecto la función `position()` para seleccionar el primer `<parrafo>` hijo del nodo de contexto. La sintaxis sin abreviar equivalente es `child::parrafo[position()=1]`.

```
parrafo[last()]
```

Selecciona el último `<parrafo>` hijo del nodo de contexto. La sintaxis sin abreviar equivalente es `child::parrafo[position()=last()]`.

```
parrafo[@numero="2"]
```

Esta ruta de localización selecciona todos los `<parrafo>` hijo del nodo de contexto que tengan un atributo `numero` con valor "2". La sintaxis sin abreviar equivalente es `child::parrafo[attribute::numero='2']`.

```
*/parrafo
```

Selecciona todos los nodos `<parrafo>` nietos del nodo de contexto. Su equivalente en sintaxis sin abreviar es `child::*/child::parrafo`.

```
capitulo[titulo="XPath"]
```

La ruta de localización selecciona el `<capitulo>` hijo del nodo de contexto que tiene uno o mas elementos hijos `<titulo>` con valor "XPath". Su sintaxis sin abreviar equivalente es `child::capitulo[child::titulo= 'XPath']`.

```
capitulo/seccion[1]/parrafo[3]
```

Selecciona el tercer hijo del nodo elemento `<parrafo>` del primer nodo elemento `<seccion>` que es hijo del nodo `<capitulo>`, que a su vez es hijo del nodo de contexto. La sintaxis sin abreviar equivalente es `child::capitulo/child::seccion[position()=1]`.

```
.//parrafo
```

La doble diagonal es otra abreviatura que se usa el eje *descendant* en lugar del eje *child*. Por lo tanto `.//parrafo` selecciona todos los nodos `<parrafo>` descendientes del nodo de contexto. La sintaxis no abreviada equivalente es `self::node()/descendant-or-self::node()/child::parrafo`.

```
capitulo//parrafo
```

Esta ruta de localización selecciona el nodo `<parrafo>` descendiente del nodo `<capitulo>` que es hijo del nodo de contexto. Su sintaxis sin abreviar equivalente es `child::capitulo/descendent::parrafo`.

```
capitulo[@numero and @titulo]
```

El predicado `[@numero and @titulo]` sólo permite `<capitulo>` hijos del nodo de contexto, que tiene ambos atributos de número y título. Su sintaxis sin abreviar equivalente es `child::capitulo[attribute::numero and attribute::titulo]`.

#### 2.2.2.2. Sintaxis abreviada con localización absoluta

Como ya se mencionó, el nodo de contexto para la localización absoluta es el nodo raíz.

```
/
```

La ruta de localización selecciona el nodo raíz; la diagonal es igual para la sintaxis abreviada que para la sin abreviar.

```
//parrafo
```

Selecciona todo los nodos <parrafo> que son descendientes del nodo raíz. Selecciona todos los nodos <parrafo> del documento. Su equivalente en sintaxis sin abreviar es /descendant::parrafo.

```
/libro/capitulo[2]/seccion[1]/parrafo[3]
```

La ruta de localización selecciona el tercer parrafo de la primera seccion el segundo capitulo del elemento <libro>, que es hijo del nodo raíz. En sintaxis sin abreviar, su equivalente es /child::libro/child::capitulo [position()=2]/child::seccion[position()=1]/child::parrafo [position()=3].

```
//capitulo/seccion
```

Selecciona los nodos <seccion> que tiene a <capitulo> como su nodo padre y que a la vez son descendientes de la raíz. Su equivalente en sintaxis sin abreviar es /descendant::capitulo/child::seccion.

```
//capitulo[@number="2"]
```

Selecciona todos los nodos <capitulo> descendientes del nodo raíz con un atributo numero con valor "2". Su sintaxis equivalente en sintaxis sin abreviar es /descendant::capitulo[attribute::numero='2']

#### 2.2.2.3. Errores comunes

Los errores más comunes se dan por la duplicación de sintaxis de una sin abreviar a una abreviada, por lo que las rutas de localización deben de ser construidas

con mucho cuidado para asegurar que se hace la selección que se pretende. Se debe tener especial cuidado con:

- La ruta de localización `//parrafo[1]` no significa lo mismo que `/descendant::parrafo[1]`. Esta última sintaxis selecciona el primer nodo `<parrafo>` descendiente, mientras que la primer sintaxis selecciona todos los nodos descendientes `<parrafo>`, que sean el primer hijo de sus padres.

- Es importante estar seguro del eje de orden de trabajo en el documento, y usar con cuidado el eje reverso.

- Cuando use la función `position()`, recuerde que el primer nodo seleccionado retorna uno, el resultado de esta función es diferente de cero.

- Es necesario distinguir en el código los nodos elemento basados en el valor de sus atributos usando `child::capitulo[attibute::numero]`, mientras que para la localización de la ruta que selecciona el atributo en sí se debe utilizar `child::capitulo/attibute::numero`.

## 2.3. Ejemplo de XSLT

Para los ejemplos, se necesita de un procesador de XSLT y se utilizará el procesador de *Michael Kay, Instant Saxon* XSLT, el cual debe de instalarse y puede obtenerse de Internet en:

`http://users.iclway.co.uk/mhkay/saxon/index.html`

Para saber como navegar, a través de un documento de XML por medio de XPath, se presenta un ejemplo con un árbol que tiene suficientes niveles en su

jerarquía para aprender como trabaja XPath. A continuación, se presenta el documento `fuente.xml`, el cual será utilizado como base del ejemplo.

```
<?xml version="1.0" ?>
<libro>
 <titulo>Lenguaje Extensible de Estilo de Páginas XSL</titulo>
 <capitulos>
  <capitulo>
   <capituloNo>1</capituloNo>
   <capituloNombre>Conceptos generales</capituloNombre>
   <capituloDescrip>Marco teórico, historia</capituloDescrip>
   <capituloSeccions>
    <capituloSeccion>Lenguaje Extensible de Marcas (XML)
    </capituloSeccion>
    <capituloSeccion>Definición de Tipos de Documento (DTD)
    </capituloSeccion>
    <capituloSeccion>Espacios de nombre</capituloSeccion>
    <capituloSeccion>Que es Lenguaje Extensible de Estilo de
    Páginas (XSL)</capituloSeccion>
    <capituloSeccion>Herramientas para XSL</capituloSeccion>
    <capituloSeccion>Otras opciones para transformar XML
    </capituloSeccion>
   </capituloSeccions>
  </capitulo>
  <capitulo>
   <capituloNo>2</capituloNo>
   <capituloNombre>XPath Lenguaje de Ruta de XML
   </capituloNombre>
   <capituloDescrip>Muestra cómo navegar y buscar partes de un
   documento de XML</capituloDescrip>
    <capituloSeccions>
    <capituloSeccion>Introducción a XPath</capituloSeccion>
    <capituloSeccion>Sintaxis</capituloSeccion>
    <capituloSeccion>Ejemplo</capituloSeccion>
     </capituloSeccions>
  </capitulo>
  <capitulo>
   <capituloNo>3</capituloNo>
   <capituloNombre>XSLT</capituloNombre>
   <capituloDescrip>Muestra cómo crear páginas de estilo con
   suficiente funcionalidad para muchas aplicaciones
   </capituloDescrip>
   <capituloSeccions>
    <capituloSeccion>Introducción</capituloSeccion>
    <capituloSeccion>Proceso de XSLT</capituloSeccion>
    <capituloSeccion>Elementos</capituloSeccion>
```

```
      <capituloSeccion>Funciones</capituloSeccion>
      <capituloSeccion>SXLT modular</capituloSeccion>
      <capituloSeccion>Ejemplo</capituloSeccion>
     </capituloSeccions>
   </capitulo>
   <capitulo>
    <capituloNo>4</capituloNo>
    <capituloNombre>XSL-FO</capituloNombre>
    <capituloDescrip>Para hacer páginas de estilo a través de la
    semántica</capituloDescrip>
    <capituloSeccions>
     <capituloSeccion>Técnicas para aplicar estilo a XML
     </capituloSeccion>
     <capituloSeccion>Introducción a CSS</capituloSeccion>
     <capituloSeccion>Aplicando CSS a XML</capituloSeccion>
     <capituloSeccion>Introducción de XSL-FO</capituloSeccion>
     <capituloSeccion>Estructura de documentos XSL-FO
     </capituloSeccion>
     <capituloSeccion>Ejemplo</capituloSeccion>
    </capituloSeccions>
   </capitulo>
  </capitulos>
</libro>
```

Para el primer ejemplo, se usara una hoja de estilo simple, sin filtrar la información del documento `fuente`. El documento de estilo se llamará `completo.xsl`, y debe de ser salvado en el mismo directorio donde instaló el *Instant Saxon.*

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

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Al aplicar la página de estilo al documento de entrada de XML, como resultado debe de desplegar el contenido de cada elemento del documento, debiendo aplicar la plantilla por defecto, a cada elemento del documento fuente.

Para crear esta salida, es necesario usar *Instant Saxon,* abrir una ventana de *MS DOS,* luego debe de colocarse en el directorio de instalación de *Instant Saxon* y escribir el siguiente comando.

```
saxon fuente.xml completo.xsl > completo.txt
```

Es importante recordar que los archivos `fuente.xml` y `completo.xsl` deben de estar en el directorio de *Instant Saxon,* y ahí será donde se generará `completo.txt`.

A continuación, se presenta el documento `completo.txt` obtenido.

Fig 3. Ejemplo de XSLT con XPath, completo.txt

```
<?xml version="1.0" encoding="utf-8"?>
   Lenguaje Extensible de Estilo de Páginas XSL


      1
      Conceptos generales
      Marco teórico, historia

        Lenguaje Extensible de Marcas (XML)
        Definición de Tipos de Documento (DTD)
        Espacios de nombre
        Que es Lenguaje Extensible de Estilo de Páginas (XSL)
        Herramientas para XSL
        Otras opciones para transformar XML


      2
      XPath Lenguaje de Ruta de XML
      Muestra cómo navegar y buscar partes de un documento de XML

        Introducción a XPath
        Sintaxis
        Ejemplo


      3
      XSLT
      Muestra cómo crear páginas de estilo con suficiente
funcionalidad para muchas aplicaciones
```

```
        Introducción
        Proceso de XSLT
        Elementos
        Funciones
        SXLT modular
        Ejemplo


      4
      XSL-FO
      Para hacer páginas de estilo a través de la semántica

        Técnicas para aplicar estilo a XML
        Introducción a CSS
        Aplicando CSS a XML
        Introducción de XSL-FO
        Estructura de documentos XSL-FO
        Ejemplo
```

Se puede notar que el archivo de salida es apenas una cosa medio bonita y que tiene muchas líneas en blanco. En el siguiente capítulo, se verá cómo manejar estos espacios. El propósito aquí es generar nuestro primer archivo de salida, sin importar que esté sea muy simple.

Nótese que el contenido de cada elemento del documento fuente es reproducido. Es notable que el archivo empieza con la declaración de un archivo de XML, a pesar de que el documento generado fue un `*.txt`.

El trabajo que hace la declaración `<xsl:templete match="/">` es encontrar el valor de la ruta de localización, que en este caso es la raíz del documento fuente. La declaración `<xsl:apply-templetes/>` selecciona cuál de los siguientes elementos será parte del documento de salida, utilizando por defecto la plantilla de XSLT cuando no se determina una plantilla específica.

