Diploma en Documentación Electrónica e Interoperabilidad de la Información

XML y la Documentación Electrónica

Renzo Angles (rangles@dcc.uchile.cl)

Departamento de Ciencias de la Computación
Universidad de Chile
2006

Índice general

Capítulo 1

Análisis de Información

1.1. Definición

Un proceso de Análisis de Información es simplemente el arte de trazar los componentes lógicos de un documento y definir su estructura.

El producto del análisis de un documento es generar un modelo jerárquico que nos permita visualizar los componentes y su relaciones de composición. El objetivo principal de este proceso es analizar un documento y diferenciar contenido, estructura y presentación.

Desde el punto de vista de documentación electrónica basado en XML, el análisis de información es un paso previo muy importante al momento de modelar datos, diseñar esquemas, y definir estilos y presentaciones para documentos XML.

A continuación vamos a describir el significado de contenido, estructura y presentación, tomando como caso de estudio la boleta de venta presentada en la figura 1.1

LaEmpresa Chile Limitada

GIRO: Fabricación, Importación y Comercialización de Aparatos Domésticos
R.U.T.: 77.145.120−6
Oficina Principal: Alcántara 23, Piso 7, Las Condes. Fono: 753 7488
Sucursales: Vicuña Mackena 1439 Ñuño. Fono/Fax: 551 5753
Sucursales. Nueva Costanera 3940 Vitacura. Fono/Fax: 953 3563

BOLETA VENTAS Y SERVICIOS
Nro. 90346

Sr. *Juan Rojas Málaga*
Calle *Romero 2385 Depto. 1307, Santiago Centro*
Ciudad *Santiago*

DIA	MES	AÑO
11	*10*	*05*

DESCRIPCION	CODIGO ARTICULO	Cantidad	PRECIO UNITARIO	TOTAL
Refrigerador Consul	*CRP34AB*	*1*	*144.000*	*144.000*
Despacho	*CL004*			*4800*

Nuestra responsabilidad cesa al salir la mercadería de nuestra Bodega.
No se aceptan devoluciones sin conformidad previa nuestra.

VALOR TOTAL *148.800*

Figura 1.1: Ejemplo de documento: Una boleta de Venta

90346

Juan Rojas Málaga
Romero 2385 Depto. 1307, Santiago Centro
Santiago

Refrigerador Consul	*CRP34AB*	*1*	*144.000*	*144.000*
Despacho	*CL004*			*4800*
				148.800

11 *10* *05*

Figura 1.2: Contenido de la boleta de Venta

1.2. Contenido

El contenido se refiere a los datos que son almacenados en el documento Los datos son los elementos de un documento que pueden variar para cada instancia. La figura 1.2 muestra los datos que pueden ser identificados en la boleta de venta.

Es de suma importancia diferenciar entre datos que contiene el documento y datos que describen características o dan información adicional sobre el documento. Estos últimos deberán ser avaluados para determinar cuales podrán ser incluidos como elementos y cuales se incluirán como metadatos (datos sobre los datos) en los documentos instancia. Por ejemplo el creador de un documento instancia, podría considerarse un metadato del documento mas que un dato que deba ir incluido como contenido de algún elemento o atributo del documento instancia.

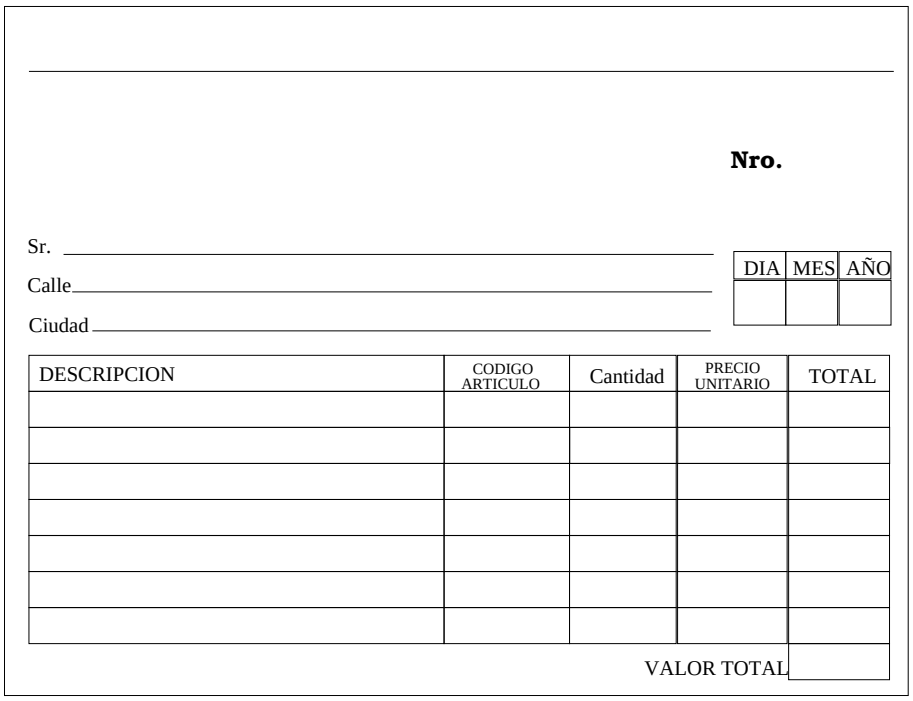
1.3. Estructura

La estructura de un documento define: los conceptos usados en el documento así como sus atributos; las relaciones entre los conceptos (principalmente relaciones de composición); y restricciones sobre el contenido de los conceptos y sus relaciones.

Al describir la estructura de un documento debemos considerar:

- Estudiar el dominio de la aplicación (ver estándares ya definidos).
- Definir el propósito y audiencia para los cuales se modela el documento.
- Al identificar los conceptos que conformarán el esquema, deben identificarse claramente cuales pertenecen al contenido, estructura y presentación respectivamente.
- Definir partes del documento que son claramente identificables.
- Definir cuando nivel de detalle es importante modelar.
- Identificar piezas o elementos repetidos que puedan ser agrupados o reutilizados (modularidad).
- Verificar si la estructura puede ser generalizada a otras instancias de documentos.
- Considerar futuras ampliaciones (extensibilidad).
- La descripción final debe identificar claramente: conceptos a modelar, estructuras componente-nivel, la naturaleza del contenido, restricciones de tipos de datos, orden y ocurrencia.

La figura 1.3 muestra los conceptos que conforman el esquema de la boleta de venta.



Nro.

Sr. _____

Calle _____

Ciudad _____

DIA	MES	AÑO

DESCRIPCION	CODIGO ARTICULO	Cantidad	PRECIO UNITARIO	TOTAL
			VALOR TOTAL	

Figura 1.3: Elementos del esquema de la boleta de Venta

Metadatos: *Metadatos* es un término que se refiere a datos sobre los propios datos. En esencia, son datos altamente estructurados que describen características de los datos, como por ejemplo procedencia, relaciones, etc.

Como resultado del análisis del contenido y la estructura de un documento, podremos elaborar un *Diccionario de Metadatos* en el cual describiremos los conceptos identificados en nuestros documentos. Cada concepto o metadato debe proveer información de su significado, tipo (formato y restricciones), contenido (estructura y restricciones), y relaciones con otros conceptos.

La descripcion de un metadato podría tener la siguiente estructura.

```
1. Identificador
2. Titulo
3. Autor
4. Descripcion
5. Fecha
6. Contenido (define el contenido que puede tener el concepto)
   6.1 Tipo
   6.1 Descripcion
   6.2 Restricciones
7. Relaciones:
   7.1 parte-de:
       7.1.1 ID-ConceptoPadre
       ...
       7.1.N
   7.2 compuesto-por:
       7.2.1 ID-ConceptoHijo,
             Restricciones de aparicion, ocurrencia, etc
       ...
       7.2.N
```

Nota. El significado de los elementos utilizados en la descripción de cada metadato, puede encontrarse en el conjunto de metadatos estándar definido por *Dublin Core* [17, 18]. El Anexo B muestra la lista de metadatos básicos propuestos por Dublin Core.

LaEmpresa Chile Limitada

GIRO: Fabricación, Importación y Comercialización de Aparatos Domésticos
R.U.T.: 77.145.120−6
Oficina Principal: Alcántara 23, Piso 7, Las Condes. Fono: 753 7488
Sucursales: Vicuña Mackena 1439 Ñuño. Fono/Fax: 551 5753
Sucursales. Nueva Costanera 3940 Vitacura. Fono/Fax: 953 3563

BOLETA VENTAS Y SERVICIOS
Nro. número

Sr. *frase*
Calle *frase*
Ciudad *frase*

DIA	MES	AÑO
dd	*mm*	*aa*

DESCRIPCION	CODIGO ARTICULO	Cantidad	PRECIO UNITARIO	TOTAL
frase	*codigo*	*numero*	*numero*	*numero*

Nuestra responsabilidad cesa al salir la mercadería de nuestra Bodega.
No se aceptan devoluciones sin conformidad previa nuestra.

VALOR TOTAL *numero*

Figura 1.4: Presentación de la boleta de Venta

1.4. Presentación

La presentación se refiere a como los datos serán mostrados al usuario final. Se utilizan formatos o diseños que definen como los datos son organizados en la presentación final. La figura 1.4 muestra los elementos que conforman el formato de presentación de la boleta de venta.

1.5. Resumen

En esta sección hemos presentado el proceso de análisis de información, recalcando la importancia que tiene la identificación del contenido, estructura y presentación de un documento. En los siguientes capítulos estudiaremos XML como una familia de estándares apropiados para procesos de análisis, administración e interoperabilidad de la información.

Capítulo 2

Introducción a XML

El e**X**tensible **M**arkup **L**anguage (XML) [2] por sus siglas en ingles o Lenguaje de Marcado Extensible es un formato de texto simple y flexible propuesto por la W3C, que fue originalmente diseñado para cumplir los desafíos de publicaciones electrónicas a gran escala y que actualmente esta siendo usado como un formato estándar para la documentación electrónica e intercambio de datos en la Web

El impacto de XML como el lenguaje estándar para representar e intercambiar información, ha generado que en los últimos años haya surgido como un área de fuerte investigación y desarrollo. En consecuencia, XML se ha convertido en una familia de tecnologías para almacenar, administrar e intercambiar datos usando como base el lenguaje XML.

El objetivo principal de este curso es brindar los conocimientos necesarios sobre los integrantes del mundo XML y poder ser aplicados en la solución de los problemas planteados por las nuevas tecnologías de información.

2.1. ¿Qué es XML?

XML puede ser visto desde 3 niveles:

- 1er. Nivel: Es un protocolo para contener y administrar información. Específicamente es una meta-lenguaje que permite describir documentos.
- 2do. Nivel: Es una familia de tecnologías para el almacenamiento, administración e intercambio de datos. Estas permiten desde representar y consultar datos hasta aplicarles formato.

- 3er. Nivel: Es una filosofía para manejo de información que busca máxima utilidad y flexibilidad para los datos, refinándolos a su forma estructurada mas pura.

2.2. Las metas de diseño de XML

La especificación de XML [2] propone las siguiente metas de diseño:

- XML debe ser utilizable directamente en Internet.
- XML debe admitir una gran variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser sencillo escribir programas para procesar documentos XML.
- El número de características opcionales en XML debe ser lo mas pequeño posible, idealmente cero.
- Los documentos XML deben ser legibles para las personas y razonablemente claros.
- El diseño de documentos XML debe ser preparado rápidamente.
- Debe ser sencillo crear documentos XML.
- El diseño de XML debe ser formal y conciso.
- La concisión en el marcado de XML tiene una importancia mínima.

2.3. Características de XML

- XML puede almacenar y organizar casi cualquier tipo de información en una forma adecuada a las necesidades del usuario.
- Es un estándar abierto, XML no pertenece a compañía alguna ni esta vinculado con alguien.
- Con Unicode como su juego de caracteres estándar, XML soporta un amplio numero de sistemas de escritura (scripts) y símbolos.
- Con su sintaxis clara y simple, mas una estructura sin ambigüedad, XML es fácil de leer y analizar por humanos al igual que por programas.

- XML ofrece muchas maneras para chequear la calidad de un documento, con reglas para la sintaxis, chequeo de enlaces internos, comparación de modelos de documentos y tipos de datos.
- Separa la estructura de un documento de su presentación. XML es fácilmente combinado con hojas de estilo para crear documentos con formato en algún estilo deseado.

2.4. Ventajas y Beneficios de usar XML

- XML es internacional.
- XML es una formato:
  - Flexible y sencillo de utilizar
  - Portable y Reusable en una variedad de aplicaciones y ambientes.
  - Estructurado y no Ambiguo: La información sobre la estructura y relaciones entre los datos así como las reglas de negocios pueden ser manejadas mas facilmente.
  - Durable y Valioso: Los datos en XML pueden sobrevivir en el entorno actual o pueden ser fácilmente compartidos en un entorno heterogéneo.
- Los documentos XML se pueden construir por composición permitiendo la reutilización.
- XML puede ser un contenedor de datos (base de datos).
- XML posee formatos estándar.

2.5. Observaciones con respecto a XML

- Es una sintaxis de documentos, no un lenguaje de programación.
- Puede fomentar la proliferación de vocabularios específicos así como formatos y vocabularios propietarios.
- Bueno para texto, malo para datos binarios.
- Poco eficiente como lenguaje de almacenamiento de bases de datos.
  - Puede requerir demasiado espacio, ancho de banda y tiempo de procesamiento.
  - Documentos largos con mucha información redundante.

2.6. La familia de especificaciones XML

- XML Syntax (Sintaxis XML): especifica como debemos escribir correctamente documentos XML .
- XML Schema (Esquemas XML): especifica como definir la estructura o esquema de un documento XML.
- XML Namespaces (Espacios de Nombres XML): especifica como identificar de forma única los elementos y atributos de un documento XML usando identificadores uniformes de recursos (URIs).
- XSL (Extensible Stylesheet Language - lenguaje de hojas de estilo extensible): es una familia de recomendaciones para definir transformaciones y formatos de presentación para documentos XML. Consiste de 3 partes:
  - XPath (XML Path Language) : un lenguaje de expresiones que permite acceder o referenciar partes de un documento XML.
  - XSLT (XSL Transformations): un lenguaje para transformar documentos XML.
  - XSL-FO (XSL Formatting Objects): un vocabulario para especificar formatos de presentación.
- XQuery (XML Query): Define la sintaxis de un lenguaje de consulta para XML.
- XLink (Extensible Linking Language): define como pueden ser enlazados los documentos XML.
- XPointer: define como apuntar o hacer referencia a un lugar específico de un documento XML.
- DOM (Document Object Model - Modelo de objetos de documento): interface que provee llamadas estándar a funciones y procedimientos que permiten trabajar con contenido, estructura, estilo o representación de documentos, todo esto basado en árboles.
- SAX (Simple API for XML): interface usada para análisis de documentos basado en eventos.

Capítulo 3

Sintaxis XML

Todos los componentes de la familia de tecnologías XML tienen como elemento base el concepto de documento XML. En esta sección describimos los conceptos principales presentados en la especificación [2] que define la sintaxis de documentos XML.

3.1. Estructura de un documento XML

Un *documento instancia XML* representa una jerarquía de datos, y puede estar compuesto de elementos, atributos, entidades, comentarios, instrucciones de procesamiento y secciones CDATA.

A continuación describiremos los distintos elementos de un documento instancia XML, usando como referencia el ejemplo *instancia01.xml*.

Declaración XML

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

Declara el inicio de un documento XML y presenta los siguientes atributos:
- *version*: indica la versión de XML a utilizar (la ultima es XML 1.1).
- *encoding*: define el tipo de codificación de datos a utilizar (UTF-8, UTF-16, iso-8859-1, etc).
- *standalone*: indica si el documento hace o no hace referencia a entidades externas.

instancia01.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<libreria>
 <!-- Lista de libros -->
 <libro codigo="34674" stock='5'>
  <titulo>El Hobbit</titulo>
  <autor>J.R.Tolkien</autor>
  <editorial>Minotauro</editorial>
 </libro>
 <libro>
  <registro codigo="83467" stock="3"/>
  <titulo>El arte de la Guerra</titulo>
  <autor>Sun-Tzu</autor>
 </libro>
</libreria>
```

Elementos: Un elemento es el componente central de un documento XML, el cuál se forma por una etiqueta de apertura del elemento `<titulo>` y otra de clausura `</titulo>`, las cuales delimitan su contenido. En caso de no tener contenido un elemento se simplifica a una etiqueta `<usuario />`.

Existen 2 tipos de elementos:

- Elementos con contenido:
  `<titulo>El Hobbit</titulo>`
- Elementos vacíos (sin contenido):
  `<registro codigo="83467"stock="3/>`

La declaración de elementos debe considerar que:

- El nombre de un elemento no puede contener espacios en blanco.
- Los elementos mantienen una jerarquía de anidamiento, por lo tanto no deben entrecruzar su contenido.
- El primer elemento de la jerarquía se denomina elemento *"raíz"* del documento, y es el que contiene todos los demás elementos.
- Si existe una etiqueta de apertura debe existir una correspondiente etiqueta de clausura.
- Los nombres de elementos son sensibles a mayúsculas y minúsculas.

Atributos: Son valores con nombre que pueden declararse en la etiqueta de apertura de un elemento, como por ejemplo:

```
<libro codigo="34674" stock='5'>
```

Los nombre de un atributo es sensible a mayúsculas y minúsculas y su valor es un cadena de caracteres escrita entre comillas simples o dobles. Un elemento puede contener varios atributos.

Comentarios: Los comentarios en un documento XML pueden ser insertados como contenido de la etiqueta de inicio de comentario "`<!--`" la etiqueta de fin de comentario "`-->`". Los comentarios no pueden incluir guiones sucesivos "`--`." o terminar con "`--->`". En nuestro ejemplo tenemos el comentario "`<!-- Lista de libros -->`".

Entidades predefinidas: Son códigos que se utilizan para representar caracteres especiales. Ver tabla 3.1.

Código de Entidad	Caracter
<	<
>	>
&	&
"	"
'	'

Cuadro 3.1: Entidades XML predefinidas

Secciones CDATA: Las secciones CDATA son usadas para contener datos binarios o texto XML que no debe ser analizado.

