Tema 6. Conversión de documentos XML

Tema 6. Conversión de documentos XML

1. Introducción

Los documentos XML permiten almacenar información de una forma estructurada, pero en muchas ocasiones necesitamos utilizar dichos datos para manipularlos o para presentarlos de una forma adecuada. A veces la forma en que se presentan los datos o se manipulan es tan importante como los propios datos en sí.

Las conversiones o transformaciones de documentos XML permiten modificar el contenido de un documento XML a nuestra voluntad. Podemos transformar un documento XML a un formato diferente, pero conservando los datos originales, como por ejemplo, a un archivo texto con sus datos separados por comas o tabuladores, o a una página HTML. También podemos convertir un documento XML en otro documento XML con parte de los datos seleccionados del original o incluso con una nueva estructura.

Para realizar conversiones XML se hace uso de XSL. *(eXtensible Stylesheet Language)* o Lenguaje Extensible de Estilos que permite definir una representación o formato para un documento XML. No se trata de un único lenguaje, sino de toda una familia de recomendaciones del World Wide Web Consortium (http://www.w3.org/Style/XSL/) para expresar hojas de estilo en lenguaje XML. El lenguaje XSL consta de tres partes:

- XSL Transformations (XSLT) es un lenguaje para transformar documentos XML en otro formato (XML, HTML, texto plano…) (http://www.w3.org/TR/xslt y http://www.w3.org/TR/xslt20/)
- XML Path Language (XPath), un lenguaje de consulta genérico usado por XSLT para acceder o referirse a partes de un documento XML. (XPath se usa también en la especificación XLink). (http://www.w3.org/TR/xpath y http://www.w3.org/TR/xpath20/)
- XSL Formatting Objects (XSL-FO) permite especificar el formato visual con el cual se quiere presentar un documento XML. Es usado principalmente para generar documentos PDF (*Portable Document Format*). http://www.w3.org/TR/xsl.

A pesar de la existencia de las CSS u Hojas de Estilo en Cascada que sirven para definir las presentaciones de documentos HTML, XHTML y XML, se ha creado otra forma específica para las presentaciones de estos tipos de documentos XML usando XSL.

Las Hojas de Estilo en Cascada que ya estudiamos son eficaces para describir formatos y presentaciones, pero no sirven para decidir qué tipo de datos deben ser mostrados y cuáles no. Normalmente las CSS se suelen utilizar con documentos XML sólo en los casos en los que debe mostrarse todo su contenido.

En cambio, XSL no sólo permite especificar cómo queremos presentar los datos de un documento XML, sino también filtrar los datos de acuerdo con ciertas condiciones. XSL es más complejo y permite muchas más funciones que las Hojas de Estilo en Cascada, ya que se asimila más a un lenguaje de programación. Además de la presentación visual, XSL permite otras opciones como la ejecución de bucles y sentencias, operaciones lógicas, ordenación de datos, selecciones por comparación, utilización de plantillas, funciones, etc.

2. XSLT.

XSLT es un lenguaje declarativo escrito en XML para transformar documentos XML mediante hojas de estilo XSL. (Un lenguaje declarativo es un lenguaje cuyas instrucciones indican qué es lo que se quiere conseguir, no cómo conseguirlo).

La principal característica de XSLT es su potencia, no sólo permite visualizar documentos, sino también transformarlos y manipularlos. Esta manipulación la realiza un programa especial denominado procesador XSLT.

Con XSLT se expresa la manera de transformar un documento XML en otro documento XML, o en otro tipo de documento que sea reconocido por un explorador, como puede ser HTML o XHTML. Esto tiene una gran ventaja: para unos mismos datos en XML, podemos elegir entre varias transformaciones. Dependiendo de la transformación, el resultado variará, pero lo que no variará son los datos del documento XML de partida.

También, como resultado de la transformación, podemos crear un nuevo documento XML con elementos y/o atributos que no están el original. De igual forma, se pueden eliminar elementos y/o atributos del original, así como cambiar la disposición y ordenación de los elementos.

Resumiendo, las ventajas de XSLT son:

- La salida, como resultado de la transformación, no tiene que por qué ser HTML para visualizarse en un navegador, sino que puede estar en muchos otros formatos.
- Permite la manipulación del documento XML añadiendo elementos, filtrando, borrando, etc.
- Permite acceder a todo el documento XML, no sólo a los contenidos de los elementos.
- Al ser un lenguaje XML es más fácil de utilizar y comprobar su validez.

Por otro lado, tiene los inconvenientes de ser algo más complejo y de mayor capacidad de proceso y memoria, ya que hay que construir en memoria un árbol con el contenido global del documento.

Las transformaciones indicadas mediante XSLT se pueden aplicar de diferentes formas:

- Visualizando directamente en un navegador el documento XML que tiene asociada una hoja de estilos XSL. (El navegador debe tener incorporado un procesador XSLT).
- Ejecutando una aplicación que incorpore un procesador XSLT de forma que se le suministra el archivo XML y la hoja de estilos XSL a utilizar, y genera la salida resultante en otro archivo.
- Realizando las transformaciones en un servidor que incluya un módulo o programa con un procesador XSLT, y enviando a los clientes sólo el resultado de la transformación.

2.1. Funcionamiento de las hojas de estilo XSL.

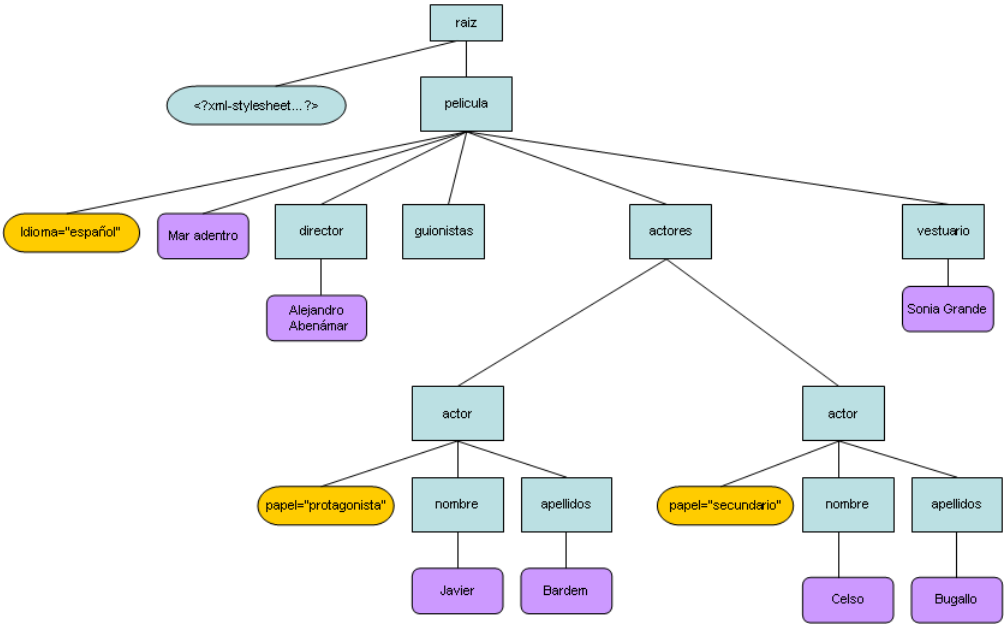
XSLT transforma la estructura de nodos de un documento XML en otra estructura de nodos. Para ello, las hojas de estilos XSL se construyen definiendo reglas. Una **regla** especifica qué es lo que hay que buscar en el árbol del documento origen y qué hay que colocar en el árbol del documento de salida.

Como ya sabemos, un documento XML bien formado tiene una estructura arborescente. Un árbol es una estructura de datos compuesta de nodos conectados entre sí, en donde en el nivel superior existe un nodo llamado raíz que conecta con sus nodos hijos o descendientes, los cuales se pueden conectar con sus propios nodos hijos y, así, sucesivamente.