A continuación, se muestra cómo crear una plantilla por defecto, para lo cual es necesario hacer una selección de elementos y así poder quedar libres de datos no

deseados en nuestro documento de salida. Se escribirá una plantilla para cada nodo del documento fuente. Este nuevo documento de XSLT será llamado `todo.xsl`.

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

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*">
<!-- Esta plantilla coincide con todos los elementos, menos la
raiz que coincide con la primer plantilla.
Esta plantilla no genera ninguna salida pues esta vacía.
-->
</xsl:template>

</xsl:stylesheet>
```

Ejecutando el siguiente comando.

```
saxon fuente.xml todo.xsl > todo.txt
```

A continuación, se presenta el documento `todo.txt` obtenido.

```
<?xml version="1.0" encoding="utf-8"?>
```

Este resultado se debe a que el nodo raíz coincide con la primer plantilla, mientras que los otros nodos coinciden con la segunda plantilla y por tener esta plantilla mayor prioridad no aplica ninguna plantilla.

En este momento, es posible hacer un ejemplo basado en el contexto, para lo cual se debe seleccionar una parte específica del documento. El código necesario, para realizar esto, se presenta en el archivo `Nombres.txt`.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="*">
<!-- Esta plantilla coincide con todos los elementos, menos la
raiz que coincide con la primer plantilla.
Esta plantilla no genera ninguna salida pues esta vacia, pero por
priorioridad las siguientes plantillas pueden hacerlo. -->
</xsl:template>

<xsl:template match="libro">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="capitulos">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="capitulo">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="capituloNombre">
  <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Dado que solo `<capituloNombre>` tiene una plantilla de salida definida; el documento resultante presenta solo los Nombre de los capítulos.

Fig 4. Ejemplo de XSLT con XPath, nombre.txt

```
<?xml version="1.0" encoding="utf-8"?>



    Conceptos generales



    XPath Lenguaje de Ruta de XML



```

```
    XSLT




    XSL-FO
```

La forma en la que trabaja es asi: mueve el contexto del elemento <libro> al elemento <capitulos>, luego a <capitulo>; una vez puesto el contexto en el elemento <capituloNombre>, puede usar el elemento <xsl:value-of> que hace que despliegue cada nombre de capítulo.

Por último y para ver un ejemplo más completo, se agregará el número y la descripción del capítulo.

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

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*">
</xsl:template>

<xsl:template match="libro">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="capitulos">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="capitulo">
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="/libro/capitulos/capitulo/capituloNo">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="/libro/capitulos/capitulo/capituloNombre">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="/libro/capitulos/capitulo/capituloDescrip">
  <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Lo que resultará en el archivo siguiente:

Fig 5. Ejemplo de XSLT con XPath, descripción.txt

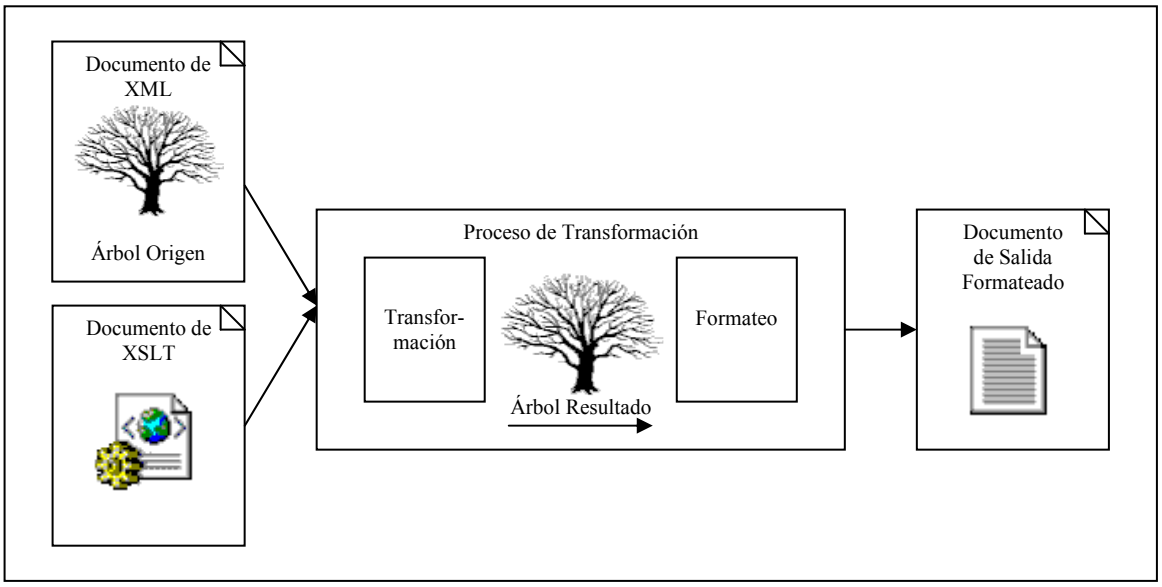
```
<?xml version="1.0" encoding="utf-8"?>



    1
    Conceptos generales
    Marco teórico, historia



    2
    XPath Lenguaje de Ruta de XML
    Muestra como navegar y buscar partes de un documento de XML



    3
    XSLT
    Muestra cómo crear páginas de estilo con suficiente
funcionalidad para muchas aplicaciones



    4
    XSL-FO
    Para hacer páginas de estilo a través de la semántica
```

# 3. XSLT

## 3.1. Modelo de procesamiento de datos

El procesamiento consiste en tomar un archivo de XML denominado origen, junto con la especificación de XSL,para hacer la transformación; el procesamiento se hace considerando al archivo de origen como un árbol y transformándolo, según dicte el documento de XSL, seobtiene un árbol resultado; a partir de este nuevo árbol, se genera el archivo de salida con un formato específico.

Fig. 6. Modelo de procesamiento de datos de XSLT




## 3.2. Procesamiento de un documento

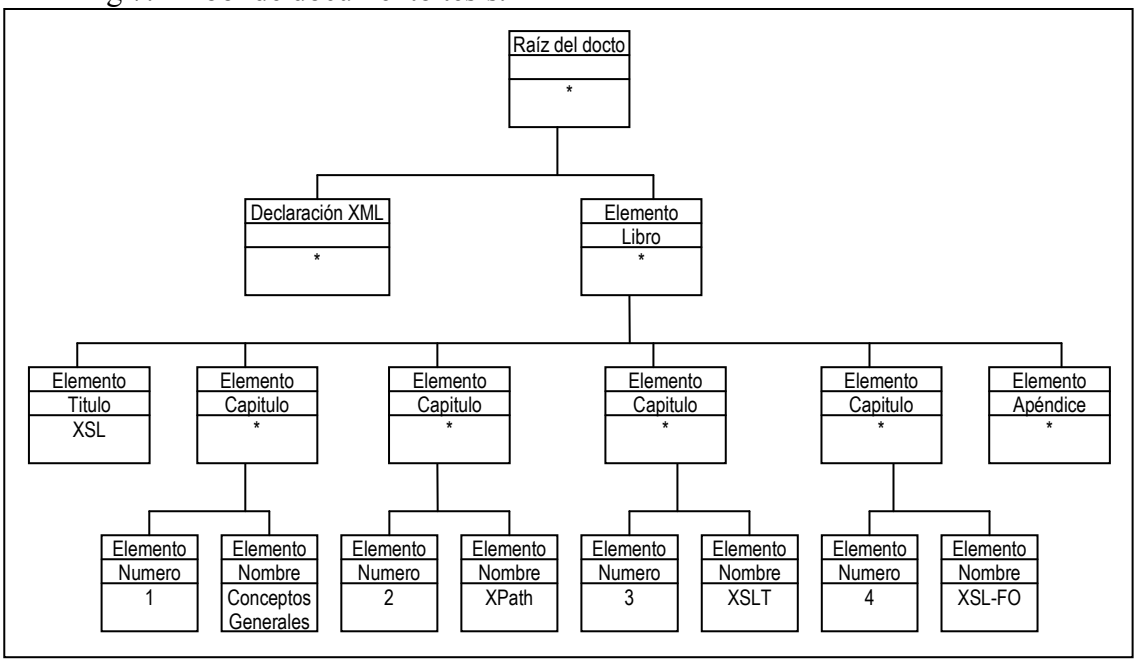
Tomando como ejemplo el siguiente documento, llamado tesis.xml.

```
<?xml version="1.0" ?>
<libro>
   <titulo>XSL</titulo>
```

```
   <capitulo>
      <numero>1</numero>
      <nombre>Conceptos generales</nombre>
   </capitulo>
   <capitulo>
      <numero>2</numero>
      <nombre>XPath</nombre>
   </capitulo>
   <capitulo>
      <numero>3</numero>
      <nombre>XSLT</nombre>
   </capitulo>
   <capitulo>
      <numero>4</numero>
      <nombre>XSL-FO</nombre>
   </capitulo>
   <apendice></apendice>
</libro>
```

La representación de un árbol para el documento anterior, basado en la estructura y el contenido, es:

Fig 7. Árbol de documento tesis.xml



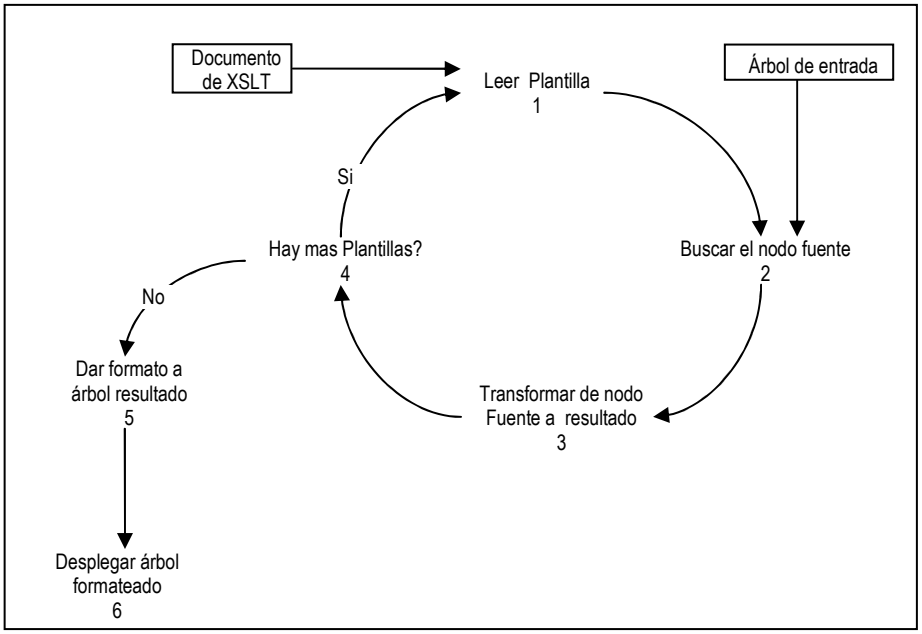

Cada nodo es descrito por un bloque de tres rectángulos. El rectángulo superior es el tipo del nodo, con el nombre del nodo en el siguiente rectángulo, y por

último el tercer rectángulo contiene un asterisco, si el nodo contiene elementos y texto para los contenidos.

La parte más alta del árbol es el nodo raíz o raíz del documento. No lo confunda con el elemento raíz o elemento documento de XML. La raíz del documento es la base del documento y tiene al elemento documento como hijo; en este caso el elemento libro. El elemento libro tiene a su vez varios hijos.

Una vez que se tiene la estructura del árbol, es posible empezar a poblar y procesar, para lo cual se debe de seguir el siguiente flujo.

Fig 8. Flujo de Procesamiento de Datos.




Antes de empezar el procesamiento tanto el documento de entrada como las especificaciones de XSLT deben de estar cargadas en memoria. A partir de las especificaciones de XSLT, se lee cada plantilla; dicha plantilla es buscada dentro del árbol del archivo fuente, luego se trasforma la información encontrada en un nodo resultado. Este proceso se repite las veces que sea necesario hasta recorrer todo el documento de XSLT. Después se formatea el árbol resultado y por ultimo se despliega o se graba en un archivo resultado.

Por ejemplo, si utiliza como especificación de transformación el siguiente documento llamado resumen.xsl y se aplica al documento texto.xml, para el proceso de transformación, es posible entender de forma más específica lo que pasa.

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

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="libro">
  <Contenido>
    <xsl:apply-templates/>
  </Contenido>
</xsl:template>

<xsl:template match="capitulo">
  <Capitulo>
    <xsl:value-of select="numero"/>.- <xsl:value-of
select="nombre"/>
  </Capitulo>
</xsl:template>

</xsl:stylesheet>
```