```
<![CDATA[ ... ]]>
```

Atributos predefinidos: Existen 2 atributos predefinidos por la especificación que pueden ser utilizados para agregar información a los elementos:

- *xml:lang* permite especificar el idioma (ej. "en", "sp").
- *xml:space* especifica como tratar los espacios en blanco.
  "preserve" = mantenerlo,
  "default" = permitir a la aplicación que lo trate como quiera.

3.2. Documento XML bien formado

Un documento XML se considera bien formado si respeta la estructura y sintaxis definida en la especificación de XML. En términos generales el documento debe considerar las siguiente reglas:

- La declaración XML es la primera declaración de un documento XML.
- Todo documento debe contener un elemento raíz.
- La declaración de elementos debe respetar que:
  - Todos los elementos deben tener etiqueta de apertura y su correspondiente etiqueta de clausura.
  - Los elementos deben estar anidados apropiadamente.
- La declaración de atributos debe respetar que:
  - Sus valores están entre comillas (simples o dobles) y no contienen el caracter "<".
  - Están declarados en la etiqueta de apertura de un elemento.
  - No existen dos atributos con el mismo nombre en un mismo elemento.
- Ciertos caracteres reservados para análisis (ej. "<"ó ">") no deben ser utilizados.

Capítulo 4

XML Schema

4.1. Introducción

Cada tipo de documento es apropiado para modelar cierto tipo de información, por ejemplo la estructura de una carta (compuesta de remitente, destinatario y contenido) es distinta de la estructura de un libro (compuesta de autor, titulo, capítulos, secciones, apéndices). El esquema de un documento lo determinan los elementos que lo componen y la forma en que dichos elementos se encuentran estructurados u organizados.

Un segundo componente de la familia XML corresponde a un lenguaje que nos permite especificar y validar la estructura o esquema que debe tener un tipo especifico de documento XML. Los DTDs y los XML Schemas son dos lenguajes que nos permiten describir esquemas de documentos. En este curso revisaremos algunos conceptos claves para el diseño de esquemas de documentos XML y nos concentraremos en el uso de XML Schema.

Describiendo la estructura Para describir la estructura de un documento debemos tener en cuenta ciertos aspectos:

- Estudiar el dominio de la Aplicación (ver estándares ya definidos).
- Definir el propósito y audiencia para los cuales se modela el documento.
- Definir partes del documento que son claramente identificables y cuando nivel de detalle es importante modelar.
- Identificar piezas o elementos repetidos que puedan ser agrupados o reutilizados (modularidad).

- Verificar si la estructura puede ser generalizada a otras instancias de documentos. Considerar futuras ampliaciones (extensibilidad).
- Tener claros los conceptos de contenido, estructura y presentación.
- La descripción final debe identificar claramente: estructuras componente-nivel; la naturaleza del contenido; reglas de orden y ocurrencia.

Schema de un documento: Un Esquema define la estructura de un tipo de documento, y es utilizado para validar o chequear que una instancia del documento cumple dicha estructura.

Para el caso de XML existen varios lenguajes que permiten definir esquemas que permitan validar documentos XML. Originalmente se utilizaron los DTDs (Document Type Definition), pero actualmente se utilizan esquemas XML (W3C XML Schema).

Importancia de los esquemas Un documento XML no necesita ir acompañado de un esquema, pero existen algunas razones por las cuales es importante su definición:

- La definición de un esquema permite establecer un contrato entre productores y consumidores de documentos XML. Este contrato especifica la estructura de los documentos XML que comparten.
- Permite a los productores de documentos XML asegurarse del contenido que ellos están proporcionando.
- Permite a los consumidores de documentos XML chequear lo que reciben de los productores y así proteger sus aplicaciones.
- Un esquema provee de un interfaz garantizada.
- Simplifican la tarea de los desarrolladores de aplicaciones para validación de documentos, dejando las tareas de detección y análisis de errores a un validador de propósito general.
- Los esquemas son un rica fuente de metadatos. Un esquema contiene información sobre los datos en los documentos instancia, por ejemplo tipos de datos, rango de valores, relaciones entre datos, etc).

instancia02.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libreria [
  <!ENTITY amp "&#38;"> <!-- Entidad-->
  <!ELEMENT libreria (libro*, revista*)>
  <!ELEMENT libro (titulo, autor+, precio?)>
  <!ELEMENT revista (titulo, numero, precio?)>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT autor (#PCDATA)>
  <!ELEMENT numero (#PCDATA)>
  <!ELEMENT precio (#PCDATA)>
  <!ATTLIST precio
    moneda CDATA #REQUIRED
>
]>
<libreria>
 <libro>
  <titulo>El Hobbit</titulo>
  <autor>J.R.R. Tolkien</autor>
  <precio moneda="USdolar">15</precio>
 </libro>
 <revista>
  <titulo>Cine &amp; Arte</titulo>
  <numero/>
  <precio moneda="peso">3500</precio>
 </revista>
</libreria>
```

Documento XML válido Inicialmente vimos que un documento XML estaba bien formado si respetaba la sintaxis XML. Adicionalmente, si el documento respeta la reglas de estructura y tipos definidas por un esquema decimos que es un "documento XML válido"'.

Document Type Definition (DTD): Uno de los primeros lenguajes para declarar esquemas de documentos XML fueron los DTDs. Las declaraciones de tipos de documentos (DTDs) nos permiten declarar reglas para instancias de documentos, como se muestra en el ejemplo *instancia02.xml*.

Limitaciones de los DTDs

- La Sintaxis no es XML (difíciles de manipular).
- No soportan espacios de nombres (ver sección 4.4).

- No permiten especificar tipos de datos (por ejemplo: enteros, flotantes, fechas, etc.).
- No hay soporte para declaraciones sensibles al contexto ya que todos los elementos se definen a nivel de documento.
- Soporte limitado para referencias cruzadas.
- No es posible formar claves a partir de varios atributos o de elementos.
- No son extensibles. Una vez definido, no es posible añadir nuevos vocabularios a un DTD.

4.2. XML Schema

XML Schema (esquema XML) [3] es un lenguaje de definición de esquemas para documentos XML propuesto por la W3C. Un esquema XML especifica los tipos de datos y como los datos XML estarán organizados en un documento instancia.

El archivo *esquema03.xsd* ilustra la declaración de un esquema XML para el documento *instancia03.xml*.

esquema03.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="libro" type="CTlibro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="CTlibro">
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
    <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="codigo" type="xsd:string" use="required"/>
   <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
  </xsd:complexType>
</xsd:schema>
```

instancia03.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="esquema03.xsd">
 <libro codigo="34674" stock="5">
  <titulo>El Hobbit</titulo>
        <autor>J.R.Tolkien</autor>
        <editorial>Minotauro</editorial>
 </libro>
 <libro codigo="83467">
  <titulo>El arte de la Guerra</titulo>
  <autor>Sun-Tzu</autor>
 </libro>
</libreria>
```

Referenciar el esquema XML en un documento instancia: La declaración `xsi:noNamespaceSchemaLocation="esquema03.xsd"` permite especificar la localización del esquema XML del documento instancia. Para poder utilizar el atributo `xsi:noNamespaceSchemaLocation` necesitamos declarar el prefijo `xsi`. Esto se hace con la declaración:
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`,
la cuál identifica el espacio de nombres para documentos instancia XML (en la sección 4.4 veremos el uso de espacios de nombres).

4.2.1. Características de XML schema

- Sintaxis XML y soporte para Espacios de Nombres.
- Mayor expresividad: restricciones numéricas, de estructura, etc.
- Tipos de datos: buena cantidad de tipos de datos predefinidos; adicionalmente soporta la creación de tipos de datos definidos por el usuario.
- Soporta construcción modular de esquemas.
- Extensibilidad: Inclusión y Redefinición de esquemas; reutilización y herencia de tipos de datos.
- Soporta Documentación.

4.2.2. Ventajas de XML Schemas

- Define tipos de datos primitivos mejorados y soporta la definición de tipos de datos personalizados.
- Tienen la misma sintaxis de los documentos instancia XML.
- Varias opciones para la definición de contenido de un elemento.
- Soporta la definición de múltiples elementos con el mismo nombre pero con diferente contenido.
- Permite definir elementos de contenido nulo.
- Soporta la definición de elementos sustituibles.
- Permite definir elementos únicos, claves y referencias a claves.
- Presenta mecanismos de orientación a objetos.

4.3. Sintaxis de XML Schema

En esta sección estudiaremos la sintaxis definida por la especificación de XML Schema [4, 5, 6].

4.3.1. Estructura Básica

El elemento `Schema`

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.libreria.org"
            xmlns="http://www.libreria.org"
            elementFormDefault="qualified"
            attributeFormDefault="qualified">
 ...
</xsd:schema>
```

- El elemento `schema` es el elemento contenedor de todas las declaraciones de elementos, atributos y tipos de datos del esquema XML.
- Los atributos `targetNamespace` y `xmlns` permiten definir espacios de nombres globales (esto será discutido en la sección 4.4).

- Todo esquema debe hacer referencia al espacio de nombres de XML Schema (`xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema"`).
  Este espacio de nombres contiene elementos para la definición del esquema XML (ej. xsd:schema, xsd:element, etc.) y tipos de datos definidos por XML Schema (ej. xsd:integer, xsd:string, etc.).

- Los atributos `elementFormDefault` y `attributeFormDefault` permiten establecer si ciertos elementos o atributos deben ser calificados en el documento instancia (esto será discutido en la sección 4.4).

Anotaciones El elemento `xsd:annotation` es usado para documentar el schema y su contenido no afecta la validación del mismo. Este elemento puede ir antes o después de algún componente global y solo al inicio de componentes no globales.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation xml:lang="es">
   Esquema XML para datos de una Libreria
  </xsd:documentation>
  <xsd:appinfo>
   <archivo>esquema03.xsd</archivo>
  </xsd:appinfo>
 </xsd:annotation>
 ...
</xsd:schema>
```

El elemento `xsd:annotation` contiene dos subelementos:

- `xsd:documentation` se usa para describir un comentario para personas. Puede tener 2 atributos: `source` que contiene un URL de un archivo con información complementaria; y `xml:lang` que especifica el lenguaje de la documentación.

- `appinfo` se usa para proporcionar comentarios a programas. Su contenido es algún texto XML bien formado y puede contener el atributo `source`.

Declaraciones globales y locales: XML schema permite definir la estructura de un documento declarando elementos simples, elementos complejos, tipos simples, tipos complejos y atributos. Estos pueden ser declarados para uso global o local:

- Los elementos, tipos o atributos *globales* son creados por declaraciones que aparecen como hijos del elemento `xsd:schema`.
- Los elementos, tipos o atributos *locales* son definidos dentro de otros elementos distintos de `xsd:schema`.

Los elementos y atributos globales son reutilizados en cualquier parte del esquema usando el atributo `ref`. Los tipos globales son reutilizados usando el atributo `type`. El ejemplo (*esquema04.xsd*) presenta definiciones globales y locales.

esquema04.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="libro" type="CTlibro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <!-- tipo complejo global -->
 <xsd:complexType name="CTlibro">
  <xsd:sequence>
   <xsd:element ref="titulo"/>
   <xsd:element name="autor" type="xsd:string"/>
   <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="codigo" type="STcodigo" use="required"/>
  <xsd:attribute ref="stock" use="optional"/>
 </xsd:complexType>
 <!-- elemento global -->
 <xsd:element name="titulo" type="xsd:string"/>
 <!-- atributo global -->
 <xsd:attribute name="stock" type="xsd:integer"/>
 <!-- tipo simple global -->
 <xsd:simpleType name="STcodigo">
  <xsd:restriction base="xsd:string">
   <xsd:length value="5"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

4.3.2. Declaración de Tipos

Tipos Simples : son elementos que representan datos atómicos, por consiguiente no pueden contener elementos ni atributos. Existen 2 clases de tipos simples:

- Predefinidos o built-in: Definidos en la especificación de XML Schema (Primitivos, Derivados predefinidos). La figura A.1 presenta la Jerarquía de tipos de XML schema. Las tablas A.1 y A.2 describen los tipos primitivos y derivados predefinidos.
- Definidos por el usuario: son los tipos simples definidos a partir de tipos built-in. Mas adelante veremos como son declarados.

Tipos Complejos : Son tipos que tienen contenido complejo en el sentido que pueden contener subelementos y atributos. La definición de su contenido lo constituyen la declaración de los subelementos seguido de las declaraciones de atributos.

El orden de los atributos no es relevante en el documento instancia, sin embargo existen distintas formas de estructurar los subelementos de un tipo complejo (*esquema05.xsd*):

- Alternativa: `xsd:choice` define un conjunto de elementos alternativos (or-exclusiva), por lo tanto el elemento puede contener solo uno de los subelementos.

```
<xsd:complexType name="CTpublicacion">
 <xsd:choice>
  <xsd:element name="libro" type="CTlibro"/>
  <xsd:element name="revista" type="CTrevista"/>
 </xsd:choice>
</xsd:complexType>
```

- Secuencia ordenada: `xsd:sequence` define un conjunto de elementos que deben ir ordenados.

```
<xsd:complexType name="CTlibro">
 <xsd:sequence>
  <xsd:element name="titulo" type="xsd:string"/>
  <xsd:element name="autor" type="xsd:string"/>
  <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
 </xsd:sequence>
 <xsd:attribute name="codigo" type="xsd:string" use="required"/>
 <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
</xsd:complexType>
```

- Secuencia no ordenada: `xsd:all` define un conjunto de subelementos que pueden ir en cualquier orden.

```
<xsd:complexType name="CTrevista">
 <xsd:all>
  <xsd:element name="titulo" type="xsd:string"/>
  <xsd:element name="numero" type="xsd:string"/>
 </xsd:all>
 <xsd:attribute name="codigo" type="xsd:string" use="required"/>
 <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
</xsd:complexType>
```

  Observaciones:
  - Los elementos `xsd:choice` y `xsd:sequence` pueden contener a su vez otras declaraciones `xsd:sequence` y `xsd:choice`, pero no `xsd:all`.
  - El elemento `xsd:all` solo puede contener declaraciones de elementos, no puede contener elementos *xsd:sequence, xsd:choice, xsd:all*.
  - Los elementos declarados dentro de `xsd:all` deben tener los atributos `maxOccurs` y `minOccurs` con valores de "0"ó "1".

- Contenido mixto: al declarar el atributo `mixed="true"` definimos un elemento cuyo contenido consiste en una mezcla de texto y subelementos (*esquema06.xsd*).

```
<xsd:element name="carta">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="cuerpo">
    <xsd:complexType mixed="true">
     <xsd:sequence>
      <xsd:element name="enfatizar" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

  Esta definición permite el siguiente contenido:

```
<cuerpo>
 Estimados clientes.
 Esta carta es para desearles
 una <enfatizar>feliz navidad</enfatizar>.
</cuerpo>
```

4.3.3. Restricciones de ocurrencia para elementos y atributos

Existen una serie de atributos que permiten restringir la ocurrencia y los posibles valores que tomarán los elementos y atributos (ver tabla 4.1). Estos atributos llamados *facetas* son:

- `maxOccurs`: numero máximo de veces que un elemento puede aparecer.
- `minOccurs`: numero mínimo de veces que un elemento puede aparecer.
- `use`: define si un atributo será opcional("optional"), requerido("required"') o prohibido ("prohibited").
- `fixed`: define un valor fijo para un elemento simple, complejo o un atributo. Si no se incluye, se utiliza el valor fijo. Si se incluye, debe coincidir con el valor definido.
- `default`: define un valor por defecto para un elemento (de contenido simple) o atributo.

faceta/valor	Elementos	Atributos	Default
`minOccurs`	entero positivo o "unbounded"	–	"1"
`maxOccurs`	entero positivo o "unbounded"	–	"1"
`use`	– – –	"required" "optional" "prohibited"	"optional"
`fixed`	algún valor	algún valor	–
`default`	algún valor	algún valor	–

Cuadro 4.1: Restricciones de ocurrencia ("–"significa no aplicable)

Observaciones:
- El uso de los atributos `fixed` y `default` es excluyente.
- El valor por defecto (default) de un atributo solo tiene sentido si el atributo es opcional.
- Los valores por defecto de los atributos se aplican cuando éstos no están presentes, y los valores por defecto de los elementos se aplican cuando estos están vacíos.
- Los atributos `minOccurs`, `maxOccurs` y `use`, no pueden aparecer en la declaración de elementos o tipos globales.
- Al usar los atributos `fixed` o `default` en un elemento de contenido complejo, será necesario el uso de entidades predefinidas para representar símbolos no permitidos como contenido (Ver tabla 3.1).

4.3.4. Elementos y atributos

Declaración de elementos La instrucción `xsd:element` permite declarar un elemento, ya sea éste de tipo simple o complejo.

Existen 2 formas de declarar un elemento (*esquema07.xsd*):

- Haciendo referencia a un tipo simple (C1), a un tipo complejo (C2) o a un elemento definido globalmente (C3).

```
(C1) <xsd:element name="autor" type="xsd:string"/>
(C2) <xsd:element name="libro" type="CTlibro" maxOccurs="unbounded"/>
(C3) <xsd:element ref="titulo"/>
```

  Observación: Si al definir un elemento no se especifica el atributo `type`, el tipo asignado por defecto es `xsd:anyType`, el cual permite que el elemento contenga cualquier cosa.

- Definiendo "inline"el contenido del elemento, ya sea como un tipo simple (C4) o un tipo complejo (C5).

