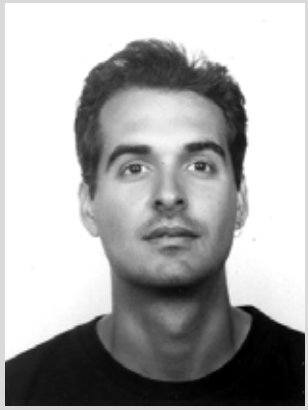
Artículos

Lenguajes de marcas aplicados a la transformación de estructuras documentales

Resumen: *Este artículo se ocupa de las tecnologías CSS (cascading style sheet) y Xslt (extensible style sheets language: transformations). Se trata de un trabajo de carácter práctico, sencillo y demostrativo más que un estudio teórico o exhaustivo. La idea que se pretende transmitir es que la unidad básica de gestión de información vuelve a ser el documento, con su marca de documento digital. La tecnología en la web se orienta hacia el control absoluto de su ciclo vital, funcionalidades y componentes estructurales. Este artículo presenta Xslt como una de las opciones más interesantes para controlar la estructura de los documentos digitales mediante transformaciones.*

Palabras clave: *CSS (cascading style sheet), Documento digital, DOM (document object model), Hojas de estilo, html (hypertext markup language), Internet Explorer, sgml (standard generalized markup language), XSL (extensible style sheets language), Xslt (extensible style sheets language: transformations), Xslf (extensible style sheets language: formatting), xml (extensible markup language).*



Antonio de la Rosa Piñero

Title: Markup languages applied to the transformation of documentary structures.

Abstract: *This article addresses CSS (cascading style sheet) and Xslt (extensible style sheets language: transformations) technologies, from a practical, straightforward and demonstrative perspective, rather than attempting to provide a theoretical or exhaustive study. The central idea is that the basic unit of information management again becomes the document, marked up as a digital document. Web technology is directed at an absolute control over its life-cycle, functionalities and structural elements. This article presents Xslt as one of the most interesting options for controlling digital documents' structure through transformations.*

Keywords: *CSS (cascading style sheet), Digital document, DOM (document object model), Style sheets, html (hypertext markup language), Internet Explorer, sgml (standard generalized markup language), XSL (extensible style sheets language), Xslt (extensible style sheets language: transformations), Xslf (extensible style sheets language: formatting), xml (extensible markup language).*

Rosa Piñero, Antonio de la. *"Lenguajes de transformación de estructuras aplicados al intercambio de documentación". En: El profesional de la información, 2001, enero-febrero, v. 10, n. 1-2, pp. 4-22.*

Introducción

¿De dónde viene la idea de tratar un documento digital como una estructura arbórea de nodos? En realidad se trata de la evolución lógica de una de las áreas de estudio clásicas de la informática: el desarrollo de aplicaciones para la generación y edición automática de documentos digitales. Este campo abarca un amplio espectro de programas de todo tipo (procesadores de textos sencillos, sistemas de ayuda basados en hipertexto, producción automática de bases de datos, edición, diseño, etc.) dirigidas a todos los tipos de usuarios: principiantes, profesionales o programadores.

Históricamente, la evolución del procesamiento de textos se puede esquematizar en los siguiente pasos: durante la década de los 60 el sistema utilizado era simple: tras generar el documento se aplicaba el formato deseado. Por lo general, la salida de este texto era impresa, y aquel tenía asociado, junto al dato propiamente dicho, la descripción deseada (llamada reproducción) que era convertida por el programa en una presentación. Algunas de las notaciones utilizadas en esta época para la reproducción siguen estando hoy en día en vigencia (con modificaciones, claro está): rtf (rich text format) o troff.

> **«Para gestionar información de forma eficaz es necesario contar con un sistema que permita estructurarla lógicamente»**

Poco tiempo después de que este sistema empezara a tener éxito, apareció el marcado de formato: se procedía a marcar el texto con una serie de etiquetas o códigos. Con la aparición de sistemas *wysiwyg* (*what you see is what you get*) se produjo una separación. Por un lado se desarrollaron lenguajes de etiquetas más complejos, por otro, se unió la potencia de los sistemas de reproducción a la capacidad de las nuevas interfaces. El producto final de esta opción son los actuales procesadores de texto (*AmiPro*, *Pagemaker*, *Word*, *WordPerfect*, etc.).

*Por **Antonio de la Rosa Piñero,** consultor e ingeniero de programación, Wisdom B. V., Assen, Holanda.*
antonio@wisdom.nl

Original recibido el 9-9-00
Aceptación definitiva: 26-10-00

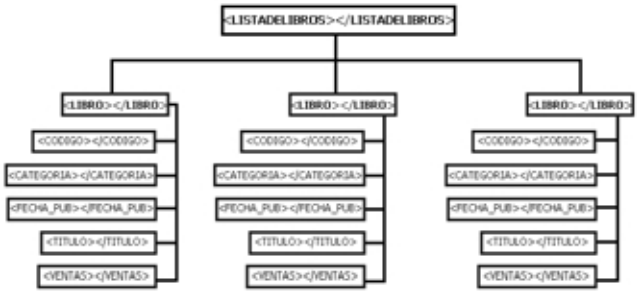


Figura 1

La proliferación de diversos formatos generó un problema: la información tenía diferentes maneras de representarse. Dentro de este contexto apareció, a finales de los 60, sgml. Lo interesante de esta evolución es constatar cómo en un momento determinado se produjo una separación entre presentación y estructuración.

El siguiente paso en la web ha sido asimilar la idea de que para gestionar información de forma eficaz es necesario contar con un sistema que permita estructurarla lógicamente, ya sea de manera tabular o arbórea, puesto que esa ordenación facilitará la automatización de las operaciones que se realicen sobre ella: recuperación, intercambio, integración, etc. Por otra parte, mantener una estructura de datos rígida dentro de un modelo abierto, como es el caso de la www, es un contrasentido.

Bibliografía recomendada

Cascading style sheets, level 2 Css2 specification. W3C recommendation, 12 de mayo, 1998.
http://www.w3.org/TR/REC-CSS2/

Cascading style sheets home page.
http://www.w3.org/Style/CSS/

Extensible markup language (xml) 1.0 (second edition). W3C recommendation, 6 de octubre, 2000.
http://www.w3.org/TR/2000/REC-xml-20001006

Francis, Brian, et al. IE5 dynamic html programmer's reference. Chicago (Illinois): Wrox Press Inc., 1999. Isbn 1861001746.

Kay, Michael. Xslt programmer's reference. Chicago (Illinois): Wrox Press Inc., 2000. Isbn 1861003129.

Xsl transformations (Xslt), version 1.0. W3C recomendation, 16 de noviembre, 1999.
http://www.w3.org/TR/xslt

De ahí el éxito del formato xml (extensible markup language), válido para la representación digital de documentos de cualquier tipo. Para trabajar con él es necesario estructurar la información. Éste es un proceso muy sencillo dentro del campo de la documentación, ya que la mayoría de "materiales" con los que se trabaja es susceptible de ser dividida en componentes. Así, un libro tiene título, capítulos, índices o un artículo resumen, apartados, subapartados, notas al pie, etc. Al mismo tiempo, cada una de estas partes tiene a su vez párrafos, frases, imágenes, etc. Xml denomina a todos ellos integrantes.

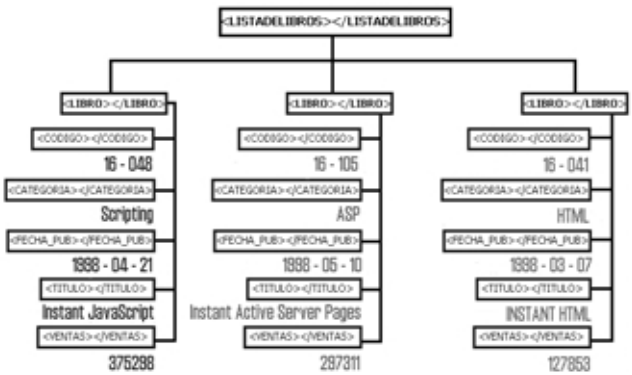


Figura 2

La siguiente tabla muestra una lista de libros formateada en xml:

```
<?Xml version="1.0"?>
<?xml-stylesheet type="text/css" href="listadelibros.css"?>
<listadelibros>
    <libro>
        <codigo>16-048</codigo>
        <categoria>scripting</categoria>
        <fecha_pub>1998-04-21</fecha_pub>
        <titulo>instant javascript</titulo>
        <ventas>375298</ventas>
    </libro>
    <libro>
        <codigo >16-105</codigo>
        <categoria>asp</categoria>
        <fecha_pub>1998-05-10</fecha_pub>
```

```
        <titulo>instant active server pages</titulo>
        <ventas>297311</ventas>
    </libro>
    <libro>
        <codigo>16-041</codigo>
        <categoria>html</categoria>
        <fecha_pub>1998-03-07</fecha_pub>
        <titulo>instant html</titulo>
        <ventas>127853</ventas>
    </libro>
</listadelibros>
```

Esto lo podríamos traducir gráficamente al árbol de etiquetas presentado en la figura 1. El cual, algo más desarrollado, genera el árbol de contenidos de la figura 2.

Figura 3

Éste es el estado de la cuestión actual por lo que se refiere a documentación digital en la www y xml. Los documentos se digitalizan en este formato de modo que puedan ser gestionados más fácilmente por aplicaciones específicas. Ahora bien, ¿cuál va a ser el siguiente paso en la evolución? Este artículo se centra en dicho tema. El próximo avance probablemente sea la gestión de información al nivel de elemento. Salvando las distancias se podría decir que, en un futuro próximo, un documento digital en la www se considerará como una pequeña base de datos y, para las aplicaciones que lo gestionen, la interacción con su contenido será algo parecido a la forma en que *SQL* interactúa con las bases de datos convencionales en la actualidad.

Una pista fundamental para basar esta aseveración es el hecho de que los lenguajes de programación tradicionales (aquellos que operan con la información) y los de etiquetas (los que la describen), se aproximan cada vez más a un territorio intermedio. Éste es el principio de *CSS* (*cascading style sheets*) y de *xslt* (*extensible style sheets language: transformations*), las dos especificaciones en las que se centra este estudio. Como se intentará demostrar, las implicaciones de su planteamiento de trabajo van mucho más allá de la simple presentación de datos.