Las primeras líneas del código a discutir son:

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

La primera línea es una plantilla con el valor de expresión XPath "/" que indica que se habla de la raíz del documento, y debido a que la plantilla a aplicar es vacía, no generará nada hacia el árbol de salida.

```
<xsl:template match="libro">
  <Contenido>
    <xsl:apply-templates/>
  </Contenido>
</xsl:template>
```

Bajando el siguiente nivel del árbol encontraremos el nodo "libro", que coincide con la siguiente plantilla y que indica que la información de libro debe de ser anidada bajo un nodo llamado "Contenido".

```
<xsl:template match="capitulo">
  <Capitulo>
    <xsl:value-of select="numero"/>.- <xsl:value-of
select="nombre"/>
  </Capitulo>
</xsl:template>
```

Por último y como el proceso es recursivo, se buscan los nodos "capitulo" y serán generados los nodos necesarios con el nombre de "capitulo" y tendrán como información la concatenación de los nodos "numero" y "nombre", según indica la plantilla a aplicar. Esto quiere decir que por cada elemento "capitulo" se creara un nodo "Capitulo" en el árbol resultado, y dentro de él se generara tanto el número como el nombre del capítulo, separado por un punto, un signo de separación y un espacio.

En este momento, ya esta almacenado en memoria el árbol resultado; ahora es necesario darle formato, por medio del cual obtendremos el siguiente archivo; esto se consigue ejecutando el siguiente comando:

```
saxon tesis.xml resumen.xsl > resumen.xml
```

Fig. 9 Ejemplo de XSL resume.xlm

```
<?xml version="1.0" encoding="utf-8"?>
<Contenido>XSL
   <Capitulo>1.- Conceptos generales</Capitulo>
   <Capitulo>2.- XPath</Capitulo>
   <Capitulo>3.- XSLT</Capitulo>
   <Capitulo>4.- XSL-FO</Capitulo>
</Contenido>
```

### 3.2.1. Modelo de procesamiento de datos

Existen dos formas de procesar los documentos; estos son modelo descendente (*Push*) y modelo ascendente (*Pull*).

El modelo descendente utiliza el documento fuente para controlar el formato, mientras la especificación de transformación se encarga de la apariencia de dicha estructura. En el modelo ascendente la especificación de transformación provee la estructura y el documento fuente actúa como fuente de datos.

Con un fragmento de "El Señor Presidente" de Miguel Ángel Asturias se podrá notar la diferencia entre ambos modelos. ElSeñorPresidente.xml será nuestro archivo de entrada:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="Senor.xsl"?>
<Fragmento>
 <Parte>
  <Titulo>Tercera Parte</Titulo>
  <Capitulo>
   <No>XXVIII</No>
   <Titulo>Habla en la Sombra</Titulo>
   <Seccion>
    <Persona>Primera voz</Persona>
    <Linea>Que dia sera hoy?</Linea>
   </Seccion>
   <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>De veras, pues, que día sera hoy?</Linea>
   </Seccion>
   <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>Esperen, a mi me capturaron el viernes:  viernes..., 
sabado..., domingo..., lunes..., </Linea>
    <Linea>Pero, Cuanto hace que estoy aqui...?  De veras, pues, 
que día sera hoy?</Linea>
   </Seccion>
   <Seccion>
    <Persona>Primera voz</Persona>
    <Linea>Siento, ustedes no saben como ...?</Linea>
```

```
    <Linea>Como si estuvieramos muy lejos, muy lejos ...</Linea>
   </Seccion>
   <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>Nos olvidaron en una tumba del cementerio viejo
enterrados para siempre ...</Linea>
   </Seccion>
   <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>No hable asi!</Linea>
   </Seccion>
   <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>No ha...</Linea>
    <Linea>...blemos aassii!!</Linea>
   </Seccion>
   <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>Pero no se callen; el silencio me da miedo, tengo
miedo,</Linea>
    <Linea>se me figura que una mano alargada en la sombra va a
cogerme por el cuello para estrangularme</Linea>
   </Seccion>
  </Capitulo>
 </Parte>
</Fragmento>
```

Utilizando el método ascendentes y el documento de transformación siguiente, llamado senor.xsl, se obtiene la siguiente presentación para el navegador.

```
<xsl:template match="Fragmento">
  <HTML>
    <HEAD>
      <H1><Center>El Senor Presidente</Center></H1>
    </HEAD>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="Parte/Titulo">
  <H1><xsl:value-of select="."/></H1>
</xsl:template>
```

```
<xsl:template match="Capitulo/No">
  <H2>Capitulo <xsl:value-of select="."/> </H2>
</xsl:template>

<xsl:template match="Capitulo/Titulo">
  <H2>   <xsl:value-of select="."/></H2>
</xsl:template>

<xsl:template match="Persona">
  <DIV><I><xsl:value-of select="."/></I></DIV>
</xsl:template>

<xsl:template match="Linea">
  <DIV><xsl:value-of select="."/></DIV>
</xsl:template>

</xsl:stylesheet>
```

Fig. 10 Ejemplo del método ascendente de XSL

Ahora utilizando el método descendente y el documento de transformación siguiente, llamado senor.xsl, se obtiene la siguiente salida para Internet.

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="Fragmento">
  <HTML>
   <HEAD>
    <H1>Contador de Datos</H1>
   </HEAD>
   <BODY>
    <P>Existen <xsl:value-of select="count(//Seccion)"/> secciones
en el fragmento de El Senor Presidente.</P>
      <xsl:for-each select="//Seccion">
      <P>
        El personaje <xsl:value-of select="Persona"/> tiene
<b><xsl:value-of select="count(Linea)"/></b> lineas,
        </P>
      </xsl:for-each>
      Haciendo un total de <b><xsl:value-of
select="count(//Linea)"/></b> lineas.
   </BODY>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

Fig. 11 Ejemplo del método descendente de XSL

## 3.3. Elementos

Los elementos más comúnmente usados son:

### 3.3.1. <xsl:stysheet>

Este es un elemento simple que contiene otros elementos propios de la hoja de estilo. Esta línea implica que el elemento documento contiene un documento de XSL. La hoja de estilo puede ser parte de otro documento, para lo cual será necesario hacer referencia a este hecho.

Dentro de este elemento, se define la versión sobre la que se está trabajando, así como la definición de espacios de nombre y detalles globales del documento.

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

### 3.3.2. <xsl:output>

Este elemento de alto nivel es usado para indicar al procesador de XSL, así como el formato que debe de tomar el árbol resultante.

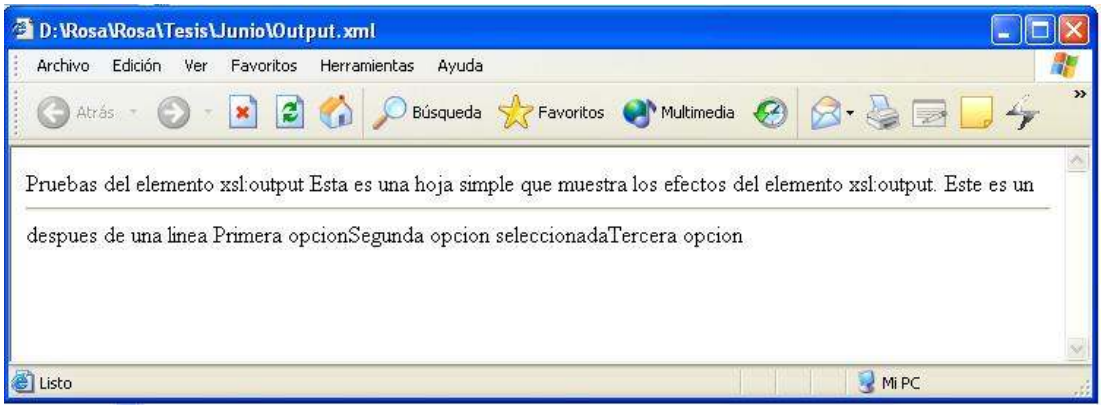
Este elemento es opcional específica el formato que la salida va a tener, sus tres posibles valores son `xml`, `html` y `text`. La opción de `xml` genera un documento bien formado. El valor `html` maneja casos de presentación para convertir a las etiquetas m<s usadas como <HR> y `text` genera una salida de texto pura, quitando las etiquetas y convirtiendo las etiquetas en texto puro. También se puede especificar si la salida será identada o no utilizando `indent` = '`yes`' o '`no`'.

Un ejemplo simple puede mostrar las diferencias, utilizando el siguiente archivo de xsl como hoja de transformación:

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/REC-html40">

<xsl:output method="text" indent="yes"/>

<xsl:template match="/">
  <HTML>
  <HEAD><H1>Pruebas del elemento xsl:output</H1></HEAD>
  <BODY>
    <P>
      Esta es una hoja simple que muestra los efectos del elemento
xsl:output.
      Este es un &lt;HR/&gt; despues de una linea
    </P>
    <HR/>
    <SELECT>
      <OPTION value="1">Primera opcion</OPTION>
      <OPTION selected="selected" value="2">Segunda opcion
seleccionada</OPTION>
      <OPTION value="3">Tercera opcion</OPTION>
    </SELECT>
  </BODY>
  </HTML>
</xsl:template>

</xsl:stylesheet>
```

Utilizando este archivo se obtendrá la salida siguiente.