```
          (C4)                              (C5)
<xsd:element name="editorial">  |   <xsd:element name="libreria">
 <xsd:simpleType>               |    <xsd:complexType>
  ...                           |     ...
 </xsd:simpleType>              |    </xsd:compleType>
</xsd:element>                  |   </xsd:element>
```

Declaración de atributos: La instrucción `xsd:attribute` permite la declaración de atributos. Un atributo puede ser global al declararse como hijo del elemento `xsd:schema`, o puede ser ser local al declararse como contenido de cualquier otro elemento. Un atributo puede ser declarado de 2 maneras:

- Haciendo referencia a un tipo simple predefinido (C1), a un tipo simple personalizado (C2) o a un atributo definido globalmente (C3) .

```
(C1) <xsd:attribute name="issn" type="xsd:string"/>
(C2) <xsd:attribute name="stock" type="STstock"/>
(C3) <xsd:attribute ref="isbn"/>
```

- Definición *"inline"* del valor del atributo.

```
<xsd:attribute name="codigo" use="required">
 <xsd:simpleType>
  ...
 </xsd:simpleType>
</xsd:attribute>
```

Agrupar Declaraciones: Con la finalidad de organizar un esquema XML, podemos usar las instrucciones `xsd:group` y `xsd:attributeGroup` para agrupar declaraciones de elementos y atributos respectivamente. Para poder referenciar un grupo, este debe ser declarado globalmente (*esquema08.xsd*).

```
<xsd:complexType name="CTlibro">
 <xsd:sequence>
  <xsd:group ref="elementosPublicacion"/>
  <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
 </xsd:sequence>
 <xsd:attributeGroup ref="atributosPublicacion"/>
</xsd:complexType>

<xsd:group name="elementosPublicacion">
 <xsd:sequence>
  <xsd:element name="titulo" type="xsd:string"/>
  <xsd:element name="autor" type="xsd:string"/>
 </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="atributosPublicacion">
 <xsd:attribute name="codigo" type="xsd:string" use="required"/>
 <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
</xsd:attributeGroup>
```

4.3.5. Derivación de Tipos

La Derivación de tipos consiste en crear *tipos derivados* desde tipos ya existentes llamados tipos base. XML esquema soporta la creación de tipos derivados simples y tipos derivados complejos.

Tipos simples Derivados (Tipos simples personalizados): Un nuevo tipo simple derivado puede crearse restringiendo el valor de otro tipo llamado tipo base, el cual puede ser un tipo simple primitivo (Ej. xsd:string) u otro tipo simple derivado. El nuevo tipo simple es creado usando el elemento `xsd:restriction` y especificando valores para una o mas facetas opcionales, las cuales dependen del tipo que será extendido:
- Facetas del *tipo string*: length, minLength, maxLength, pattern, enumeration, whitespace.
- Facetas del *tipo integer*: totalDigits, maxInclusive, maxExclusive, minInclusive, minExclusive, enumeration, whitespace, pattern.
Nota. La faceta *pattern* permite el uso de expresiones regulares. La tabla A.3 presenta ejemplos de expresiones regulares.

Algunos ejemplos de tipos simples derivados (*esquema07.xsd*):

- Aplicando facetas `xsd:minInclusive` y `xsd:maxInclusive`. El valor predefinido para ambas facetas es "1".

```
<xsd:simpleType name="STstock">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="2"/>
  <xsd:maxExclusive value="10"/>
 </xsd:restriction>
</xsd:simpleType>
```

- Aplicando faceta `xsd:enumeration`.

```
<xsd:element name="editorial">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="Minotauro"/>
   <xsd:enumeration value="O'Reilly"/>
   <xsd:enumeration value="McGraw Hill"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

- Aplicando faceta `xsd:length`

```
<xsd:attribute name="codigo" use="required">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:length value="5"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:attribute>
```

- Aplicando faceta `xsd:pattern`.

```
<xsd:simpleType name="STrut">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="\d{8}-(\d{1}|k)"/>
 </xsd:restriction>
</xsd:simpleType>
```

Adicionalmente existen dos instrucciones que nos ayudan a definir el contenido de un tipo simple:

- `xsd:union` permite definir un tipo simple como la unión de 2 o mas tipos simples.

```
<xsd:simpleType name="STnota1">
 <xsd:restriction base="xsd:float">
  <xsd:minInclusive value="0" />
  <xsd:maxInclusive value="7" />
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="STnota2">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="NSP" />
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="STnota">
 <xsd:union memberTypes="STnota1 STnota2"/>
</xsd:simpleType>
```

- `xsd:list` permite definir que el contenido de un tipo simple es una lista de valores separados por espacios.

```
<xsd:simpleType name="STlistaCodigos">
 <xsd:list itemType="xsd:positiveInteger"/>
</xsd:simpleType>
...
<xsd:element name="codigos" type="STlistaCodigos"/>
```

  La definición anterior permitiría un elemento de la forma:
  <codigos>637 746 348 829</codigos>

  Observaciones:
  - No se pueden crear listas de listas, ni listas de tipos complejos
  - En el documento instancia los elementos de la lista deben ir separados por espacios en blanco.
  - Las facetas disponibles para `xsd:list` son: length, minLength, maxLength, enumeration, pattern.

Tipos complejos Derivados: Existen 2 formas de crear tipos complejos derivados, por extensión y por restricción de tipos (*esquema09.xsd*).

- Derivar por extensión: extiende el tipo complejo base agregando mas elementos o atributos. Este tipo de derivación permite aplicar el concepto de polimorfismo.

```
<xsd:complexType name="CTpublicacion">
 <xsd:sequence>
  <xsd:element name="titulo" type="xsd:string"/>
  <xsd:element name="autor" type="xsd:string" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="codigo" type="xsd:string" use="required"/>
 <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
</xsd:complexType>

<xsd:complexType name="CTlibro">
 <xsd:complexContent>
  <xsd:extension base="CTpublicacion">
   <xsd:sequence>
    <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Un documento instancia permitirá los siguientes elementos:

```
<publicacion codigo="42345">
 <titulo>Comunidades Virtuales</titulo>
 <autor>J.H.Gray</autor>
 <autor>S.Kling</autor>
</publicacion>

<libro codigo="34674" stock="5">
 <titulo>El Hobbit</titulo>
 <autor>J.R.Tolkien</autor>
 <editorial>Minotauro</editorial>
</libro>
```

- Derivar por restricción: define un tipo el cual es subconjunto del tipo base. Existen 2 maneras:

  - Redefinir el tipo base para restringir el contenido eliminando o agregando elementos. Siguiendo el ejemplo anterior, podemos definir un elemento revista como una publicación que no tiene autores.

```
<xsd:complexType name= "CTrevista">
 <xsd:complexContent>
  <xsd:restriction base="CTpublicacion">
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="numero" type="xsd:string"/>
   </xsd:sequence>
  </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

    Un documento instancia permitirá el siguiente elemento:

```
<revista codigo="38475">
 <titulo>PC-Magazine</titulo>
 <numero>Enero2006</numero>
</revista>
```

  - Redefinir el tipo base para tener un número más restringido de ocurrencias o valores. Esto se logra modificando los valores para `minOccurs` y `maxOccurs` para los subelementos y `use` para los atributos. En el ejemplo inicial, podemos podemos definir un tipo de publicación que tenga un solo autor.

```
<xsd:complexType name="CTpublicacionPersonal">
 <xsd:complexContent>
  <xsd:restriction base="CTpublicacion">
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

    Un documento instancia permitirá el siguiente elemento:

```
<publicacionPersonal codigo="43434">
 <titulo>Remembranzas de guerra</titulo>
 <autor>A. Scholl</autor>
</publicacionPersonal>
```

Tipos complejos especiales:

- Definir un tipo complejo con atributos y contenido simple

```
<xsd:element name="precioA">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:decimal">
    <xsd:attribute name="moneda" type="xsd:string"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

  Un documento instancia permitirá el siguiente elemento:

```
<precioA moneda="peso">5000</precioA>
```

- Definir un tipo complejo con atributos y contenido vacío

```
<xsd:element name="precioB">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:restriction base="xsd:anyType">
    <xsd:attribute name="moneda" type="xsd:string" use="required"/>
    <xsd:attribute name="valor" type="xsd:decimal" use="required"/>
   </xsd:restriction>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

  Un documento instancia permitirá el siguiente elemento:

```
<precioB moneda="peso" valor="5000"/>
```

Controlar la derivación de tipos complejos: Al definir un elemento complejo, podemos evitar su posible derivación definiendo el atributo `final` con alguno de los siguientes valores:
- "restriction": El elemento no permite derivaciones por restricción.
- "extension": El elemento no permite derivaciones por extensión.
- "#all"El elemento no permite derivaciones ni por extensión ni por restricción.

```
<xsd:complexType name="..." final="restriction | extension | #all" >
```

4.4. Espacios de nombres (XML Namespaces)

4.4.1. Introducción

Un *Uniform Resource Identifier (URI)* [7] o Identificador Uniforme de Recursos es una cadena compacta de caracteres que permite identificar un recurso abstracto o físico (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc).

Normalmente un URI consta de dos partes, el identificador del método de acceso o protocolo al recurso (ej. http:, mailto:, ftp) y el nombre del recurso (ej. "//www.libreria.org"). Pueden identificarse dos tipos de URI:

- URL (Uniform Resource Locator - Localizador Uniforme de Recursos) [8]: permite identificar recursos mediante una representación de su mecanismo primario de acceso (ej. localización en la red), en vez de identificar el recurso por su nombre o por otro atributo del recurso.

  `http://www.libreria.org/libros/elhobbit.xml`

- URN (Uniform Resource Name - Nombre Uniforme de Recursos) [9]: permite etiquetar persistentemente un recurso con un identificador aún cuando el recurso desaparezca o no este disponible. El recurso es identificado independientemente de su localización.

  `urn:libreria:libros:elhobbit:xml`

4.4.2. Objetivo de los Espacios de Nombres

Supongamos que tenemos los siguientes fragmentos XML; Un primer fragmento contiene datos de Chile, el segundo representa una inversión y el tercero junta los datos anteriores.

```
<!-- fragmento 1 -->
<pais nombre="Chile">
  <capital>Santiago</capital>
</pais>

<!-- fragmento 2 -->
<inversion>
  <capital>$7000</capital>
</inversion>
```

```
<!-- fragmento 3 -->
<inversiones>
 <pais nombre="Chile">
  <capital>Santiago</capital>
  <capital>$1200</capital>
 </pais>
 ...
</inversiones>
```

Vemos que el significado del elemento *"capital"* depende del contexto en el cual se utiliza. Este problema se conoce como *Homonimia* (mismo nombre con diferentes propósitos) y puede causar muchos problemas cuando queremos reutilizar o integrar datos.

El problema de homonimia, puede solucionarse asociando a cada elemento un URI que indica cual es el contexto (o espacio de nombres) al que pertenece. Por ejemplo, podríamos definir dos espacios de nombres, uno para identificar datos sobre geografía (`"http://www.geografia.org/terminos#"`) y otro para datos sobre economía (`"http://www.economia.org/terminos#"`). De esta manera los fragmentos iniciales se representarían como:

```
<http://www.geografia.org/terminos#pais nombre="Chile">
 <http://www.geografia.org/terminos#capital>
  Santiago
 </http://www.geografia.org/terminos#capital>
</http://www.geografia.org/terminos#pais>

<http://www.economia.org/terminos#inversion>
 <http://www.economia.org/terminos#capital>
  $7000
 </http://www.economia.org/terminos#capital>
</http://www.economia.org/terminos#inversion>

<inversiones>
 <http://www.geografia.org/terminos#pais nombre="Chile">
  <http://www.geografia.org/terminos#capital>
   Santiago
  </http://www.geografia.org/terminos#capital>
  <http://www.economia.org/terminos#capital>$1200</capital>
 </http://www.geografia.org/terminos#pais>
 ...
</inversiones>
```

Esta solución genera un nuevo problema, al agregar a cada elemento el URI al que pertenece sobrecargamos el documento con información repetida y perdemos legibilidad.

Para evitar la información redundante de la solución anterior, se propuso definir un *alias* para cada URI utilizado en el documento, el cual podrá ser usado como un *prefijo* al utilizar los elementos de un espacio de nombres.

```
<!--
Prefijos:
geo =  http://www.geografia.org/terminos#
eco = http://www.economia.org/terminos#
-->
<inversiones>
 <geo:pais nombre="Chile">
  <geo:capital>Santiago</geo:capital>
  <eco:capital>$1200</eco:capital>
 </geo:pais>
 ...
</inversiones>
```

En resumen, los espacios de nombres nos permiten solucionar el problema de homonimia y son ampliamente utilizados para combinar vocabularios ya que facilitan la incorporación de elementos no previstos inicialmente

4.4.3. Definiendo y utilizando espacios de nombres

Definiendo espacios de nombres en un esquema XML .
El elemento *schema* define los siguientes atributos relacionados a espacios de nombres (ver el ejemplo *esquema10.xsd*):

- `targetNamespace` (espacio de nombres de destino) define el espacio de nombres del esquema. El targetNamespace asocia los elementos definidos en el esquema con un espacio de nombres.
- `xmlns` define un espacio de nombres por defecto (default Namespace). Este será usado para identificar elementos que no tienen asignado un prefijo dentro del esquema. En el ejemplo el namespace por defecto es idéntico al targetNamespace, pero puede referenciar otro namespace.
- `xmlns:alias` define un prefijo "alias"que permitirá referenciar a un namespace en el ámbito del esquema (namespace global). De esta manera se pueden definir distintos alias a namespaces, por ejemplo `xmlns:xsd` asigna el alias *"xsd"* al namespace de XML Schema.
- Los atributos `elementFormDefault` y `attributeFormDefault` especifican si los elementos o atributos globales definidos en el esquema deben ser explícitamente calificados ("qualified") o no calificados ("unqualified") en un documento instancia (Ver mas adelante).

esquema10.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.libreria.org"
            xmlns="http://www.libreria.org"
            xmlns:lib="http://www.libreria.org"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="unqualified"
            attributeFormDefault="unqualified">
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="libro" type="CTlibro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:complexType name="CTlibro">
  <xsd:sequence>
   <xsd:element ref="titulo"/>
   <xsd:element name="autor" type="xsd:string"/>
   <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="codigo" type="xsd:string" use="required"/>
  <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
 </xsd:complexType>
 <xsd:element name="titulo" type="xsd:string"/>
</xsd:schema>
```

A excepción de los esquemas sin espacio de nombres, un esquema XML define al menos dos espacios de nombres, el *targetNamespace* y el *XML Schema namespace* (`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`).

Espacios de nombres en instancias XML .
Si el esquema no define un espacio de nombres, se hace uso del atributo `xsi:noNamespaceSchemaLocation` para referenciarlo.

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="esquema04.xsd">
  ...
  ...
</libreria>
```

Cuando un esquema define un espacio de nombres usando el atributo *targetNamespace*, sus documentos instancia deberán hacer referencia a este esquema usando el atributo `xsi:schemaLocation` y declarar los prefijos de espacios de nombres que sean necesarios. Observar el ejemplo *instancia10.xml*.

instancia10.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:libreria
  xmlns:lib="http://www.libreria.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.libreria.org esquema10.xsd">
 <libro codigo="34674" stock="5">
  <lib:titulo>El Hobbit</lib:titulo>
  <autor>J.R.Tolkien</autor>
  <editorial>Minotauro</editorial>
 </libro>
 <libro codigo="83467">
  <lib:titulo>El arte de la Guerra</lib:titulo>
  <autor>Sun-Tzu</autor>
 </libro>
</lib:libreria>
```

Calificar o no calificar elementos en los documentos instancia .
Si un elemento o atributo especifica su espacio de nombres usando un prefijo (por ejemplo `<lib:titulo>`), o usando el namespace por defecto, decimos que es un elemento o atributo calificado.

Los atributos `elementFormDefault`(forma de elemento por defecto) y `attributeFormDefault`(forma de atributo por defecto) permiten especificar si los elementos o atributos deberán ir calificados o no en un documento instancia:

- Si su valor es 'unqualified"(valor por defecto) *solo* los elementos o atributos declarados globalmente en el esquema deben ir calificados en un documento instancia. En contraste, los elementos y atributos definidos localmente no deben ir calificados.
- Si su valor es "qualified", todos los elementos o atributos usados en un documento instancia deben ir calificados.

Observaciones:
- Los elementos y atributos definidos globalmente en el esquema deberán ir necesariamente calificados en un documento instancia, sin importar el valor de los atributos `elementFormDefault` o `attributeFormDefault`.
- El uso del default namespace como medio para calificar no es aplicable para *attributeFormDefault="qualified"*.
- La calificación puede especificarse de forma separada para cada declaración local utilizando el atributo *form*.
- Si un esquema no tiene un namespace asociado, los atributos `elementFormDefault` y `attributeFormDefault` no afectan los documentos instancia.
- Los ejemplos *esquema10a, esquema10b, esquema10c, esquema10d* presentan las posibles situaciones de calificar y no calificar.

4.5. Conceptos Avanzados

4.5.1. Elementos Nulos

El atributo `nillable="true"` permite especificar que un elemento puede aceptar un valor nulo, sin implicar que el elemento este vacio. Por ejemplo considere la siguiente declaración (*esquema14.xsd*):

```
<xsd:element name="persona">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="nombre" type="xsd:string"/>
   <xsd:element name="primerApellido" type="xsd:string"/>
   <xsd:element name="segundoApellido" type="xsd:string" nillable="true"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Un documento instancia permitirá el siguiente elemento:

```
<persona>
 <nombre>Luis</nombre>
 <primerApellido>Rojas</primerApellido>
 <segundoApellido xsi:nil="true"/>
</persona>
```

Observaciones:
- El atributo `xsi:nil` se define como parte del espacio de nombres para instancias del esquema XML.
- Un elemento con atributo `xsi:nil="true"` puede no tener contenido pero puede tener atributos.

4.5.2. Elementos `any`, `anyType` y `anyAttribute`

Elementos con cualquier contenido El tipo de dato *xsd:anyType* no restringe el contenido de un elemento de modo alguno, permitiendo cualquier mezcla de caracteres excepto elementos.