CSS

El ejemplo bajo estas líneas es un documento html a partir del cual se van a introducir algunos conceptos sobre esta clase de hojas de estilo:

```
<!doctype html public "-//W3C//dtd html 4.0//en">
<html>
    <head>
        <title> Página html simple </title>
    </head>
    <body>
        <h1> Ejemplo html – CSS </h1>
        <p> Así se aplica CSS (cascading style sheets) </p>
    </body>
</html>
```

Para hacer que el contenido del elemento <*h1*> aparezca en azul, se podría usar la siguiente regla *CSS*:

```
h1 {color:blue}
```

Una regla *CSS* consta de dos componentes principales: un selector (’*h1*’) y una declaración (’*color:blue*’) la cual, a su vez, tiene dos partes: propiedad (’*color*’) y valor (’*blue*’). El código anterior podría verse como una hoja de estilo muy simple; combinada con otras (su característica fundamental[1] es que pueden combinarse) servirá para determinar la presentación final del documento.

La especificación html 4.0 define cómo se deben aplicar las reglas *CSS* a los documentos html: tanto dentro del mismo documento como mediante una hoja de estilo externa. Para incluirla internamente se usa el elemento <*style*>:

```
<!doctype html public "-//W3C//dtd html 4.0//en">
<html>
    <head>
        <title> Página simple html </title>
        <style type="text/css">
            body {color:red}
            h1 {color:blue}
        </style>
    </head>
        <body>
            <h1> Ejemplo html - CSS </h1>
            <p> Así se aplica CSS (cascading
style sheets) </p>
        </body>
</html>
```

Figura 4. Apariencia por defecto del documento book.xml en IE5

La hoja de estilo (en este caso se corresponde con el elemento <*style*>) contiene dos reglas: la primera hace que el elemento <*body*> adopte por defecto el color rojo mientras que la segunda hace posible que el contenido de <*h1*> aparezca en azul. No se especifica ningún color para <*p*> y por lo tanto heredará el color de su elemento padre (aquel en el cual está contenido): <*body*>. <*h1*> es también hijo de <*body*> pero en este caso se aplica la segunda regla de la hoja de estilo. ’*Color*’ es sólo una de las aproximadamente 100 propiedades incluidas en *CSS-2*. En el siguiente código se utilizan algunas otras:

```
<!doctype html public "-//W3C//dtd html 4.0//en">
<html>
    <head>
        <title> Página simple html </title>
        <style type="text/css">
            body {
                font-family:"Gill Sans" sans-serif;
                font-size:12pt ;
                margin:3em ;
                }
            h1 {color:blue}
        </style>
    </head>
    <body>
        <h1> Ejemplo html - CSS </h1>
        <p> Así se aplica CSS (cascading style sheets) </p>
    </body>
</html>
```

El primer punto a comentar es que hay varias declaraciones agrupadas en un bloque de código delimitado por llaves (*{...}*), separadas con (*;*), aunque la última puede aparecer también seguida por (*:*). La primera declaración asociada con el elemento <*body*> fija el tipo de letra en *gill sans*. Si no está disponible, el navegador usará *sans-serif*, que es uno de los genéricos que todos los navegadores "conocen". Los hijos del elemento <*body*> heredarán el valor de la propiedad *font-family* si no especifican los suyos propios. La segunda declaración fija el tamaño de la letra del elemento <*body*> en 12 puntos[2]. La tercera usa una unidad de dimensión relativa, la unidad *"em"*, y se refiere a la medida de la letra; en este caso, los márgenes alrededor de dicho elemento serán tres veces el tamaño de la letra (por lo tanto, 36 puntos).

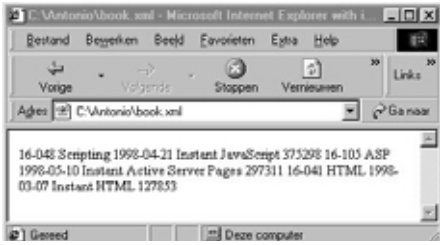


Figura 5. Book.xml asociado a la hoja de estilo Xslt expresada en el código anterior

1. Elementos-objeto (selectores). El siguiente ejemplo es un poco más complicado. El código define un estilo para todos los elementos <*p*>: 14 puntos, color verde, letra *arial*; y otro para todos los que presenten un atributo *"class*" con el valor *"miclase*": 12 puntos, color azul, letra *courier*:

```
<head>
    <style type="text/css">
        p          {font-family:Arial; font-size:14pt; color:green}
        .miclase   {font-family:Courier; font-size:12pt; color:blue}
    </style>
</head>
```

Como se ha visto antes, *CSS* permite definir un selector o elemento-objeto del estilo a aplicar, usando el nombre de dicho elemento, como se acaba de hacer con <*p*>. Es posible ser más precisos a la hora de especificar los selectores indicando su forma de anidamiento. En este otro ejemplo el estilo se aplica solamente a los elementos <*em*> anidados dentro de <*p*>:

```
p em        {font-family:Arial; font-size:14pt; color:green}
```

El siguiente código es ligeramente diferente, puesto que separa los nombres de los selectores con una coma. Con esta sintaxis, el estilo se aplicará a todos los elementos <*p*> y a todos los <*em*>:

```
p, em       {font-family:Arial; font-size:14pt; color:green}
```

2. Clases–objeto. El ejemplo anterior también definía un estilo para una clase llamada *miclase*. Para indicar que es una clase propietaria[3], se antecede el nombre con un punto:

```
.miclase        {font-family:Courier; font-size:12pt; color:blue}
```

Ahora se puede asignar este estilo a cualquier elemento usando el atributo (*class*) de la siguiente forma:

```
<p class="miclase">Texto</p>
```

a. Estilos dinámicos. Usar clases objeto, como en el ejemplo anterior, es una técnica muy popular cuando se utiliza *dhtml* (*dynamic html*). La razón es que un script puede cambiar el estilo de un elemento en respuesta a un determinado evento. Si se definen varias clases, sólo habrá que ajustar la propiedad *className* del elemento en cuestión a la clase apropiada usando el script. En el siguiente ejemplo se cambia la propiedad *className* de un elemento <*p*> en respuesta a un evento *onmouseover*:

```
    <style type="text/css">
        .grande{font-family:Arial; font-size:24pt; font-weight:bold}
        .peque{font-family:Courier; font-size:12pt; font-
weight:normal }
    </style>

<p class="peque"
id="miEncabez"onmouseover="hazGrande();"onmouseout="haz-
Peque();">Texto</p>

...
<script language="JScript">
    function hazGrande()
    {
        document.all('miEncabez').className = 'grande';
    }
    function hazPeque()
    {
        document.all('miEncabez').className = 'peque';
    }
</script>
```

b. Hojas de estilo enlazadas. El ejemplo anterior utiliza una sección <*style*> dentro de un documento html para definir la información relativa a su presentación. Sin embargo, también se puede vincular una hoja de estilo independiente con un documento html usando tanto el elemento <*link*> como la instrucción de estilo *@import*. Los siguientes ejemplos enlazan al documento html una hoja independiente llamada *miestilo.css*:

```
    <link rel="stylesheet" type="text/css" href="miestilo.css">
```

El elemento <*link*> especifica:

— El tipo de archivo con el que se enlaza: *"stylesheet"*.

— La ruta a ese archivo mediante el atributo *"href"*.

— El tipo de hoja de estilo: *"text/css"*.

```
<style type="text/css">
    @import url(miestilo.css);
</style>
```

Es siempre recomendable usar hojas de estilo externas por dos motivos fundamentales: pueden modificarse sin tocar el documento fuente html y es posible compartirlas entre diferentes documentos.

3. Usar *CSS* con documentos xml. A las aplicaciones documentales les resulta más interesante el uso que se pueda hacer de *CSS* a partir de documentos xml. ¿Cómo se trabaja con esto? No es posible usar bloques de código delimitados por el elemento <*style*> ya que no tiene sentido en xml. Sucede lo mismo con <*link*>; el navegador presentará el contenido de ambos como simple texto; al fin y al cabo éste es el planteamiento tras xml: cada elemento significa lo que su creador decida.

«Salvando las distancias se podría decir que, en un futuro próximo, un documento digital en la www se considerará como una pequeña base de datos»

a. La instrucción de proceso *xml-stylesheet.* La solución al problema es usar una instrucción de proceso xml. Este lenguaje define el elemento <*?xml...?*> como una instrucción de proceso que se usa para dar órdenes al parser o procesador xml. Una versión especial de esa instrucción de proceso se usa para enlazar documentos xml con las hojas de estilo que contienen la información sobre su presentación:

```
<?xml-stylesheet type="text/css" href="miestilo.css"?>
```

El parser xml buscará la hoja de estilo en la dirección especificada en el atributo *href* y la usará para dar formato al contenido del documento xml. También se pueden definir diferentes hojas de estilo usando varias instrucciones <*?xml-stylesheet..*>. En ese caso los estilos se fusionan y, en caso de duplicación, los últimos en aplicarse tienen precedencia.

b. Un ejemplo. El siguiente código muestra cómo una hoja de estilo *CSS* puede servir para controlar la presentación de un documento xml, llamado *book.xml*[4], que es una lista de tres libros:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="listadelibros.css"?>
<listadelibros>
    <libro>
        <codigo>16-048</codigo>
        <categoria>scripting</categoria>
        <fecha_pub>1998-04-21</fecha_pub>
        <titulo>instant javascript</titulo>
        <ventas>375298</ventas>
    </libro>
    <libro>
        <codigo >16-105</codigo>
        <categoria>asp</categoria>
        <fecha_pub>1998-05-10</fecha_pub>
        <titulo>instant active server pages</titulo>
        <ventas>297311</ventas>
    </libro>
    <libro>
        <codigo>16-041</codigo>
        <categoria>html</categoria>
        <fecha_pub>1998-03-07</fecha_pub>
        <titulo>instant html</titulo>
        <ventas>127853</ventas>
    </libro>
</listadelibros>
```

Se puede observar que la segunda línea es la instrucción de proceso que especifica la hoja de estilo llamada *listadelibros.css*. A continuación se muestra su contenido:

```
libro        {display:block; margin:15px}

codigo     {display:inline
        font-family:tahoma, arial, sans-serif;
        font size:10pt;
        font weight: bold}

categoria     {display:inline
        font-family:tahoma, arial, sans-serif;
        color:darkgray;
        font size:12pt;
        font weight: bold}

fecha_pub     {display:inline
        font-family:tahoma, arial, sans-serif;
        color:red;
        font size:10pt}

titulo     {display:inline
        font-family:tahoma, arial, sans-serif;
        font size:12pt;
        color:white;
        background-color:black}

ventas     {display:none }
```

Para cada uno de los elementos del documento fuente xml (excepto <*listadelibros*>), se especifica un conjunto de propiedades de estilo. Para el elemento <*libro*> se define como *display:block*. Por lo tanto, cada <*libro*> aparecerá formando un bloque[5] separado de los demás por un salto de línea. Esto es análogo a lo que se consigue con el elemento html <*div*>. También

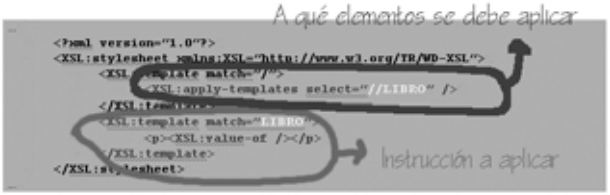


Figura 6

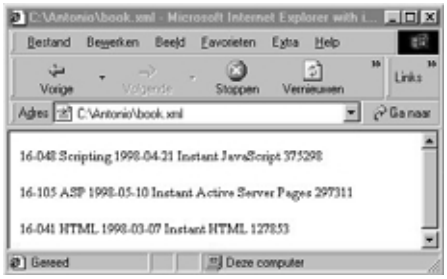


Figura 7

se ha incrementado el espacio entre elementos <*libro*> al fijar la propiedad *margin* en 15 píxeles.

<*codigo*>, <*categoria*>, <*fecha_pub*> y <*titulo*> presentan la propiedad *display:inline*, por lo que se presentan en la misma línea, algo similar a lo que se obtiene usando el elemento html <*span*>. Cada uno de ellos, sin embargo, posee diferentes propiedades de estilo y por eso aparecerán en diferentes tamaños y colores. Por otra parte, no se quiere presentar el contenido de los elementos <*ventas*> y por eso la propiedad *display:* para los cuales se fijó en *none*. La imagen de la figura 3 muestra el resultado de la hoja de estilo *listadelibros.css* actuando sobre el documento *book.xml:*

Xslt

En este apartado se cambia de tema: de *CSS* a *xslt*; de hojas de estilo para la presentación del documento fuente a otras para su transformación. Si se quisiera analizar *xslt* desde un punto de vista global, sería un tema demasiado amplio para un trabajo como éste. Por lo tanto, esta sección se centrará en su rendimiento práctico en el navegador *Internet explorer 5*. Lo que se pretende es, por un lado, ver cómo funciona en la práctica esta tecnología y tener muy en cuenta que uno de los posibles entornos de aplicación es una herramienta tan asequible y globalmente difundida como *IE5*; por otro, llegar a ciertas conclusiones que nos ayudarán a la hora de plantear posibles soluciones en el mundo de la documentación digital.

1. Introducción.

a. Transformar código xml para su presentación. La función de *xslt* se describe a menudo como: "transformar xml para su presentación". Lo que esto realmente significa es que no hay una forma normalizada de presentar elementos xml, ya que no poseen ningún significado intrínseco.

Lo que hace *xslt* es permitir que el autor, o el receptor del documento xml, puedan especificar cómo se debe presentar su contenido. Esto lo consigue al transformarlo en html o en un lenguaje equivalente que posea características de presentación normalizadas[6]. Generalmente los elementos xml se convertirán a html; en otras palabras, se especificará código html para cada elemento xml que será el encargado de producir la presentación deseada. Si los elementos creados son válidos en html y xml, el navegador los mostrará en pantalla. La necesidad de que todo el código cumpla con la sintaxis xml, sin dejar de ser html válido, hace que los elementos vacíos como <*hr*> deban presentar su propio cierre (<*hr*></*hr*> o <*hr*/>). También es posible incluir secciones <*script*> en el código que serán presentadas como html y después ejecutadas por el navegador.

Lo más importante es la posibilidad de usar *xslt* para crear nuevos elementos o eliminar otros incluidos en el documento fuente. Esto significa que *xslt* no está limitado al modelo *CSS* de aplicación de estilo a elementos existentes. *Xslt* presenta la posibilidad de copiar elementos desde documento fuente al nuevo, usar estructuras de decisión para presentar sólo ciertos elementos preseleccionados, crear nuevos u ordenar el contenido del documento fuente de forma diferente a la original antes de presentarlo.

b. ¿Qué hace *xslt*? El principio básico de *xslt* en *IE5* es que actúa como un "procesador" para transformar cualquier documento xml[7] en otro diferente. Sin embargo, las hojas de estilo *xslt* están escritas en xml como cualquier otro tipo de documento. Por lo tanto, la entrada de datos del proceso *xslt* puede ser perfectamente una hoja de estilo *xslt* en lugar de un documento xml convencional[8], según se ve en la figura 12.

El resultado del proceso es otro documento, pero éste podría asumir cualquier formato. La forma en que *xslt* procesa los datos permite que un nodo xml pueda ser transformado en casi cualquier tipo de cadena de texto, incluyendo ascii. La gran ventaja de este planteamiento es que proporciona una técnica automática de conversión entre diferentes tipos de datos o diferentes formatos.

¿Qué ocurre cuando se lee un documento xml con *IE5*? Lo que se obtiene en pantalla es el resultado del trabajo de una hoja de estilo *xslt*. *IE5* contiene una que se utiliza por defecto cuando se carga un documento xml que no especifica otra hoja. Su propósito es mostrar el documento de una forma más legible y obvia (desde la perspectiva xml). Se podría decir que *IE5*

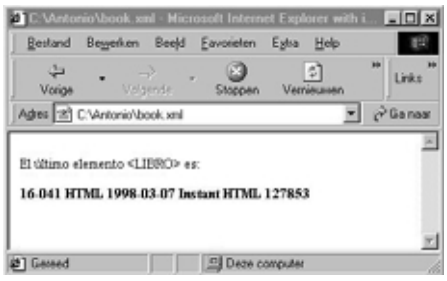


Figura 8

"entiende" xml puesto que añade saltos de línea, indentaciones y signos + y – delante de los nodos con subelementos o los que se encuentran vacíos. Lo que ocurre es que *IE5* está aplicando su hoja de estilo por defecto al documento xml para producir un resultado como el que se ve en la imagen de la figura 4.

c. ¿Cómo se trabaja con *xslt*? En términos generales, *CSS* relaciona la información sobre el estilo con los elementos donde lo va a aplicar especificando el nombre de estos elementos (como *<p>* o **) o usando una clase previamente definida (como *.miestilo*) que se aplica a un elemento utilizando su atributo *class*.

Xslt es completamente diferente. Usa instrucciones de proceso especiales llamadas *templates* para especificar los nodos–objeto[9] a los que se les aplicará un conjunto de instrucciones de estilo. Los *templates* declaran un patrón o un filtro que pueden asociar con mucha precisión la información de estilo que contienen con elementos o grupos de ellos específicos. Los elementos-objeto no deben tener el mismo nombre o contener atributos especiales. Hay muchas formas de especificar un patrón, de forma que se seleccionen sólo los apropiados en el documento fuente xml.

d. *Xslt* en el futuro. Aunque es imposible estar absolutamente seguro de lo que va a ocurrir con *xslt*, las recomendaciones actuales del *W3C* dan valiosas pistas al respecto. Actualmente las transformaciones realizadas con esta tecnología son sobre todo xml en html para que el contenido pueda ser presentado en el navegador. Hay planes para extender sus posibilidades y así permitir conversiones a otros lenguajes, particularmente aquellos mejor adaptados a funciones como resultado impreso, hablado, etc.

El borrador del *W3C* para *xslt* también propone técnicas para dar formato a la información usando "objetos de formato" o de "flujo" y un vocabulario de formato. Estos temas están en constante evolución, pero probablemente resultarán en una nueva especificación *xslfo* (extensible style sheets: formatting) que se ocupará del estilo en profundidad mientras *xslt* se concentra en las transformaciones del contenido.

2. Enlazar hojas de estilo *xslt* con documentos xml. Una hoja de estilo *xslt* es simplemente un archivo de texto, igual que una del tipo *CSS*. Se puede vincular un documento xml con una *xslt* usando una instrucción de proceso de forma similar a lo que se hizo anteriormente para unirlo con una *CSS*:

```
<?xml-stylesheet type="text/xsl" href="miestilo.xsl"?>
```

Sólo es posible asociar una hoja de estilo *xslt* a un documento; si se definen más de una, sólo la primera será interpretada y ejecutada. Si se especifican dos hojas de estilo, una *xslt* y otra *CSS*, *IE5* elegirá el primer formato. Otra circunstancia a tener en cuenta es que las hojas de estilo *xslt* sólo pueden cargarse desde el mismo dominio del documento xml al que están asociadas. Esto es así por motivos de seguridad, ya que pueden contener código script.

Figura 9

3. *Templates xslt*. Como ya se ha dicho, una hoja de estilo *xslt* se compone de una o más instrucciones especiales llamados *templates*. Cada uno especifica un patrón (o un filtro) que se utiliza para emparejar el *template* con el/los nodo/s objeto sobre los que actuará. También contienen información sobre cómo deben ser presentados.

a. Una hoja de estilo *xslt*. Todas las instrucciones en una hoja de estilo *xslt* están escritas en sintaxis xml y se podrían considerar "elementos" *xslt*. En realidad son elementos xml ordinarios con el prefijo *"xsl:"*. El *namespace*[10] definido para ese prefijo asigna significados especiales a cada uno de esos elementos. El siguiente código muestra una hoja de estilo *xslt* sencilla que contiene un solo *template*. Obsérvese que el *namespace* debe especificarse tal y como aparece en el ejemplo para que pueda ser procesado por *IE5*:

```
<?xml version="1.0"?>[11]
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <xsl:value-of />
   </xsl:template>
</xsl:stylesheet>
```

El código anterior presentará el texto del documento fuente *book.xml* exactamente como está, con el tipo de letra por defecto del navegador. El código tiene un elemento raíz: *<xsl:stylesheet>*, y dentro de él un sólo elemento *template*: *<xsl:template>*, que especifica un atributo de asociación *"match"* con el patrón *"/"* como valor, lo que significa que ese *template* se emparejará con el elemento raíz de *book.xml*. Finalmente, el contenido del *template* (la acción) es la instrucción *xslt <xsl:value-of>*. Este elemento le dice al procesador: "considera como texto el contenido del nodo objeto y sus descendientes e insértalo en el documento resultado". Lo que se obtiene puede verse en la figura 5.

La clave del funcionamiento de las hojas de estilo *xslt* es la forma en que los *templates* se asocian con los elementos del documento fuente xml a través del patrón especificado en el atributo *"match*=" del elemento <*xsl:template*>. *Xslt* utiliza la recursión para aplicar a los elementos–objeto las instrucciones de estilo o transformación especificadas dentro de <*xsl:template*> (en este caso la instrucción es simplemente <*xsl:value-of* />).

El ejemplo anterior usaba el elemento <*xsl:template*>:

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

Aquí se especifica el patrón "/" mediante el cual se selecciona el nodo raíz del documento fuente xml. De modo que este *template* se emparejará unívocamente con ese elemento. La instrucción dentro del *template* era el siguiente elemento *xslt*:

```
<xsl:value-of />
```

Esto provoca que el procesador *xslt* inserte en el resultado tan sólo el texto del nodo seleccionado y sus descendientes. En este caso es toda la cadena de caracteres del contenido del documento fuente, puesto que el nodo raíz contiene al resto de los nodos. Se pueden especificar diferentes *templates* para cada tipo de nodo usando la instrucción <*xsl: apply-templates*> con los patrones adecuados:

Figura 10

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

La forma más simple de seleccionar un nodo es referenciándolo por su nombre —lo que asegura que la acción se llevará a cabo sobre todos los que tengan ese nombre—, lo cual, en la mayoría de los casos, es lo que se pretende conseguir. Sin embargo, la técnica mencionada (<*xsl: apply-templates*>) permite asociar *templates* a los elementos hijo de la raíz, seleccionándolos mediante el patrón, especificado en el atributo *"select*". Después, sucesivamente, se pueden añadir más instrucciones <*xsl: apply-templates*> a esos nuevos *templates* para relacionarlos con los elementos hijo de los elementos anteriores y así sucesivamente, llegando a todos los niveles de profundidad del árbol documental.

«Es siempre recomendable usar hojas de estilo externas por dos motivos fundamentales: pueden modificarse sin tocar el documento fuente html y es posible compartirlas entre diferentes documentos»

En la hoja de estilo anterior un solo *template* se asociaba con el nodo raíz del documento fuente xml[12], y la acción consistía en aplicarlo (y las instrucciones anidadas en él) a los elementos descendientes del nodo raíz. Mediante la recursión, los *templates* de la hoja de estilo se asocian a diferentes elementos del documento fuente xml, comenzando por el nivel superior (el elemento raíz) y recorriendo el árbol del documento hacia abajo hasta localizar todos los elementos seleccionados. Para ejemplificar este funcionamiento se puede añadir un nuevo *template* a la hoja de estilo anterior:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/">
            <xsl:apply-templates select="//libro" />
        </xsl:template>
        <xsl:template match="libro">
            <p><xsl:value-of /></p>
        </xsl:template>
    </xsl:stylesheet>
```

Lo que se ha hecho es sustituir el elemento <*xsl:value-of*> en el *template* original con <*xsl:apply-templates select="//libro"*>, que comunica al procesador *xslt* que debe aplicar el *template* asociado con el patrón *//libro* a cada elemento <*libro*> en el documento fuente xml (el patrón *//libro* se asocia con todos <*libro*> a cualquier nivel por debajo del nodo raíz). El nuevo *template* que se ha añadido a la hoja de estilo se asocia con *//libro* y las instrucciones anidadas dentro de él se ejecutarán para cada <*libro*>. Quizás se ve un poco más claro en la imagen de la figura 6.

En el *template* rodeado por la línea roja se usa de nuevo el elemento <*xsl:value-of*> para dar formato al resultado de cada elemento <*libro*>. Se puede observar que, en este caso, se rodea el elemento <*xsl:value-of*> con la etiqueta html <*p*><*/p*>, con lo que cada <*libro*> (y sus descendientes[13]) aparecerá en un párrafo aparte. En la imagen de la figura 7 se puede apreciar el resultado.

b. Sintaxis *IE5-xslt* de asociación de patrones. El atributo *"match"* del elemento <*xsl:template*> proporciona varias posibilidades a la hora de especificar exactamente qué nodos (elementos o atributos) del documento fuente xml deben ser procesados por un *template* particular. La primera hoja de estilo usaba *"/"* para referirse al nodo raíz, pero hay muchas más posibilidades. Básicamente hay dos formas de especificar nodos:

— Mediante la posición del nodo o grupo de nodos en el árbol jerárquico del documento fuente.

— A través de la aplicación de un filtro que seleccione uno o varios nodos.

b.1. Operadores de posición para *xslt* e *IE5*. Para seleccionar un nodo o nodos usando su posición en el árbol del documento fuente se usa un conjunto de operadores de posición o ruta; combinándolos se obtiene un patrón que, como se ha visto, se utiliza para escoger el nodo o nodos apropiados junto a sus descendientes sobre los que posteriormente se ejecutarán las acciones previstas en el *template*. Cuando no se especifica ninguno, el nodo "corriente" será la raíz del documento fuente. Los operadores de posición son los de la tabla 1, mientras que los ejemplos de la tabla 2 ayudan a comprender esta sintaxis.

Figura 11

Los operadores de posición siempre devuelven todos los elementos que coinciden con el patrón. La función *index()* puede usarse para especificar un elemento particular dentro del conjunto de aquellos recuperados mientras que *end()* define el último nodo (tabla 3).

Por ejemplo, para retornar los detalles del último elemento <*libro*> del documento fuente xml, se usaría la siguiente hoja de estilo:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <xsl:apply-templates select="//libro" />
    </xsl:template>
    <xsl:template match="libro[end()]">
        El último elemento <libro> es:<p><b><xsl:value-of
/></b></p>
    </xsl:template>
</xsl:stylesheet>
```

La cual, actuando sobre el documento fuente: *book.xml* daría en el navegador lo que se ve en la figura 8. Obsérvese que, como muchos operadores, la función *end()* sólo puede aplicarse a nodos que son hijos del mismo padre. Por ejemplo, para encontrar el elemento <*titulo*> del último elemento <*libro*> tendríamos que usar:

```
listadelibros/libro[end()]/titulo
```

Si utilizáramos:

```
listadelibros/libro/titulo[end()]
```

obtendríamos el último elemento <*titulo*> del primer <*libro*>.

b.2. Operadores de filtro para *xslt* e *IE5*. Un filtro *xslt* tiene la forma: *[operador patrón]*, donde *operador* es un filtro opcional que define cómo se va a aplicar el patrón y *patrón*, es el filtro *xslt* que selecciona uno o varios elementos en el documento fuente basándose en diferentes criterios. Es posible elegir elementos basándose en:

— La existencia de elementos hijos.

— El valor de un elemento o de un elemento hijo.

— La presencia de atributos.

— El valor de éstos.

— Cualquier combinación de las posibilidades anteriores.

El siguiente código, por ejemplo, usa el filtro *xslt [titulo]* para seleccionar solamente los elementos <*categoria*> de libros que tengan elementos hijo <*titulo*> (por lo tanto no se seleccionarán elementos <*libro*> sin hijos <*titulo*>):

```
libro[titulo]/categoria
```

Para encontrar todos los libros que tengan hijos <*titulo*> y <*categoria*> se usarán dos filtros:

```
libro[titulo][categoria]
```

Para seleccionar un elemento por su valor (por ejemplo, elementos <*libro*> que contengan un elemento <*categoria*> cuyo valor sea *"Scripting"*) se podría usar el siguiente código:

```
libro[categoria = 'Scripting']
```

Hay, además del =, varios operadores de comparación que pueden usarse:

— *libro[ventas > 10000].*

— *libro[fecha_pub <= '1998-02-07'].*

— *libro[titulo != 'Instant JavaScript'].*

— *libro[titulo ine 'Instant JavaScript'].*

El último ejemplo usa un operador de comparación "no igual a" insensible a mayúsculas o minúsculas. Es diferente del anterior, que sí tiene en cuenta esa diferencia. Un filtro puede usar también el operador @ para especificar un atributo del elemento actual. El siguiente, por ejemplo, especifica que el elemento <*libro*> debe tener un atributo *fecha_impr*:

```
libro[@fecha_impr]
```

Cuando un filtro especifica el valor de un atributo, puede usar * para definir cualquier nombre de atribu-

to. Con este, por ejemplo, sólo se seleccionan elementos <*autor*> con un atributo tipo cuyo valor sea *"contrato"*:

```
,    autor[@tipo = 'contrato']
```

El siguiente filtro obtiene elementos <*autor*> en los que cualquiera de sus atributos tenga un valor *"contrato"*:

```
autor[@* = 'contrato']
```

Finalmente, es posible construir filtros más complejos usando operadores booleanos para combinar otros simples. Los disponibles son: *and*, *or* y *not*. En el siguiente código se seleccionan todos los elementos <*libro*> que tengan un elemento hijo <*categoria*> con el valor *'Scripting'* pero no uno con el valor *'html'*:

```
libro[categoria = 'Scripting' $and$ categoria != 'html']
libro[categoria = 'Scripting' $and$ $not$ categoria = 'html']
```

b.3. Coincidencia de varios patrones con un sólo elemento. Si hay más de un *template* en una hoja de estilo, es fácil que algunos elementos del documento fuente estén asociados a más de un patrón en los *templates*. Para impedir la duplicación en el resultado, el procesador *xslt* no permitirá que se aplique más de un *template* a cada elemento y decide cuál debe usarse comprobando todos los que tiene asociados y eligiendo el que mejor se corresponda con él. Sin embargo, es mejor evitar estas situaciones. Un ejemplo: un elemento <*autor*> hijo de un elemento <*libro*>, podría relacionarse a varios *templates* diferentes: *"libro/autor"*, *"libro/*"* o *"*"*. En este caso el procesador escogería el primero porque se trata de la asociación más restrictiva, es decir, la que mejor se corresponde con el elemento <*autor*>.

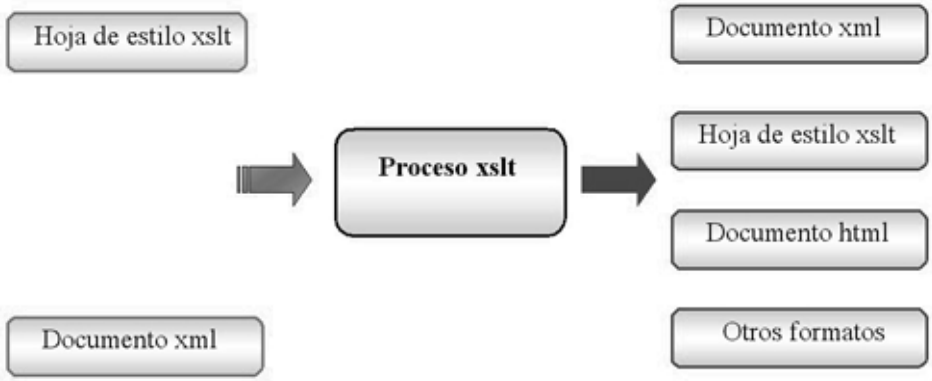


Figura 12

3. Elementos *xslt* básicos. *IE5* soporta varios elementos o instrucciones *xslt* que pueden usarse en una hoja de estilo *xslt*. Los fundamentales son:

— <*xsl:stylesheet*>.

— <*xsl:template*>.

— <*xsl:define-template-set*>.

— <*xsl:apply-templates*>.

— <*xsl:copy*>.

— <*xsl:value-of*>.

1. <*xsl:stylesheet*>. Es el elemento raíz de una hoja de estilo *xslt* y, por lo tanto, contiene a los demás elementos. En él se puede especificar el lenguaje de script usado en la hoja, si se utiliza alguno, y si necesitan preservarse o no los espacios blancos del documento fuente en el que se da como resultado. Además contiene una declaración de *namespace* para el prefijo xsl. La sintaxis es:

```
<xsl:stylesheet    xmlns:xsl="http://www.w3.org/TR/WD-xsl"
          indent-result="yes"
          language="JavaScript1.2">

</xsl:stylesheet>
```

El *namespace* (*xmlns*) debe teclearse exactamente como en el ejemplo para que pueda ser leído por *IE5*. El atributo *"indent-result"* con valor positivo hace que los espacios en blanco sean preservados en el documento resultado.

2. <*xsl:template*>. Define un *template* específico dentro de una hoja de estilo *xslt*. Utiliza el atributo *"match"* cuyo valor es un patrón de asociación dado para especificar los elementos en el documento fuente xml que serán seleccionados y procesados por la hoja de estilo. También acepta un atributo para especificar el lenguaje script que se va a usar en el *template*. La sintaxis es:

```
<xsl:template match="patrón" language="VBScript">
   ...
</xsl:template>
```

3. <*xsl:define-template-set*>. Este elemento se usa para agrupar un conjunto de *templates* que tienen un ámbito diferente al del resto de la hoja de estilo. Puede usarse dentro de los elementos <*xsl:stylesheet*> o <*xsl:template*>. El siguiente código define dos *templates* que sólo se aplicarán cuando se use el elemento (<*xsl:template match="*"*>) en el que se hallan anidadas:

```
<xsl:template match="*">
   <xsl:define-template-set>
      <xsl:template match="libro">...</xsl:template>
      <xsl:template match="autor">...</xsl:template>
   </xsl:define-template-set>
</xsl:apply-templates>
```

Operador	Descripción
/	Es el operador de posición que identifica a la descendencia. Selecciona elementos que son descendientes directos del nodo especificado. Es similar al sistema usado para especificar direcciones en un URL, por ejemplo: libro/categoria elige todos los elementos <categoria> descendientes de <libro>. Para definir el nodo raíz se sitúa este operador al principio del patrón de la siguiente forma: /libro/categoría.
//	Indica descendencia recursiva. Selecciona todos los nodos especificados a cualquier nivel de profundidad bajo el nodo corriente. Por lo tanto listadelibros//titulo seleccionaría todos los elementos <titulo> que sean descendientes a cualquier nivel del elemento <listadelibros>. Cuando este operador aparece al comienzo del patrón: //titulo obtiene todos los elementos <titulo> que sean descendientes del nodo raíz (por lo tanto todos los elementos <titulo>).
.	Es el operador de contexto corriente. Se usa para indicar el nodo corriente o "contexto". De esta forma .//titulo seleccionará todos los elementos <titulo> a cualquier nivel por debajo del nodo corriente. La combinación ./ siempre indica el contexto actual y normalmente es superflua: ./libro/categoria es lo mismo que libro/categoria.
@	Identifica la posición de los atributos. Indica que esa sección del patrón se está refiriendo a atributos del elemento corriente. Así libro@fecha_impr seleccionará el valor del atributo "fecha_impr" para cada uno de los elementos <libro>.
*	El asterisco es un operador comodín que se usa cuando se desean seleccionar todos los elementos o atributos sin tener en cuenta sus nombres. Por ejemplo libro/* seleccionara todos los hijos de todos los elementos <libro>, libro/@* obtendrá todos los atributos de esos mismos elementos.

Tabla 1

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

4. <*xsl:apply-templates*>. Este elemento le indica al procesador que debe buscar y ejecutar los *templates* asociados a los elementos hijos del elemento actual. Si hay un atributo *"select*" cuyo valor sea un patrón de asociación, los elementos seleccionados según ese patrón serán los que se procesen. También acepta el atributo *"order-by*" que determina el orden en que se presentarán dichos elementos. El valor de este atributo es una lista de patrones separados por *";"* cada uno de los cuales puede ser precedido por "+" o "-" para indicar orden ascendente o descendente. Los patrones en la lista se procesarán en relación con aquel dado en el atributo *"select*". La sintaxis es:

```
<xsl:apply-templates select="patrón" order-by="lista de patrones"/>
```

Para confeccionar una lista de libros ordenados alfabéticamente por el nombre del autor, se usaría el siguiente código:

```
<xsl:apply-templates      select="libro"
                  order-by="+autor/apellido; +autor/nombredepila">
```

5. <*xsl:copy*[14]>. Simplemente copia los elementos seleccionados al resultado como están, es decir, incluyendo sus etiquetas. Ésta es la diferencia respecto de <*xsl:value-of..*> que sólo copiaba el contenido de los elementos elegidos:

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

6. <*xsl:value-of*>. Devuelve el valor de un nodo xml (y de sus descendientes si los tiene) en forma de texto que se inserta en el resultado. Se puede usar opcionalmente un patrón (como valor del atributo *"select*") para seleccionar uno o varios elementos en el documento fuente. Si se omite el atributo *"select*" se optará por el elemento actual en el documento fuente. Si el patrón se asocia con más de un elemento, sólo se procesará el primero de ellos. Si el elemento tiene descendientes, sus valores se concatenarán en el resultado. La sintaxis es:

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

e. Generar html mediante *xslt*. Los elementos *xslt* que se han comentado hasta ahora son suficientes para construir hojas de estilo de una complejidad razonable y con capacidad semejante a las construidas con *CSS*. Se ha visto cómo *xslt* procesa los documentos xml emparejando los *templates* de la hoja de estilo con los elementos del documento fuente y generando a partir de ahí un documento resultante en el que los elementos originales han sido simplemente copiados o bien procesados hasta cierto punto. La mayoría de los ejemplos anteriores generaban como resultado código html que podía ser gestionado por el navegador de la forma usual.

Para ver mejor cómo puede usarse *xslt* para dar formato a un documento xml, en los siguientes ejemplos se va a definir una hoja de estilo *xslt* que, asociada al archivo *book.xml*, producirá el mismo resultado que la hoja de estilo *CSS* que se usó al principio. En la primera sección de la hoja de estilo *xslt* se puede ver que el *template* raíz (cuyo atributo *"match"* tiene el valor *"/"*) contiene código html con el que se creará la estructura básica del documento resultante. Dentro del html hay un elemento <*xsl:apply-templates...*> que selecciona los elementos <*libro*> del documento fuente:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
    <html>
        <body>
            <xsl:apply-templates select="//libro" />
        </body>
    </html>
</xsl:template>
```

La hoja de estilo anterior transformará *book.xml* en el siguiente documento html:

```
<html>
    <body>
        resultado del proceso del primer elemento <libro>
        resultado del proceso del segundo elemento <libro>
        resultado del proceso del tercer elemento <libro>
        etc.
    </body>
</html>
```

Al procesar el *template* raíz, el procesador *xslt* encuentra el patrón *"//libro"* y busca todos los elementos <*libro*> bajo la raíz en el documento fuente. Al encontrarlos, busca en la hoja de estilo un *template* que se asocie con ellos, como el siguiente:

```
<xsl:template match="libro">
    <span style="display:block; margin:15px">
        <xsl:apply-templates select="codigo" />
        <xsl:apply-templates select="categoria" />
        <xsl:apply-templates select="fecha_pub" />
        <xsl:apply-templates select="titulo" />
    </span>
</xsl:template>
```

Este *template* simplemente inserta un elemento html <*span*> para cada elemento <*libro*>, dentro del cual se incluyen las referencias a los *templates* asignados a cada uno de los elementos hijos de <*libro*>: <*codigo*>, <*categoria*>, etc.