Fig. 12 Ejemplo de `output` en XSLT con método Texto



Mientras que si se sustituye la opción de salida por `html` o `xml`, no muestra ninguna diferencia gráfica, pero internamente generar <HR/> o <HR/&gt. respectivamente, la presentación en un navegador es:

Fig. 13 Ejemplo de `output` en XSLT con método Html o XML


### 3.3.3. <xsl:template>

Este es otro elemento de alto nivel que contiene la información requerida para producir un nodo en el árbol resultado. Ya se han presentado ejemplo del uso de `Template` utilizando el atributo `match`, con la siguiente sintaxis:

```
<xsl:template match="Persona">
```

El valor del atributo es una expresión de XPath que deberá de coincidir con el árbol de entrada, lo que hace que la plantilla que se define a continuación se ejecute. Por ejemplo, aquí diferencia entre un título y otro, presentando él `titulo` de parte entre etiquetas <H1> y el `capitulo` entre <H2>.

```
<xsl:template match="Parte/Titulo">
  <H1><xsl:value-of select="."/></H1>
</xsl:template>
```

```
<xsl:template match="Capitulo/Titulo">
  <H2>   <xsl:value-of select="."/></H2>
</xsl:template>
```

Si no fuera necesario hacer esta diferencia, se podría definir así:

```
<xsl:template match="Titulo">
  <H1><xsl:value-of select="."/></H1>
</xsl:template>
```

Otros atributos con los que cuenta `template` son `priority`, que selecciona por prioridad que nodo debe o no agregarse al árbol de salida; `name` que sirve para luego ser usado a través de `<xsl:call-template>`, y por último `nodo` que provee de una flexibilidad adicional a `templete`.

### 3.3.4. <xsl:apply-template>

Junto con los elementos `<xsl:template>` y `<xsl:value-of>`, que son la fuerza de trabajo de XSLT, ya que con ellos se controla que plantillas son usadas en cada punto, mientras se construye el árbol resultado.

Sus atributos son `select` y `mode`. `Select` toma el valor de una expresión de XPath, para controlar qué elementos del árbol fuente deben de ser procesados. `Select` es opcional, y si queda nulo, todos los hijos serán procesados. `Mode` sirve para definir la apariencia de dicha elemento.

También cuenta con dos sub-elementos, `<xsl:sort>` y `<xsl:with-param>`, los cuales sirven para determinar el orden del procesamiento y los parámetros que serán trasladados a la plantilla, respectivamente.

### 3.3.5. <xsl:value-of>

Este elemento contiene una plantilla, que a través del atributo `select` define la expresión de XPath que sirve para generar el árbol resultado, que funciona valuando la función XPath, como lo hace `<xsl:apply-templates>`, pero utiliza únicamente la primer instancia seleccionada. El valor de la expresión es siempre escrito como una cadena de caracteres; si se trata de un número, es convertido a cadena y si es un grupo de nodos retorna todo la sección contenida, por ejemplo:

```
<xsl:value-of select = "Seccion[2]">
```

Se puede notar que devuelve la concatenación de sección, persona y línea.

```
  <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>De veras, pues, que día será hoy?</Linea>
  </Seccion>
```

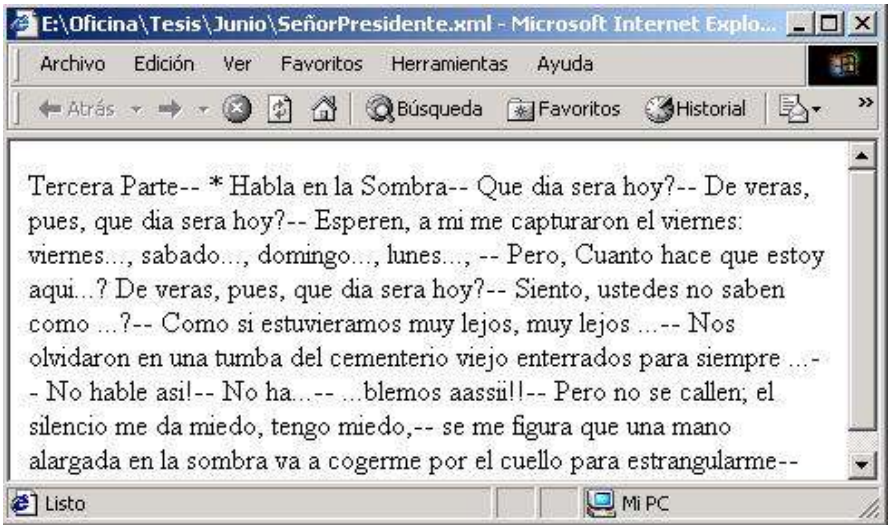
### 3.3.6. <xsl:copy> y <xsl:copy-of>

Estos dos elementos copian información directamente de un nodo origen a un nodo resultado. <xsl:copy> hace una copia superficial, o sea que copia solo el nodo y su espacio de nombre, mientras que <xsl:copy-of> hace una copia a profundidad, ques es una copia todos los atributos y sus descendientes, la cual se usa para copiar secciones completas de XML del árbol fuente hacia el árbol resultado, sin ningún cambio. Por ejemplo

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="text()"/>

<xsl:template match="Capitulo/No">
  <xsl:copy>
      <xsl:apply-templates/> *
```

```
  </xsl:copy>
</xsl:template>

<xsl:template match="Titulo | Linea">
  <xsl:copy-of select="."/> --
</xsl:template>

</xsl:stylesheet>
```

Fig. 14 Ejemplo de `copy` en XSLT




Se puede notar que de la opción `copy` solo aparece un *, mientras que de `copy-of` aparece todo el resto de la información.

A continuación, se presentas elementos que sirven para el control de flujo, los cuales son `<xsl::if>`, `<xsl::choose>`, `<xsl::when>`, `<xsl::otherwise>` y `<xsl::for-each>` y son parecidos a los que se usan en programación de procedimientos.

### 3.3.7. <xsl:if>

Este elemento es usado con una plantilla para ejecutar una secuencia encerrada que dependen de un valor booleano condicional. Es necesario que el atributo `test`

contenga una expresión que devuelva un valor booleano. La secuencia encerrada será ejecutada, si el valor de la expresión es verdadero. La expresión a valuar puede contener funciones de XSLT, que serán presentadas más adelante.

### 3.3.8. <xsl:choose>, <xsl:when>, <xsl:otherwise>

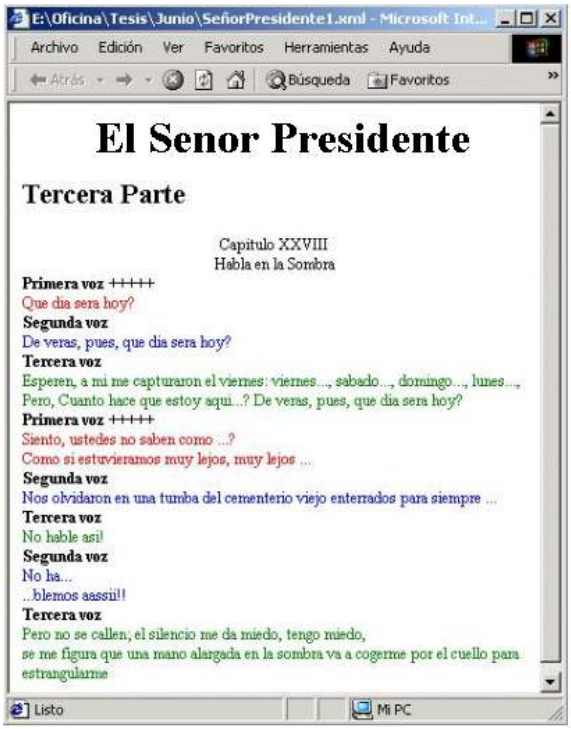
Provee el uso de tener varias expresiones de valuación; de esta manera es posible tener varias secuencias encerradas, y por último una secuencia que se ejecuta, cuando ninguna de las anteriores fue verdadera, lo cual permite más flexibilidad.

A continuación, se presenta un ejemplo que incluye el uso de `if` y de `choose`.

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="*|/"><xsl:apply-templates/></xsl:template>

<xsl:template match="Fragmento">
  <HTML>
    <HEAD>
      <H1><Center>El Senor Presidente</Center></H1>
    </HEAD>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="Parte/Titulo">
  <H1><xsl:value-of select="."/></H1>
</xsl:template>
<xsl:template match="Capitulo/No">
  <DIV><Center>Capitulo <xsl:value-of select="."/></Center></DIV>
</xsl:template>
<xsl:template match="Capitulo/Titulo">
  <Center> <xsl:value-of select="."/></Center>
</xsl:template>
<xsl:template match="Persona">
  <DIV><B><xsl:value-of select="."/>
    <xsl:if test="../Persona='Primera voz'">+++++</xsl:if>
  </B></DIV>
```

```
</xsl:template>
<xsl:template match="Linea">
  <DIV>
     <xsl:choose>
       <xsl:when test="../Persona='Primera voz'">
         <DIV style="color:red">
           <xsl:value-of select="."/>
         </DIV>
       </xsl:when>
       <xsl:when test="../Persona='Segunda voz'">
         <DIV style="color:blue">
           <xsl:value-of select="."/>
         </DIV>
       </xsl:when>
       <xsl:otherwise>
         <DIV style="color:green">
           <xsl:value-of select="."/>
         </DIV>
       </xsl:otherwise>
     </xsl:choose>
   </DIV>
</xsl:template>
</xsl:stylesheet>
```

Fig. 15 Ejemplo de `if` y `choose` en XSLT



### 3.3.9. <xsl:for-each>

Esta función sirve para hacer ciclos, los cuales se ejecutan para cada elemento que cumpla con la selección especificada. En el ejemplo del método descendente, se utilizó un ciclo para contar cuántas líneas tiene cada persona en una sección.

### 3.3.10. <xsl:sort>

Este elemento es usado para alterar el orden proceso de los nodos por medio de un `<xsl:apply-templetes>` o `<xsl:for-each>`. El ordenamiento puede llevarse a cabo por medio de múltiples criterios, donde se utiliza como primer orden el primer criterio y, partiendo de eso, se ordenan los siguientes, utilizando el principio de prioridad.

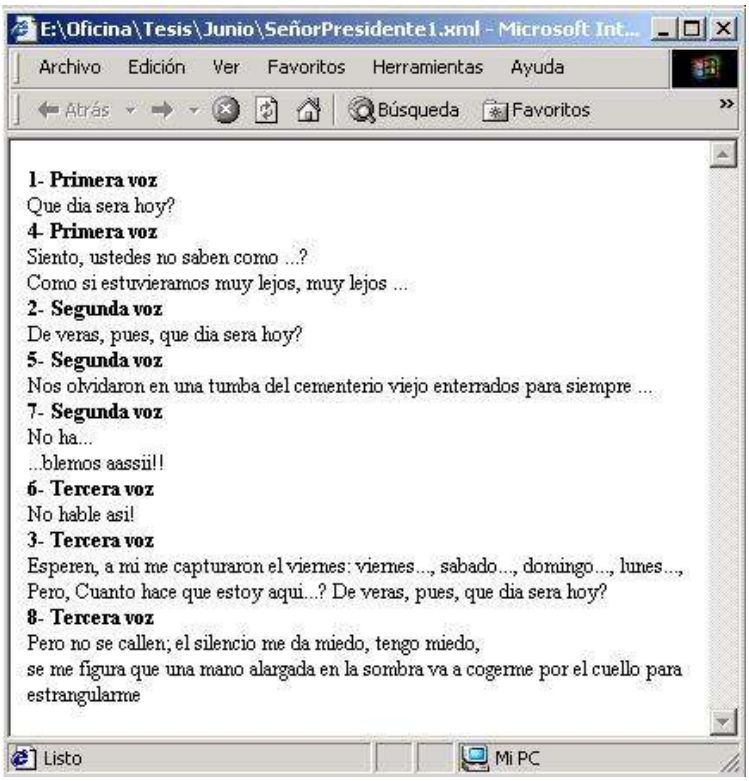
### 3.3.11. <xsl:number>

Este elemento es usado para determinar la posición del nodo en el árbol fuente. Tiene asociado el atributo `level`, que puede tener uno de los siguientes valores: `single`, `any` o `múltiple`. El primero o simple se usa para determinar la posición de un nodo, respecto a sus hermanos; puede usarse para viñetas o para posiciones en una tabla; el segundo o ninguno es usado para no considerar el nivel del documento en la jerarquía del árbol; puede usarse para numeración de pie de páginas, y él último es, según la jerarquía del árbol, el que sirve para numerar subsecciones.