```
<xsd:element name="cualquier_cosa" type="xsd:anyType">
```

anyType es el tipo usado por defecto cuando no se especifica alguno. Por lo tanto la declaración anterior es equivalente a :

```
<xsd:element name="cualquier_cosa"/>
```

Permitiendo cualquier elemento o atributo .
El elemento `xsd:any` permite que en un documento instancia contenga elementos no definidos por el esquema.
De manera similar, el elemento `xsd:anyAttribute` permite que un elemento contenga cualquier atributo no definido por el esquema.

El siguiente ejemplo (*esquema15.xsd*), muestra la definición del elemento `persona`, el cual permite agregar elementos y/o atributos no definidos en el esquema como `id`, `edad`, `direccion` y `telefono`.

```
<xsd:element name="persona">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="nombre" type="xsd:string"/>
   <xsd:element name="apellidos" type="xsd:string"/>
   <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute/>
 </xsd:complexType>
</xsd:element>
```

Un documento instancia (*instancia15.xml*) permitirá el siguiente elemento:

```
<persona id="427" edad="26">
 <nombre>Luis</nombre>
 <apellidos>Rojas</apellidos>
 <direccion>Romero 2385</direccion>
 <telefono>89428282</telefono>
</persona>
```

Nota: Un documento instancia cuyo esquema utiliza `xsd:any` o `xsd:anyAttribute` se denomina *extensible*.

El atributo `processContents` especifica como se evaluará el contenido del elemento o atributo, pudiendo tomar uno de los siguientes valores:
- `"strict"` el contenido debe estar declarado en el esquema (default).
- `"skip"` el contenido solo debe ser bien formado.
- `"lax"`, aceptar cualquier contenido y validar cuando sea posible.

4.5.3. Grupos de sustitución

Los grupos de sustitución permiten que los elementos sean sustituidos por otros elementos. Se definen declarando un elemento denominado "head"(cabecera) y luego declarando otros elementos sustituibles usando el atributo `substitutionGroup`.
El siguiente ejemplo (*esquema18.xsd*) declara el grupo de sustitución `grupoTrabajo`:

```
<xsd:element name="grupoTrabajo">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="integrante" minOccurs="3" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<!-- declaramos un elemento que actuará como "head" -->
<xsd:element name="integrante" type="xsd:string"/>
<!-- asignamos elementos al grupo de sustitucion-->
<xsd:element name="jefe" substitutionGroup="integrante"
             type="xsd:string"/>
<xsd:element name="analista" substitutionGroup="integrante"
             type="xsd:string"/>
<xsd:element name="programador" substitutionGroup="integrante"
             type="xsd:string"/>
```

Un instancia válida sería (*instancia18.xml*):

```
<grupoTrabajo>
 <integrante>Jorge</integrante>
 <jefe>Adrian</jefe>
 <analista>Luis</analista>
 <programador>Andres</programador>
</grupoTrabajo>
```

Observaciones:
- Los elementos del *substitutionGroup* pueden ser utilizados en cualquier lugar que pudiésemos utilizar el elemento cabecera.

- El elemento cabecera así como los elementos del *substitutionGroup* deben ser declarados globalmente.
- Cuando se crean grupos de sustitución a nivel de tipos, se debe considerar que el tipo de cada elemento del substitutionGroup debe ser el mismo o un derivado del tipo del elemento cabecera.

4.5.4. Elementos y tipos abstractos

Los tipos o elementos abstractos sirven como plantilla para crear tipos derivados. Un elemento o tipo se declara como *abstracto* definiendo el atributo `abstract="true"`.

Elementos Abstractos: son elementos que pueden ser reemplazados por algún elemento perteneciente a un grupo de sustitución. Un elemento abstracto no puede usarse en un documento instancia.

El siguiente ejemplo (*esquema16.xsd*) declara el elemento abstracto `integrante`:

```
<!-- declaramos un elemento que actuara como cabecera -->
<xsd:element name="integrante" type="xsd:string" abstract="true"/>

<!-- asignamos elementos al grupo de sustitucion-->
<xsd:element name="jefe" substitutionGroup="integrante"
             type="xsd:string"/>
<xsd:element name="analista" substitutionGroup="integrante"
             type="xsd:string"/>
<xsd:element name="programador" substitutionGroup="integrante"
             type="xsd:string"/>
<xsd:element name="grupoTrabajo">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="integrante" minOccurs="2" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Un instancia válida sería (*instancia16.xml*):

```
<grupoTrabajo>
 <jefe>Adrian</jefe>
 <analista>Luis</analista>
 <programador>Andres</programador>
</grupoTrabajo>
```

Tipos Abstractos: se implementan definiendo un tipo abstracto y uno o mas tipos derivados. En un documento instancia se puede usar el elemento abstracto pero requiere el atributo `xsi:type=‘‘..."` para referenciar el tipo derivado que implementará.
El siguiente ejemplo (*esquema17.xsd*) declara el tipo abstracto `CTpublicacion`:

```
<!-- Elemento que implementa el tipo abstracto -->
<xsd:element name="publicacion" type="CTpublicacion"/>
<!-- declaracion del tipo abstracto -->
<xsd:complexType name="CTpublicacion" abstract="true">
 <xsd:sequence>
  <xsd:element name="titulo" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<!-- declaracion del tipo derivado CTlibro -->
<xsd:complexType name="CTlibro">
 <xsd:complexContent>
  <xsd:extension base="CTpublicacion">
   <xsd:sequence>
    <xsd:element name="autor" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<!-- declaracion del tipo derivado CTrevista -->
<xsd:complexType name="CTrevista">
 <xsd:complexContent>
  <xsd:extension base="CTpublicacion">
   <xsd:sequence>
    <xsd:element name="numero" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

La declaración anterior permite el siguiente contenido (*instancia17.xml*):

```
<publicacion xsi:type="CTlibro">
 <titulo>El Hobbit</titulo>
 <autor>J.R.Tolkien</autor>
</publicacion>
<publicacion xsi:type="CTrevista">
 <titulo>PC-Magazine</titulo>
 <numero>Enero2006</numero>
</publicacion>
```

4.5.5. Unicidad y declaración de Claves

Unicidad: el elemento `xsd:unique` permite indicar que el valor de un atributo o elemento debe ser único dentro de un cierto contexto.

El siguiente ejemplo (*esquema19.xsd*) declara el atributo `rut` como único para elementos `vendedor` dentro del contexto del elemento `vendedores`:

```
<xsd:element name="vendedores">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="vendedor" type="CTvendedor" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:unique name="Urut">
  <xsd:selector xpath="vendedor"/>
  <xsd:field xpath="@rut"/>
 </xsd:unique>
</xsd:element>
<xsd:complexType name="CTvendedor">
 <xsd:sequence>
  <xsd:element name="nombre" type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="rut" type="xsd:integer"/>
</xsd:complexType>
```

Un instancia válida sería (*instancia19.xml*):

```
<vendedores>
 <vendedor rut="100">
  <nombre>Juan</nombre>
 </vendedor>
 <vendedor rut="101">
  <nombre>pedro</nombre>
 </vendedor>
</vendedores>
```

Observaciones:
- El elemento `xsd:selector` define el elemento que deseamos definir como "único".
- El elemento `xsd:field` define el atributo o subelemento que será usado para identificar al elemento.
- Es posible crear combinaciones de campos que deben ser únicos agregando elementos `xsd:field` para identificar los valores involucrados.

Claves: Una clave es un valor o conjunto de valores que identifican unívocamente un elemento. El elemento `xsd:key` permite definir claves haciendo referencia a un atributo, un elemento e incluso se permiten claves compuestas por una combinación de elementos y atributos. El valor de una clave debe ser único y no puede ser igualado a "nil".

Referencias a Claves: Cuando el valor de un atributo o elemento tiene el valor (referencia) de una clave, este elemento o atributo se denomina *clave externa*. El elemento `xsd:keyref` permite declarar clave externas. El declarar un elemento o atributo como clave externa no implica que su valor debe ser único, pero significa que debe existir como clave en algún lugar de la instancia.

Por ejemplo, dado el siguiente documento instancia (*instancia20.xml*):

```
<libreria>
 <catalogo>
  <producto codigo="100">
   <nombre>PC Magazine</nombre>
   <precio>400</precio>
  </producto>
  <producto codigo="101">
   <nombre>PC World</nombre>
   <precio>200</precio>
  </producto>
 </catalogo>
 <ventas>
  <venta>
   <codigoProducto>101</codigoProducto>
   <cantidad>5</cantidad>
  </venta>
 </ventas>
</libreria>
```

El siguiente esquema (*esquema20.xsd*), define que el atributo `codigo` es la clave de cada elemento `producto`, y que el atributo `codigoProducto` de un elemento `venta` es una clave externa que hace referencia a un código de producto. Ambas restricciones deben ser respetadas dentro del contexto del elemento `libreria`.

```
<xsd:element name="libreria">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="catalogo" type="CTcatalogo"/>
   <xsd:element name="ventas" type="CTVentas"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:key name="PKcodigoProducto">
  <xsd:selector xpath="catalogo/producto"/>
  <xsd:field xpath="@codigo"/>
 </xsd:key>
 <xsd:keyref name="FKcodigoProducto" refer="PKcodigoProducto">
  <xsd:selector xpath="ventas/venta"/>
  <xsd:field xpath="codigoProducto"/>
 </xsd:keyref>
</xsd:element>
<xsd:complexType name="CTcatalogo">
 <xsd:sequence>
  <xsd:element name="producto" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="nombre" type="xsd:string"/>
     <xsd:element name="precio" type="xsd:double"/>
    </xsd:sequence>
    <xsd:attribute name="codigo" type="xsd:string"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CTVentas">
 <xsd:sequence>
  <xsd:element name="venta" maxOccurs="unbounded">
   <xsd:complexType name="CTventa">
    <xsd:sequence>
     <xsd:element name="codigoProducto" type="xsd:string"/>
     <xsd:element name="cantidad" type="xsd:integer"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

4.6. Reutilización de Esquemas

4.6.1. Inclusión de esquemas

El elemento `xsd:include` permite incluir definiciones y declaraciones desde otro esquema, el cual no debe definir un targetNamespace o este debe ser igual al targetNamespace del esquema principal.

Si un esquema incluido no define un targetNamespace, este adquiere el namespace del esquema que lo incluye. Esto se conoce como *efecto camaleón* y los componentes del esquema incluido se llaman *componentes camaleón*. Por ejemplo, el siguiente esquema (*esquema11a.xsd*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="libro">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
    <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="codigo" type="xsd:string" use="required"/>
   <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

es incluido en el siguiente esquema principal (*esquema11.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.libreria.org"
 xmlns:lib="http://www.libreria.org"
 elementFormDefault="qualified">
 <xsd:include schemaLocation="esquema11a.xsd"/>
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="lib:libro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

el cual permite validar la siguiente instancia (*instancia11.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria xmlns="http://www.libreria.org"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.libreria.org esquema11.xsd">
 <libro codigo="34674" stock="5">
  <titulo>El Hobbit</titulo>
  <autor>J.R.Tolkien</autor>
  <editorial>Minotauro</editorial>
 </libro>
 ...
</libreria>
```

4.6.2. Importación de esquemas

El elemento `xsd:import` permite importar componentes desde otro esquema con un targetNamespace distinto al del esquema importador.

Por ejemplo, el siguiente esquema (*esquema12a.xsd*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.libros.org"
            xmlns:libros="http://www.libros.org">
 <xsd:element name="libro">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
    <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="codigo" type="xsd:string" use="required"/>
   <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

es importado en el siguiente esquema principal (*esquema12.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.libreria.org"
            xmlns:libros="http://www.libros.org">
 <xsd:import schemaLocation="esquema12a.xsd" namespace="http://www.libros.org"/>
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
```

```
    <xsd:element ref="libros:libro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

el cual permite validar la siguiente instancia (*instancia12.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:libreria xmlns:lib="http://www.libreria.org"
              xmlns:libros="http://www.libros.org"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.libreria.org esquema12.xsd">
 <libros:libro codigo="34674" stock="5">
  <titulo>El Hobbit</titulo>
  <autor>J.R.Tolkien</autor>
  <editorial>Minotauro</editorial>
 </libros:libro>
 ...
</lib:libreria>
```

Cuando `xsd:import` especifica el atributo `namespace`, este debe coincidir con el targetNamespace del esquema importado; los componentes importados mantienen su espacio de nombres. Cuando solo se especifica el atributo `schemaLocation`, el esquema a importar no debe definir un targetNamespace; los nuevos componentes no tendrán espacio de nombres.

4.6.3. Redefinición de esquemas

El elemento `xsd:redefine` es similar a `xsd:include` pero posibilita la redefinición de tipos complejos, tipos simples, grupos de elementos y grupos de atributos que se obtienen desde otros esquemas.

Por ejemplo, el tipo complejo "CTlibro"definido en el siguiente esquema (*esquema13a.xsd*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:complexType name="CTlibro">
  <xsd:sequence>
   <xsd:element name="titulo" type="xsd:string"/>
   <xsd:element name="autor" type="xsd:string"/>
   <xsd:element name="editorial" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="codigo" type="xsd:string" use="required"/>
 </xsd:complexType>
</xsd:schema>
```

El esquema anterior es redefinido en el siguiente esquema (*esquema13.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:redefine schemaLocation="esquema13a.xsd">
  <xsd:complexType name="CTlibro">
   <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="codigo" type="xsd:string" use="required"/>
   <xsd:attribute name="stock" type="xsd:integer" use="optional"/>
  </xsd:complexType>
 </xsd:redefine>
 <xsd:element name="libreria">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="libro" type="CTlibro" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

en el cual hemos eliminando el elemento "editorial", y a cambio hemos agregado el atributo "stock". Este ultimo esquema permite validar la siguiente instancia (*instancia13.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.libreria.org esquema13.xsd">
 <libro codigo="34674" stock="5">
  <titulo>El Hobbit</titulo>
  <autor>J.R.Tolkien</autor>
 </libro>
 ...
</libreria>
```

Al igual que con *xsd:include*, los esquemas redefinidos deben tener el mismo targetNamespace que el esquema principal. El efecto Camaleón también es posible.

4.7. Desventajas de XML Schema

- No soporta entidades como un mecanismo para crear macros (los DTDs si soportan)
  <!ENTITY &texto; "Este texto se repite muchas veces">
- La especificación de restricciones es limitado. Ejemplo: Verificar que un valor total es correcto acorde a la suma de un conjunto de valores parciales.
- Sensibilidad al contexto limitada. Ejemplo: especificar que el contenido de un elemento depende del valor de un atributo.
- Tamaño de archivos XML Schema puede ser excesivo y complejo.
- La flexibilidad que presenta (existen varias opciones para declarar estructuras), hace necesaria la especificación de buenas practicas.
- Falta de Legibilidad en especificaciones complejas y poco modulares.

4.8. Buenas prácticas para el diseño de esquemas XML

4.8.1. Aspectos a considerar al diseñar esquemas XML

- *Modularidad:* Determinar como se dividirá la información a modelar (en términos de documentos y elementos). Esto influirá en la facilidad de lectura, escritura y procesamiento de los documentos.
- *Flexibilidad:* Diseñar esquemas flexibles en términos de la estructuración y división de los datos. Acá debe considerarse cuando modelar datos como elementos o atributos.
- *Extensibilidad:* Diseñar esquemas flexibles manejando adecuadamente los espacios de nombre y la reutilizacion de tipos simples y complejos.
- *Consistencia:* Evitar características incoherentes
- *Nivel de abstracción:* Buscar término medio en nivel de detalle

  ```
  <fecha>10 Marzo 2003</fecha>
  ```

  versus

  ```
  <fecha><día>10</dia><mes>Marzo</mes><año>2003</año></fecha>
  ```

4.8.2. Tipos complejos anónimos Vs. Tipos complejos con nombre

Considerar el siguiente ejemplo:

```
<!-- Tipo complejo anonimo -->
<xsd:element name="alumno">
 <xsd:complexType>
  ...
 </xsd:complexType>
</xsd:element>

<!-- Tipo complejo con nombre -->
<xsd:ComplexType name="TipoAlumno">
 ...
</xsd:ComplexType>
...
<xsd:element name="alumno" type="TipoAlumno"/>
```

- Si se busca legibilidad o el concepto no se reutilizara, entonces definir como tipo complejo anónimo.
- Si el concepto sera utilizado en distintas partes del esquema (o importado por otros esquemas), entonces definir como tipo complejo con nombre.
- Al definir tipos complejos anónimos se genera un anidamiento de definiciones conocido como *Diseño de Muñeca Rusa*.

4.8.3. ¿Atributo o SubElemento?

- *Atributos:* Información sobre el contenido (meta-información). Valores asociados con objetos sin identidad propia (ej. edad).
- *Subelementos:* Datos que se consideran contenido de otro elemento. valores con identidad propia (ej. fecha-nacimiento).

Por ejemplo, modelar `edad` como elemento o como atributo:

```
<persona>
 <nombre>Juan</nombre>
 <edad>23</edad>
</persona>
```

```
<persona edad="23">
 <nombre>Juan</nombre>
</persona>
```

Capítulo 5

Consulta de datos en XML

En este capítulo revisaremos dos miembros de la familia XML que son utilizados para consultar documentos XML. XPath [10] permite definir expresiones de caminos para consultar partes de un árbol XML. XQuery [12] es el lenguaje de consulta para documentos XML propuesto por la W3C.