Las propiedades de estilo usadas con el elemento <*span*> son exactamente las mismas que se utilizaron en la hoja de estilo *CSS*. El resultado es simplemente que el contenido de cada elemento <*libro*> aparecerá en un bloque aparte, dado que se ha vuelto a usar la propiedad *display:block*. Lo único que falta para completar la hoja de estilo *xslt* es, por lo tanto, añadir el formato que deseemos a los *templates* asignados a los elementos: <*codigo*>, <*categoria*>, <*fecha_pub*> y <*titulo*>. Para <*categoria*>, por ejemplo, podría ser el siguiente:

```
<xsl:template match="categoria">
    <span style="
display:inline;
            font-family:Tahoma, Arial,sans-serif;
            color:darkgray;
font-size:12pt;
            font-
weight:bold">
        <xsl:value-of />
    </span>
</xsl:template>
```

De nuevo, es un simple elemento <*span*> dentro del cual se inserta el contenido del elemento <*categoria*> (en el documento fuente *book.xml*) mediante el uso de la instrucción <*xsl:value-of..*>. El estilo especificado es idéntico al que se usó en la hoja de estilo *CSS*. Los otros tres *templates*, para los elementos <*codigo*>, <*fecha_pub*> y <*titulo*>, siguen exactamente el mismo criterio. La hoja de estilo *xslt* completa es:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/tr/wd-xsl">

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

<xsl:template match="libro">
    <span style="display:block; margin:15px">
        <xsl:apply-templates select="codigo" />
        <xsl:apply-templates select="categoria" />
        <xsl:apply-templates select="fecha_pub" />
        <xsl:apply-templates select="titulo" />
    </span>
</xsl:template>

<xsl:template match="codigo">
    <span style="    display:inline;
                font-family:tahoma, arial,sans-serif;
                font-size:10pt;
                font-weight:bold">
        <xsl:value-of />
    </span>
</xsl:template>

<xsl:template match="categoria">
    <span style="    display:inline;
                font-family:tahoma, arial,sans-serif;
                color:darkgray; font-size:12pt;
                font-weight:bold">
        <xsl:value-of />
    </span>
</xsl:template>

<xsl:template match="fecha_pub">
    <span style="    display:inline;
                font-family:tahoma, arial,sans-serif;
                color:red;
                font-size:10pt">
```

Patrón	Resultado
/	Solamente el nodo raíz.
libro/autor	Elementos <autor> hijos de elementos <libro>.
//	El nodo raíz y todos los nodos bajo el.
//*	Todos los elementos bajo el nodo raíz.
libro//autor	Elementos <autor> descendientes a cualquier nivel de elementos <libro>.
.//autor	Elementos <autor> descendientes a cualquier nivel del elemento actual.
*	Todos los elementos excepto el nodo raíz.
libro/*	Todos los elementos hijos de <libro>.
libro//*	Todos los elementos descendientes a cualquier nivel de <libro>.
libro/*/autor	Elementos <autor> nietos de elementos <libro>.
libro/@fecha_impr	Atributos "fecha_impr" asociados a elementos <libro>.
*/@fecha_impr	Atributos "fecha_impr" asociados a cualquier elemento.

Tabla 2

```
        <xsl:value-of />
    </span>
</xsl:template>

<xsl:template match="titulo">
    <span style="     display:inline;
                font-family:tahoma, arial,sans-serif;
                font-size:12pt;
                color:white;
                background-color:black">
        <xsl:value-of />
    </span>
</xsl:template>

</xsl:stylesheet>
```

Y el resultado en el navegador es idéntico al conseguido con *CSS* (figura 9).

Una diferencia sutil que no puede apreciarse en el ejemplo es que con *CSS* los elementos se presentan en el orden en que están dispuestos dentro del documento fuente xml, mientras que con *xslt* aparecen según están dispuestos en los elementos <*xsl:apply-templates...*> en la hoja de estilo *xslt*. Después de haber comprobado que *xslt* es capaz de llevar a cabo todo lo que *CSS* puede hacer, sería interesante ver si también es posible realizar otras que *CSS* no puede realizar. La forma más coherente de comprobarlo es llevando el ejemplo anterior un poco más lejos. La imagen de la figura 10 es el resultado de aplicar una nueva hoja de estilo *xslt* al mismo documento *book.xml*. Es fácil apreciar que se trata de un resultado completamente diferente.

«El principio básico de *xslt* en *IE5* es que actúa como un 'procesador' para transformar cualquier documento xml en otro diferente»

Aunque parezca muy distinto, básicamente sigue tratándose de la transformación de un documento xml en html. Sin embargo se usan algunos trucos como añadir nuevos elementos al resultado o cambiar el orden de presentación de los elementos xml originales. Ahora <*categoria*> aparece en primer lugar seguido de <*titulo*>. Esto podría haberse hecho usando las propiedades de posicionamiento absoluto en *CSS*, pero lo que ocurre en esta página es que los elementos están en celdas de una tabla html. Además los libros aparecen ordenados alfabéticamente. ¿Qué ha cambiado en la hoja de estilo?

d.1. La hoja de estilo del ejemplo anterior puede dividirse en tres partes.

— Primera parte:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
```

Patrón	Resultado
/listadelibros/libro[0]	Primer elemento <libro> dentro del elemento <listadelibros>.
/libro/autor[2]	Tercer elemento <autor> dentro del elemento <libro>.
/listadelibros/libro[end()]	Último elemento <libro> dentro del elemento <listadelibros>.

Tabla 3

```
    <body>
        <div style="     font-family:Tahoma,Arial,sans-serif;
                    font-size:24pt; color:green;
                    text-align:center; letter-spacing:8px;
                    font-weight:bold">
            Una lista de libros para estudiar
            </div>
        <hr/>
        <table width="100%" cellpadding="5">
            <xsl:apply-templates select="//libro" order-by="+cate-
goria" />
        </table>
        <div style="     font-family:Arial,sans-serif;
                    font-size:8pt; margin-left:10px">
            2000 Antonio de la Rosa
        </div>
    </body>
</html>
</xsl:template>
```

Para el encabezamiento se usa un elemento html <*div*> al que se aplican las propiedades de estilo apropiadas. Después una línea horizontal <*hr*/>. A continuación se incluye una tabla donde se inserta la instrucción <*xslt:apply-templates...*> encargada de seleccionar los elementos <*libro*> del documento fuente. A la tabla le sigue otro elemento <*div*> que contiene una nota al pie. En otras palabras, se crea una tabla html que va a contener el contenido del documento xml *book.xml*. Para hacer que cada elemento <*libro*> se inserte en una fila separada en la tabla, se añade al código anterior lo siguiente.

— Segunda parte:

```
<xsl:template match="libro">
    <tr>
        <xsl:apply-templates select="categoria" />
        <xsl:apply-templates select="titulo" />
        <xsl:apply-templates select="codigo" />
    </tr>
    <tr>
        <xsl:apply-templates select="fecha_pub" />

    </tr>
    <tr><td colspan="3"><hr/></td></tr>
</xsl:template>
```

En este ejemplo se puede ver cómo se cambia en el resultado el orden de los elementos. Sólo es necesario especificar el *template* para cada uno según se desea que aparezcan: <*categoria*> en primer lugar, seguido por <*titulo*>, <*codigo*> y <*fecha_pub*>. Este *template* es más complicado por el hecho de que las celdas que van a contener los elementos <*categoria*> y <*titulo*> deben abarcar dos columnas, mientras que la columna final contiene <*codigo*> y <*fecha_pub*> en celdas ordinarias. Esa es la razón para emplear la etiqueta <*tr*></*tr*> como se ha hecho porque, de esa forma, se fuerza al contenido del elemento <*fecha_pub*> a

aparecer en la primera celda disponible dentro de la siguiente fila, es decir: debajo de la celda de <*codigo*>. Entre cada elemento <*libro*> hay otra fila que abarca las tres columnas descritas y que contiene una línea horizontal. Gráficamente se podría describir según la tabla 4.

Para completar la hoja de estilo *xslt* que produce la imagen mostrada anteriormente, sólo se necesitan los cuatro *templates* referenciados en el código anterior y correspondientes a los elementos <*categoria*>, <*titulo*>, <*codigo*> y <*fecha_pub*>.

— Tercera parte:

```
<xsl:template match="codigo">
    <td style="     font-family:Tahoma, Arial,sans-serif;
            Font-size:10pt">
        Código: <b><xsl:value-of /></b>
    </td>
</xsl:template>

<xsl:template match="categoria">
    <td rowspan="2" style="     font-family:Comic Sans MS,
Arial,sans-serif;
                Color:darkblue; font-size:16pt;
                Font-weight:bold">
        <xsl:value-of />
    </td>
</xsl:template>

<xsl:template match="fecha_pub">
    <td style="     font-family:Tahoma, Arial,sans-serif;
            Font-size:10pt">
        Fecha:<b><xsl:value-of /></b>
    </td>
</xsl:template>

<xsl:template match="titulo">
    <td rowspan="2" style="
                Font-family:Lucida Casual Italic,sans-serif;
                Font-size:18pt;
                Color:darkred;
                Font-weight:bold">
        <i>"<xsl:value-of />"</i>
    </td>
</xsl:template>

</xsl:stylesheet>
```

Puede observarse que las etiquetas <*td*></*td*> se insertan en las posiciones adecuadas con los atributos *"rowspan"* correspondientes para las celdas que contienen los elementos <*categoria*> y <*titulo*>. No hay necesidad de reorganizar los *templates* porque son ejecutados en el orden especificado según el principal (el de <*libro*> en la segunda parte). En definitiva, sólo hay que unir las tres partes comentadas en una sola hoja de estilo *xslt*, y ésta, asociada al documento *book.xml*, producirá el resultado que ya se ha visto (figura 10).

e. Crear elementos xml en el resultado. Con *xslt* se pueden crear nuevos elementos xml en el documento resultante, con lo cual se proporciona un método para mantener la documentación constantemente actualizada sin tener que cambiar, necesariamente, los documentos fuente. Los elementos *xslt* con los que se consigue esto son:

— *Xsl:attribute*.

— *Xsl:cdata*.

— *Xsl:comment*.

— *Xsl:element*.

— *Xsl:entity-ref*.

— *Xsl:pi*.

— *Xslt* también tiene un elemento que devuelve el nombre del nodo actual en forma de texto: *xsl:node-name*.