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

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

```
<xsl:template match="Capitulo"><xsl:for-each select="Seccion">
  <xsl:sort select="Persona"/>
  <xsl:sort select="count(Linea)"/>
  <DIV><B>
    <xsl:number/>-   <xsl:value-of select="Persona"/>
  </B></DIV>
  <xsl:for-each select="Linea">
    <DIV><xsl:value-of select="."/></DIV>
  </xsl:for-each>
</xsl:for-each></xsl:template>

</xsl:stylesheet>
```

Fig. 16 Ejemplo de `for-each`, `sort` y `number` en XSLT




### 3.3.12. <xsl:text>

Este elemento inserta literalmente su contenido en el documento de salida. No es muy usado porque se refiere únicamente a agregar el texto, pero hay que recordar que xsl:text tiene dos ventajas importantes. La primera es que preserva los espacios

en blanco, exactamente como fueron definidos, aun cuando el nodo sea vacío, ya que por defecto el procesador de XSLT borra todos los espacios en blanco del texto del nodo, los espacios utilizando xsl:text prevalecen. La segunda ventaja es que si se incluye el texto `<` en la hoja de XSL, en el documento resultado, pueden aparecer `<` o >; esto depende del valor que se le asigne al atributo `disable-output-escaping`; si el atributo tiene `no` o es omitido, aparecerá `<`, pero si el atributo tiene `yes`, aparecerá `>`.

### 3.3.13. <xsl:element>

Por medio de este elemento, se puede crear elementos en el árbol de salida. Por ejemplo, el siguiente código de xsl, genera una salida simple.

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

<xsl:template match="/">
  <xsl:apply-templates select="Fragmento/Parte/Capitulo"/>
</xsl:template>

<xsl:template match="Capitulo">
  <xsl:element name="Cap{No}"/>
<xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

El resultado obtenido utilizado sobre SenorPresidente.xml es

```
<?xml version="1.0" encoding="utf-8" ?>
<CapXXVIII/>
```

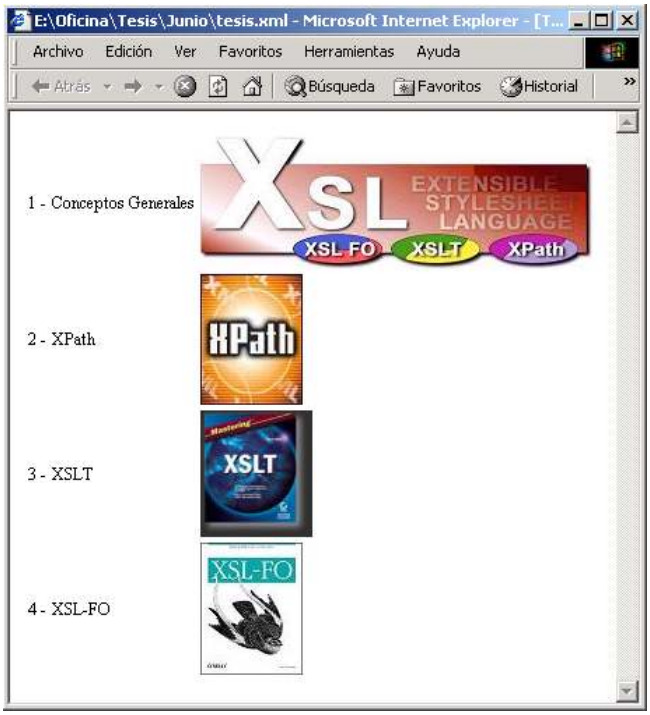
### 3.3.14. <xsl:attribute>

Similar a elemento, pero genera atributos, basando en el árbol fuente crea el nombre del atributo y su valor hacia el árbol resultado.

Por ejemplo, si se desea colocar un dibujo para cada capítulo del archivo tesis.xml, y asociar un archivo con extensión jpg al número del capítulo, se debe de proceder así:

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:apply-templates select="libro"/>
</xsl:template>
<xsl:template match="libro">
  <TABLE><xsl:apply-templates select="capitulo"/></TABLE>
</xsl:template>
<xsl:template match="capitulo">
  <TR>
    <TD><xsl:value-of select="numero"/> - </TD>
    <TD><xsl:value-of select="nombre"/></TD>
    <TD> <IMG>  <xsl:attribute name="src">
          <xsl:value-of select="numero"/>.jpg
        </xsl:attribute> </IMG> </TD>
  </TR>
</xsl:template>
```

Fig.17 Ejemplo de `attribute` en XSLT




## 3.4. Funciones de XSLT

Las principales funciones de XSLT son: position(), last(), name() y count(), las cuales serán detalladas a continuación.

### 3.4.1. position()

Sirve para seleccionar nodos basados en la posición que ocupan, según su aparición en el nodo de prueba; dicho nodo de prueba se forma después de hacer un ordenamiento utilizando el elemento `<sort>` sobre algún nodo específico; esto lo hace diferente al elemento `<number>`, ya que él ordena según el árbol fuente. La primera posición devolverá como valor 1.

### 3.4.2. last()

Sirve para determinar que un nodo especifico es el último; se utiliza comparándose con la función position().

### 3.4.3. name()

Retorna el nombre del nodo y fue creado con la idea de describir la información que presenta.

### 3.4.4. count()

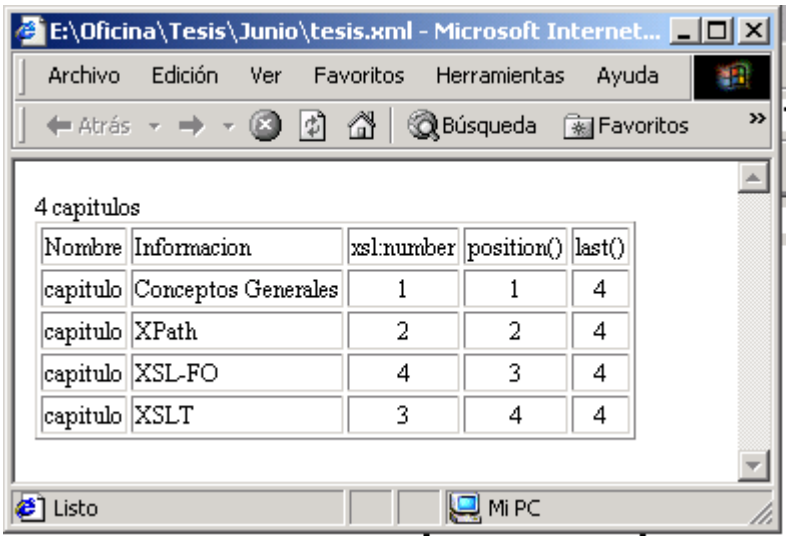
Función que devuelve la cantidad de nodos en un nodo de prueba.

Un ejemplo utilizando estas funciones es:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/REC-html40">

<xsl:template match="/">
  <TABLE BORDER="1">
    <TR>
      <TD>Nombre</TD>
      <TD>Informacion</TD>
      <TD>xsl:number</TD>
      <TD>position()</TD>
      <TD>last()</TD>
    </TR>

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

    <xsl:apply-templates select="libro/capitulo">
      <xsl:sort select="nombre"/>
    </xsl:apply-templates>
  </TABLE>
```

```
</xsl:template>

<xsl:template match="libro">
  <xsl:value-of select="count(capitulo)"/> capitulos
</xsl:template>
```

Fig.18 Ejemplo de `table` en XSLT



# 4. XSL-FO

XLS fue diseñado para producir presentaciones en múltiples medio con un alto grado de precisión, siguiendo con la idea de tomar información de un documento y presentarlo donde sea y de la forma que sea.

## 4.1. Técnicas para aplicar estilo a XML

El XLST es una herramienta poderosa para transforma documentos XML en muchos nuevos documentos adecuados y consistentes, pero cuando se trata de aplicarles estilo el resultado no es tan satisfactorio, ya que se debe de recurrir a HTML. Existen dos tecnologías para hacer la labor de estilo del XML, éstas son CSS y XSL-FO. CSS fue creada con un enfoque de presentación para Internet, mientras que XSL-FO fue creada con mas recursos para controlar exactamente el esquema. En la practica, ambos están lejos de realizar su potencial total.

La versión inicial de las recomendaciones de CSS o CSS nivel 1 fue producida en diciembre de 1996 y fue modificada el enero de 1999, mientras que CSS nivel 2 fue finalizada en mayo de 1998; esto hizo que los navegadores más populares no implementaran partes muy particulares las recomendaciones. Sin embargo, es posible utilizar con mucha confianza `Internet Explorer 5` o superior y `Netscape 6`. Por otro lado, las herramientas disponibles para producir y desplegar XSL-FO son aun rudimentarias, pero prometen muchas ventajas a corto plazo.

## 4.2. Introducción a CSS

Uno de sus principios básicos es mantener la separación entre el contenido y el estilo. Su principal beneficio es que aplicando estilo con CSS a una determinada clase de documentos, se obtiene una presentación consistente de información.

Las CSS pueden ser aplicadas usando alguna de las técnicas siguientes:

- Enlazar una CSS externa utilizando la notación `<LINK>` de html.
- Importando una CSS externa, a través de la notación `@import`.
- Anidar la hoja de estilo en un elemento `<Style>`, o bien en html, se puede utilizar el atributo `style` de cada elemento.
- Información de estilo en línea.

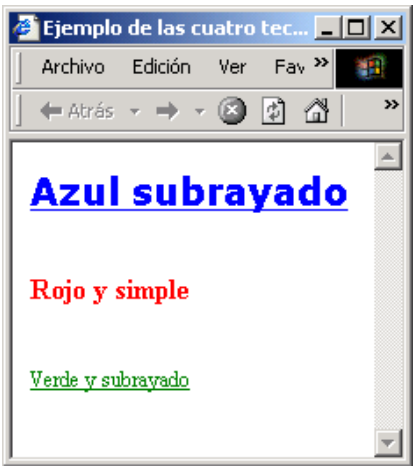
A continuación, se presenta un ejemplo que contiene las cuatro técnicas simultáneamente; contiene redundancia innecesaria utilizada sólo para mostrar las cuatro técnicas a la vez; esto no debería de hacerse en la práctica. Para propósitos prácticos utilización de enlace o importación es igual, y cada una es capaz de asociarse a más de una hoja de estilo a la vez.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Ejemplo de las cuatro tecnicas de CSS</TITLE>
<LINK REL=STYLESHEET TYPE="text/css"
            HREF="Tecnicas.css">
<STYLE TYPE="text/css">
            @import url(Tecnicas.css);

  H1 {color:blue; text-decoration:underline; font-family:Verdana }
  H2 { color: red; text-decoration:none; }
</STYLE>
</HEAD>

<BODY>
<H1>Azul subrayado</H1>
<BR>
<H2>Rojo y simple</H2>
<BR>
<P STYLE="color:green; text-decoration:underline;">Verde y
subrayado</P>
</BODY>
</HTML>
```

El archivo tecnicas.css provee las reglas utilizadas por el elemento <STYLE> y por @import.