Para los procesadores XSLT, que son los encargados de realizar las transformaciones, el documento XML es considerado como un árbol en donde los nodos pueden ser cualquier componente del documento. Por lo tanto, los nodos pueden ser: elementos, texto, atributos, espacios de nombres, instrucciones de procesamiento y comentarios, en donde el propio documento es el nodo raíz (no confundir con el elemento raíz). Supongamos el siguiente documento XML:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pelicula.xsl"?>
<pelicula idioma="español">
    Mar adentro
    <director>
        Alejandro Amenábar
    </director>
    <guionistas></guionistas>
    <actores>
    <actor papel="protagonista">
        <nombre>Javier</nombre>
        <apellidos>Bardem</apellidos>
    </actor>
    <actor papel="secundario">
        <nombre>Celso</nombre>
        <apellidos>Bugallo</apellidos>
    </actor>
    </actores>
    <vestuario>
        Sonia Grande
    </vestuario>
</pelicula>
```

El diagrama de la estructura de árbol con la que trabaja un procesador XSLT sería:



Así, un procesador XSLT usa como entrada el árbol de nodos que representa un documento XML y proporciona como salida un nuevo árbol de nodos que representa un nuevo documento XML. Para realizar esta tarea el lenguaje de transformación XSL dispone de operadores para seleccionar nodos del árbol, reordenar los nodos y generar nodos como salida.

2.2. Documentos XSL.

Un documento XSLT está escrito en XML, por lo tanto, existen elementos y atributos XSLT especiales que se usan para crear las hojas de estilo.

Los documentos XSLT se componen de una serie de *reglas* que indican cómo se va a realizar la transformación. Cada *regla* se compone de un *patrón* y de un *contenido*. El *patrón* especifica un modelo en el árbol origen, de forma que la regla se aplicará a todos los nodos del árbol origen que coincidan con el modelo patrón. El *contenido* de la regla permite especificar lo que formará parte del árbol de salida.

Cuando un procesador XSLT transforma un documento XML, recorre la estructura de nodos del documento empezando por la raíz y continúa según el orden indicado en las reglas de transformación. El procesador visita cada nodo del documento, comprueba si el nodo cumple el patrón de cada regla y, si cumple con dicho patrón, generará la salida indicada por el contenido de dicha regla. En el proceso de transformación, XSLT usa XPath para expresar los patrones de los nodos involucrados en la regla.

Generalmente la acción que se realiza mediante la transformación, es enviar a la salida algunas marcas, nuevos datos, o datos obtenidos del documento XML.

2.3. Estructura XSLT.

Como los documentos XSLT tienen formato XML, se usará este formato para especificar las reglas, los patrones y las acciones a realizar.

Todas las hojas de estilo XSL empiezan indicando el tipo de documento mediante el elemento xsl:stylesheet, o bien, xsl:transform. Semánticamente indican lo mismo.

En la etiqueta de inicio es obligatorio indicar la versión mediante el atributo version. La mayoría usa la versión 1.0, aunque la versión 2.0 se irá imponiendo a medida que los procesadores XSLT la vayan implementando.

También es obligatoria la declaración del espacio de nombres oficial del W3C. De esta forma, cualquier otra etiqueta que no lleve el prefijo xsl identificará a elementos que no son XSLT. El prefijo puede ser cualquiera, aunque lo más habitual es usar xsl o xslt.

Por ejemplo:

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

2.3.1. El elemento <xsl:template>. Reglas.

Como hemos comentado, una hoja de estilo XSL consta de una serie de **reglas** que determinan cómo va a ocurrir la transformación. Cada regla se establece en un documento XSLT mediante un elemento de tipo xsl:template. El patrón para dicha regla se especifica mediante el atributo match del elemento; y la acción a realizar para generar la salida es el contenido del elemento xsl:template. Cada regla afecta a uno o varios elementos del documento XML.

Sintácticamente, las reglas tienen tres partes:

- La marca de apertura que contiene un atributo match que describe a qué partes del árbol origen se aplica la regla (qué nodos están afectados). La sintaxis del patrón (valor del atributo match) debe seguir las especificaciones del lenguaje **XPath**. Opcionalmente se puede usar el atributo name que indica un nombre o identificador para la regla. No puede haber dos reglas con el mismo nombre.
- La parte central o contenido describe la salida que se insertará en el árbol de salida.
- La marca de cierre.

```
<xsl:template match ="sección del árbol origen" name="nombre de la regla" >
   Salida que se insertará en el árbol de resultados
</xsl:template>
```

Así, cada elemento xsl:template se asocia con un fragmento del documento XML (que puede ser un nodo o un conjunto de nodos) y genera como salida otro fragmento de XML, HTML, texto plano, etc. de acuerdo a lo que se especifique en el contenido de la regla.

Podemos hacer un resumen paso a paso de cómo se realiza la transformación.

1. El documento XML origen se pasa al procesador XSLT y se convierte en una estructura de árbol.
2. El procesador carga la hoja de estilo XSL indicada por el documento XML.
3. El procesador se posiciona en la raíz del documento XML.
4. El procesador recorre el documento XML origen, nodo por nodo.
5. Si para el nodo que está visitando existe una regla (elemento xsl:template), realizará las acciones que indique la regla, en caso contrario se limitará a colocar en la salida el contenido del elemento del nodo visitado.
6. El procesador sólo aplica una regla a cada nodo, si existen dos reglas para el mismo nodo, sólo aplica la última regla en aparecer.

El orden en que aparecen las reglas en la hoja de transformaciones no es significativo. El orden lo marca el recorrido por parte del procesador del árbol del documento XML.

Todo lo que esté dentro de las etiquetas de inicio y fin de la regla, es lo que será la salida en el árbol de resultados. En particular, "*los elementos XML normales o texto que aparezcan como contenido del elemento xsl:template terminarán en el documento de salida tal cual están; y aquellos que comiencen con el prefijo del espacio de nombres declarado en el prólogo (normalmente xsl o xslt), son elementos especiales que indicarán al procesador acciones o tareas a realizar*".

En el documento XML que se va a transformar hay que indicar mediante una instrucción de procesamiento la hoja de estilos XSL a usar. Dicha instrucción se añade al prólogo de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="nombre_archivo.xsl"?>
  . . .
```

Donde nombre_archivo.xsl sería el archivo de disco que contiene la transformación XSL.

Prácticamente la totalidad de los navegadores o clientes web actuales disponen de un procesador XSLT incluido, de forma que pueden transformar un documento XML según la hoja de estilos XSL que se indique en el documento XML. También hay software específico que permite realizar dichas transformaciones como XML Copy Editor o Xanan, así como entornos de desarrollo como es el caso de Visual Studio.

2.3.2. Operatoria de las transformaciones.

Básicamente podemos encontrarnos con tres tipos de transformaciones.

A. Si la hoja de transformaciones no contiene reglas (elementos xsl:template), sólo el contenido textual del documento XML se envía directamente a la salida. Los valores de los atributos, comentarios o instrucciones de procesamiento no se envían a la salida. Por ejemplo:

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

B. Si la hoja de transformaciones sólo contiene una regla cuyo patrón es el nodo raíz del documento XML, pero sin ningún contenido, no se envía nada a la salida. Ejemplo:

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

(La expresión XPath que expresa la raíz del documento origen es "/". Dichas expresiones las comentaremos posteriormente.)

C. Si la hoja de transformación no tiene ninguna regla cuyo patrón sea el nodo raíz, pero sí otras asociadas a otros nodos, se va recorriendo el árbol del documento XML y enviando a la salida el contenido textual del documento y, cuando se encuentre un nodo con una regla asociada, se le aplicará las acciones definidas en la regla.

Por ejemplo, supongamos que al documento *película.xml* le aplicamos la transformación:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="nombre">
          nnnnnnnnnn
     </xsl:template>
     <xsl:template match="apellidos">
          aaaaaaaaa
     </xsl:template>
 </xsl:stylesheet>
```

La salida resultante sería:

```
Mar adentro Alejandro Amenábar nnnnnnnnnn aaaaaaaaa nnnnnnnnnn aaaaaaaaa Sonia Grande
```

En general, si la hoja de transformación tiene alguna regla, se recorre el árbol del documento XML, y en el momento que aparezca alguna regla cuyo patrón coincida con el del nodo en que nos encontramos, se le aplica la regla.

- Ejemplo 1º.

La transformación más simple sería por ejemplo la propuesta por el siguiente documento XSL en donde existe una sola regla donde el patrón es la raíz del documento XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match ="/">
    <html>
        <head>
            <title>Pepe Cohete</title>
        </head>
        <body>
            <p>Hola, soy Pepe Cohete</p>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Cuando el procesador lea el documento XML, empezará por visitar el nodo raíz, y como la regla establece como patrón la raíz del documento, se produce la coincidencia y se aplicarán las acciones que indica la regla. Como el contenido de la regla es sólo texto y no existe ninguna acción de procesamiento

que involucre a algún nodo del árbol de entrada, el procesador se limitará a incluir en el árbol de salida todo el contenido de forma literal:

```
<html>
    <head>
        <title>Pepe Cohete</title>
    </head>
    <body>
        <p>Hola, soy Pepe Cohete</p>
    </body>
</html>
```

2.3.3. Otros elementos de XSL.

En el ejemplo anterior, la regla en su definición se limitaba a enviar a la salida contenido textual, pero existen otros elementos del espacio de nombres xsl que podemos usar como contenido de una regla para realizar otras tareas. Los principales elementos que se usan en XSL son:

- xsl:apply-templates hace que se apliquen las reglas que siguen a todos los nodos seleccionados.
- xsl:value-of extrae un valor concreto (literal o cadena) del documento.
- xsl:for-each aplica una acción repetidamente para cada nodo de un conjunto de nodos. En definitiva, se usa para iterar sobre una serie de nodos.
- xsl:if sirve para evaluar condiciones sobre valores de atributos o elementos.
- xsl:choose sirve para evaluar condiciones múltiples (tipo switch).
- xsl:sort ordena un conjunto de nodos de acuerdo a algún elemento. Por ejemplo, para ordenar alfabéticamente el contenido de un conjunto de elementos.

Pero antes de analizar los elementos anteriores es necesario conocer las expresiones XPath necesarias para direccionar las secciones de un árbol de documento XML.

2.4. Introducción a XPath.

Los patrones para las reglas pueden ser complejos y, como hemos indicado, se especifican en un lenguaje llamado **XPath**. Es un lenguaje declarativo, contextual (los resultados dependen del llamado *nodo de contexto*), y admite expresiones comunes: operadores de comparación, lógicos y matemáticos (=, <, and, or, *, +, etc.)

XPath es un lenguaje de consulta para direccionar las secciones de un documento XML obteniendo las partes de información que se necesiten.

Para direccionarnos dentro de un documento se utilizan las llamadas *expresiones XPath* que expresan una ruta de acceso a la parte del documento que queremos seleccionar. Pero para seleccionar una parte del documento necesitamos saber de dónde partimos; esto se conoce como el *nodo de contexto*: la sección del documento XML desde donde se inicia el trayecto.

Para entender mejor algunos conceptos, usaremos el archivo *pedido.xml* siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<pedido id="5432">
    <fecha>12/08/2010</fecha>
    <cliente id="24123456H">Dolores Fuertes</cliente>
    <lineaPedido>
        <numeroPieza almacen="Central">HH-123</numeroPieza>
        <descripcion>muelle del saltabalates</descripcion>
        <cantidad>5</cantidad>
    </lineaPedido>
    <lineaPedido>
        <numeroPieza almacen="Sur">MK-442</numeroPieza>
        <descripcion>freno del saltabalates</descripcion>
        <cantidad>10</cantidad>
    </lineaPedido>
</pedido>
```

2.4.1. Rutas de acceso de ubicación.

Lo primero que debemos considerar es el concepto de raíz del documento, que no debe confundirnos con el elemento raíz del documento. XPath necesita una raíz que actúe como raíz de la jerarquía del documento. De hecho, el elemento raíz del documento XML es un hijo de la raíz del documento.

La raíz del documento no apunta a ningún elemento del documento, simplemente es la raíz conceptual del documento. En XPath la raíz del documento se indica con un carácter "/". Por eso en la regla de nuestro primer ejemplo XSLT, que se comparaba con "/", se aplicaba a todo el documento.

- **Construcción de rutas de acceso de ubicación.**

Si en una regla sólo quisiéramos que se aplicara a los elementos <pedido> del archivo *pedido.xml*, en lugar de aplicarla a todo el documento, podríamos usar la expresión XPath "/pedido" que significa: empezar en el elemento raíz del documento y devolver los elementos <pedido> que son hijos del nodo raíz. En este caso el nodo de contexto, de donde partimos para localizar los nodos, es la raíz del documento porque es la sección de donde se empieza.

Aunque lo más cómodo es la leer la expresión de la siguiente manera: "seleccionar todos los elementos denominados <pedido> que son hijos de la raíz del documento".

Las expresiones XPath pueden ser relativas o absolutas. Si son relativas indican que comienzan en el nodo de contexto, si son absolutas implican que comienzan en la raíz del documento.

Por lo tanto, la expresión "pedido" sería una ruta relativa que significa: "seleccionar todos los elementos <pedido> que son hijos del nodo de contexto".

Para recorrer la jerarquía de un árbol, XPath utiliza el carácter "/" para separar los nodos. Así la expresión "/pedido/lineaPedido/numeroPieza" significa: "seleccionar todos los elementos denominados <numeroPieza> que son hijos de los elementos <lineaPedido> que son hijos de los elementos <pedido>, que son hijos de la raíz del documento."

Para direccionar atributos se usa el símbolo @. Así "@id" significa seleccionar el atributo id del nodo de contexto; y "/pedido/cliente/@id" selecciona el atributo id de los nodos <cliente>

Operador	Descripción
/	Selección de hijo. Selecciona únicamente a los descendientes directos. Si se usa al principio del patrón, indica la raíz del documento.
//	Selección de descendientes. Selecciona todos los descendientes.
.	Selección del elemento actual (el de contexto).
..	Selección del elemento padre del elemento actual
*	Selección de todos (en el sentido habitual de este operador)
@	Prefijo que se antepone al nombre de un atributo.
[ ]	Filtro sobre el conjunto de nodos seleccionado.

La localización de nodos basados en su nombre, independientemente del lugar en donde aparezcan en el documento, se hace utilizando la notación "//". Así "//@id" selecciona todos los atributos id del documento, independientemente del elemento al que pertenece y de su ubicación en el árbol de nodos.

Algunos ejemplos más:

./cliente Selecciona todos los elementos <cliente> dentro del contexto actual. (Es decir, todos los hijos del nodo actual que tengan como etiqueta <cliente>).

/cliente Selecciona los elementos con etiqueta <cliente> que cuelgan directamente de la raíz.

//cliente Selecciona todos los elementos <cliente> en cualquier parte del documento.

//@* Selecciona todos los atributos del documento.

- **Predicados XPath.**

Los predicados permiten localizar nodos que tengan valores específicos o que cumplan ciertas condiciones. Para ello se usan corchetes que actúan como un filtro sobre los nodos seleccionados. Así podemos comprobar los elementos con determinados nodos hijos poniendo simplemente el nombre del nodo hijo entre los corchetes. Por ejemplo:

./pedido[cliente] Busca la coincidencia de cualquier elemento <pedido> que es hijo del nodo de contexto y que tiene un elemento hijo <cliente>.

pedido[@id] Busca la coincidencia de todos los elementos <pedido> que sean hijos del nodo de contexto y que tengan un atributo id.

pedido[cliente ='Dolores Fuertes'] Busca la coincidencia de todos los elementos <pedido> que sean hijos del nodo de contexto y que tengan un hijo <cliente> con el valor 'Dolores Fuertes'.

//cliente[@id='12345678A '] Selecciona todos los elementos <cliente> en cualquier parte del documento y de éstos, aquellos que tengan el valor 12345678A en el atributo id.

/pedido/lineaPedido[1] Selecciona el primer elemento <lineaPedido> hijo de <pedido>

/pedido/lineaPedido[last()-1] Selecciona el penúltimo elemento <lineaPedido> hijo del elemento <pedido>

/pedido/lineaPedido[position()<5] Selecciona los cuatro primeros elementos <lineaPedido> hijos del elemento <pedido>

/pedido/lineaPedido[cantidad>3]/numeroPieza Selecciona los elementos <numeroPieza> de aquellas líneas de pedido que has suministrado más de 3 piezas.

- **Operadores XPath.**

Las expresiones XPath no sólo pueden retornar un nodo o conjunto de nodos, sino también una cadena, un valor booleano o un número debido a la posibilidad de usar operadores. Estos son:

- Aritméticos: +, - *, div, mod
- Relacionales =, !=, >, >=, <, <=
- Lógicos: not(), or, and
- Operador | (permite seleccionar dos conjuntos de nodos). Por ejemplo: //libros | //cd

- **Funciones XPath.**

Las expresiones XPath disponen de gran cantidad de funciones que permiten por ejemplo direccionar nodos o conjuntos de nodos que no pueden ser encontrados por medio de las relaciones normales de padre/hijo o elemento/atributo. También existen funciones para utilizar con cadenas de caracteres o números, bien para recuperarlos o para formatearlos en la salida.

Todas las funciones terminan con paréntesis (). Algunas necesitan información para operar y se suministran a la función mediante parámetros que se colocan entre los paréntesis. Por ejemplo, la función string-lenght('hola') retorna el número de caracteres de la cadena pasada como parámetro; en este caso 4.

