Conversión y adaptación de documentos XML.

Caso práctico

jonny goldstein (CC BY)

La empresa de **Félix** y **María** continúa creciendo.

Dado que los documentos de los clientes de la empresa están en formato XML, **Juan** les plantea una nueva posibilidad para su página web. Dar la posibilidad a los clientes de consultar sus datos en la empresa a través de su página web.

Aunque a **Félix** y a **María** no les parece una idea muy seductora inicialmente, debido a que no consideran que el formato de los documentos XML sea el más atractivo para los clientes de la misma, deciden pedir consejo.

Una vez más, Juan les explica que mediante un sistema XSL se pueden transformar y formatear los contenidos almacenados en documentos XML. Se podrían presentar los datos en distintos formatos como XHTML, HTML o incluso PDF.

1.- Técnicas de transformación de documentos XML.

Caso práctico

Let Ideas Compete (CC BY-NC-ND)

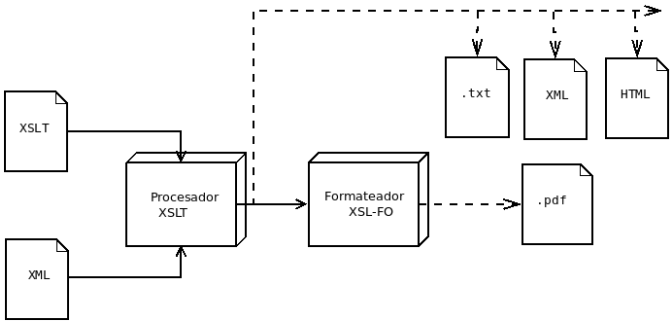
Félix escucha atentamente las explicaciones de **Juan** sobre las posibilidades que da la transformación de documentos.

Gracias a ellas acaba de pensar que, inicialmente se permitirá a los clientes consultar los datos a través de la web de la empresa, pero que en un futuro también puede servirles para generar informes en formato PDF, que pueden utilizar para la comunicaciones con sus clientes, bien a través del correo ordinario o del electrónico.

Extensible Stylesheet Language (**XSL**) es un lenguaje que interpreta hojas de estilo. Una hoja de estilo XSL describen cómo se debe mostrar un documento XML, algo parecido a lo que hace un archivo CSS (Cascading Style Sheets) con un archivo HTML.

La especificación XSL define unas características y una sintaxis que se agrupan en tres partes:

XSL Transformation Language (XSLT: Un lenguaje para la transformación de documentos XML.
XML Path Language (XPath): Un lenguaje de consulta de elementos en un documento XML.
XSL Formatting Object (XSL-FO): Un vocabulario XML para especificar la semántica del formato, es decir, indicar dónde debe aparecer cada elemento, con qué espaciado, con qué colores, con qué tipo de letra, ...



José Antonio Molina Bautista. *Transformaciones XSL* (GNU/GPL)

Para realizar las transformaciones se utilizan unos programas llamados **procesadores XSL** o parser XSL. A estos programas se les debe indicar los archivos de entrada XML y XSL. Una vez procesados si no tienen errores se generará el archivo deseado en el formato indicado por las instrucciones XSL.

XSLT fue diseñado para realizar transformaciones usando el vocabulario XML especificado por XSL-FO pero también para usarse con independencia de XSL y dicho vocabulario.

A nivel académico podemos realizar transformaciones de documentos XML a documentos HTML u otros XML de estructura diferente sin usar XSL-FO. En esta unidad usaremos XSLT y XPath de esta forma para evitar toda la complejidad añadida.

Para saber más

Puedes ver la recomendación XSL 1.0 completa en la web del consorcio W3C.

Si quieres ampliar información puedes buscar en el apartado dedicado a los lenguajes de hojas de estilos extensibles también en la web del consorcio W3C.

Autoevaluación

Los lenguajes de marcas que no permiten la transformación de documentos son:

- [ ] HTML.
- [ ] XHTML.
- [ ] XPath.
- [ ] XSL.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto

2.- XML Path Language (XPath)

Caso práctico

Nate Steiner (Dominio público)

María empieza a convencerse de que el proyecto puede suponer una ventaja al permitir exponer la información de la empresa en múltiples formatos. **Juan** le dice que en lo que a intercambio de información se refiere si es así.