5.1. XPath

La especificación de XPath [11], define un lenguaje de expresiones para acceder o referenciar partes de un documento XML. XPath define un conjunto de funciones estándar para la navegación y selección sobre documentos XML, las cuales sirven de base para definir transformaciones (XSL) y consultas (XQuery).

Expresiones XPath XPath se basa en el uso de expresiones en forma de camino para especificar un recorrido (navegación) a través de la estructura de árbol propia de un documento XML y adicionalmente seleccionar nodos (elementos y atributos) encontrados en este camino. Estas expresiones son similares a las utilizadas al trabajar con un sistema de archivos.

Funciones Estándar XPath XPath define algo mas de 100 funciones, entre las cuales se incluyen funciones para:
- comparación de cadenas, valores numéricos, fecha y hora;
- manipulación de nodos;
- manipulación de secuencias;
- valores verdadero-falso, etc.

5.1.1. Nodos XPath

Existen 7 tipos de nodos en XPath, los cuales están descritos en la tabla 5.1

Tipo de Nodo	Representa	función asociada
root	al elemento "root"(raíz), es decir todo el documento	/
elemento	un elemento	*
atributo	el atributo de un elemento	@*
texto	el texto de un elemento	*text()*
comentario	un comentario del autor	*comment()*
instrucción	instrucciones de otro lenguaje	*processing-instruction()*
namespace	la definición de un namespace	*namespace()*

Cuadro 5.1: Tipos de Nodos XPath

Adicionalmente, la estructura de árbol própia de un documento XML, describe ciertas relaciones entre los nodos:

- *Padre*: cada elemento o atributo tiene un solo padre.
- *Hijos*: los nodos elemento pueden tener cero, uno o mas hijos.
- *Hermanos*: nodos que tienen el mismo padre.
- *Ancestros*: nodos que contienen un elemento
- *Descendientes*: nodos que contiene un elemento

Por ejemplo, en el documento *instancia21.xml*, podemos identificar los siguiente nodos:
- Nodo documento o raiz: <libreria>
- Nodo Elemento: <autor>J.R.Tolkien<autor>
- Nodo Atributo: stock="5"
- Valores atómicos: J.R.Tolkien, "5"

instancia21.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria>
 <libro codigo="34674" stock="5">
  <titulo>El Hobbit</titulo>
  <autor>J.R.Tolkien</autor>
  <editorial>Minotauro</editorial>
  <precio moneda="pesos">6000</precio>
 </libro>
 <revista codigo="82029" stock="4">
  <titulo>Pc Magazine</titulo>
  <numero>Nov2005</numero>
  <precio moneda="pesos">4567</precio>
  <contenido>
   <articulo pagina="3">
    <titulo>John Sardini</titulo>
    <autor>Los nuevos microprocesadores</autor>
   </articulo>
  </contenido>
 </revista>
 <libro codigo="83467" stock="2">
  <titulo>El arte de la Guerra</titulo>
  <autor>Sun-Tzu</autor>
  <editorial>Planeta</editorial>
  <precio moneda="dolar">4567</precio>
 </libro>
 <revista codigo="82030" stock="7">
  <titulo>Pc Magazine</titulo>
  <numero>Nov2005</numero>
  <precio moneda="pesos">4567</precio>
  <contenido>
   <articulo pagina="5">
    <autor>Mike Morwood</autor>
    <autor>Thomas Sutikna</autor>
    <titulo>La isla que el tiempo olvidó</titulo>
   </articulo>
   <articulo pagina="16">
    <autor>Josh Fischman</autor>
    <titulo>Antiguos Aventureros</titulo>
   </articulo>
  </contenido>
 </revista>
 <libro codigo="33312" stock="3">
  <titulo>Historia de la Ciencia</titulo>
  <autor>John Gribbin</autor>
  <editorial>Crítica</editorial>
  <precio moneda="dolar">15</precio>
 </libro>
</libreria>
```

5.1.2. Sintaxis XPath

Selección de nodos: XPath usa expresiones con estructura de camino para seleccionar nodos en un documento XML. Los nodos son seleccionados al seguir el camino descrito por la expresión y el resultado puede ser un conjunto de nodos, una cadena, un número, o un valor boleano.

La tabla 5.2 muestra las expresiones XPath mas útiles, y la tabla 5.3 presenta algunos ejemplos de expresiones de camino

Expressión	Descripción
Nodo	Selecciona todos los hijos (nodos) de un nodo
/	Selección desde el nodo raiz
//	Permite acceso rápido a los descendientes del nodo actual
.	Selecciona el nodo actual
..	Selecciona el padre del nodo actual
@∗	Selecciona atributos
|	permite seleccionar varios caminos
∗	Selecciona cualquier nodo sea cual sea su nombre

Cuadro 5.2: Entidades XML predefinidas

Expressión XPath	Selecciona:
*/libreria/**	todos los nodos hijo del elemento *libreria*
/libreria/revista/contenido/articulo	todos los articulos de revistas
/libreria/revista/@codigo	solo códigos de revistas
*//**	todos los elementos del documento
//autor	todos los elementos *autor* sin importar donde se ubican en el documento
//@codigo	todos los atributos *codigo* sin importar donde se ubican en el documento
/libreria//titulo	todos los elementos *titulo* que son descendientes de *libreria*
/libreria//@moneda	todos los atributos *moneda* que son descendientes de *libreria*
//libro/titulo | //articulo/titulo	los titulos de libros y artículos
/libreria/libro/autor/text()	el texto desde todos los nodos autor

Cuadro 5.3: Ejemplos de expresiones de camino XPath

Selección de nodos desconocidos: XPath define tres símbolos comodín (wilcards) para seleccionar elementos desconocidos. La tabla 5.4 muestra estos símbolos.

Wildcard	Descripción
***	empareja cualquier nodo elemento
*@**	empareja cualquier nodo atributo
node()	empareja cualquier nodo de algún tipo

Cuadro 5.4: Wildcards

Predicados: Los predicados permiten agregar condiciones a las expresiones de camino. Los predicados son incluidos entre corchetes, en cualquier parte de una expresión de camino. La tabla 5.5 muestra algunos ejemplos de expresiones XPath con predicados.

Expressión XPath	Selecciona:
libreria/libro[1]	el primer elemento *libro* que es hijo de *libreria*
libreria/libro[last()]	el ultimo elemento *libro* que es hijo de *libreria*
libreria/libro[last()-1]	el penultimo elemento *libro* que es hijo de *libreria*
libreria/libro[position()<2]	los elementos *libro* que estén en las primeras 2 posiciones
//libro/precio[@moneda='pesos']	todos los libros con precio en pesos
//libro[@stock<5]/titulo	todos los titulos de libros con stock menor a 5

Cuadro 5.5: Ejemplos de predicados XPath

Existe un conjunto de operadores que pueden ser usados para definir condiciones en los predicados. Estos pueden ser, operadores matemáticos (ver tabla 5.6) y operadores booleanos (ver tabla 5.7)

Operador	Descripción
+	Suma
-	Resta
***	Multiplicación
div	División
mod	Módulo

Cuadro 5.6: Operadores Matemáticos

Operador	Descripción
=	Igualdad
<	Menor
<=	Menor e igual
>	Mayor
>=	Mayor e igual
!=	Diferente
and	Y
or	O

Cuadro 5.7: Operadores Boleanos

Ejes XPath (XPath Axes) Un *Eje* define un conjunto de nodos relativos al nodo actual. La tabla 5.8 muestra los ejes definidos por XPath y la tabla 5.9 algunos ejemplos.

Nombre del Eje	Selecciona
*self::**	selecciona el nodo actual
*parent::**	el padre del nodo actual
*child::**	todos los hijos del nodo actual
*attribute::**	todos los atributos del nodo actual
*ancestor::**	todos los ancestros del nodo actual (padre, abuelo, etc)
*ancestor-or-self::**	todos los ancestros del nodo actual, incluido este último
*descendant::**	todos los descendientes del nodo actual (hijos, nietos, etc)
*descendant-or-self::**	todos los descendientes del nodo actual, incluido este último
*following::**	cualquier cosa despues del tag de clausura del nodo actual
*following-sibling::**	todos los hermanos despues del nodo actual
*preceding::**	cualquier cosa antes del tag de apertura del nodo actual
*preceding-sibling::**	todos los hermanos antes del nodo actual
*namespace::**	todos los nodos namespace de nodo actual

Cuadro 5.8: Ejes XPath

Expresión XPath	Selecciona:
*/libreria/child:** */libreria/child::node()*	todos los nodos hijo de *libreria*
/libreria/child::libro	los nodos hijo del nodo *libreria*, que tengan el nombre *libro*
/libreria//attribute::**	los atributos de cualquier nodo hijo de *libreria*
/libreria/descendant::[name()='titulo']*	cualquier nodo descendiente del nodo *libreria*, con nombre *titulo*
//titulo/parent::[@pagina]*	cualquier nodo cuyo nodo padre tenga un atributo *pagina*

Cuadro 5.9: Ejemplos del uso de ejes XPath

5.1.3. Funciones XPath

La especificación de XPath define una serie de funciones que pueden ser utilizadas al construir expresiones. Existen funciones para trabajar con números (tabla 5.10), cadenas (tabla 5.11), datos booleanos (tabla 5.12), y nodos (tabla 5.13).

Función	Descripción
number()	Convierte cualquier objeto en un número.
round()	Redondea el valor entre paréntesis.
floor()	Redondea hacia abajo el valor entre paréntesis.
ceiling()	Redondea hacia arriba el valor entre paréntesis.
sum()	Suma el contenido.

Cuadro 5.10: Funciones Numéricas

Función	Descripción
string()	Convierte cualquier objeto en una cadena de caracteres.
starts-with()	Devuelve verdadero o falso en función de que la primera cadena empiece o no con la segunda. Ej. *starts-with('abcd','a')* – Result: verdadero Ej. *starts-with('abcd','b')* – Result: falso
contains()	Devuelve verdadero o falso en función de que la primera cadena contenga o no la segunda. Ej. *contains('abcd','a')* – Result: verdadero Ej. *contains('abcd','e')* – Result: falso
substring-before()	Devuelve la subcadena anterior a la segunda cadena encontrada en la primera cadena. Ej. *substring-before('primera-segunda','-')* – Result:"primera"
substring-after()	Devuelve la subcadena posterior a la segunda cadena encontrada en la primera cadena. Ej. *substring-after('primera-segunda','-')* – Result:"segunda"
substring()	Devuelve una subcadena especificando una posición inicial y opcionalmente una cantidad de caracteres a extraer. Ej. *substring('abcdef', 4)* – Result:"def" Ej. *substring('abcdef', 3, 2)* – Result:"cd"
string-length()	Devuelve un numero con la longitud de la cadena. Ej. *string-length('abcdef')* – Result: 6
normalize-space()	Elimina los espacios anteriores y posteriores de la cadena y sustituye las sucesiones de espacios por un solo espacio.
concat()	Devuelve una cadena concatenada de los objetos entre paréntesis.
translate()	Devuelve una cadena como el producto de sustituir una subcadena por otra. Ej. *translate('a-b-c','-','_')* – Result:"a_b_c"

Cuadro 5.11: Funciones para Cadenas

Función	Descripción
true()	Siempre devuelve verdadero.
false()	Siempre devuelve falso.
not()	Cambia el valor de verdadero a falso y de falso a verdadero.
boolean()	Convierte cualquier objeto en verdadero o falso.

Cuadro 5.12: Funciones Boleanas

Función	Descripción
text()	Devuelve el texto del nodo actual. Ej. */libreria/libro/titulo/text()*
position()	Devuelve el numero de posición del nodo actual. Ej. */libreria/revista[position()=1]*
last()	Devuelve el número de posición del último nodo. Ej. */libreria/libro[position()=last()] es el último*
count()	Devuelve el número total de nodos. Ej. *count(//libreria/libro)*
id()	Devuelve un conjunto de nodos que tengan ese identificador. El atributo con el identificador debe estar definido en el esquema como tipo ID.
name()	Devuelve una cadena de caracteres con el nombre completo del elemento (incluyendo el prefijo *namespace*). Ej. *name(element)* – Result:"xsd:element"
local-name()	Devuelve una cadena de caracteres con el nombre local del elemento (sin el prefijo *namespace*). Ej. *local-name(element)* – Result:"element"
namespace-uri()	Devuelve una cadena de caracteres con la URI asociada al "namespace"del elemento. Ej. *namespace-uri(element)* Result:"http://www.w3.org/2001/XMLSchema"
lang()	Devuelve verdadero o falso en función de que el código de idioma sea igual o no al atributo xml:lang más cercano del nodo actual.

Cuadro 5.13: Funciones para Nodos

5.2. XQuery

XQuery [12] es una propuesta de lenguaje de consulta para XML, el cual permite encontrar y extraer elementos y atributos desde documentos bien formados. XQuery esta basado en expresiones XPath (soporta la mismas funciones y operadores) y permite expresar consultas en una sintaxis similar a SQL.

XQuery aún no ha sido aprobado como una recomendación de la W3C y una descripción completa de su sintaxis puede consultarse en [13].

Expresiones FLWOR de XQuery La estructura básica de una consulta en XQuery es una expresión FLWOR (**F**or-**L**et-**W**here-**O**rder-**R**eturn), la cual generaliza las expresiones SELECT-FROM-HAVING-WHERE utilizadas en SQL.

Por ejemplo, usando los datos de los documentos instancia *autores.xml* y *libros.xml*, planteamos la siguiente consulta:

Los títulos de libros, ordenados por precio, cuyo autor tiene el nombre "Pablo"

La solución a esta consulta puede estar dada por la siguiente expresión XQuery (*consulta01.xq*).

```
<respuesta>
{
for $x in doc("d:/ejemplos/libros.xml")/libreria/libro
let $y := doc("d:/ejemplos/autores.xml")
          /autores/autor/nombre[text()='Pablo']/../@codigo
where $x[@autor=$y]
order by $x/precio/text() ascending
return
  <libro>{$x/titulo/text()}</libro>
}
</respuesta>
```

El resultado de la consulta, será el siguiente documento XML (*resultado01.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<respuesta>
 <libro>Viajes</libro>
 <libro>El habitante y su esperanza</libro>
</respuesta>
```

Si analizamos esta expresión XQuery tendremos que:

- La clausula **for** selecciona todos los elementos *libro* del archivo "libros.xml", y almacena la lista resultante en la variable *$x*.

- La clausula **let** obtiene desde el archivo "autores.xml", el *codigo* del autor cuyo elemento *nombre* es igual a 'Pablo' y lo guarda en la variable *$y*.
- La clausula **where** filtra la lista de libros de la variable *$x*, a libros cuyo atributo *autor* es igual a la variable *$y*.
- La clausula **Order by** nos permite ordenar los elementos de la lista filtrada deacuerdo al valor del elemento *precio*.
- La clausura **result** nos permite ordenar y formatear la salida.

autores.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<autores>
  <autor codigo="A001">
    <nombre>Pablo</nombre>
    <apellidos>Neruda</apellidos>
    <nacionalidad>Chile</nacionalidad>
  </autor>
  <autor codigo="A002">
    <nombre>Mario</nombre>
    <apellidos>Vargas Llosa</apellidos>
    <nacionalidad>Peru</nacionalidad>
  </autor>
</autores>
```

libros.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<libreria>
  <libro codigo="3474" autor="A001" stock="5">
    <titulo>El habitante y su esperanza</titulo>
    <precio>6000</precio>
  </libro>
  <libro codigo="2651" autor="A002" stock="3">
    <titulo>Los Cachorros</titulo>
    <precio>5000</precio>
  </libro>
  <libro codigo="44261" autor="A002" stock="6">
    <titulo>La Casa Verde</titulo>
    <precio>4000</precio>
  </libro>
  <libro codigo="14351" autor="A001" stock="3">
    <titulo>Viajes</titulo>
    <precio>5000</precio>
  </libro>
</libreria>
```

Capítulo 6

XSLT

XSLT [14] es un lenguaje que permite transformar un documento XML en otro documento XML o en cualquier otro tipo de documento (Ej. HTML). Usa plantillas (templates) y expresiones XPath para navegar a través de elementos y atributos en un documento origen XML, y definir que partes del documento serán escritos en un documento destino.

XSLT permite: añadir o eliminar elementos y atributos; re/ordenar elementos; llevar a cabo verificaciones para determinar que elementos procesar; y otras actividades relacionadas a la transformación de documentos. Un proceso de transformación de un documento XML puede traducirse en transformar un árbol XML fuente en un árbol XML destino.

A continuación describiremos las instrucciones XSLT básicas que permiten la transformación de documentos XML. Recordar que en XSLT podemos utilizar las funciones descritas en la sección que describe XPath. La especificación de XSLT [15] presenta información mas detallada.

6.1. Transformaciones XSLT

Documentos XSLT: Los documentos XSLT contienen reglas de transformación que podrán ser aplicadas a un documento instancia XML. Estos documentos son almacenados en archivos con extensión **.xsl*.

- El elemento `<xsl:stylesheet>`**:** El elemento raíz que permite declarar que un documento contiene la definición de un XSLT es `xsl:stylesheet`. Este elemento contiene todas las definiciones de templates y reglas de transformación que serán aplicadas a un documento origen XML. Por ejemplo:

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

Como alternativa al elemento `xsl:stylesheet` se puede usar `xsl:transform`. El espacio de nombres de XSLT es ‘‘`http://www.w3.org/1999/XSL/Transform`" y usaremos el prefijo “xsl”para referenciarlo.

6.2. Templates

6.2.1. El elemento <xsl:template>

XSLT se basa en la definición de *templates*, donde cada template contiene reglas que serán aplicadas cuando un específico nodo coincide con una expresión XPath. El elemento `xsl:template` es usado para declarar templates y contiene un atributo `match` que permite asociar el template con un elemento XML. Un template permite especificar una posición o contexto de navegación dentro del árbol que representa el documento XML de entrada.