A continuación, se presenta una lista de las funciones más usuales en las expresiones XPath:

name(). Obtiene el nombre del nodo actual.

root(). Obtiene el elemento raíz.

text(). Retorna el contenido de un nodo.

node(). Retorna el propio nodo y todos sus descendientes.

position(). Obtiene la posición del nodo de un conjunto de nodos.

last(). Obtiene la posición del último nodo de un conjunto de nodos.

count(). Retorna la cantidad de nodos de un conjunto de nodos.

number(). Convierte el contenido de un elemento o valor de atributo en un valor numérico.

sum(). Retorna la suma de los valores numéricos de un conjunto de nodos.

avg(). Obtiene la media de los valores numéricos de un conjunto de nodos.

max(). Valor máximo de un conjunto de nodos.

min(). Valor mínimo de un conjunto de nodos.

format-number(). Devuelve un número con el formato que se especifique.

string(). Convierte cualquier valor a cadena de caracteres.

string-length(). Retorna el número de caracteres de una cadena.

concat(). Concatena o une varias cadenas.

contains(). Indica si una cadena contiene en su interior una cadena determinada.

starts-with(). Indica si una cadena comienza con el contenido de otra cadena.

substring(). Obtiene una cadena a partir del contenido de otra cadena.

boolean(). Evalúa una expresión XPath para verificar si es verdadera o falsa.

not(). Niega una expresión.

true(). Siempre devuelve verdadero.

false(). Siempre devuelve falso.

processing-instruction(). Retorna instrucciones de procesamiento.

comment(). Retorna comentarios.

Ejemplos. Usaremos el documento XML de *pedido.xml* anterior

- Selecciona la descripción de las piezas de los almacenes Central o Sur

  /pedido/lineaPedido/numeroPieza[@almacen = 'Central' or @almacen = 'Sur']/descripción

- Selecciona todos los nodos que comiencen con el carácter 'c'.

  //*[starts-with(name(),'c')]

- Suma las cantidades para el pedido 5432.

  sum(/pedido[@id = '5432']/lineaPedido/cantidad)

2.5. Ejemplos de transformación.

- Ejemplo 2º.

Para comprobar cómo se aplican las reglas partiremos del documento *película.xml* al cual le aplicaremos varias transformaciones.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pelicula.xsl"?>
<pelicula idioma="español">
    Mar adentro
    <director>Alejandro Amenábar</director>
    <guionista></guionista>
    <actores>
        <actor papel="protagonista">
            <nombre>Javier</nombre>
            <apellidos>Bardem</apellidos>
        </actor>
        <actor papel="secundario">
            <nombre>Celso</nombre>
            <apellidos>Bugallo</apellidos>
        </actor>
    </actores>
    <vestuario>Sonia Grande</vestuario>
</pelicula>
```

Vamos a realizar una primera transformación suponiendo que el archivo XSL *película.xsl* es:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/pelicula/actores">
        -- No se obtienen los actores porque aunque el patrón de la regla
           es válido, no se especifica ninguna acción sobre ningún elemento.--
  </xsl:template>
</xsl:stylesheet>
```

El resultado de la transformación, otro documento XML, es el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
Mar adentro Alejandro Amenábar -- No se obtienen los actores por que aunque el patrón de la regla es válido, no se especifica ninguna acción sobre ningún elemento. -- Sonia Grande
```

Cuando se realiza una transformación el procesador empieza recorriendo el árbol de nodos del documento desde el nodo raíz visitando todos los nodos descendientes. *Si para el nodo que está visitando existe una regla (elemento **xsl:template**), realizará las acciones que indique la regla, en caso contrario se limitará a colocar en la salida el contenido del elemento del nodo visitado.*

Al empezar desde el nodo raíz, al documento XML de salida se van agregando los contenidos de los elementos de los nodos visitados: Mar adentro Alejandro Amenábar. Cuando se llega al nodo <actores> al existir una regla para él y sus descendientes se aplicará. Pero como la regla no indica que hacer con los contenidos de los nodos, sólo tiene texto, éste se enviará directamente a la salida. El recorrido continúa por el resto de los nodos, llegando al nodo <vestuario> y añadiendo su contenido: Sonia Grande a la salida.

Ahora bien, si en la regla anterior se hubiera indicado alguna operación a efectuar con los contenidos, sería el resultado de dicha operación lo que se enviaría a la salida. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/pelicula/actores">
     <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

La salida habría sido:

```
<?xml version="1.0" encoding="utf-8"?>
 Mar adentro
    Alejandro Amenábar
      Javier
      Bardem
      Celso
      Bugallo
   Sonia grande
```

Aunque después se explicará, el elemento <xsl:value-of> permite obtener el valor de "algo" y colocarlo en la salida. Este "algo" se especifica mediante la expresión XPath asignada al atributo select.

Para explicar el resultado debemos tener en cuenta lo siguiente. *La expresión XPath utilizada en atributo match de la regla se convierte en el nodo de contexto de la regla; por lo que todas las expresiones XPath que aparezcan en el interior de la regla serán relativas a dicho nodo.*

La regla del ejemplo se aplica para cada uno de los elementos <actores> que sean hijos de elementos <pelicula>, que a su vez sean hijos de la raíz del documento. Cuando se alcance a algún un elemento <actores> éste será el nodo de contexto para la regla, y por lo tanto la expresión XPath **"."** hace referencia a dicho nodo. Por lo tanto, el resultado de <xsl:value-of select="."/> es el contenido del elemento del nodo de contexto y *el de sus descendientes.*

En nuestro caso se obtiene en la salida el contenido de todos los elementos <actores>, (que no tienen ningún contenido…) y el de todos sus descendientes: es decir, los contenidos de los elementos <actor> (sin contenido), y los contenidos de los elementos <nombre> y <apellidos>.

- Ejemplo 3º

En esta ocasión, la salida será un documento HTML. El archivo XSLT en este caso será:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
    <html>
        <head>
            <title><xsl:value-of select="/pelicula/text()" /></title>
        </head>
        <body>
            <p><strong>Contenido del nodo película y sus descendientes:</strong><xsl:value-of
            select="/pelicula" />
            </p>
```

```
            <p><strong>Contenido del nodo actores y sus descendientes</strong>
                <xsl:value-of select="/pelicula/actores" />
            </p>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Y la salida que se obtiene es:

```
<html>
    <head><title>Mar  adentro</title></head>
    <body>
        <p><strong>Contenido del nodo película y descendientes:</strong> Mar adentro
            Alejandro Amenábar Javier Bardem Celso Bugallo Sonia grande
        </p>
        <p><strong>Contenido del nodo actores y descendientes</strong> Javier Bardem Celso Bugallo</p>
    </body>
</html>
```

Cuando el procesador lea el documento XML, empezará por visitar el nodo raíz, y como la regla establece como patrón la raíz del documento, se produce la coincidencia y se aplicarán las acciones que indica la regla. Las acciones comienzan con trasladar a la salida: el texto contenido en la regla, (las etiquetas <html><head><title>); a continuación, aparece el elemento <xsl:value-of> que permite como ya sabemos obtener el valor de "algo" y colocarlo en la salida. El valor de ese "algo" se obtendrá del nodo y de sus descendientes que se indique mediante el atributo select.

El valor obtenido dependerá del tipo de nodo. Si el elemento del nodo tuviera contenido, lo enviará a la salida y además enviará a la salida los contenidos de los elementos descendientes. Si en ese recorrido algún nodo descendiente no tuviera contenido, lógicamente no se enviará nada de dicho nodo a la salida.

Ahora bien, si requiere obtener sólo el contenido del nodo especificado en el atributo select, ignorando el contenido de los descendientes, habría que usar la función text() para dicho nodo.

Por lo tanto, el elemento <xsl:value-of select="/pelicula/text()" /> enviará a la salida sólo el contenido del elemento, es decir Mar adentro. A continuación, se enviarán a la salida las etiquetas </title></head><body><p> y el texto Contenido del nodo película y descendientes: seguido de la etiqueta .

El elemento <xsl:value-of select="/pelicula" /> enviará a la salida Mar adentro Alejandro Amenábar Javier Bardem Celso Bugallo Sonia grande, es decir, los contenidos del nodo y de sus descendientes.

La salida se acaba de forma parecida: enviando de forma literal etiquetas HTML y texto, junto a los valores obtenidos por <xsl:value-of select="/pelicula/actores"/> que son el contenido del nodo <actores> y todos sus descendientes.

2.6. Elementos básicos de transformación.

2.6.1. Elemento <xsl:apply-templates>

Este elemento se usa desde dentro de una regla para llamar o invocar a otras reglas. Si el elemento usa el atributo opcional select, entonces se evaluará la expresión XPath y el resultado proporcionará el nodo de contexto que sería usado por las reglas invocadas. Como este atributo es opcional, si no se especifica, el nodo de contexto no cambiaría manteniéndose el vigente.