1. *Xsl:attribute*. Crea un atributo xml con el nombre y contenido que se especifique:

```
<xsl:attribute name="nombre del atributo">Valor del
atributo</xsl:attribute>
```

Por ejemplo, para añadir a todos lo elementos <*autor*> un atributo llamado *"tipodeautor"* y el valor *"investigador"* se usaría el siguiente código:

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

<xsl:template match:"autor">
    <xsl:copy>
        <xsl:attribute name="tipodeautor">Investigador</xsl:attri-
bute>
    </xsl:copy>
</xsl:template>
```

Si se quisiera añadir el atributo solamente al último autor, habría que modificar el código de la siguiente forma:

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

<xsl:template match:"au-
tor[end()]">
    <xsl:copy>
        <xsl:attribute na-
me="tipodeautor">investi-
```

<*categoria*>	<*titulo*>	<*codigo*> <*fecha_pub*>
<*hr/*>		
<*categoria*>	<*titulo*>	<*codigo*> <*fecha_pub*>
<*hr/*>		
<*categoria*>	<*titulo*>	<*codigo*> <*fecha_pub*>
<*hr/*>		

Tabla 4

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

Este elemento puede ser muy útil, por ejemplo, para construir enlaces en los documentos xml. Para conseguirlo se necesitará envolver el texto del vínculo con la etiqueta html <*a*></*a*> y añadir la URL apropiada como su atributo *"href*", suponiendo que quiera usarse como texto del enlace, y que es el contenido del elemento <*mienlace*> en el documento fuente. Podría construirse de la siguiente forma:

```
<xsl:template match:"//mienlace">
    <a>
        <xsl:attribute name="href">
            <xsl:value-of />
        </xsl:attribute>
    </a>
</xsl:template>
```

```
...
<?xml version="1.0"?>
<?xml-stylesheet                    type="text/xsl"
href="listadelibros.xsl"?>

<listadelibros>

 <libro>
 <codigo>16-048</codigo>
 <categoria>scripting</categoria>
 <fecha_pub>1998-04-21</fecha_pub>
 <titulo>instant javascript</titulo>
 <ventas>375298</ventas>
 </libro>

 <libro>
 <codigo>16-105</codigo>
 <categoria>asp</categoria>
 <fecha_pub>1998-05-10</fecha_pub>
 <titulo>instant active server pages</titulo>
 <ventas>297311</ventas>
 </libro>

 <libro>
 <codigo>16-041</codigo>
 <categoria>html</categoria>
 <fecha_pub>1998-03-07</fecha_pub>
 <titulo>instant html</titulo>
 <ventas>127853</ventas>
 </libro>

</listadelibros>
...
```

```
...
<?xml version="1.0"?>
<!—nueva versión del documento book.xml-->


<nuevoslibros>

 <codigo>16-048</codigo>
 <titulo>instant javascript</titulo>
 <ventas>375298</ventas>

 <codigo>16-105</codigo>
 <titulo>instant active server pages</titulo>
 <ventas>297311</ventas>

</nuevoslibros>
...
```

Tabla 5

2. *Xsl:cdata*. Crea una sección *cdata*[15] xml y su contenido. El procesador *xslt* no lo procesará y, por lo tanto, esta posibilidad puede usarse para crear elementos *xslt* en el resultado si es necesario:

```
<xsl:cdata>contenido de la sección cdata</xsl:cdata>
```

3. *Xsl:comment*. Crea un comentario xml y su contenido:

```
<xsl:comment>Contenido del comentario</xsl:comment>
```

El siguiente código, por ejemplo produciría como resultado < *! - -Esto es un comentario xml - - >*:

```
<xsl:comment>Esto es un comentario xml</xsl:comment>
```

4. *Xsl:element*. Crea un elemento xml con el nombre y contenido que se especifique:

```
<xsl:element name="nombre del elemento">Contenido del elemento</xsl:element>
```

El siguiente código produciría como resultado <*mielemento*>*hola*</*mielemento*>:

```
<xsl:element name="mielemento">hola</xsl:element>
```

5. *Xsl:entity-ref*. Crea una entidad de referencia xml con el nombre y valor que se defina:

```
<xsl:entity-ref name="nombre de la entidad">Contenido</xsl:entity-ref>
```

6. *Xsl:pi*. Produce una instrucción de proceso xml con el nombre y contenido señalados:

```
<xsl:pi name="nombre">Contenido</xsl:pi>
```

El siguiente caso, por ejemplo produciría la instrucción <*?xml version='1.0'?*>:

```
<xsl:pi
name="xml">'1.0'</xsl:pi>
```

7. *Xsl:node-name*. Devuelve el nombre del elemento actual en el documento fuente xml en forma de texto, de modo que puede insertarse en el documento resultante como texto, y no como elemento en sí. Sólo funciona con nombres de elementos y no con comentarios, instrucciones de proceso etc. Para hacer una lista con los nombres de los elementos de un documento, por ejemplo, se podría usar el siguiente código:

```
<xsl:template match="/">
    <xsl:for-each se-
lect:"//">
        Nombre del ele-
mento: <xsl:node-name
/><br/>
    </xsl:for-each>
</xsl:template>
```

f. Transformar un documento xml en un documento xml diferente. Los ejemplos anteriores convertían xml en html para que aquel pudiera ser presentado sin problemas en el navegador. Pero en general, el planteamiento tras *xslt* es la mutación de unos formatos en otros. Esto incluye la idea de transformar un documento xml en otro diferente. Usando las características de *xslt* es posible modificar la estructura y contenido de un documento xml, cambiar el orden y la apariencia de los elementos, seleccionar los datos que se quieren o no presentar y sumar nuevos elementos a la estructura original.

El siguiente ejemplo de hoja de estilo *xslt* toma el documento *book_xml* y crea otro nuevo a partir de él insertando una instrucción de proceso y un comentario, seleccionando solamente tres elementos del documento original. También se limitan los libros presentados usando un filtro que obtiene sólo aquellos publicados después del 31 de marzo de 1998:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
    <xsl:pi name="xml">version="1.0"</xsl:pi>
    <xsl:comment>Nueva versión del documento
book.xml</xsl:comment>
    <xsl:element name="nuevoslibros">
        <xsl:apply-templates select="//libro" />
    </xsl:element>
</xsl:template>

<xsl:template match="libro[fecha_pub $gt$ '1998-03-31']">
    <xsl:apply-templates select="codigo" />
    <xsl:apply-templates select="titulo" />
    <xsl:apply-templates select="ventas" />
</xsl:template>

<xsl:template match="codigo">
    <xsl:copy />
</xsl:template>

<xsl:template match="titulo">
    <xsl:copy />
</xsl:template>

<xsl:template match="ventas">
    <xsl:copy />
</xsl:template>

</xsl:stylesheet>
```

Después de seleccionar los elementos <*libro*> que se quieren insertar en el documento resultante, se especifican los *templates* para los elementos hijo: <*codigo*>, <*titulo*> y <*ventas*> que deben copiar los elementos a los que se refieren en el documento resultante usando la instrucción <*xslt:copy*>. El resultado de aplicar la hoja de estilo anterior al documento *book_xml* (a la izquierda) es un nuevo documento xml (a la derecha) con estructura y contenido diferentes (tabla 5).

«La forma en que *xslt* procesa los datos permite que un nodo xml pueda ser transformado en casi cualquier tipo de cadena de texto, incluyendo ascii»

g. Programación en *xslt*. Además de dar formato al contenido de un documento xml o crear nuevos elementos, el procesador *xslt* también puede controlar el resultado usando elementos de *loop* y estructuras de decisión. Es parecido a la forma en que *SQL* puede examinar los resultados obtenidos al consultar una base de datos. Además, también se pueden incluir scripts en las hojas de estilo *xslt*, lo que resulta particularmente útil pues es actualmente la única técnica para incluirlos en documentos xml puros.

g.1. *Loops* y estructuras de decisión. Hay cinco elementos básicos en *xslt* que proporcionan a las hojas de estilo la capacidad de definir *loops* y estructuras de decisión:

— *Xsl:for-each*

— *Xsl:if*

— *Xsl:choose*

— *Xsl:when*

— *Xsl:otherwise*

1. *Xsl:for-each*. Se utiliza para detallar series de elementos *xslt* sobre los que se va a repetir un determinado proceso. Esta instrucción permite controlar el orden de los elementos a los que afecta sin tener que especificar *templates* en un orden determinado (como ocurría con el elemento <*xsl:apply-templates...*>). Si el atributo *"select"* presenta algún valor, sólo se procesarán los elementos hijos del elemento o aquellos a los que se refiera ese valor. También acepta un atributo opcional *"order-by"* que define el orden en que se presentarán los elementos seleccionados. El valor de este atributo es un conjunto de patrones delimitados por *";"*, cada uno de los cuales puede estar precedido por *"+"* o *"-"* para indicar orden ascendente o descendente. Su sintaxis general es:

```
<xsl:for-each select="patrón" order-by="conjunto de patrones">…
</xsl:for-each>
```

Para conseguir, a partir de *book_xml* una lista de nombres de autores en orden alfabético, se podría usar el siguiente código:

```
<xsl:for-each select="libro/autor" order-by="+apellido; +nombre-depila">…
Aquí los elementos xslt que deban procesarse para cada elemento <autor>.
</xsl:for-each>
```

2. *Xsl:if*. Se usa para obtener un resultado condicionado al producido por una prueba, la cual no es más que un patrón *xslt*:

```
<xsl:if match="patrón">Resultado condicionado</xsl:if>
```

El siguiente código insertará el texto *Investigador* solamente si el atributo *"tipodeautor"* de <*autor*> presenta un valor *"investigador"*:

```
<xsl:if match="autor[@tipodeautor='investigador']">
    Investigador
</xsl:if>
```

3. Xsl:choose, *Xsl:when* y *Xsl:otherwise*. Se usan conjuntamente para proporcionar al procesador *xslt* capacidad de bifurcación condicional. <*xsl:when..*> pre-

senta un atributo *"match*", cuyo valor, como en el caso de <*xsl:if..*>, es la prueba sobre la que se construye la condición. Si el contenido del documento fuente xml supera esa prueba, entonces, el contenido de <*xsl:when..*> se incluirá en el resultado. Solamente se procesa el primer elemento <*xsl:when..*> que supera la condición y si no hay ninguno, se utiliza <*xsl:otherwise..*> en su lugar. Por ejemplo, dado el documento xml *numeros.xml*:

```
<?xml version="1.0"?>
 <?xml-stylesheet type="text/xsl" href="numeros.xsl"?>
  <pruebanumeros>
    <numero>16</numero>
    <numero>0</numero>
    <numero>-78</numero>
    <numero>1687698</numero>
    <numero>-2</numero>
    <numero>-0</numero>
  </pruebanumeros>
```

se puede crear una hoja de estilo (*numeros.xsl*) que indique cuándo un número es positivo, negativo o cero usando los elementos <*xsl:choose*>, <*xsl:when*> y <*xsl:otherwise*>:

```
<?xml version="1.0" ?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/tr/wd-xsl">
 <xsl:template match="/">
  <html>
  <body>
   <h3>prueba con cifras</h3>
   <xsl:for-each select="//numero">
    <xsl:value-of />
    <xsl:choose>
     <xsl:when match="*[. $gt$ 0]">positivo</xsl:when>
     <xsl:when match="*[. $lt$ 0]">negativo</xsl:when>
     <xsl:otherwise>cero</xsl:otherwise>
    </xsl:choose>
   <br/>
   </xsl:for-each>
   </body>
  </html>
  </xsl:template>
</xsl:stylesheet>
```

El *template* raíz crea la estructura básica de la página html que va a ser el resultado. Después se usa un *loop* <*xsl:for-each..*> para iterar el programa sobre todos los elementos <*numero*>. El elemento <*xslt:value-of*> inserta en el resultado el valor actual del número que está siendo procesado, por lo que <*xsl:choose...*> añade a ese valor el texto apropiado dependiendo de si es positivo, negativo o cero. El resultado de la hoja de estilo *numeros.xsl* actuando sobre el documento numeros.xml puede observarse en la figura 11.

«X*slt* proporciona un elemento especial que permite asociar scripting a los documentos xml definiéndolo en la hoja de estilo»

La traducción del código <*xsl:when match="*[. gt 0]"*>*positivo*</*xsl:when*> podría ser: "si cualquiera de los elementos seleccionados presenta un valor mayor que cero, entonces escribe el valor de ese elemento seguido de la cadena de caracteres positivo". Si asociamos cada parte de la expresión anterior con su correspondiente en el código resultaría algo así: "si cualquiera (por eso el '*') de los elementos seleccionados (<*xsl:for-each select="//numero"*>) presenta un valor (por eso el '.': el valor del elemento que se esté procesando en ese momento) mayor que cero (*"*[. gt 0]" (gt = greater than) (lt = less than)*), entonces escribe el valor de ese elemento (<*xsl:value-of* />) seguido de la cadena de caracteres positivo".

«Es posible incluir scripts en las hojas de estilo *xslt*, lo que resulta particularmente útil pues es actualmente la única técnica para incluirlos en documentos xml puros»

g.2. Añadir script a hojas de estilo *xslt*. *Xslt* proporciona distintos medios de manipular y presentar documentos xml en el navegador. Por lo tanto ¿se podría pensar que a partir de ahora es posible abandonar html y utilizar simplemente xml? Una de los temas que se deben considerar es que en xml la etiqueta html <*script*> no significa nada. Para superar esta limitación, *xslt* proporciona un elemento especial que permite asociar scripting a los documentos xml definiéndolo en la hoja de estilo:

— *Xsl:script*

— *Xsl:eval*

1. *Xsl:script*. Este elemento define una sección script de la hoja de estilo que puede ejecutarse desde cualquier parte de la hoja de estilo simplemente invocándolo. <*xsl:script...*> acepta un atributo *"language"* que especifica el lenguaje de scripting que se va a usar:

```
<xsl:script language="lenguaje de scripting">
...
</xsl:script>
```

El siguiente ejemplo define una función que devuelve la hora y fecha actuales:

```
<xsl:script language="VBScript">
    Function getDateTime()
        GetDateTime = Now()
    End Function
</xsl:script>
```

2. *Xsl:eval*. Se utiliza para evaluar una cadena de caracteres del script e insertar el resultado de la evaluación en el nuevo documento. También acepta un atributo *"language*":

```
<xsl:eval language="lenguaje de scripting">
...
```

Si sus documentos son estratégicos para su compañía, también lo son para nosotros

BASISPLUS

La Solución Documental

BASIS*plus* es el único sistema de gestión de documentos que permite gestionar de un modo integrado toda la información que circula por su organización.

Con BASIS*plus* su trabajo será más fácil, su tiempo más rentable y su información más segura y accesible.

Más de 1 millón de usuarios, 2.500 instalaciones en 75 países y en 24 idiomas

TECHLIBPLUS

La Gestión de Bibliotecas

TECHLIB*plus* es una solución completa totalmente diseñada bajo entorno Web para gestionar los recursos de información y actualizar el trabajo diario de las bibliotecas.

Aplicaciones basadas en las tecnologías **BASIS*plus*** y **TECHLIB*plus*:**

Centros de Documentación
Sistemas de Gestión de Calidad
Gabinetes de Prensa
Difusión de Boletines Oficiales
Gestión de Expedientes
Normativa y Procedimientos
Producción Editorial
Gestión de Archivos

1969 30 ANIVERSARIO 1999

CENTRISA
Tecnologías de la Información
Imaginación

08037 BARCELONA
Consejo, 373
Tel. 93 207 65 11
Fax 93 459 00 14

28006 MADRID
Castelló, 98
Tel. 91 562 73 54
Fax 91 562 48 54

48009 BILBAO
Alameda Rekalde, 27. 1º izq.
Tel. 94 423 00 75
Fax 94 423 94 18

31008 PAMPLONA
Iturrama, 11. 1º D
Tel. 948 17 87 29
Fax 948 17 16 49

46009 VALENCIA
Ricardo Micó, 5
Tel. 96 346 32 50
Fax 96 346 32 53

CGC
48003 BILBAO
Avda. San Adrián, 45-47
Tel. 94 412 31 43
Fax 94 410 04 46

INFINITY
08820 EL PRAT DE LLOBREGAT (Barcelona)
Solidaridad, 2, esc. D, Pl. 2
Tel. 93 478 57 21
Fax 93 478 51 62

SDC
28006 MADRID
Castelló, 98
Tel. 91 562 73 54
Fax 91 562 48 54

CENTRISA-PORTUGAL
1050 LISBOA
Avda. João Crisóstomo, 38 C
Escritorio 1
Tel. 0035-11 93 54 15 99
Fax 0035-11 93 54 16 08

CENTRISA-ARGENTINA
1386 BUENOS AIRES
Avda. Presidente Roque Sáenz Peña, 832, 7º Pl.
Tel. 0054-11 43 39 91 58
Fax 0054-11 43 28 80 95

CENTRIMEX
MÉXICO D.F.
Mussat, 353
Colonia Polanco, 11550
Tel. 0052-5 254 39 15
Fax 0052-5 254 17 93

CENTRIVEN
CARACAS
Avda. La Industria
Centro Industrial
Palo Verde, piso 2º
Tel. 0058-2 251 35 51
Fax 0058-2 251 61 52

CENTRISA-COLOMBIA
SANTAFÉ DE BOGOTÁ D.C.
Transversal 40, 21-40

CENTRISA-BRASIL
SALVADOR DE BAHIA

GRUPO CENTRISA

www.centrisa.es

```
</xsl:eval>
```

Como ejemplo, se puede llamar a la función *getDateTime()* definida antes:

```
<xsl:eval language="VBScript">getDateTime()</xsl:eval>
```

El mismo código podría servir para llamar a la función *now()*:

```
<xsl:eval language="VBScript">Now()</xsl:eval>
```

Conclusión

En estas páginas se ha analizado una de las especificaciones del entorno xml con más proyección de futuro. Si se tuvieran que resumir las características de *xslt* se podría hablar de su estructura jerárquica de forma y de su proximidad conceptual con la orientación a objetos.

> **«X*slt* pretende situarse justo a medio camino, estructurando los objetos como xml, pero gestionando también su comportamiento como en un lenguaje de programación»**

¿Cuál es la diferencia fundamental entre un lenguaje de programación orientado a objetos y un lenguaje de etiquetas como xml? La disparidad reside en el concepto de comportamiento de los objetos. Una parte importante de la orientación a objetos se basa en la especificación del comportamiento del objeto. Xml especifica sólo la estructura de los objetos (entiéndanse éstos el documento y sus elementos), excluyendo intencionadamente su comportamiento para dar a los desarrolladores la máxima flexibilidad a la hora de utilizar esos datos. Pues bien, *xslt* pretende situarse justo a medio camino, estructurando los objetos como xml, pero gestionando también su comportamiento como en un lenguaje de programación.

La flexibilidad, compatibilidad y vocación www de esta especificación son otras tres características muy importantes. *Xslt* es un tema muy amplio. Las posibilidades examinadas muy brevemente en este trabajo han sido simplemente: seleccionar, modificar, dar formato y presentar documentos xml transformándolos en código html. Hay que tener presente que las funcionalidades reales de aplicación de esta tecnología van mucho más allá. Por otra parte no se ha comentado nada sobre otros parsers (más sofisticados que *IE5*) como: *Oracle xsl*, *Saxon*, *Xalan*, *xt*, *ixslt*, *Stylus* o *4xslt* simplemente para subrayar el hecho de que es una tecnología al alcance de todo aquel que pueda descargar en su sistema la última versión del navegador *IE*.

Las preguntas que quedan por hacerse son: ¿dónde sería más útil emplear esta tecnología?, ¿qué tipo de aplicaciones documentales podrían desarrollarse con ella? Las respuestas deberían darse en forma de proyectos concretos.

Notas

1. De ahí el nombre *cascading style sheets*, porque se aplican en cascada a un documento.

2. El "punto" es una unidad absoluta de dimensión usada en tipografía.

3. Definida por el autor del código.

4. Este ejemplo volverá a usarse en varias ocasiones a lo largo del trabajo.

5. Este bloque en realidad estará formado por el contenido de los descendientes del elemento <*libro*>, que no es más que una forma abstracta de agrupar ese contenido.

6. Aceptadas por la aplicación en la que el documento vaya a ser presentado. Generalmente un navegador.

7. También otros formatos.

8. Por el mismo razonamiento, el input del proceso *xslt* también podría ser una *DTD*, un esquema, etc., lo que nos da una idea aproximada del potencial de esta tecnología.

9. Generalmente elementos.

10. Xml permite a los desarrolladores crear sus propios lenguajes de etiquetas para sus propios proyectos. Estos lenguajes pueden compartirse con individuos que trabajen en el mismo tipo de proyecto a través de la web. Un ejemplo claro es *xslt*. El lenguaje de transformación *xslt* debe generar xml bien-formado que incluya etiquetas *xslt*. Por lo tanto se necesita un sistema para distinguir claramente cuándo esos elementos xml son en realidad "instrucciones" *xslt* y cuándo son simples elementos de salida (incluso si ambos elementos tienen el mismo nombre). Esa es la razón de ser de los *namespaces*, pues permiten que cada elemento pueda ser entendido en un entorno específico. Por ejemplo, los elementos xml que conciernen a las instrucciones de transformación *xslt* están en el *namespace*: *http://www.w3.org/xsl/transform/1.0*, mientras que los que sólo son elementos xml de salida están en: *http://www.w3.org/xsl/format/1.0*.

11. Because *xslt* style sheets are xml documents they should begin with the usual xml declaration.

12. El nodo raíz de *book.xml* era <*listadelibros*></*listadelibros*>.

13. <*codigo*/>, <*categoria*/>, <*fecha_pub*/> y <*ventas*/>.

14. <*xslt:copy*> es particularmente útil cuando el documento fuente *XM* contiene etiquetas html como <*b*>, <*i*> o <*table*>. Usar el elemento *xslt:copy* es como decir al procesador: "Trata estas etiquetas xml como si tuvieran su significado html habitual".

15. En xml el contenido de las secciones *cdata* o *character data* es tratado como texto plano.