```
H1 {    color: blue;
        text-decoration:underline;   }
H2 {    color: red;
        text-decoration:none; }
H1 {   font-family : Verdana }
```

Fig 19 Ejemplo de CSS, tecnicas.css




### 4.2.1. Plantillas CSS

Las plantillas CSS son descripciones detalladas del formato de las entidades de XML. Como se presento anteriormente, pueden ser parte del archivo XML o bien pueden estar en un archivo externo, esto es lo más recomendable, para poder así aprovechar el beneficio de reutilización.

Estas plantillas no tiene ninguna cabecera especial; lo único que se recomienda es que su nombre contenga la extensión ***.css**. En él se almacenan reglas, las cuales pueden estar dispuestas en cualquier orden y no es importante que sobren o faltan reglas, ya que todo código erróneo es automáticamente ignorado.

### 4.2.2. Reglas de estilo de CSS

Las CSS trabajan basadas en reglas de estilo. Trabajan creando una asociación entre el nombre del elemento de un documento de XML y las propiedades de CSS.

Una regla de estilo se compone de dos partes principales: el **selector** que es el elemento que aparece en primer lugar y la **declaración**, que es todo lo que esta entre los signos de "llave" (**{** y **}**). A su vez una declaración se compone de una pareja de entidades, las cuales son: **propiedad** y **valor**, separadas por dos puntos ( **:** ).

Para el ejemplo anterior selector es `H1;` la declaración de este selector esta formada por las propiedades `color` y `text-decoration` con los valores `blue` y `underline`.

Es posible repetir las veces que sean necesarias un selector, hasta definir perfectamente su formato. Es importante considerar que si se utiliza una técnica más específica, ese estilo prevalecerá sobre una definición general.

### 4.2.3. Agrupaciones de CSS

Las agrupaciones son una ventaja más de las CSS, ya que con ellas se pueden compartir reglas para varios selectores a la vez. Los nombres de los selectores deben de ir separados por coma, y luego se define la declaración de la manera general. Las agrupaciones se usan para las declaraciones, que son iguales para un grupo de selectores, pero si algún selector tiene otra característica importante, puede definirse una declaración por aparte específica para el, porque como se mencionó con anterioridad, las declaraciones específicas tiene mayor jerarquía.

Por ejemplo, si se tiene un selector `H3` que tenga características combinadas de `H1` y `H2`, la definición que utiliza agrupadores seria:

```
H1 {   color: blue;
       text-decoration:underline;  }
H2, H3 { color: red;
       text-decoration:none; }
H1, H3 {  font-family : Verdana }
```

### 4.2.4. Herencia de propiedades en CSS

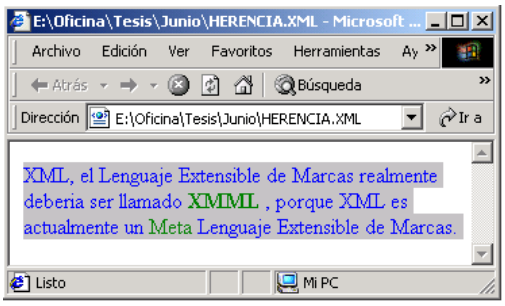
Se utiliza con los documentos que tiene información en varios niveles; los elementos descendientes heredan las propiedades de sus ancestros siendo aplicadas, siempre que no exista una regla específica del elemento que la anule o la invalide. Por ejemplo si se tiene la siguiente plantilla.

```
P{    background-color: silver;
      color: blue;
      text-decoration:undeline}
B,I{  color: green}
B {   font-weight:bold}
```

Si el documento de html fuera.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
   <LINK REL=STYLESHEET TYPE="text/css"  HREF="HerenciaHTML.css">
 </HEAD>
 <BODY>
  <P>XML, El lenguaje extensible de marcas realmente debería ser
llamado <B> XMML </B>, porque XML es actualmente un <I>meta</I>
lenguaje extensible de marcas. </P>
 </BODY>
</HTML>
```

Fig. 20 Ejemplo de CSS sobre html, herencia.css



Si se desea que cierta declaración sea aplicada solo en el caso de que un selector sea hijo de otro, la sintaxis es la siguiente

```
P B {color:red}
```

Esto significa que solo si B es hijo de P, se aplicará la declaración.

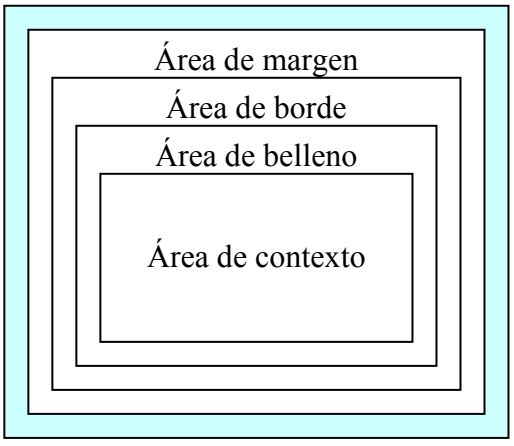
Si se necesita que se aplique la declaración, cuando se trate de un descendiente en general, no sólo para los hijos, la sintaxis es:

```
P > B {color:red}
```

### 4.2.5. Otras característica de las CSS

El modelo de formato para CSS asume la existencia de cajas o bloques (box), donde cada elemento se comporta como un rectángulo formado por cuatro zonas diferentes. Cada caja o bloque incluye siempre una zona con el contenido (*context area*), que puede ser un texto, una imagen u otro objeto. El tamaño del contenido marca el tamaño de dicha área. Alrededor de la zona de contenido, se encuentran las zonas de relleno (*padding*), de borde (*border*) y de margen (*margin*), que pueden existir o no, ya que son opcionales. El relleno y el margen son espacios que separan el contenido del borde. El relleno toma el mismo color del fondo del contenido y el margen es transparente.

Fig. 21 Bloques de CSS



Un ejemplo de utilización de estas entidades podría ser:

```
B {padding:20px; border:2px; margin:.5cm}
```

Cuando se definen los valores del relleno, del borde y del margen, se aplica a todos los lados del rectángulo; para ajustar los valores para uno, o varios lados debe añadir a continuación de las palabras `padding`, `border` y `margin`, los atributos `-left`, `-top`, `-right` o `-bottom`, que se refieren a los lados izquierdo, superior, derecho o inferior, respectivamente. Con estas posibilidades, se podría definir un código como:

```
P {padding-left:20px; padding-top:35px; padding-right:20px;
padding-bottom:35px;
border-left:1px; border-top:2px; border-right:1px; border-
bottom:4px;
margin-left:45px; margin-top:5px; margin-right:20px; margin-
bottom:27px}
```

De esta misma forma, se puede definir para los bordes el grosor (`-width`), estilo (`-style`) y color (`-color`) que da lugar a diseños tan detallados como:

```
P {padding:20px; border-left-width:thin; border-left-color:black;
border-left-style:dotted; border-top-width:medium; border-top-
color:blue; border-top-style:double; border-right-width:thick;
border-right-color:green; border-right-style:solid; border-bottom-
width:2px; border-bottom-color:red; border-bottom-style:groove;
margin:45px}
```

Las cajas pueden diseñarse para que sean posicionadas, de acuerdo con cuatro posibilidades principales: normal, absoluto, relativo y flotante.

El posicionamiento normal es cuando no se especifica posicionamiento alguno, entonces cada caja de bloque se coloca debajo de la anterior, todas en posición vertical, y cada caja en línea se sitúa a la derecha de la anterior, en posición horizontal.

El posicionamiento absoluto permite ajustar el punto exacto, respecto a la esquina superior izquierda donde se desea que comience a posicionarse una determinada caja. Se expresa con `position:absolute` y las coordenadas del punto medido normalmente en horizontal, desde el borde izquierdo (`left`) y en vertical desde el borde superior (`top`). Por ejemplo

```
B {position:absolute; left:70px; top:50px}
```

El posicionamiento relativo permite ajustar el punto exacto, respecto al punto que le correspondería como caja de bloque o como caja en línea, donde se desea que comience a posicionarse una determinada caja. Se expresa con `position:relative` y las mismas coordenadas comentadas para el posicionamiento absoluto. Por ejemplo.

```
B {position:absolute; left:70px; top:50px}
```

El posicionamiento flotante sirve para dejar flotar libremente una determinada caja, que será desplazada a la izquierda o a la derecha de la línea de texto donde se encuentre. Se expresa con `float:left` o `float:right`, y las mismas coordenadas comentadas para el posicionamiento absoluto.

En el caso de utilizar el posicionamiento absoluto para varios elementos XML, puede que se solapen total o parcialmente, produciendo situaciones indeseadas, pero si se controla adecuadamente, puede dar lugar a diseños especiales. Para controlar qué objetos deben mostrarse superpuestos a otros objetos, se puede utilizar la propiedad `z-index`, que permite dar un valor a cada elemento de manera que los que tengan valores más altos se sitúen encima de los que los tengan más bajos, por ejemplo.

```
P {position:absolute;left:50px;top:50px;z-index:3}
B {position:absolute;left:70px;top:70px;z-index:2}
H {position:absolute;left:90px;top:90px;z-index:1}
```

Aparecerá P por encima de B, que a su vez estará encima de H, todos ligeramente solapados entre sí.

Por ultimo para incluir comentarios dentro de un archivo de plantillas CSS se deben de utilizar el signo de división y el asterisco (/*) como muestra de inicio del comentario y para finalizar con el asterisco y signo de división (*/).