```
<xsl:apply-templates select="patron"/>
```

Es importante comprender bien el efecto de este elemento: cuando en el recorrido del árbol del documento aparezca <xsl:apply-templates select="patron"/> se invocarán las reglas que existan para el nodo o nodos que se determinen mediante el patron, y el resultado obtenido se incrustará en dicho punto del árbol de salida. En el caso de que no se usara el atributo select, sólo se aplicarían aquellas reglas cuyo atributo match coincidiera con el del nodo que se está visitando, es decir con el de contexto.

También pudiera darse el caso de que no existieran nuevas reglas a aplicar, y por lo tanto su efecto sería nulo, limitándose la acción a cambiar el nodo de contexto en el caso de que se hubiere usado el atributo select y a continuar con el procesamiento normal de los nodos pendientes.

Como aclaración, comentaremos algunos ejemplos:

-Ejemplo 4º.

Usaremos el siguiente archivo *libros.xml* con datos de una colección de libros:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type= "text/xsl" href="libros.xsl"?>
<biblioteca>
    <libro prestado="SI">
        <titulo>Visual C#</titulo>
        <autor>Fco. Javier Ceballos</autor>
        <editorial>Ra-Ma</editorial>
        <paginas>936</paginas>
        <precio>52,75</precio>
    </libro>
    <libro prestado="NO">
        <titulo>Programación en C</titulo>
        <autor>Luis Joyanes Aguilar</autor>
        <editorial>McGraw-Hill</editorial>
        <paginas>735</paginas>
        <precio>45,25</precio>
    </libro>
</biblioteca>
```

Si usamos la transformación siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <em>
    <xsl:apply-templates/>
  </em>
</xsl:template>
</xsl:stylesheet>
```

Como la regla afecta a todos los nodos del árbol, se empezaría colocando en la salida , a continuación <xsl:apply-templates/> provocaría el procesamiento normal de los nodos enviándolos a la salida salvo que al alcanzar alguno de éstos encontráramos una regla para él. Como no hay ninguna regla adicional, ni tampoco cambia el nodo del contexto, el efecto es nulo. Por lo tanto se sigue el procesamiento normal enviándose a la salida el contenido del nodo raíz y los de sus descendientes. Acabado lo anterior se termina añadiendo a la salida. El resultado final sería:

```
<em>Visual C# Javier Ceballos Ra-Ma 936 52,75  Programación en C Luis Joyanes Aguilar McGraw-Hill 735 45,25
</em>
```

Ahora bien, si usamos la transformación siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="editorial">
  <em>
    <xsl:apply-templates />
  </em>
</xsl:template>
</xsl:stylesheet>
```

Se empezaría colocando en la salida el contenido del nodo <biblioteca> (que está vacío) y el de todos sus descendientes: es decir el contenido de <libro> (que está vacío) el de <titulo>,<autor> y al llegar al nodo <editorial> la regla traslada a la salida , a continuación <xsl:apply-templates/> provocaría el procesamiento normal de los nodos pendientes enviándolos a la salida salvo que al alcanzar alguno de éstos encontráramos una regla para él. Como no hay ninguna regla adicional, ni tampoco cambia el nodo del contexto, su efecto es nulo, y por lo tanto se sigue el procesamiento normal enviándose a la salida el contenido del nodo <editorial> y el de sus descendientes junto a . Terminaríamos enviando a la salida el contenido del resto de nodos: <paginas> y <precio>. La salida sería:

```
Visual C# Javier Ceballos<em>Ra-Ma</em>936 52,75Programación en C Luis Joyanes Aguilar<em>McGraw-Hill</em>735 45,25
```

- Ejemplo 5º.

Usemos ahora la siguiente transformación para el archivo *libros.xml*

```
?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="libro">
   <em>
      <xsl:apply-templates select="editorial"/>
   </em>
</xsl:template>
</xsl:stylesheet>
```

En la salida se añadiría directamente el contenido del nodo <biblioteca> (que está vacío) y el de sus descendientes. Como la regla se establece para los nodos <libro>, **por cada libro** se añadiría a la salida , a continuación <xsl:apply-templates select="editorial"/> provocaría el procesamiento normal de los nodos enviándolos a la salida salvo que al alcanzar alguno de éstos encontráramos una regla para él. En este caso tampoco hay reglas adicionales, **pero sí se cambia el nodo de contexto al nodo** <editorial>, por lo que sólo para este nodo y sus descendientes se trasladaría a la salida su contenido; finalmente se añadiría a la salida . En este caso, la salida final sería:

```
<em>Ra-Ma</em><em>McGraw-Hill</em>
```

- Ejemplo 6ºA. Supongamos la siguiente hoja de transformación:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
   <h2>Lista de libros</h2>
   <xsl:apply-templates /><hr/>
</xsl:template>

<xsl:template match="libro">
     <p><xsl:value-of select="editorial" /></p>
</xsl:template>
</xsl:stylesheet>
```

En la transformación anterior como la regla afecta a todos los nodos del árbol, se empezaría colocando en la salida <h2>Lista de libros</h2>, a continuación <xsl:apply-templates/> provocaría el procesamiento normal de los nodos enviándolos a la salida salvo que al alcanzar alguno de éstos encontráramos una regla para él. En este caso **si existe una regla para <libro>** que se aplicará cuando se alcance dicho nodo. La regla indica que para cada libro se coloque en la salida , el contenido del nodo <editorial> y el de sus descendientes, y . Se terminaría colocando en la salida <hr/>.

La salida sería:

```
<h2>Lista de libros</h2>
<p> Ra-Ma </p>
<p> McGraw-Hill </p>
<hr/>
```

- Ejemplo 6ºB. Supongamos ahora la siguiente transformación:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
     <h2>Lista de libros</h2>
     <xsl:apply-templates /><hr/>
</xsl:template>

<xsl:template match="editorial">
     <em><xsl:value-of select="." /></em>
</xsl:template>
</xsl:stylesheet>
```

Se empezaría colocando en la salida <h2>Lista de libros</h2>, a continuación <xsl:apply-templates/> provocaría el procesamiento normal de los nodos enviándolos a la salida salvo que al alcanzar alguno de éstos encontráramos una regla para él. En este caso, se enviarían a la salida los datos de cada libro y cuando se alcanzara un nodo <editorial>, se colocaría en la salida , el contenido del nodo <editorial> y el de sus descendientes, y posteriormente se envía . Finalmente se terminaría colocando en la salida <hr/>.

La salida sería:

```
<h2>Lista de libros</h2>
Visual C#
Fco. Javier Ceballos
<em>Ra-Ma</em>
936
52,75

Programación en C
Luis Joyanes Aguilar
<em>McGraw-Hill</em>
735
45,25
<hr />
```

- Ejemplo 7º. La siguiente transformación, va aplicando reglas sucesivamente hasta llegar al nodo <libro>, apartir de este nodo sólo se aplican las reglas para visualizar los contenidos de <autor> y <titulo>, los cuales tienen sus propias reglas.

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

<xsl:template match="/">
    <html>
        <head>
            <title>Lista de libros</title>
            <style>
                .neg {font-weight:bold}
                .roj {color:# FF0000}
                .az {color:#0000FF}
            </style >
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>

<xsl:template match="biblioteca">
    <h2>Mis librillos:</h2>
    <xsl:apply-templates/>
    <hr/>
</xsl:template>

<xsl:template match="libro">
    <p>
        <strong class="neg">Autor: </strong><xsl:apply-templates select="autor"/>
        <strong class="neg">. Título: </strong ><xsl:apply-templates select="titulo"/>
    </p>
</xsl:template>

<xsl:template match="titulo">
    <span class="az">
        <xsl:value-of select="." />
    </span>
</xsl:template>

<xsl:template match="autor">
    <span class="roj">
        <xsl:value-of select="." />
    </span>
</xsl:template>

</xsl:stylesheet>
```

La salida ya procesada por un explorador web sería:

Mis librillos:

Autor: Fco. Javier Ceballos**. Título:** Visual C#

Autor: Luis Joyanes Aguilar**. Título:** Programación en C

Idéntica salida se hubiera obtenido con la siguientre transformación:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <head>
            <title>Lista de libros</title>
            <style>
                .neg {font-weight:bold}
                .roj {color:# FF0000 }
                .az {color:#0000FF }
            </style >
        </head>
        <body>
            <xsl:apply-templates />
        </body>
    </html>
 </xsl:template>

<xsl:template match="biblioteca">
    <h2>Mis librillos:</h2>
    <xsl:apply-templates /><hr/>
</xsl:template>

<xsl:template match="libro">
    <p><strong class="neg">Autor: </strong><span class="az"><xsl:value-of select="autor"/></span>
     <strong class="neg"> Título:</strong><span class="roj"><xsl:value-of select="titulo"/></span></p>
</xsl:template>

</xsl:stylesheet>
```

2.6.2. Elemento <xsl:output>

El elemento <xsl:output> permite especificar el formato del documento de salida, es decir podemos elegir un formato XML, HTML o incluso en formato texto simple. Cuando se utilice debe colocarse como un hijo directo de <xsl:stylesheet>. El formato es:

```
<xsl:output method = "xml | html | text" version= "version"  encoding = "codigo caracteres" indent = "yes | no"/>
```

El atributo method especifica qué tipo de salida a va a generar: “xml”, “html” o “text”. Si no se incluye <xsl:output> y el elemento raíz en el árbol de salida es <html>, entonces el método predeterminado de salida es “html”, en caso contrario es “xml”.