Su pregunta ahora es, ¿de qué modo van a lograr seleccionar partes de la información que tienen en sus extensos archivos XML? ¿cada cliente solamente querrá ver la información que le corresponde?

Juan les explica que para eso existe un lenguaje llamado XPath, cuya sintaxis se asemeja a la que se usa para desplazarse a través de un árbol de directorios pero añadiéndole condiciones y funciones extras.

XML Path Language (XPath) es un lenguaje de consulta de elementos en un documento XML. Diseñado inicialmente para ser usado con XSLT (eXtensible Stylesheet Language for Transformations) y XPointer (XML Pointer Language). En versiones posteriores también es usado por XQuery como veremos en próximas unidades.

El consorcio W3C (The World Wide Web Consortium) aprobó su primera versión XPath 1.0 en noviembre de 1999. XPath usa una sintaxis compacta y sin etiquetas por lo que no se asemeja a XML. Su nombre surge por el uso de las rutas en la navegación a través de estructuras jerárquicas dentro de los documentos XML.

Posteriormente siguieron las versiones:

XPath 2.0 en enero de 2007,
XPath 2.0 segunda edición en diciembre de 2010,
XPath 3.0 en abril de 2014 y
XPath 3.1 en marzo de 2017.

Para saber más

Puedes ver la recomendación XPath 1.0 completa en la web del consorcio W3C.

Si quieres ampliar información de las distintas versiones de XPath puedes buscar en el apartado correspondiente de la web del consorcio W3C.

Autoevaluación

XPath es un lenguaje que permite:

- ○ Transformar el formato de los datos de un fichero XML.
- ○ Definir un vocabulario que ha de cumplir un documento XML.
- ○ Acceder a los datos de una base de datos XML.
- ○ Acceder a los datos de un fichero XML.

No es correcta porque eso lo hace XSLT.

Incorrecta, porque de ello se encarga DTD y XML Schema.

No es la respuesta correcta.

Muy bien. Has captado la idea.

Solución

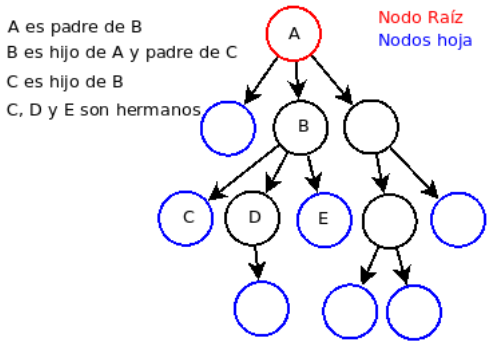
1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

2.1.- Árbol de nodos

XPath modela los documentos XML en árboles de nodos.

Un árbol es una estructura abstracta muy usada en informática que representa una serie de elementos (nodos) unidos por líneas (vértices). Además entre dos nodos cualesquiera sólo existe un único camino y ninguno de los caminos forma un bucle.

El nombre de esta estructura le viene porque tiene forma de copa de árbol invertida. Partiendo de un nodo raíz van añadiéndose ramas que contienen nodos hijos. A los nodos que tiene al menos un nodo hijo se le llama nodo padre. A los nodos que no tienen nodos hijos se les llama nodos hoja. Los los nodos que comparten padre se les llaman nodos hermanos.



José Antonio Molina Bautista. *Árbol de nodos* (GNU/GPL)

En XPath existen distintos tipos de nodos que recogen los distintos tipos de información de los documentos XML. Los siete distintos tipos de nodos que podemos tener serán utilizadas posteriormente como 'selectores de nodos' en los 'pasos de localización'.