El valor del atributo `match` consiste en una expresión XPath que hace referencia a algún nodo del documento instancia, a cuyo contenido se le podrá aplicar las reglas definidas dentro del template. Por ejemplo, el siguiente template se ubica en el nodo *libreria* que es el elemento raíz del documento, y define instrucciones HTML que serán escritas directamente en el archivo de salida.

```
<xsl:template match="/libreria">
  <html>
    <body>
      ...
    </body>
  </html>
</xsl:template>
```

6.2.2. El elemento <xsl:value-of>

Este elemento es usado para extraer el valor de un nodo XML. Este valor permite realizar alguna validación o puede ser usado para escribirlo en el archivo de salida.

El elemento `xsl:value-of` contiene el atributo `select`, el cuál debe contener una expresión XPath que hace referencia a algún nodo tomando como contexto base el template en el cual está la declaración `xsl:value-of`.

```
<xsl:template match="libro">
 <tr>
 <td><xsl:value-of select="@codigo"/></td>
 <td><xsl:value-of select="titulo"/></td>
 <td><xsl:value-of select="autor"/></td>
 <td><xsl:value-of select="precio"/></td>
 </tr>
</xsl:template>
```

6.2.3. El elemento <xsl:apply-templates>:

Esta instrucción permite aplicar un template al nodo actual o a los nodos hijo del elemento actual. Si la instrucción no contiene atributos, es decir se declara como
<xsl:apply-template />
el procesador buscará templates que coincidan con los nombres de nodos hijo del nodo contexto.

El elemento `xsl:apply-templates` contiene un atributo `select` que permite especificar una expresión XPath para seleccionar los nodos hijos a los cuales se les aplicará un template. Este atributo puede ser usado para especificar el orden en el cual los hijos serán evaluados.

```
<xsl:template match="/libreria">
  <html>
    <body>
     <h1>Lista de Libros</h1>
     ...
     <xsl:apply-templates select="libro"/>
     ...
     <h1>Lista de Revistas</h1>
     ...
     <xsl:apply-templates select="revista"/>
     ...
    </body>
  </html>
</xsl:template>
```

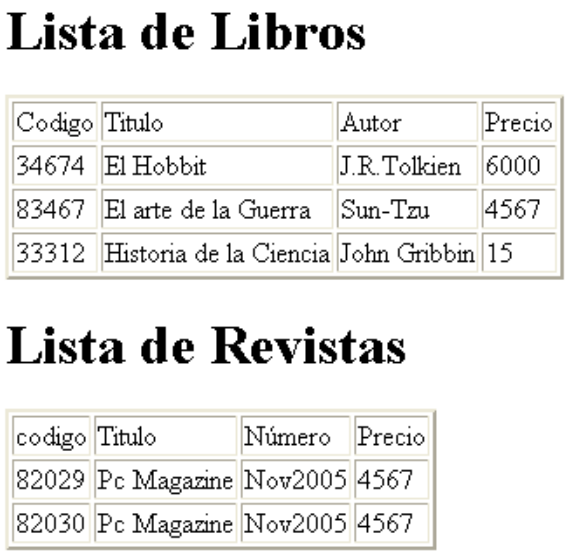
Para ilustrar las declaraciones anteriores consideremos el documento instancia *instancia21.xml*. La declaración `xml-stylesheet`, nos permite vincular el documento XML con un documento que contiene transformaciones XSLT.

En nuestro ejemplo tenemos el archivo *xslt01.xsl*, el cual nos permite generar una página HTML con 2 tablas, una para libros y otra para revistas. En este XSLT podemos observar que:

- Se tienen declarados tres templates que permiten referenciar elementos existentes en el XML origen.
- El template con `match=/libreria"` define como su contexto el elemento libreria. En otras palabras se ubica en el nodo *libreria*, dentro del árbol XML que representa el XML origen.
- Como contenido del template hemos insertado código HTML que será escrito directamente en el archivo de salida. En caso de ser código XML debe estar bien formado.

- Hacemos uso de la instrucción `xsl:apply-templates` para invocar la aplicación de otro template indicado por el atributo `select`. Esto puede resultar en una serie de invocaciones (con posible recursividad) al estilo de un lenguaje de programación funcional.
- Dentro de los templates con `match="libro"` y `match=‘‘revista"` insertamos nuevamente código HTML junto con expresiones `xsl:value-of` para obtener datos desde el documento XML origen.

El resultado del XSLT anterior es la pagina HTML *output01.html* presentada en la figura 6.1.



Lista de Libros

Codigo	Titulo	Autor	Precio
34674	El Hobbit	J.R.Tolkien	6000
83467	El arte de la Guerra	Sun-Tzu	4567
33312	Historia de la Ciencia	John Gribbin	15

Lista de Revistas

codigo	Titulo	Número	Precio
82029	Pc Magazine	Nov2005	4567
82030	Pc Magazine	Nov2005	4567

Figura 6.1: Ejemplo de transfomación usando XSLT

instancia21.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xslt01.xsl"?>
<libreria>
  <libro codigo="34674" stock="5">
    <titulo>El Hobbit</titulo>
    <autor>J.R.Tolkien</autor>
    <editorial>Minotauro</editorial>
    <precio moneda="pesos">6000</precio>
  </libro>
  <revista codigo="82029" stock="4">
    <titulo>Pc Magazine</titulo>
    <numero>Nov2005</numero>
    <precio moneda="pesos">4567</precio>
  </revista>
  <libro codigo="83467" stock="2">
    <titulo>El arte de la Guerra</titulo>
    <autor>Sun-Tzu</autor>
    <editorial>Planeta</editorial>
    <precio moneda="dolar">4567</precio>
  </libro>
  <revista codigo="82030" stock="7">
    <titulo>Pc Magazine</titulo>
    <numero>Nov2005</numero>
    <precio moneda="pesos">4567</precio>
  </revista>
  <libro codigo="33312" stock="3">
    <titulo>Historia de la Ciencia</titulo>
    <autor>John Gribbin</autor>
    <editorial>Critica</editorial>
    <precio moneda="dolar">15</precio>
  </libro>
</libreria>
```

xslt01.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/libreria">
<html>
 <body>
  <h1>Lista de Libros</h1>
  <table border="2">
   <tr>
    <td>Codigo</td>
    <td>Titulo</td>
    <td>Autor</td>
    <td>Precio</td>
   </tr>
   <xsl:apply-templates select="libro"/>
  </table>
  <h1>Lista de Revistas</h1>
  <table border="2">
   <tr>
    <td>codigo</td>
    <td>Titulo</td>
    <td>Número</td>
    <td>Precio</td>
   </tr>
   <xsl:apply-templates select="revista"/>
  </table>
  </body>
</html>
</xsl:template>
<xsl:template match="libro">
 <tr>
 <td><xsl:value-of select="@codigo"/></td>
 <td><xsl:value-of select="titulo"/></td>
 <td><xsl:value-of select="autor"/></td>
 <td><xsl:value-of select="precio"/></td>
 </tr>
</xsl:template>
<xsl:template match="revista">
 <tr>
 <td><xsl:value-of select="@codigo"/></td>
 <td><xsl:value-of select="titulo"/></td>
 <td><xsl:value-of select="numero"/></td>
 <td><xsl:value-of select="precio"/></td>
 </tr>
</xsl:template>
</xsl:stylesheet>
```

output01.html

```
<html>
   <body>
      <h1>Lista de Libros</h1>
      <table border="2">
         <tr>
            <td>Codigo</td><td>Titulo</td><td>Autor</td><td>Precio</td>
         </tr>
         <tr>
            <td>34674</td>
            <td>El Hobbit</td>
            <td>J.R.Tolkien</td>
            <td>5000</td>
         </tr>
         <tr>
            <td>83467</td>
            <td>El arte de la Guerra</td>
            <td>Sun-Tzu</td>
            <td>5</td>
         </tr>
         <tr>
            <td>33312</td>
            <td>Historia de la Ciencia</td>
            <td>John Gribbin</td>
            <td>25</td>
         </tr>
      </table>
      <h1>Lista de Revistas</h1>
      <table border="2">
         <tr>
            <td>codigo</td><td>Titulo</td><td>Numero</td><td>Precio</td>
         </tr>
         <tr>
            <td>82029</td>
            <td>Pc Magazine</td>
            <td>Oct2005</td>
            <td>4500</td>
         </tr>
         <tr>
            <td>82030</td>
            <td>Pc Magazine</td>
            <td>Nov2005</td>
            <td>5000</td>
         </tr>
      </table>
   </body>
</html>
```

6.2.4. El elemento <xsl:for-each>

Esta instrucción permite iterar sobre un conjunto de nodos del documento origen. Este conjunto de nodos es seleccionado usando una expresión XPath definida en el atributo `select` del elemento `xsl:for-each`.

El siguiente ejemplo (*xslt02.xsl*) itera sobre el conjunto de elementos *libro* que son parte del elemento *libreria*, y va construyendo los elementos de una tabla HTML.

```
<xsl:template match="/">
 ...
 <xsl:for-each select="libreria/libro">
   <tr>
   <td><xsl:value-of select="@codigo"/></td>
   <td><xsl:value-of select="titulo"/></td>
   <td><xsl:value-of select="autor"/></td>
   <td><xsl:value-of select="precio"/></td>
   </tr>
 </xsl:for-each>
 ...
</xsl:template>
```

6.2.5. El elemento <xsl:sort>:

Este elemento puede incluirse dentro del elemento `xsl:for-each` para ordenar la salida. `xsl:sort` contiene un atributo `select`, que permite especificar el nodo que será usado para ordenar.

En el siguiente ejemplo (*xslt03.xsl*), ordenamos la lista de libros en base al atributo *codigo*.

```
<xsl:template match="/">
 ...
 <xsl:for-each select="libreria/libro">
   <xsl:sort select="@codigo"/>
   <tr>
   <td><xsl:value-of select="@codigo"/></td>
   <td><xsl:value-of select="titulo"/></td>
   <td><xsl:value-of select="autor"/></td>
   <td><xsl:value-of select="precio"/></td>
   </tr>
 </xsl:for-each>
</xsl:template>
```

6.2.6. El elemento <xsl:if>:

Esta instrucción define una expresión condicional sobre el contenido del documento origen XML. *xsl:if* permite aplicar una serie de transformaciones si es que su atributo `test` es evaluado como válido. El valor del atributo `test` es una expresión XPath.

En el siguiente ejemplo (*xslt04.xsl*), establecemos un filtro sobre la lista de libros, seleccionando solo aquellos libros cuyo elemento *precio* tenga el atributo *moneda* igual a "pesos".

```
<xsl:for-each select="libreria/libro">
  <xsl:if test="precio[@moneda='pesos']">
    <tr>
    <td><xsl:value-of select="@codigo"/></td>
    <td><xsl:value-of select="titulo"/></td>
    <td><xsl:value-of select="autor"/></td>
    <td><xsl:value-of select="precio"/></td>
    </tr>
  </xsl:if>
</xsl:for-each>
```

6.2.7. El elemento <xsl:choose>:

Esta instrucción es usada junto con `xsl:when` y `xsl:otherwise` para definir una expresión condicional de múltiples opciones.

En el siguiente ejemplo (*xslt05.xsl*), ponemos una condición sobre el atributo moneda.

```
<xsl:for-each select="libreria/libro">
 <tr>
  <td><xsl:value-of select="@codigo"/></td>
  <td><xsl:value-of select="titulo"/></td>
  <td><xsl:value-of select="autor"/></td>
  <xsl:choose>
   <xsl:when test="precio[@moneda='pesos']">
    <td>$ <xsl:value-of select="precio"/></td>
   </xsl:when>
   <xsl:when test="precio[@moneda='dolar']">
    <td>US$ <xsl:value-of select="precio"/></td>
   </xsl:when>
   <xsl:otherwise>
    <td><xsl:value-of select="precio"/></td>
   </xsl:otherwise>
  </xsl:choose>
 </tr>
</xsl:for-each>
```

Capítulo 7

XSL-FO

XSL-FO (Extensible Stylesheet Language Formatting Objects) [16] define un vocabulario para especificar reglas de formateo para documentos XML. En este capítulo examinaremos en que consiste XSL-FO y su relación con XSLT.

7.1. Documentos XSL-FO

Los documentos XSL-FO contienen reglas que especifican un formato de presentación. Estos contienen información sobre el diseño y el contenido del documento de salida. Los documentos XSL-FO son almacenados en archivos con extensión **.fo*.

Los documentos XSL-FO siguen la siguiente estructura (o jerarquía de elementos) de la figura 7.1. Un ejemplo de estructura básica se muestra a continuación:

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

 <fo:layout-master-set>
   <fo:simple-page-master master-name="PaginaEjemplo">
     <!-- contenido del template de pagina -->
   </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="PaginaEjemplo">
   <!-- contenido de la pagina -->
 </fo:page-sequence>

</fo:root>
```

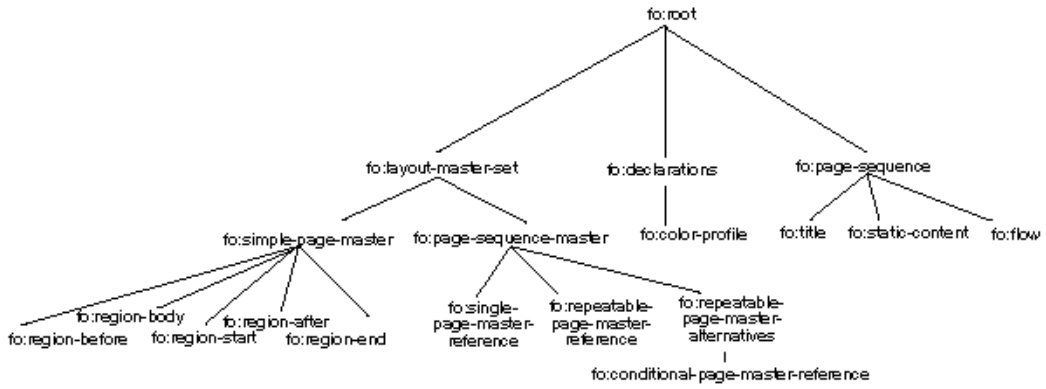


Figura 7.1: Estructura de un documento XSL-FO

Explicación de la estructura anterior:

- Los documentos XSL-FO son documentos XML, por lo tanto deben incluir la declaración XML:

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

- El elemento <`fo:root`> es el elemento raíz de un documento XSL-FO. Este elemento declara el espacio de nombres de XSL-FO:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <!-- cuerpo del documento XSL-FO -->
</fo:root>
```

- El elemento <`fo:layout-master-set`> contiene uno o mas templates para páginas:

```
<fo:layout-master-set>
  <!-- templates para paginas -->
</fo:layout-master-set>
```

- Cada elemento <`fo:simple-page-master`> contiene el template para una página simple. Cada template debe tener un nombre único (`master-name`):

```
<fo:simple-page-master master-name="PaginaEjemplo">
  <!-- uno o mas templates -->
</fo:simple-page-master>
```

- Uno o mas elementos <`fo:page-sequence`> describen el contenido de una página El atributo `master-reference` hace referencia al template `simple-page-master` con el mismo nombre:

```
<fo:page-sequence master-reference="PaginaEjemplo">
  <!-- contenido de la pagina -->
</fo:page-sequence>
```

7.2. Areas XSL-FO

XSL-FO define un conjunto de áreas rectangulares (cajas) para mostrar la salida. Toda salida (texto, figuras, etc) será formateada en estas cajas para luego ser mostradas o impresas en algún medio elegido.

Las áreas definidas son las siguientes:

- *Páginas:* La salida de XSL-FO es formateada en muchas páginas separadas.
- *Regiones:* Cada página contiene un conjunto de regiones (ver figura 7.2).
  - *region-body:* el cuerpo de la página
  - *region-before:* la cabecera de página
  - *region-after:* el pié de página
  - *region-start:* el recuadro o barra lateral de la izquierda
  - *region-end:* el recuadro o barra lateral de la derecha

  Nota: Las regiones region-before, region-after, region-start y region-end son parte de region-body.
- *Bloques:* Cada región contiene pequeños elementos denominados bloques, como párrafos, tablas y listas. Un bloque puede contener otros bloques pero con más frecuencia contiene Líneas.
- *Linea:* Un área de linea define lineas de texto dentro de bloques.
- *Inline:* Un área de línea contiene áreas Inline, las cuales definen texto dentro de líneas (viñetas, caracteres simples, gráficos).