### 4.3. Aplicación de CSS a XML

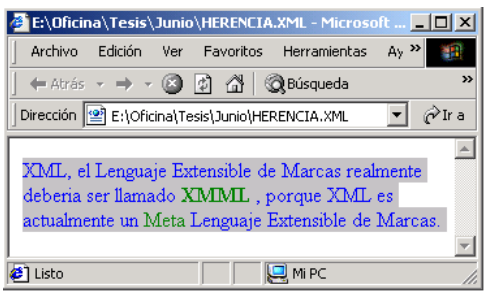
La aplicación de CSS a XML es muy similar a la hecha con HTML, únicamente se necesita direccionar la plantilla que se utilizara para el documento XML, para lo cual se utiliza la instrucción <?xml-stylesheet> como se presenta en el siguiente ejemplo donde el documento de XML es:

```
<?xml version='1.0'?>
<?xml-stylesheet href="herencia.css" type="text/css"?>

<PARRAFO>
XML, El lenguaje extensible de marcas realmente debería ser
llamado <NEGRITAS> XMML </NEGRITAS> porque XML es actualmente un
<VERDE>meta</VERDE> lenguaje extensible de marcas.
</PARRAFO>
```

El archivo herencia.css es:

```
PARRAFO {  background-color: silver; color: blue;
           text-decoration:undeline}
NEGRITAS,VERDE { color: green}
NEGRITAS  { font-weight:bold}
```

Fig. 22 Ejemplo de CSS sobre xml, herencia.css



Con el fin de mostrar en mas detalle las características de las CSS, se presenta a continuación un ejemplo un poco mas extenso y detallado. El archivo de XML que se utiliza es el siguiente:

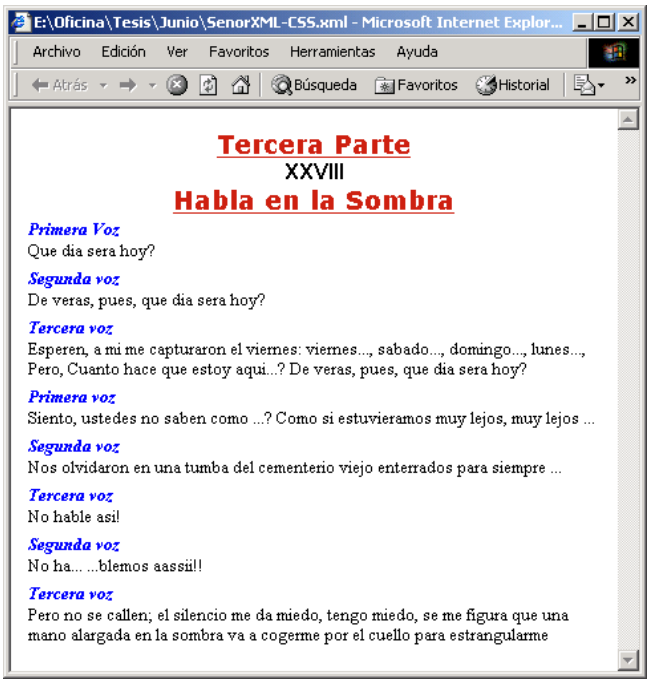
```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="Senor.css"?>
<Fragmento>

<Parte>
  <Titulo>Tercera Parte</Titulo>
<Capitulo>
  <No>XXVIII</No>
  <Titulo>Habla en la Sombra</Titulo>
  <Seccion>
    <Persona>Primera Voz</Persona>
    <Linea>Que dia sera hoy?</Linea>
  </Seccion>
  <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>De veras, pues, que día sera hoy?</Linea>
  </Seccion>
  <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>Esperen, a mi me capturaron el viernes:  viernes...,
sabado..., domingo..., lunes..., </Linea>
    <Linea>Pero, Cuanto hace que estoy aqui...?  De veras, pues,
que día sera hoy?</Linea>
  </Seccion>
  <Seccion>
    <Persona>Primera voz</Persona>
    <Linea>Siento, ustedes no saben como ...?</Linea>
    <Linea>Como si estuvieramos muy lejos, muy lejos ...</Linea>
  </Seccion>
  <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>Nos olvidaron en una tumba del cementerio viejo
enterrados para siempre ...</Linea>
  </Seccion>
  <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>No hable asi!</Linea>
  </Seccion>
  <Seccion>
    <Persona>Segunda voz</Persona>
    <Linea>No ha...</Linea>
    <Linea>...blemos aassii!!</Linea>
  </Seccion>
```

```
  <Seccion>
    <Persona>Tercera voz</Persona>
    <Linea>Pero no se callen; el silencio me da miedo, tengo
miedo,</Linea>
    <Linea>se me figura que una mano alargada en la sombra va a
cogerme por el cuello para estrangularme</Linea>
  </Seccion>
</Capitulo>
</Parte>
</Fragmento>
```

La plantilla que se aplicará es:

```
Titulo{
  font-family: Verdana;;
  font-size:14pt;
  font-weight:bold;
  text-decoration:underline;
  display:block;
  Text-align:center;
  color: rgb(200,33,20) }

No{
  font-family: Arial;
  font-size:12pt;
  font-weight:bold;
  display:block;
  Text-align:center  }

Seccion{
  font-family: TimesRoman;
  font-size: 10pt;
  font-style:normal;
  display:block;
  padding:3  }

Persona{
  font-style:italic;
  font-weight:bold;
  display:block;
  color:blue   }
```

Fig. 23 Ejemplo de CSS sobre xml, senor.css




## 4.4. Introducción XSL-FO

XSL-FO es un lenguaje complejo que contiene vocabulario y formato en un archivo XML. XLS-FO fue diseñado para desplegar información en múltiples medios; permite especificar a detalle las características de formato y apariencia imposibles de obtener con XSLT o con CSS. Su principal deficiencia es que aún no ha sido implementada una parte importante de sus especificaciones; esto deja expectativas de que más se podrá hacer en el fututo con el.

Mediante los objetos de formato y sus propiedades específicas se pueden describir cómo se verán los elementos de un documento. Objetos de formato son:

- Las características de página
- Los párrafos
- Las listas
- Los enlaces, etc.

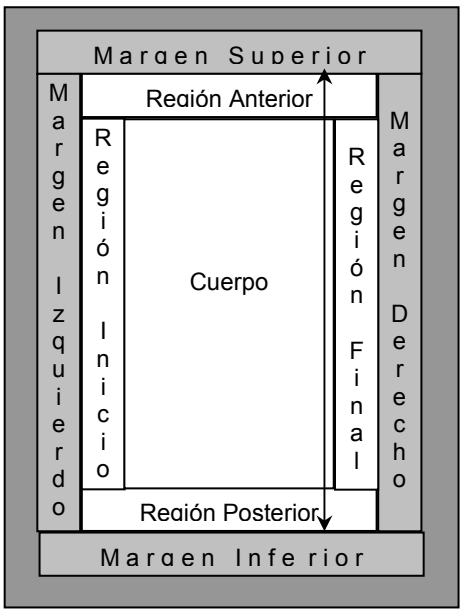
Cabe destacar que XSL-FO tiene un gran potencial en el área de documentación, pero esto requiere de mucha programación por el amplio grado de detalle en sus documentos; por esto puede ser considerado un lenguaje difícil a esto; se puede agregar que existe escaso soporte en aplicaciones comerciales.

Actualmente es posible tener todo el contenido de un libro en un archivo de XML y por medio de XSL-FO hacer una página de estilo, para imprimir este libro con las normas mas estrictas de edición, y con otra página de estilo generar la presentación para un navegador de Internet.

## 4.5. Páginas de XSL-FO

Una de las claves para tener el dominio de XSL-FO es entender la estructura y terminología del esquema de una página.

Fig. 24 Esquema de páginas deXSL-FO



El esquema anterior muestra las regiones de una página de XSL-FO para un lenguaje de 'Izquierda a Derecha', como es el caso del español y el inglés. La parte más oscura es una región de borde que no presenta contenido. Las partes de gris claro, llamadas márgenes, son utilizadas para definir anchos y no presentan contenido. El área blanca esta dividida en cinco partes, para presentar información; dentro de ellas se define el contenido general de una página. Cada una de estas áreas es capaz de presentar información independiente; no son áreas obligatorias en las páginas, y pueden no aparecer.

Para lenguajes como el árabe o el hebreo que son lenguajes de "Derecha a Izquierda" las áreas van invertidas, o sea la región inicio está en la derecha y la región final, a la izquierda. Para lenguajes verticales, la rotación es a 90 grados.

El cuerpo del documento esta limitado por las regiones; por esoo debe de tenerse cuidado en la definición de los anchos de estos, para dejar un tamaño apropiado al cuerpo del documento.

## 4.6. Estructura de un documento en XSL-FO

XLS-FO es esencialmente un lenguaje de descripción de documentos o, mejor dicho, un lenguaje de descripción de páginas. Para un documento de muchas páginas, puede tenerse un esquema diferente para cada página, por ejemplo, cuando inicia un capítulo o para páginas izquierdas y derechas con márgenes diferentes.

Un documento de XSL-FO se divide en:

- Maestro de esquema o "`layout-page-master`"
- Descripción de tipos de página o "`simple-page-master`"
- Secuencia en la que parecen los tipos de página o "`page-sequence`"
- Páginas propiamente dichas o "`flow`"

Para los ejemplos, se necesita de un procesador de XSL-FO, se utilizará el software *XSL Formater*, el cual debe de instalarse y puede obtenerse de Internet en:

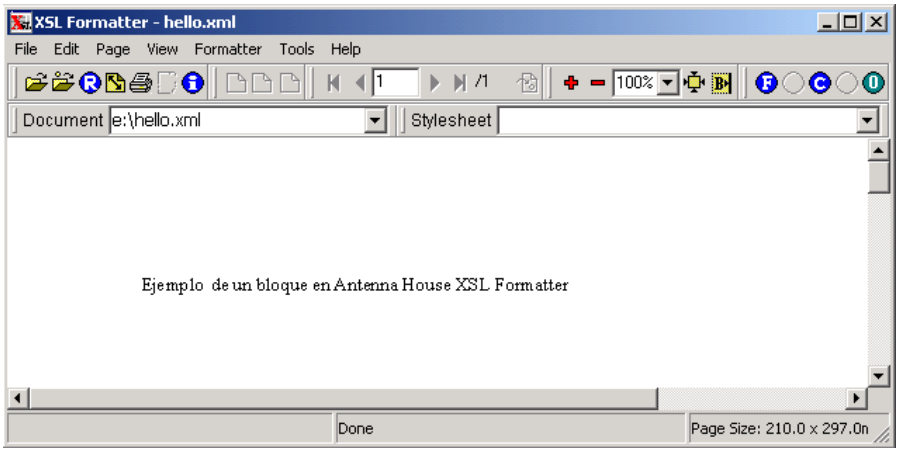
`http://www.antennahouse.com/xslformatter.html`

*XSL Formater* es un programa ejecutable que genera una presentación con base a el documento de origen y a la página de estilo que se le asocie.