Los atributos version y encoding sólo se pueden usar cuando el método de salida es “xml”o “html”. Los valores de estos atributos se usarán para crear la declaración XML del árbol de salida.

El atributo indent permite generar una salida más legible indentando el texto, pero los procesadores XSL tienen libertad en la menara de implantar dicha legibilidad. Por ejemplo:

```
<xsl:output method = "xml" version= "1.0"  encoding = "UTF-8"/>
```

2.6.3. Elemento <xsl:value-of>

Este elemento permite extraer información de los nodos del documento a transformar. Su sintaxis es:

```
<xsl:value-of  select="patron"/>
```

Simplemente selecciona el valor de un nodo o conjunto de nodos que han sido especificados mediante la expresión XPath del atributo **select**. Importante: si el atributo **select** especifica un nodo del que existen más en el conjunto de nodos, se enviará a la salida el valor del primer nodo del conjunto, no los valores de todos los nodos.

- Ejemplo 8º. La transformación siguiente envía a la salida el contenido completo de <biblioteca>

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

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

```
        <html>
            <head>
                <title>Lista de libros</title>
            </head>
            <body><xsl:apply-templates/></body>
        </html>
    </xsl:template>


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

    </xsl:stylesheet>
```

La salida que se obtendría sería un documento html cuyo contenido es:

Visual C# Fco. Javier Ceballos Ra-Ma 936 52,75 Programación en C Luis Joyanes Aguilar McGraw-Hill 735 45,25

En cambio, si sólo cambiamos el elemento <xsl:value-of select="."/> por este otro:

```
<xsl:value-of select="libro"/>
```

Obtendríamos en el documento html sólo el contendido del primer nodo <libro>:

Visual C# Fco. Javier Ceballos Ra-Ma 936 52,75

2.6.4. Elemento <xsl:call-template>

Permite invocar a una regla por su nombre. A diferencia de <xsl:applyl-templates>, <xsl:call-template > no cambia el nodo actual o la lista de nodos actual. Por lo que en la regla invocada hay que usar el mismo direccionamiento de elementos que el de la regla invocante.

Ejemplo 9º:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <xsl:call-template name="ListaEditorial"/>
</xsl:template>

<xsl:template name="ListaEditorial">
    <p><xsl:value-of select="biblioteca/libro/editorial" /></p>
</xsl:template>

</xsl:stylesheet>
```

Cuando se invoca la regla, el nodo de contexto es el raíz, por lo que en la regla invocada, si queremos acceder a los nodos <editorial>, se necesita indicar la ruta desde el nodo raíz hasta los nodos <editorial>. Como ya comentamos anteriormente al explicar el elemento <xsl:value-of>, la salida sólo presenta el valor del primer nodo del conjunto de ellos.

```
<?xml version="1.0" encoding="utf-8"?>
<p>Ra-Ma</p>
```

2.6.5. Elemento <xsl:text>

Permite escribir texto de forma literal en la salida.

Ejemplo 10º:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  linea 1
  linea 2
  linea 3<xsl:text>&#10;&#9;&#9;</xsl:text>linea 4
</xsl:template>
</xsl:stylesheet>
```

Entre los textos linea 3 y linea 4 se envía un salto de línea y dos tabuladores usando los códigos UNICODE de dichos caracteres.

2.7. Elementos de control.

Los elementos de control nos permiten seleccionar varios nodos y repetir las acciones para cada uno de ellos, al igual que imponer condiciones para que se procesen o no determinados nodos.

2.7.1. Elemento <xsl:for-each>

Este elemento nos permite procesar todos y cada uno de los nodos pertenecientes a un conjunto de nodos. A nivel de proceso permite iterar o realizar bucles. El atributo select especifica el conjunto de nodos que se van a procesar individualmente. Su sintaxis es:

```
<xsl:for-each select="expresion">
  ...
</xsl:for-each>
```

-Ejemplo 11º. La transformación siguiente genera como salida un documento HTML en donde para cada nodo <libro> se presentan los contenidos de los nodos <titulo>, <autor> y <editorial>. El elemento <xsl:for-each> indica que deben procesarse todos y cada uno de los nodos <libro>.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <head>
            <title>Informe</title>
            <style>
                body {background-color:#D2D2D2}
                h1, p {border-botom:1px double black}
                .neg {font-weight:bold}
                .cur {font-style:italic }
            </style >
        </head>
        <body>
            <h1>Informe de libros</h1>
            <xsl:for-each select="biblioteca/libro">
                <p>
                    <strong class="neg">Titulo: </strong><em class="cur"><xsl:value-of select="titulo"/></em>
                </p>
                <p>
                    <strong class="neg">Autor: </strong><em class="cur"><xsl:value-of select="autor"/></em>
                </p>
                <p>
                    <strong class "neg">Editorial: </strong><em class ="cur"><xsl:value-of select="editorial"/></em>
                </p>
            </xsl:for-each>
        </body>
    </html>
</xsl:template>


</xsl:stylesheet>
```

El documento HTML obtenido como salida, se presentaría en un navegador de la forma:

Informe de libros

Titulo: *Visual C#*
Autor : *Fco. Javier Ceballos*
Editorial: *Ra-Ma*

Titulo: *Programación en C*
Autor : *Luis Joyanes Aguilar*
Editorial: *McGraw-Hill*

2.7.2. Elemento <xsl:if>

Es un elemento condicional, que nos permite establecer una salida dependiendo de que la condición que se evalúa sea verdadera o falsa. Con el atributo test se especifica la condición. El formato es:

```
<xsl:if test = "condición">
   ... salida si la condición es verdadera
</xsl:if>
```

Si la condición que se establece se evaluara como falsa, no se procesaría nada.

-Ejemplo 12º. Vamos a modificar la transformación anterior de forma que sólo se obtengan datos de aquellos libros que tuvieran más de 900 páginas. Para ello añadiríamos a la transformación la condición:

```
<xsl:if test="paginas &gt; 900"> . . . </xsl:if>
```

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

 <xsl:template match="/">
    <html>
        <xsl:apply-templates select="biblioteca" />
    </html>
 </xsl:template>
 <xsl:template match="biblioteca">
    <head>
        <title>Libros gordos</title>
        <style>
            body {background-color:#FAFAFA}
            .neg {font-weight:bold}
            .cur {font-style:italic }
        </style >
    </head>
    <body>
        <h1>Libros tochos</h1>
        <p>Los libros que tienen más de 900 páginas son:</p>
        <ol>
            <xsl:apply-templates select="libro" />
        </ol>
    </body>
 </xsl:template>

 <xsl:template match="libro">
    <xsl:if test="paginas &gt; 900">
        <li>
            <strong class="neg"><xsl:value-of select="titulo"/></strong>(<em class="cur"><xsl:value-of
            select="autor"/></em>): <xsl:value-of select="paginas"/> págs.
        </li>
    </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

La salida sería:

Libros tochos

Los libros que tienen más de 900 páginas son:

1. **Visual C#** (*Fco. Javier Ceballos*): 936 págs.

2.7.3. Elemento <xsl:choose>

Con el elemento anterior sólo podíamos establecer una única condición. Con <xsl:choose> podemos establecer condiciones múltiples, es decir, podremos evaluar más de una condición y establecer salidas diferentes según se cumplan o no cada una de las condiciones. La sintaxis es:

```
<xsl:choose>
  <xsl:when test="expression">   alguna salida ...  </xsl:when>
  <xsl:when test="expression">   alguna salida ...  </xsl:when>
  . . .
  [<xsl:otherwise>   alguna salida...  </xsl:otherwise>]
```

```
</xsl:choose>
```

En los bloques <xsl:when test="expression"> mediante el atributo test se fijan las condiciones, las cuales se evaluaran descendentemente; por lo tanto se pueden colocar tantos bloques <xsl:when> como condiciones a evaluar. El bloque <xsl:otherwise> es opcional, y si se usa permite establecer la salida cuando ninguna de las condiciones de los bloques <xsl:when> se cumplan.

-Ejemplo 13º. Se realizará una transformación de forma que la salida presentará los datos de los libros de la editorial Ra-Ma de color azul, los de McGraw-Hill en rojo, y en verde para el resto de editoriales.

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

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


<xsl:template match="biblioteca">
    <head>
        <title>Libros</title>
        <style>
            body {background-color:#FAFAFA}
            .az {color:#0000FF }
            .roj {color:#FF0000}
            .verd {color:#00FF00}
        </style >
    </head>
    <body>
        <h1>Listado Libros</h1>
        <xsl:apply-templates select="libro" />
    </body>
 </xsl:template>

<xsl:template match="libro">
    <xsl:choose>
        <xsl:when test="editorial = 'Ra-Ma'">
            <p class="az"><xsl:value-of select="."/></p>
        </xsl:when>
        <xsl:when test="editorial = 'McGraw-Hill'">
            <p class="roj"><xsl:value-of select="."/></p>
        </xsl:when>
        <xsl:otherwise>
            <p class="verd"><xsl:value-of select="."/></p>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

El documento HTML obtenido como salida, se presentaría en un navegador de la forma:

Listado de libros

Visual C# Fco. Javier Ceballos Ra-Ma 936 52,75

Programación en C Luis Joyanes Aguilar McGraw-Hill 735 45,25

2.7.4. Elemento <xsl:sort>

Este elemento permite establecer el orden de salida para un conjunto de nodos. El elemento <xsl:sort> se puede utilizar junto a los elementos <xsl:apply-templates> y <xsl:for-each>. Su sintaxis es:

```
<xsl:sort select="patron" order="ascending | descending"   data-type="text | number" lang="idioma" />
```

El atributo select permite seleccionar qué se va ordenar. Usando el atributo order podemos establacer una ordenación ascendente asignándole el valor *ascending* o descendente *descending*. El atributo data-type permite establecer la ordenación: alfabética (*text*) o numérica (*number*). La ordenación por defecto es alfabética ascendente. Con lang se especifica el código del idioma para ordenar.

-Ejemplo 14º.

Modificaremos la transformación anterior para conseguir la misma salida pero ordenada por <titulo>. Para ello sólo deberemos modificar en la segunda regla, el elemento:

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

por este otro en donde se añade un elemento de ordenación sobre el contenido de <titulo>:

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

Ahora en este caso la salida se presentaría ordenada ascendentemente por el título del libro:

Listado de libros

Programación en C Luis Joyanes Aguilar McGraw-Hill 735 45,25

Visual C# Fco. Javier Ceballos Ra-Ma 936 52,75

-Ejemplo 15º.

La salida que se obtuvo en el ejemplo 11º que usaba el elemento <xsl:for-each> para recorrer todos los nodos <libro> y presentar los contenidos de <titulo>, <autor> y <editorial>, podemos modificarla para que también presente el contenido de <precio>, y además quede ordenado por <precio>. Para ello le hemos añadido a la salida el contenido de <precio> y hemos indicado dentro del elemento <xsl:for-each> un elemento <xsl:sort> para que el recorrido por los diferentes nodos se presenten en la salida ordenados por precio:

```
<xsl:for-each select="/biblioteca/libro">
    <xsl:sort select="precio" data-type="number"/>
    <p><strong id="neg">Titulo: </strong><em id="cur"><xsl:value-of select="titulo" /></em></p>
    <p><strong id="neg">Autor : </strong><em id="cur"><xsl:value-of select="autor" /></em></p>
    <p id="lin">
        <strong id="neg">Editorial: </strong><em id="cur"><xsl:value-of select="editorial"/></em>
        <strong id="neg">Precio: </strong><em id="cur"><xsl:value-of select="precio"/></em>
    </p>
</xsl:for-each>
```

La nueva salida será:

Informe de libros

Titulo: *Visual C#*
Autor : *Fco. Javier Ceballos*
Editorial: *Ra-Ma*
Precio: *45,25*

Titulo: *Programación en C*
Autor : *Luis Joyanes Aguilar*
Editorial: *McGraw-Hill*
Precio: *52,75*

2.8. Generación de nuevos elementos y atributos.

2.8.1. Elemento <xsl:copy>

El elemento xsl:copy realiza una copia del nodo actual, pero no copia ni los descendientes ni lo atributos que tuviera. La mayor utilidad de este elemento es cuando se quiere usar el elemento, pero cambiando la estructura de su contenido o añadiendo o eliminando atributos.

```
<xsl:copy>. . .</xsl:copy>
```

- Ejemplo 16º.

Con la siguiente transformación crearíamos como salida un archivo XML con los nodos <biblioteca>, <libro> y <autor>, copiando el contenido del nodo <autor>.

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

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

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

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

</xsl:stylesheet>
```

La salida sería:

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
    <libro>
        <autor>Fco. Javier Ceballos</autor>
    </libro>
    <libro>
        <autor>Luis Joyanes Aguilar</autor>
    </libro>
</biblioteca>
```

2.8.2. Elemento <xsl:copy-of>

Este elemento copia en la salida un nodo y todos los descendientes, incluidos los atributos que tuvieran los nodos. Su principal uso es para enviar a la salida un fragmento del documento XML sin aplicarle ninguna transformación.

Su sintaxis es:

```
<xsl:copy-of select="expresion"/>
```

- Ejemplo 17º.

La transformación siguiente crea un nuevo documento XML en el que se sustituye el nodo <biblioteca> por otro llamado <libreria>. El resto de los nodos se copian todos.

```
<?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>
        <xsl:apply-templates select="biblioteca"/>
    </libreria>
</xsl:template>

<xsl:template match="biblioteca">
    <xsl:copy-of select ="libro"/>
</xsl:template>

</xsl:stylesheet>
```

2.8.3. Elemento <xsl:attribute>

El elemento <xsl:attribute> permite añadir atributos a un elemento. Si el atributo que se añade ya existiera, lo reemplazaría. La sintaxis es:

```
<xsl:attribute name="nombre_atributo">
    …algo que defina el valor del atributo
</xsl:attribute>
```

Donde nombre_atributo es el nombre del atributo a añadir.

Por ejemplo, para añadir el atributo ruta al elemento <imagen> y asignarle el valor "images/im.jpg", sería:

```
<imagen>
  <xsl:attribute name="ruta">
      <xsl:value-of select="images/im.jpg" />
  </xsl:attribute>
</imagen>
```

-Ejemplo 18º.

Supongamos que modificamos el documento *bibloteca.xml* añadiéndole a cada libro un nuevo elemento denominado <web> que contiene la URL de la empresa editora del libro, y queremos realizar una transformación para obtener un documento HTML con la lista de los títulos de los libros seguidos de un enlace al sitio web de la editorial.

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/xsl/transform">
<xsl:template match="/">
      <html>
          <head>
          <title>Informe de libros y editoriales</title>
          <style>
              body {background-color:#FAFAFA}
          </style >
          </head>
          <body>
              <h1>Libros y editoriales</h1>
              <ol>
                  <xsl:for-each select="biblioteca/libro">
                      <li>
                          <xsl:value-of select="titulo" /><xsl:text> (</xsl:text>
                          <a>
                              <xsl:attribute name="href">
                                  <xsl:text>http://</xsl:text>
                                  <xsl:value-of select="web"/>
                              </xsl:attribute>
                              <xsl:value-of select="editorial"/>
                          </a>
                          <xsl:text>)</xsl:text>
                      </li>
                  </xsl:for-each>
              </ol>
          </body>
   </html>
   </xsl:template>
  </xsl:stylesheet>
```

La salida es:

```
<html>
    <head>
        <title>Informe de libros y editoriales</title>
        <style>
            body {background-color:#FAFAFA}
        </style >
    </head>
    <body>
        <h1>Libros y editoriales</h1>
        <ol>
            <li>Visual C# (<a href="http://www.ra-ma.es">Ra-Ma</a>)</li>
            <li>Programación en C (<a href="http://www.mcgraw-hill.es">McGraw-Hill</a>)</li>
        </ol>
    </body>
</html>
```

2.8.4. Elemento <xsl:element>

El elemento <xsl:element> permite generar un nuevo elemento en el documento de salida. Mediante el atributo name se especifica el nombre del elemento a añadir; y con el atributo opcional use-attribute-sets el nombre de uno o varios conjuntos de atributos creados mediante el elemento <xsl:attribute-set>

-Ejemplo 19º.