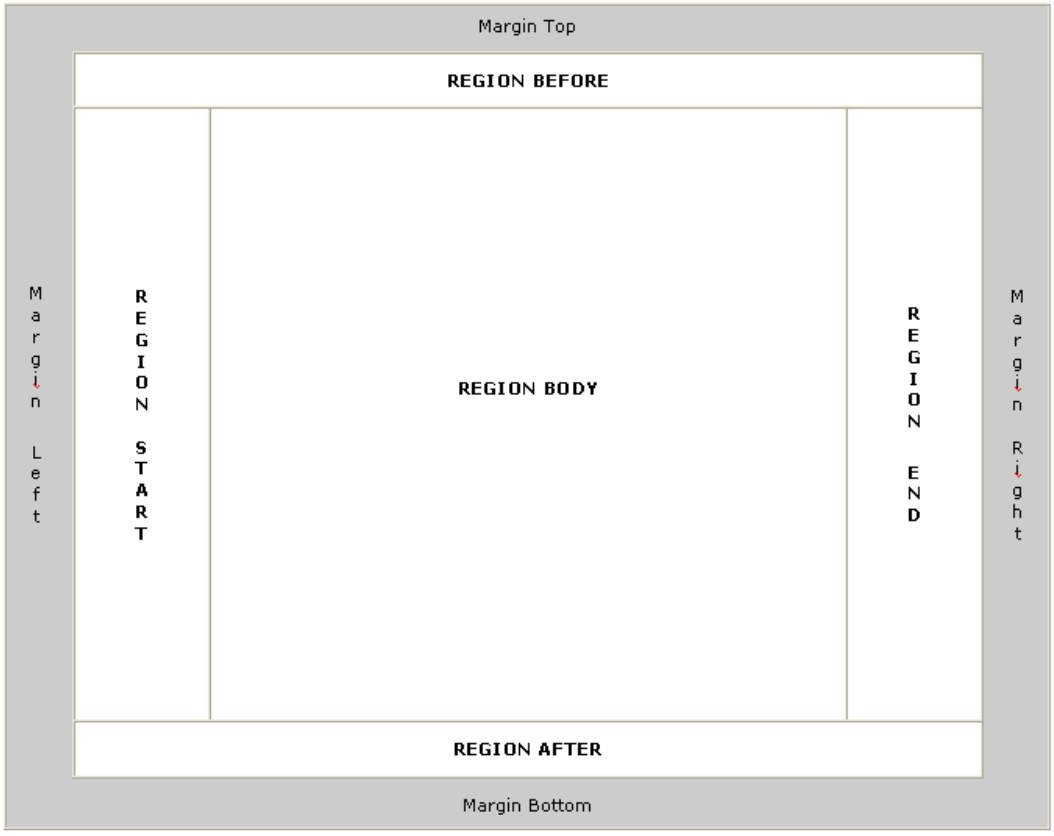


Figura 7.2: Regiones de una página

7.3. Definiendo el formato de salida

Elemento <fo:layout-master-set>**:** Este elemento contiene un conjunto de declaraciones de plantillas para páginas que podrán ser usadas en el documento XSL-FO. Este elemento puede contener uno o mas elementos <fo:simple-page-master>.

Elemento <fo:simple-page-master>**:** Este elemento permite definir plantillas (templates) o diseños (layouts) para páginas llamados "Page Masters"(patrónes de página). Cada plantilla debe tener un nombre único.

El elemento `fo:simple-page-master` permite los siguientes atributos:

- `page-width`: define el ancho de la página.
- `page-height`: define el alto de la página.
- `margin-top`: define el márgen superior de la página.
- `margin-bottom`: define el márgen inferior de la página.
- `margin-left`: define el márgen izquierdo de la página.
- `margin-right`: define el márgen derecho de la página.
- `margin`: define el valor de los cuatro márgenes.

Adicionalmente podemos definir características individuales de las distintas regiones de la página. Por ejemplo:

```
<fo:simple-page-master master-name="plantilla"
                       page-width="21.6cm"
                       page-height="27.9cm"
                       margin-top="2cm"
                       margin-left="3cm">
  <fo:region-body background-color="grey"/>     <--
</fo:simple-page-master>
```

7.4. Definiendo los datos de salida

Normalmente la salida es definida en un conjunto anidado de elementos como se muestra en el siguiente esquema:

```
<fo:page-sequence master-reference="PaginaEjemplo">
 <fo:flow flow-name="xsl-region-body">
   <fo:block>Ejemplo de XSL-FO</fo:block>
 </fo:flow>
</fo:page-sequence>
```

A continuación describiremos los distintos elementos que nos permiten definir los datos de salida.

Elemento <fo:page-sequence>: Este elemento es usado para especificar como crear una secuencia de páginas dentro de un documento. El atributo `master-reference` permite hacer referencia a un elemento `master-page` que define el diseño de página que se utilizará para crear la presentación final. Cada elemento `page-sequence` contiene un elemento `fo:flow` que define la salida. Cada página de salida es impresa o mostrada, de acuerdo al orden secuencial en que fueron definidas.

Elemento <fo:flow>: Los datos que se desean formatear con XSL-FO, son insertados dentro del elemento `fo:flow`. Este contiene todos los elementos que serán impresos en la página.

El atributo `flow-name` permite indicar donde debe ir el contenido del elemento `fo:flow`. Los valores posibles son: xsl-region-body, xsl-region-before, xsl-region-after, xsl-region-start, xsl-region-end.

Elemento <fo:block>: Este elemento representa un bloque, cuya secuencia de salida se traduce a un conjunto de cajas rectangulares. La estructura de un bloque se muestra en la figura 7.3.

El elemento `fo:block` presenta los siguiente atributos relacionados al formato de su contenido:

- Formato de Fondo: background-color, background-image, background-repeat, background-attachment.
- Tipo de letra: font-family, font-weight, font-style, font-size, font-variant.
- Formato del texto: text-align, text-align-last, text-indent, text-start-indent, end-indent, wrap-option, break-before, break-after, reference-orientation.

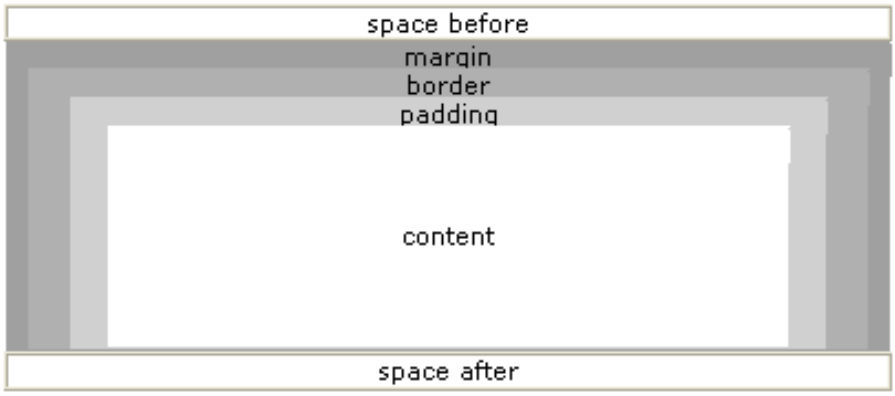


Figura 7.3: Cajas de un bloque

Existen además una serie de atributos que permiten definir el formato de los elementos de un bloque:

- `Margin`: margin, margin-top, margin-bottom, margin-left, margin-right.
- `Border`: tenemos atributos para definir el estilo el color y el ancho del borde.
  - Estilo: border-style, border-before-style, border-after-style, border-start-style, border-end-style, border-top-style, border-bottom-style, border-left-style, border-right-style.
  - Color: border-color, border-before-color, border-after-color, border-start-color, border-end-color, border-top-color, border-bottom-color, border-left-color, border-right-color.
  - Ancho: border-width, border-before-width, border-after-width, border-start-width, border-end-width, border-top-width, border-bottom-width, border-left-width, border-right-width.
- `Padding`: padding, padding-before, padding-after, padding-start, padding-end, padding-top, padding-bottom, padding-left, padding-right.

El siguiente ejemplo de documento XSL-FO (*xsl-fo-01.fo*), ilustra la declaración de algunas opciones de formato. El resultado (*output-fo-01.pdf*) de este documento XSL-FO se muestra en la figura 7.4.

xsl-fo-01.fo

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <!-- formato de la pagina -->
 <fo:layout-master-set>
  <fo:simple-page-master master-name="paginaEjemplo"
                         page-width="21.6cm"
                         page-height="27.9cm"
                         margin-top="1cm"
                         margin-left="2cm">
   <fo:region-body margin="0.5cm"/>
  </fo:simple-page-master>
 </fo:layout-master-set>
 <!-- contenido de la pagina -->
 <fo:page-sequence master-reference="paginaEjemplo">
  <fo:flow flow-name="xsl-region-body">
   <fo:block font-size="5" font-weight="bold" space-after="1cm">
     Ejemplo de XSL-FO
   </fo:block>
   <fo:block font-style="italic" space-after="1cm">
     Este es un ejemplo de parrafo
   </fo:block>
   <fo:block border-style="solid" border="2">
     Parrafo con borde
   </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

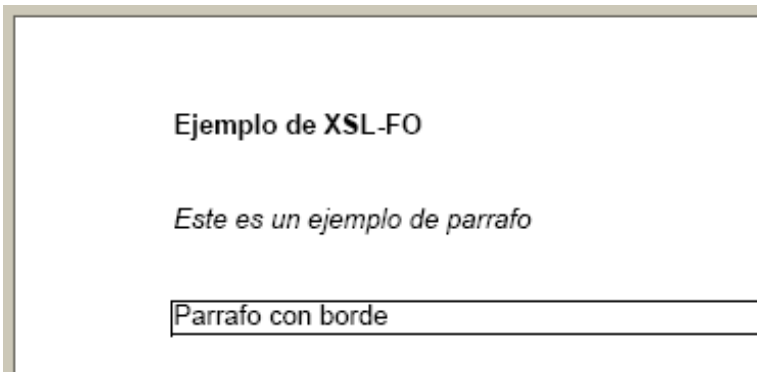


Figura 7.4: Ejemplo de documento XSL-FO

7.5. Estructuras adicionales

7.5.1. Listas

XSL-FO define cuatro elementos para crear listas:

- `fo:list-block`: elemento que contiene la lista.
- `fo:list-item`: elemento que contiene cada item de la lista
- `fo:list-item-label`: define la etiqueta para el elemento `list-item`. Tipicamente es un objeto `fo:block` conteniendo un número, un caracter, etc.
- `fo:list-item-body`: representa el contenido o cuerpo del `list-item`. Típicamente es un conjunto de objectos `fo:block`.

El siguiente ejemplo (*xsl-fo-02.fo*), muestra el uso de listas. El resultado (*output-fo-02.pdf*) de este documento XSL-FO se presenta en la figura 7.5

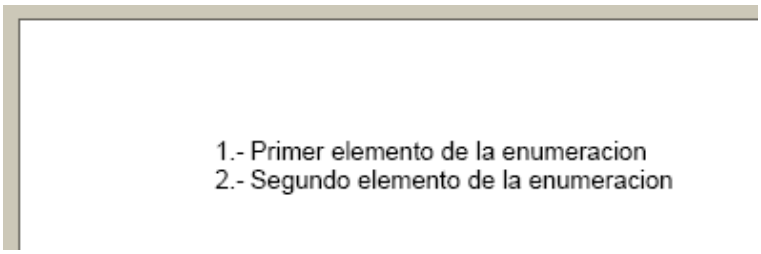


Figura 7.5: Ejemplo de Lista

xsl-fo-02.fo

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="plantilla"
                           page-width="21.6cm"
                           page-height="27.9cm"
                           margin-top="2cm"
                           margin-left="3cm">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="plantilla">
    <fo:flow flow-name="xsl-region-body">
      <fo:list-block>
        <fo:list-item>
         <fo:list-item-label start-indent="3mm">
          <fo:block>1.-</fo:block>
         </fo:list-item-label>
         <fo:list-item-body start-indent="9mm">
          <fo:block>Primer elemento de la enumeracion</fo:block>
         </fo:list-item-body>
        </fo:list-item>
        <fo:list-item>
         <fo:list-item-label start-indent="3mm">
          <fo:block>2.-</fo:block>
         </fo:list-item-label>
         <fo:list-item-body start-indent="9mm">
          <fo:block>Segundo elemento de la enumeracion</fo:block>
         </fo:list-item-body>
        </fo:list-item>
      </fo:list-block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
```

7.5.2. Tablas

XSL-FO define los siguientes elementos para crear tablas: `fo:table`, `fo:table-column`, `fo:table-header`, `fo:table-footer`, `fo:table-body`, `fo:table-row`, `fo:table-cell`.

El elemento `fo:table` contiene elementos `fo:table-column`(opcional), un elemento `fo:table-header`(opcional), un elemento `fo:table-body`, un elemento `fo:table-footer`(opcional). Cada uno de estos elementos tiene uno o más elementos `fo:table-row`, con uno o más elementos `fo:table-cell`.

El siguiente ejemplo (*xsl-fo-03.fo*) muestra la declaración de tablas en XSL-FO. El resultado (*output-fo-03.pdf*) se presenta en la figura 7.6

xsl-fo-03.fo

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="plantilla"
                           page-width="21.6cm" page-height="27.9cm"
                           margin-top="2cm"    margin-left="3cm">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="plantilla">
   <fo:flow flow-name="xsl-region-body">
     <fo:table width="12cm">
      <fo:table-column column-width="2cm"/>
      <fo:table-column column-width="5cm"/>
      <fo:table-column column-width="5cm"/>
      <fo:table-header>
       <fo:table-row>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Código</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Titulo</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Autor</fo:block>
        </fo:table-cell>
       </fo:table-row>
      </fo:table-header>
      <fo:table-body>
       <fo:table-row>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>34674</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>El Hobbit</fo:block>
```

```
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>J.R.Tolkien</fo:block>
        </fo:table-cell>
      </fo:table-row>
       <fo:table-row>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>83467</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>El arte de la Guerra</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>Sun-Tzu</fo:block>
        </fo:table-cell>
      </fo:table-row>
      <fo:table-row>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>33312</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>Historia de la Ciencia</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block>John Gribbin</fo:block>
        </fo:table-cell>
      </fo:table-row>
     </fo:table-body>
    </fo:table>
   </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Código	Titulo	Autor
34674	El Hobbit	J.R.Tolkien
83467	El arte de la Guerra	Sun-Tzu
33312	Historia de la Ciencia	John Gribbin

Figura 7.6: Ejemplo de Tabla

7.6. XSL-FO y XSLT

Como vimos en los ejemplos anteriores, el contenido de un documento XSL-FO es una mezcla de instrucciones de formato y los datos, esto hace que un documento XSL-FO no sea fácil de construir directamente, debido a la cantidad de declaraciones que este requiere. En este contexto recordamos el hecho que XSLT permite generar documentos XML a medida y por lo tanto podemos aplicar esta idea al generar documentos XSL-FO desde nuestros documentos instancia XML usando como intermediario una transformación XSLT.

El proceso descrito anteriormente se ilustra en la figura 7.7.

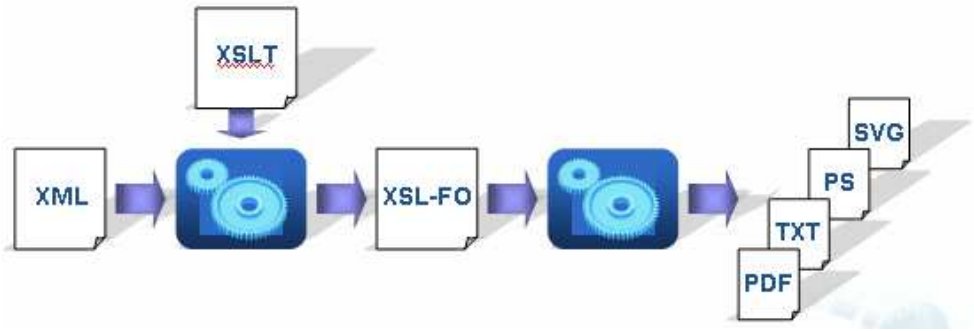


Figura 7.7: Trabajando con XSLT y XSL-FO

El siguiente ejemplo nos muestra un documento XSLT (*xslt07.xsl*) que al ser aplicado sobre un documento XML (*instancia22.xml*) el cual contiene un conjunto de elementos *libro*, producirá un documento XSL-FO (*output07.fo*) muy similar al presentado en el ejemplo anterior. El documento XSL-FO resultante de este proceso, permitirá generar distintos formatos de presentación para el documento XML instancia; por ejemplo un documento en formato PDF (*output-fo-07.pdf*) o en formato PS (*output-fo-07.ps*).

xslt07.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:output method="xml" version="1.0" indent="yes"/>

<xsl:template match="/">
 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="plantilla"
                           page-width="21.6cm"
                           page-height="27.9cm"
                           margin-top="2cm"
                           margin-left="3cm">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="plantilla">
   <fo:flow flow-name="xsl-region-body">
     <fo:table width="12cm">
      <fo:table-column column-width="2cm"/>
      <fo:table-column column-width="5cm"/>
      <fo:table-column column-width="5cm"/>
      <fo:table-header>
       <fo:table-row>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Código</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Titulo</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid" border-width="1pt">
         <fo:block font-weight="bold">Autor</fo:block>
        </fo:table-cell>
       </fo:table-row>
      </fo:table-header>
      <fo:table-body>

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

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

```
<xsl:template match="libro">
 <fo:table-row>
  <fo:table-cell border-style="solid" border-width="1pt">
   <fo:block>
    <xsl:value-of select="@codigo"/>
   </fo:block>
  </fo:table-cell>
  <fo:table-cell border-style="solid" border-width="1pt">
   <fo:block>
    <xsl:value-of select="titulo"/>
   </fo:block>
  </fo:table-cell>
  <fo:table-cell border-style="solid" border-width="1pt">
   <fo:block>
    <xsl:value-of select="autor"/>
   </fo:block>
  </fo:table-cell>
 </fo:table-row>
</xsl:template>

</xsl:stylesheet>
```

Capítulo 8

Documentación Electrónica y XML

8.1. XML

El e**X**tensible **M**arkup **L**anguage (XML) [1] por sus siglas en ingles o Lenguaje de Marcado Extensible es un formato de texto simple y flexible propuesto por la W3C, que fue originalmente diseñado para cumplir los desafíos de publicaciones electrónicas a gran escala y que actualmente esta siendo usado como un formato estándar para la documentación electrónica e intercambio de datos en la Web.

Motivado por los requerimientos de modelamiento en el área de documentación electrónica, XML se ha convertido en una familia de tecnologías que entre muchas cosas permiten trabajar con los conceptos de contenido, estructura y presentación discutidos en el capítulo 1. Las tecnologías encargadas de cubrir estos tres aspectos son:

- **XML Syntax** [1]: El *contenido* de un documento, puede modelado usando la sintaxis definida por XML, dando lugar al concepto de *documento instancia XML*.
- **XML Schema** [3]: un *esquema XML* permite definir la *estructura* de un tipo de documento.
- **XSL** [14]: Las *hojas de estilo XSL* permiten definir y aplicar formato o *presentación* a documentos instancia XML.

8.2. Documentación Electrónica

El proceso de documentar la información en un formato electrónico que facilite su almacenamiento, consulta, presentación e intercambio se denomina Documentación Electrónica. Actualmente el formato estándar para la documentación electrónica es XML.

El proceso de documentación electrónica puede ser divido en:

- Analizar la factibilidad del proceso de documentación Electrónica.
- Análisis de Información (según lo descrito en el capítulo 1).
- Definición de metadatos y creación del diccionario de metadatos.
- Creación de esquemas XML y Hojas de Estilo XSL.
- Creación de documentos instancia XML.
- Desarrollo de aplicaciones para la administración de documentos electrónicos.

8.3. Metadatos Semánticos

Un aspecto importante de considerar al crear documentos electrónicos es agregar metadatos que nos brinden información adicional sobre cada recurso. Dublin Core [17, 18] provee un lista detallada de metadatos estándar que pueden ser usados con este fin.

Metadatos para documentos instancia XML : En los documentos instancia XML puede ser útil definir algunos de los metadatos presentados en el siguiente ejemplo:

```
<?xml version="1.0" ?>
<!-- METADATOS
  1.Identificador:
  2.Titulo:
  3.Autor:
  4.Descripcion:
  5.Esquemas: Lista de esquemas usados para validar el documento
  6.Diccionario: URI del diccionario de metadatos
  7.Hoja de estilo: URI del XSL usado para presentar el documento
--->
<Documento>
 ...
</Documento>
```

Metadatos para esquemas XML : Para el caso específico de esquemas XML, estos metadatos pueden ser incluidos dentro de un elemento <METADATOS>, el cuál formara parte del contenido del elemento `appinfo`, y a su vez parte de un elemento `annotation` definido en el elemento raíz del esquema. Por ejemplo:

```
<?xml version="1.0" ?>
<xsd:schema ...