A continuación se presenta un documento para analizar

```
<?xml version='1.0'?>

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

  <fo:layout-master-set>
    <fo:simple-page-master master-name="mi-pagina">
      <fo:region-body margin="1in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="mi-pagina">
    <fo:flow flow-name="xsl-cuerpo">
      <fo:block>Ejemplo de un bloque en Antenna House XSL
Formatter</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Fig 25 Ejemplo de XSL-FO, bloque.xml en *XSL Formater*



El elemento <layout-page-master> contiene dentro del uno o más elementos <simple-page-master>; como éste es un ejemplo simple contiene sólo uno denominado "mi-pagina" y este tipo de página es el único que aparece en la secuencia <page-sequence>. Dicha secuencia tiene anidada la descripción de la página, que en este caso es el flujo "xsl-cuerpo".

Para darle una mejor apariencia a una página se deben de definir las regiones, utilizando el elemento <fo:static-content>, donde la región anterior o "xsl-region-before" es utilizada generalmente como cabecera de página, mientras que la región posterior o "xsl-region-after" se utiliza como pie de página. La definición de <fo:static-content> debe de preceder a la definición del elemento <flow> o flujo. En otras palabras, debe de definir las cabeceras y los pies de páginas, antes de definir el cuerpo de la página y para que estas sean visibles debe de existir el espacio suficiente en la región respectiva.

Dentro del elemento <flow>, puede tener elementos como <fo:block>, <fo:block-container>, <fo:table-and-caption>, <fo:table>, <fo:list>.

### 4.6.1. Crear un documento XSL-FO, usando XSLT

La mezcla entre XLS-FO y XSLT se hace definiendo el formato, a través de la sintaxis de XSL-FO y los datos que se van a presentar a través de las plantillas de XSLT, como se muestra en el siguiente ejemplo:

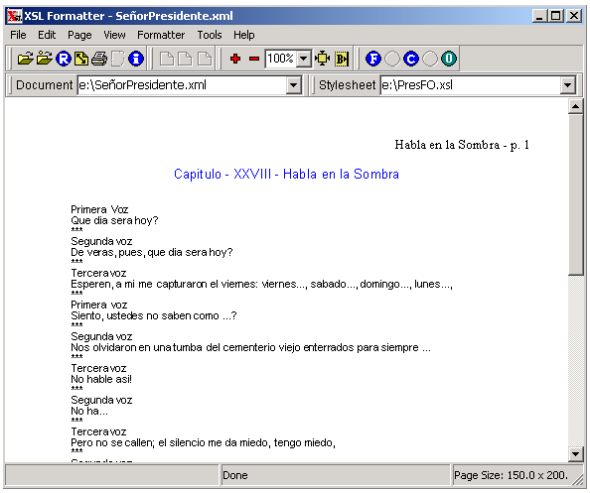
```
<?xml version='1.0'?>

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

<xsl:template match="/">
 <fo:root>
  <fo:layout-master-set>
   <fo:simple-page-master master-name="General"
```

```
    page-height="20cm"
    page-width="15cm"
    margin-top="1cm"
    margin-bottom="1cm"
    margin-left="1cm"
    margin-right="1cm">
    <fo:region-body margin-top="1cm"/>
    <fo:region-before extent="1cm"/>
    <fo:region-body margin="0.3in"/>
   </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-name="General">
   <fo:static-content flow-name="xsl-region-before">
    <fo:block text-align="end"
      font-size="10pt"
      color="Green"
      font-family="serif"
      line-height="14pt" >
      <xsl:value-of select="/Fragmento/Parte/Capitulo/Titulo"/>
      -     p.
      <fo:page-number/>
    </fo:block>
   </fo:static-content>

   <fo:flow flow-name="xsl-region-body">
    <fo:block font-size="10pt"
      font-family="sans-serif"
      line-height="10pt"
      space-after.optimum="15pt"
      color="Blue"
      text-align="center"
      padding-top="3pt">
      Capítulo  -
      <xsl:apply-templates select="Fragmento/Parte/Capitulo/No"/>
- <xsl:apply-templates select="Fragmento/Parte/Capitulo/Titulo"/>
    </fo:block>

    <fo:block font-size="8pt"
     font-family="sans-serif"
     line-height="8pt"
     space-before.optimum="8pt"
     space-after.optimum="8pt"
     text-align="start"
     padding-top="3pt">
     <xsl:apply-templates
select="Fragmento/Parte/Capitulo/Seccion"/>
    </fo:block>
```

```
    </fo:flow>
  </fo:page-sequence>
 </fo:root>
</xsl:template>

<xsl:template match="Fragmento/Parte/Capitulo/Seccion">
  <fo:block>
    <xsl:value-of select="Persona/."/>
  </fo:block>
  <fo:block>
    <xsl:value-of select="Linea/."/>
  </fo:block>
  ***
</xsl:template>

<xsl:template match="Fragmento/Parte/Capitulo/No">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="Fragmento/Parte/Capitulo/Titulo">
  <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Por medio de `Formatter` de `Antenna House`, se ve la aplicación del formato anterior al documento ElSeñorPresidente.xml, que se utilizo para ejemplo en capítulos anteriores, lo que resulta en:

Fig. 26 Ejemplode XSL-FO, SenorPresidente.xml



Para entender como se obtiene este resultado, se discutirá a continuación trozo por trozo. Por ejemplo:

```
<?xml version='1.0'?>

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

Determina qué es un archivo de XML por su primer línea, pero también define que trabajará, tanto son XSLT como con XSL-FO.

Luego se determina qué, para todos los elementos del documento fuente, debe de aplicarse la siguiente plantilla; esto en sintaxis de XSLT.

```
<xsl:template match="/">
```

El maestro de esquema se define con el nombre "`General`" y en el se determinan los márgenes y tamaños que se van utilizar para las páginas, así como los tamaños de las regiones que presentarán datos; si una región no tiene definida área, no presentará información alguna.

```
  <fo:layout-master-set>
   <fo:simple-page-master master-name="General"
    page-height="20cm"
    page-width="15cm"
    margin-top="1cm"
    margin-bottom="1cm"
    margin-left="1cm"
    margin-right="1cm">
    <fo:region-body margin-top="1cm"/>
    <fo:region-before extent="1cm"/>
    <fo:region-body margin="0.3in"/>
   </fo:simple-page-master>
  </fo:layout-master-set>
```

La secuencia en la que aparecen los bloques se define a continuación, se definen primero las regiones estáticas como indican las instrucciones de XSL-FO. Para la región cabecera se define una alineación izquierda, con letra tamaño 10, color verde y tipo *serif*, que presentara el título del capítulo, seguido de un guión y el numero de página.

```
    <fo:page-sequence master-name="General">
   <fo:static-content flow-name="xsl-region-before">
    <fo:block text-align="end"
      font-size="10pt"
      color="Green"
      font-family="serif"
      line-height="14pt" >
      <xsl:value-of select="/Fragmento/Parte/Capitulo/Titulo"/> -
p. <fo:page-number/>
    </fo:block>
   </fo:static-content>
```

El cuerpo de la página presentará un bloque de titulo con el número del capítulo y el título del capítulo, el cual será formateado en letra tamaño 10 de tipo *sans-serif* y color azul, con alineación centrada

```
   <fo:flow flow-name="xsl-region-body">
    <fo:block font-size="10pt"
      font-family="sans-serif"
      line-height="10pt"
      space-after.optimum="15pt"
      color="Blue"
      text-align="center"
      padding-top="3pt">
      Capítulo -
      <xsl:apply-templates select="Fragmento/Parte/Capitulo/No"/>
- <xsl:apply-templates select="Fragmento/Parte/Capitulo/Titulo"/>
    </fo:block>
```

Siempre en el cuerpo de la página, se presentará otro bloque con la información de las secciones escrito en letra *Sans-serif* de tamaño 8, alineado a la izquierda, con un relleno de tamaño 3 y dejando un espacio de 8 en cada lado.

```
   <fo:block font-size="8pt"
     font-family="sans-serif"
     line-height="8pt"
     space-before.optimum="8pt"
     space-after.optimum="8pt"
     text-align="start"
     padding-top="3pt">
     <xsl:apply-templates
select="Fragmento/Parte/Capitulo/Seccion"/>
    </fo:block>
```

En sintaxis de XSLT, se define la apariencia de cada elemento del documento fuente, que coincida con una plantilla específica. Para el caso de las secciones, se presentará información de la persona y de la línea en bloques separados, para que sea más presentable y también se agrega un valor fijo '***' después de ellos.

```
<xsl:template match="Fragmento/Parte/Capitulo/Seccion">
  <fo:block>
    <xsl:value-of select="Persona/."/>
  </fo:block>
  <fo:block>
    <xsl:value-of select="Linea/."/>
  </fo:block>
  ***
</xsl:template>
```

Las últimas especificaciones son más sencillas y presenta los datos tal cuales en el documento de entrada.

```
<xsl:template match="Fragmento/Parte/Capitulo/No">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="Fragmento/Parte/Capitulo/Titulo">
  <xsl:value-of select="."/>
</xsl:template>
```

# CONCLUSIONES

**1.** XSL es un lenguaje potente en la transformación de datos, a partir de un documento XML, por la utilizacion de plantillas reutilizables para la presentacion de la información.

**2.** La presentación de la información con XSL es versátil, ya que puede llegar a mostrar la misma informacion de diferentes formas, en diferente orden, con diferente aspecto y con distintos filtros de acceso.

**3.** Las salida de datos o presentaciones de información de un documento XSL puede realizarse a diferentes dispositivos electrónicos, como teléfonos celulares, palms, etc., y no solamente a páginas de Internet.

**4.** La familia de XSL está formada por tres lenguajes: XPath, XSLT y XSL-FO. XPath; es el encargado de obtener el segmento de informacion que se va a utilizar, XSLT y XSL-FO realizan la presentacion de la informacion, el primero es el más conocido y utilizado, mientras que XSL-FO es más potente, pero no está implementado en su totalidad.

# RECOMENDACIONES

**1.** A los diseñadores de páginas para Internet se les insta usar XSL, ya que este lenguaje es más abierto, con el se puede llegar a un nivel de detalle superior y, sobre todo, se pueden hacer plantillas que pueden ser utilizadas para múltiples documentos de entrada.

**2.** Para los usuarios que utilizan XLM, para la transferencia de información, se les recomienda utilizar XSL, para poder filtrar la información que se va a enviar, enviar sólo una parte de ella o bien reestructurarla de cualquier manera.

**3.** Para quienes deseen crear páginas para los buscadores más usados del momento, se debe utilizar XSLT pues es soportado en su totalidad por ellos, mientras que XSL-FO es soportado únicamente por programas específicos.

# REFERENCIAS

1. Manual de XML en Castellano http://www.programacion.net/html/xml/htmdsssl/capitulo1/capitulo1.htm#capitulo1, enero de 2004.

# BIBLIOGRAFIA

## Libros


1. Aguilera Camacho, María Cristina y Enriquez Tobar, Mario René. **XML y Técnicas de Aplicación**, Guatemala, Universidad Francisco Marroquin, 2000.

2. Bradley, Neil. **The XML Companion,** USA, Addion-Wesley, 1998.

3. Paz Echeverria, Jorge Leonel. **Estructuración de Datos en XML,** Guatemala, Universidad Francisco Marroquin, 2000.

4. Rusty Harold, Elliotte. **XML Extensible Markup Language**, USA, IDG Books, 1998.


## Documentos Electrónicos


1. **Microsoft's XML page,** http://www.microsoft.com/xml

2. **Microsoft's XSL page,** http://www.microsoft.com/xml/xsl

3. **The application of XSL for XML transformations in e-business solutions,** http://www.infoloom.com/gcaconfs/WEB/paris2000

4. **The World Wide Web Consortium,** http://www.w3c.org

5. **Tutorial de XML,** http://www.programacion.net/html/xml/htmdsssl/capitulo1/capitulo1.htm

6. **Using XSL FO with XEP 3.0,** http://www.renderx.com/Tests/doc/html/tutorial.html

7. **XSLT,** http://www.programacion.net/articulos/xsltie5.php

8. **What is XML,** http://www.xml.com/

9. **W3C Extensible Markup Language (XML) page,** http://www.w3.org/XML/

10. **W3C Technical Reports and Publications of Extensible Markup Language,** http://www.w3.org/TR/

11. **W3C Technical Reports and Publications of Extensible Stylesheet Language,** http://www.w3.org/TR/xsl/