Supongamos que a nuestro documento *bibloteca.xml* lo queremos modificar añadiéndole un nuevo elemento denominado <nombre> para almacenar el nombre de la biblioteca y algunos otros atributos. Dicho elemento será descendiente de <libreria> y estará por delante de todos los elementos <libro>.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="biblioteca">
        <xsl:copy>
            <xsl:element name="nombre" use-attribute-sets="identificacion">Pública de Maracena </xsl:element>
            <xsl:copy-of select ="libro"/>
        </xsl:copy>
    </xsl:template>
    <xsl:attribute-set name="identificacion">
        <xsl:attribute name="id">BP123-MA/GR</xsl:attribute>
        <xsl:attribute name="localidad">Marecena</xsl:attribute>
        <xsl:attribute name="cpostal">18200</xsl:attribute>
    </xsl:attribute-set>
</xsl:stylesheet>
```

La salida sería

```
<?xml version="1.0" encoding="utf-8" ?>
<biblioteca>
    <nombre id="BP123-MA/GR" localidad="Marecena" cpostal="18200">Pública de Maracena</nombre>
    <libro prestado="SI">
        <titulo>Visual C#</titulo>
        <autor>Fco. Javier Ceballos</autor>
        <editorial>Ra-Ma</editorial>
        <paginas>936</paginas>
        <precio>52,75</precio>
    </libro>
    <libro prestado="NO">
        . . .
    </libro>
</biblioteca>
```

Esta misma salida también se podría haber obtenido mediante la siguiente transformación:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="biblioteca">
        <xsl:copy>
            <xsl:element name="nombre">
                <xsl:attribute name="id">BP123-MA/GR</xsl:attribute>
                <xsl:attribute name="localidad">Marecena</xsl:attribute>
                <xsl:attribute name="cpostal">18200</xsl:attribute>
                Publica de Maracena
            </xsl:element>
            <xsl:copy-of select ="libro"/>
        </xsl:copy>
    </xsl:template>
  </xsl:stylesheet>
```

2.8.5. Elemento <xsl:comment>

Este elemento permite generar un comentario en el documento de salida. El comentario puede ser texto o bien datos obtenidos del documento XML a transformar.

2.8.6. Elemento <xsl:processing-instruction>

Permite generar instrucciones de procesamiento en el documento de salida. Esto es útil cuando un documento XML se procesa con una hoja de transformaciones en otro documento XML, el cual necesita de una segunda transformación.

-Ejemplo 20º.

Vamos a modificar la transformación anterior para que el documento de salida incluya una instrucción de procesamiento de tipo <?xml-stylesheet ... ?> y un comentario indicando el número de libros.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="biblioteca">
    <xsl:processing-instruction name="xml-stylesheet">
        type= "text/xsl" href="libros_2.xsl"
    </xsl:processing-instruction>
    <xsl:copy>
        <xsl:element name="nombre">
            <xsl:attribute name="id">BP123-MA/GR</xsl:attribute>
            <xsl:attribute name="localidad">Marecena</xsl:attribute>
            <xsl:attribute name="cpostal">18200</xsl:attribute>
            Publica de Maracena
        </xsl:element>
        <xsl:comment>
            Datos registrados de <xsl:value-of select ="count(/biblioteca/libro)"/> libros
        </xsl:comment>
        <xsl:copy-of select ="libro"/>
    </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

La salida sería

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet  type= "text/xsl" href="libros_2.xsl"?>
<biblioteca>
    <nombre id="BP123-MA/GR" localidad="Marecena" cpostal="18200">Publica de Maracena</nombre>
    <!--Datos registrados de 2 libros-->
    <libro prestado="SI">
        <titulo>Visual C#</titulo>
        <autor>Fco. Javier Ceballos</autor>
        <editorial>Ra-Ma</editorial>
        <paginas>936</paginas>
        <precio>52,75</precio>
    </libro>
    <libro prestado="NO">
        . . .
    </libro>
</biblioteca>
```

2.8.7. Elemento <xsl:variable>

Este elemento permite declarar variables asignándole un valor. Posteriormente el contenido de dicha variable se puede utilizar referenciándola o usando dicha variable. El sentido de variable no es como el que se tiene en los lenguajes de programación tradicionales. Estas variables siempre mantienen el valor que se le asignó inicialmente sin capacidad de modificación; es decir se comportan como constantes. Este elemento se puede definir de dos maneras:

```
<xsl:variable name = "nombreVariable" select = "valor"/>
```

En este caso nombreVariable es el nombre de la variable y valor el valor asignado. Por ejemplo:

```
<xsl:variable name = "color" select = "rojo"/>
```

La segunda manera es sin utilizar el atributo select, asignándose como valor a la variable el contenido del elemento:

```
<xsl:variable name = "nombreVariable">
   …algo que defina el valor de la variable
</xsl:variable>
```

Por ejemplo:

```
<xsl:variable name = "prestado">
   <xsl:value-of select = "@prestado"/>
</xsl:variable>
```

Cuando se necesite utilizar el valor asignado a la variable se usa la notación $nombreVariable, teniendo en cuenta que una variable sólo es visible dentro de la regla en que se definió.

-Ejemplo 21º.

Se va a obtener como salida un documento HTML donde aparezca el número total de páginas de todos los libros. Para generar el esquema de un documento HTML se define una variable plantillaHTML Además usaremos una variable suma cuyo valor a asignar será la suma del número de páginas de todos los libros.

Para realizar la suma, se utilizará la función sum() definida en XPath que permite sumar o acumular un conjunto de valores.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name = "plantillaHTML">
        <html>
            <head>
                <title>Informe</title>
            </head>
            <body>
                <xsl:apply-templates />
            </body>
        </html>
    </xsl:variable>
    <xsl:template match="/">
        <xsl:copy-of select="$plantillaHTML"/>
    </xsl:template>
    <xsl:template match="biblioteca">
        <xsl:variable name = "suma" select = "sum(libro/paginas)"/>
        Nº total de páginas de los libros: <xsl:value-of select = "$suma"/>
    </xsl:template>
</xsl:stylesheet>
```

La salida se presenta como:

Nº total de páginas de los libros 1671

2.9. Ejemplo resumen.

En esta transformación se presentan todos los datos de los libros de forma tabular.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
        <head>
            <title>Tabla ordenada de libros</title>
            <style>
                body {background-color:#FAFAFA}
                h1 {text-align:center}
                table {align:center; border:2px solid black}
                tr#cabecera {background-color:#FFFF00}
                td.centro {text-align:center}
            </style >
        </head>
        <xsl:apply-templates select="biblioteca" />
        </html>
  </xsl:template>
  <xsl:template match="biblioteca">
        <body>
            <h1>Listado de libros ordenados por número de páginas.</h1>
            <table>
                <tr id ="cabecera">
                    <th>Titulo</th>
                    <th>Autor</th>
                    <th>Editorial</th>
                    <th>Paginas</th>
                    <th>Precio</th>
                    <th>Prestado</th>
                </tr>
                <xsl:for-each select="libro">
                    <xsl:sort select="paginas" order="ascending" />
```

```
                <tr>
                    <td><xsl:value-of select="titulo"/></td>
                    <td><xsl:value-of select="autor"/></td>
                    <td><xsl:value-of select="editorial"/></td>
                    <td><xsl:value-of select="paginas"/></td>
                    <td><xsl:value-of select="precio"/></td>
                    <td class ="centro">
                        <xsl:if test="@prestado='SI'">
                            <strong style="color:#00FF00;"> <xsl:value-of select="@prestado"/></strong>
                        </xsl:if>
                        <xsl:if test="@prestado='NO'">
                            <strong style="color:#FF0000;"><xsl:value-of select="@prestado"/></strong >
                        </xsl:if>
                    </td>
                </tr>
            </xsl:for-each>
        </table>
    </body>
  </xsl:template>
</xsl:stylesheet>
```

La salida se presenta de la siguiente forma:

Listado de libros ordenados por número de páginas

Titulo	Autor	Editorial	Paginas	Precio	Prestado
Programación en C	Luis Joyanes Aguilar	McGraw-Hill	735	45,25	NO
Visual C#	Fco. Javier Ceballos	Ra-Ma	936	52,75	SI