  <xsd:annotation>
    <xsd:documentation xml:lang = "es">
     ...
    </xsd:documentation>

    <xsd:appinfo>
      <METADATOS>
              <!-- metadatos semanticos -->
              <Identificador>ESXML4536-2</Identificador>
              <Titulo>Esquema para boleta</Titulo>
              <Autor>Juan Rojas</Autor>
              <Descripcion>Este esquema define ...</Descripcion>
              <Creacion>2005-05-24</Creacion>
              <Modificado>2005-06-04</Modificado>
              <Modificado>2005-06-30</Modificado>
              <Estado>Aprobado</Estado>
              <Version>3.0</Version>
               ...
      </METADATOS>
    </xsd:appinfo>

  <xsd:/annotation>
  ...
<xsd:schema>
```

8.4. Decreto 81

En Junio del 2004 el gobierno de Chile promulgó el Decreto Supremo Nro. 81, que establece las características mínimas obligatorias de interoperabilidad que deben cumplir los documentos electrónicos en su generación, envío, recepción, procesamiento y almacenamiento. En este contexto, la tabla 8.1 enumera un conjunto de puntos que pueden ser considerados al evaluar si un documento electrónico cumple con lo especificado en el Decreto 81.

Punto	Descripción	Fuente.
P1	Documento bien formado según codificación XML v.1.1	Art. 8°
P2	Utiliza XML Schemas	Art. 8°
P3	Esquemas publicos y de libre acceso	Art. 8°
P4	Indica los esquemas utilizados	Art. 9°
P5	Indica los diccionarios de datos	Art. 9°
P6	Indica las formas de presentación alternativas	Art. 9°
P7	Utiliza la codificación UTF-8	Art. 9°
P8	Incluye metadatos semánticos	Art. 10°
P9	Utiliza XFORMS	Art. 11°
P10	Soporta firma y cifrado conforme normas técnicas	Art. 12°
P11	La firma electrónica se hace conforme XML Signature	Art. 12°
P12	Los esquemas son consistentes con el diccionario	Art. 13°
P13	Incorpora adecuadamente etiquetas del diccionario	Art. 13°
P14	Tiene un identificador único asociado a su localización	Art. 14°
P15	Incorpora metadatos sobre la vida del documento	Art. 14°
P16	Tiene metadatos para documentar su significado y uso	Art. 14°
P17	Tiene un esquema XML validador debidamente documentado	Art. 15°
P18	Posee una presentacion definida por defecto utilizando XSL	Art. 15°
P19	Dispone al menos de una visualización en XHTML	Art. 15°
P20	Provee un mecanismo para verificar la integridad y autenticidad de una representación impresa	Art. 15°

Cuadro 8.1: Puntos a evaluar

Apéndice A

Figuras y Tablas

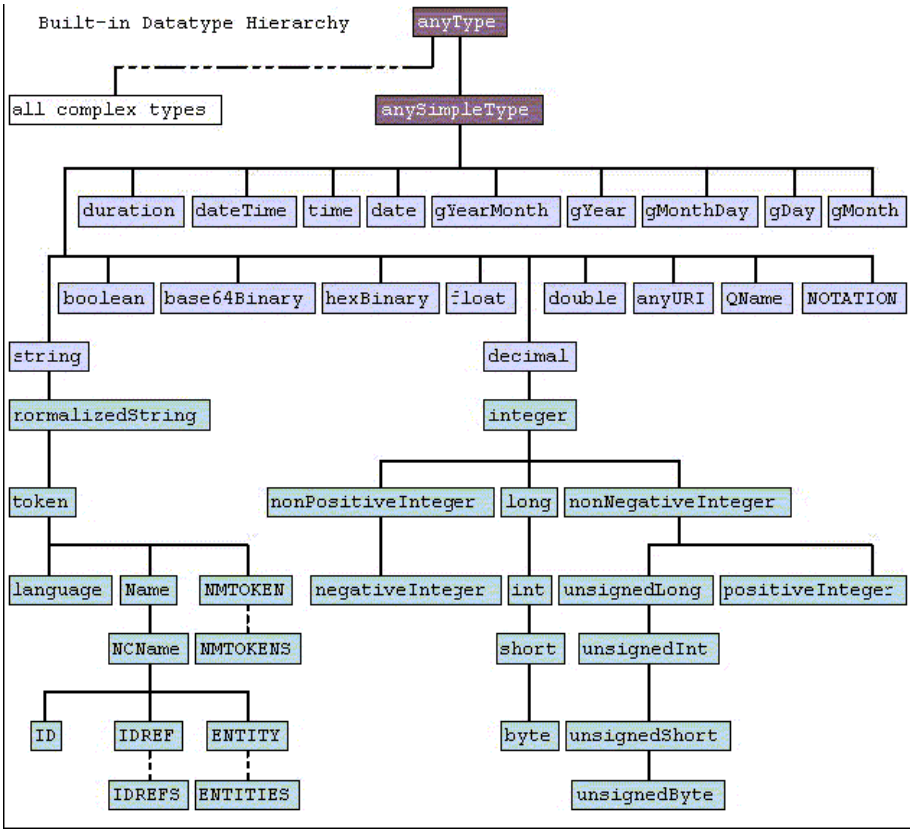


Figura A.1: Jerarquía de tipos en XML Schema

Tipo Primitivo	Formato
string	"Hola mundo"
boolean	true, false, 1, 0
decimal	3.1416
float	12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
double	12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
duration	P1Y2M3DT10H30M12.3S
dateTime	CCYY-MM-DDThh:mm:ss
time	hh:mm:ss.sss
date	CCYY-MM-DD
gYearMonth	CCYY-MM
gYear	CCYY
gMonthDay	$--MM-DD$
gDay	$---DD$
gMonth	$--MM$
hexBinary	una cadena en hexadecimal
base64Binary	una cadena en base64
anyURI	http://www.w3.com
QName	a namespace qualified name
NOTATION	NOTATION desde la especificación XML

Cuadro A.1: Tipos primitivos y su formato ("T"=separador fecha/hora, INF=infinito, NAN=no numérico)

Tipo Primitivo	Formato
normalizedString	cadena sin espacios (tabulaciones, saltos de linea, etc)
token	cadena con espacios
language	algún $xml : language$ valido (EN, FR)
integer	456
long	desde -9223372036854775808 hasta 9223372036854775807
int	desde -2147483648 hasta 2147483647
shot	desde -32768 hasta 32767
byte	desde -127 hasta 128
nonPositiveInteger	desde infinito negativo hasta 0
negativeInteger	desde infinito negativo hasta -1
nonNegativeInteger	desde 0 hasta infinito positivo
positiveInteger	desde 1 hasta infinito positivo
unsignedLong	desde 0 hasta 18446744073709551615
unsignedInt	desde 0 hasta 4294967295
unsignedShort	desde 0 hasta 65535
unsignedByte	desde 0 hasta 255
IDREF, IDREF, IDREFS	algún identificador
NMTOKEN, NMTOKENS,	US
ENTITY, ENTITIES	US UK

Cuadro A.2: Tipos derivados y su formato

Expression	Posibles Valores
\d{2}	2 digitos
$a?b$	b, ab
$a+b$	ab, aab, aaab, ...
$a*b$	b, ab, aab, aaab, ...
$[xyz]b$	xb, yb, zb
$[a-c]x$	ax, bx, cx
$[a-zA-Z-[xy]]*$	cualquier letra excepto x o y
[^0-9]x	Caracter no dígito seguido de x
\Dx	Caracter no dígito seguido de x
$(pa)\{2\}s$	papas
$.abc$	Cualquier caracter (uno solo) seguido de abc
$(a\|b)+x$	ax, bx, aax, bbx, abx, bax,...
$a\{1,3\}x$	ax, aax, aaax
\\	backslash
|	barra vertical
\^	el simbolo ^
\?	signo de interrogración
\+ * \- \.	suma, asterisco, guión, punto
\{ \}	los simbolos { y }
\(\)	abrir - cerrar parentesis
\[\]	abrir - cerrar corchetes
\w	un caracter (alfanumérico o guión)
\s \n \r \t	espacio, salto de linea, retorno de carro, tabulación
\p{L} \p{Lu} \p{Ll}	Alguna: Letra, letra mayuscula, letra minuscula
\p{N} \p{Nd}	un numero, un digito
\p{P}	un simbolo de puntuación
\p{Sc}	Símbolo de moneda

Cuadro A.3: Ejemplos de Expresiones Regulares

Apéndice B

Dublin Core

A continuación se presenta una descripción de los elementos básicos del conjunto de metadatos "Dublin Core". Podemos clasificar estos elementos en tres grupos que indican la clase o el ámbito de la información que se guarda en ellos:

- Elementos relacionados principalmente con el contenido del recurso: *título, claves, descripción, fuente, lengua, relación, cobertura.*
- Elementos relacionados principalmente con el recurso cuando es visto como una propiedad intelectual: *creador, editor, otros colaboradores, derechos.*)
- Elementos relacionados principalmente con la instanciación del recurso: *fecha, tipo del recurso, formato, identificador del recurso.*

Descripción de los elementos:

1. *Título*
   Etiqueta: DC.Title
   Descripción: El nombre dado a un recurso, usualmente por el autor.

2. *Autor o Creador*
   Etiqueta: DC.Creator
   Descripción: La persona u organización responsable de la creación del contenido intelectual del recurso. Por ejemplo, los autores en el caso de documentos escritos, artistas, fotografos e ilustradores en el caso de recursos visuales.

3. *Claves*
   Etiqueta: DC.Subject
   Descripción: Los tópicos del recurso. Típicamente, Subject expresará las claves o frases que describen el título o el contenido del recurso. Se fomentará el uso de vocabularios controlados y de sistemas de clasificación formales.

4. *Descripción*
   Etiqueta: DC.Description Descripción: Una descripción textual del recurso, tal como un resumen en el caso de un documento o una descripción del contenido en el caso de un documento visual.

5. *Editor*
   Etiqueta: DC.Publisher
   Descripción: La entidad responsable de hacer que el recurso se encuentre disponible en la red en su formato actual, por ejemplo la empresa editora, un departamento universitario u otro tipo de organización.

6. *Otros Colaboradores*
   Etiqueta: DC.Contributor
   Descripción: Una persona u organización que haya tenido una contribución intelectual significativa en la creación del recurso pero cuyas contribuciones son secundarias en comparación a las de las personas u organizaciones especificadas en el elemento Creator (por ejemplo, editor, ilustrador y traductor).

7. *Fecha*
   Etiqueta: DC.Date
   Descripción: Una fecha en la que el recurso se puso a disposición del usuario en su forma actual. Esta fecha no ha de confundirse con la que pertenece al elemento Coverage, que sería asociada con el recurso sólo en la medida en que el contenido intelectual está de algún modo relacionado con esa fecha.
   Recomendamos la utilización de uno de los formatos definidos en el documento "Date and Time Formats",
   `http://www.w3.org/TR/NOTE-datetime`
   basado en la norma ISO 8601 que incluye, entre otras, fechas en el formato AAAA y AAAA-MM-DD. De esta forma la fecha 1994-11-05 correspondería al 5 de Noviembre de 1994.

8. *Tipo del Recurso*
   Etiqueta: DC.Type

Descripción: La categoría del recurso, por ejemplo página personal, romance, poema, minuta, diccionario. Para asegurar la interoperabilidad, Type debería ser seleccionado de entre una lista de valores que actualmente se encuentra bajo desarrollo en un grupo de trabajo.
En http://sunsite.berkeley.edu/Metadata/types.html se puede consultar el estado actual de la discusión en torno a este tema.

9. *Formato*
   Etiqueta: DC.Format
   Descripción: El formato de datos de un recurso, usado para identificar el software y posiblemente, el hardware que se necesitaría para mostrar el recurso. Para asegurar la interoperabilidad, los valores de Format deberían ser seleccionados de entre una lista de valores que actualmente se encuentra bajo desarrollo en un grupo de trabajo.

10. *Identificador del Recurso*
    Etiqueta: DC.Identifier
    Descripción: Secuencia de caracteres usados para identificar unívocamente un recurso. Ejemplos para recursos en línea pueden ser URLs y URNs (cuando estén implementados). Para otros recursos pueden ser usados otros formatos de identificadores, como por ejemplo ISBN ("International Standard Book Number Número Internacional Normalizado para Libros)

11. *Fuente*
    Etiqueta: DC.Source
    Descripción: Secuencia de caracteres utilizado para identificar unívocamente un trabajo a partir del cual proviene el recurso actual. Por ejemplo, es posible usar Source con la fecha de 1603 como descripción de una película basada en una obra de Shakespeare, pero es preferible, en ese caso, usar Relation "EstaBasadoEnçon una referencia a un recurso distinto cuya descripción contenga el elemento Date con valor 1603.

12. *Lengua*
    Etiqueta: DC.Language
    Descripción: Lengua/s del contenido intelectual del recurso. Prácticamente el contenido de este campo debería coincidir con los Tags para la identificación de lenguas de la RFC 1766, por ejemplo: *en, es, de, fi, ja y zh* (`http://ds.internic.net/rfc/rfc1766.txt`).

13. *Relación*
    Etiqueta: DC.Relation
    Descripción: Un identificador de un segundo recurso y su relación con el recurso actual. Este elemento permite enlazar los recursos relacionados y las descripciones de los recursos. Por ejemplo: EsVersionDe, EstaBasadoEn, EsParteDe, EsFormatoDe.
    Para asegurar la interoperabilidad, las relaciones deberían ser seleccionadas de una lista de elementos que actualmente se encuentra bajo desarrollo en un grupo de trabajo.

14. *Cobertura*
    Etiqueta: DC.Coverage
    Descripción: La característica de cobertura espacial y/o temporal del contenido intelectual del recurso.
    La cobertura espacial se refiere a una región física (por ejemplo, sector celestial); uso de coordenadas (por ejemplo, longitud y latitud) o nombres de lugares extraidos de una lista controlada.
    La cobertura temporal se refiere al contenido del recurso en vez de a cuando fue creado o puesto accesible ya que este último pertenece al elemento Date. Se usa el mismo formato basado en:
    `http://www.w3.org/TR/NOTE-datetime`.

15. *Derechos*
    Etiqueta: DC.Rights
    Descripción: Una referencia (URL, por ejemplo) para una nota sobre derechos de autor, para un servicio de gestión de derechos o para un servicio que dará información sobre términos y condiciones de acceso a un recurso. Una especificación formal del elemento Rights se encuentra catualmente en discusión y por lo tanto su uso se considera experimental.

Bibliografía

[1] http://www.w3.org/XML/

[2] Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004. "http://www.w3.org/TR/2004/REC-xml-20040204/"'.

[3] http://www.w3.org/XML/Schema

[4] XML Schema Part 0: Primer Second Edition, "W3C Recommendation 28 October 2004. "http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/".

[5] XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004. "http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/".

[6] XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004. "http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/".

[7] Uniform Resource Identifiers (URI): Generic Syntax. "http://www.ietf.org/rfc/rfc2396.txt".

[8] Uniform Resource Locators (URL). "http://www.ietf.org/rfc/rfc1738.txt".

[9] URN Syntax." http://rfc.net/rfc2141.html".

[10] http://www.w3.org/TR/xpath

[11] XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. "http://www.w3.org/TR/1999/REC-xpath-19991116.html".

[12] http://www.w3.org/XML/Query/

[13] XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005. "http://www.w3.org/TR/2005/CR-xquery-20051103/".

[14] http://www.w3.org/Style/XSL/

[15] XSL Transformations (XSLT), Version 1.0. W3C Recommendation 16 November 1999. "http://www.w3.org/TR/1999/REC-xslt-19991116".

[16] Extensible Stylesheet Language (XSL) version 1.0. W3C Recommendation 15 October 2001. "http://www.w3.org/TR/2001/REC-xsl-20011015/".

[17] http://dublincore.org/documents/dces/

[18] http://www.rediris.es/metadata/