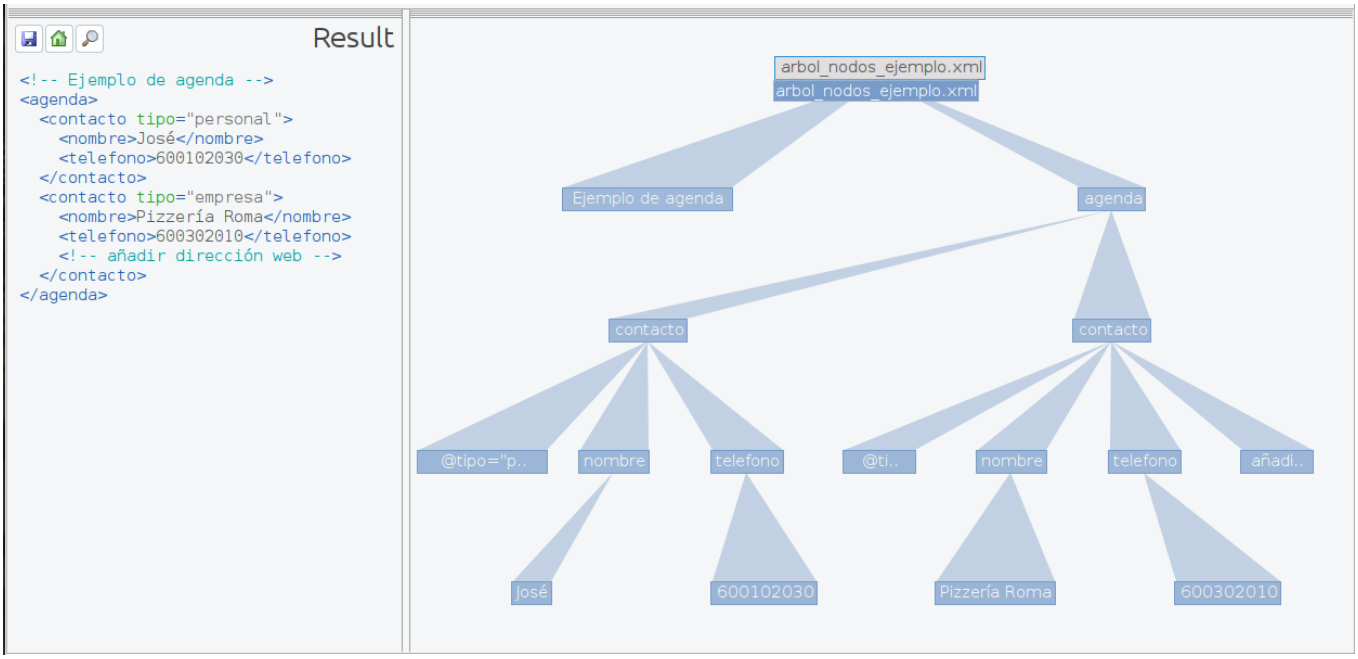
Los tipos de nodos son los siguientes:

Nodo raíz: Es el primer nodo del árbol, es único y el único que no tiene padre. No confundir el nodo raíz con el elemento raíz del documento XML. El nodo raíz tiene como hijos al elemento raíz y en su caso los comentarios y las instrucciones de procesamiento que formen el prólogo del documento XML.
Nodo elemento: Hay un nodo elemento por cada elemento XML del documento. Todos los nodos elemento tiene un sólo padre que puede ser otro nodo elemento o el nodo raíz. Los nodos elemento pueden tener un identificador único (ID). Para ser así debe estar declarado de tipo ID en su DTD o en el XML Schema asociado.
Nodo atributo: Los atributos de un documento XML se almacenan en nodos atributo. Un nodo atributo estará asociado a un único nodo elemento que será el padre de este. Pero desde el punto de vista del nodo elemento no se considerará a sus atributos como nodos hijos. Los nodos atributo son nodos hoja, es decir, no pueden tener nodos hijos.
Nodo texto (o contenido): Los valores alfanuméricos de los contenidos de los elementos de un documento XML serán almacenados en nodos texto. Los nodos texto son nodos hoja, es decir, no pueden tener nodos hijos. Nótese que los valores de los atributos no se almacenan en nodos texto sino en los propios identificadores de dichos nodos.
Nodo comentario: Los comentarios de un documento XML son almacenados en nodos comentario. El nodo almacenará el contenido del comentario sin incluir el inicio .
Nodo espacio de nombres: Cada nodo elemento puede tener un conjunto asociado de nodos espacio de nombres, uno para cada uno de los distintos prefijos de espacio de nombres incluyendo si es el caso el espacio de nombres por defecto. Tienen un funcionamiento parecido a los atributos. Un nodo espacio de nombres tiene un único nodo elemento como padre. Pero desde el punto de vista del nodo elemento no son considerados como nodos hijos de este. Los nodos espacio de nombres son nodos hoja, es decir, no pueden tener nodos hijos.
Nodo instrucción de procesamiento: Hay un nodo instrucción de procesamiento para cada instrucción de procesamiento. No incluye la terminación ?>.

Además se establece un **orden entre los nodos** que se corresponde con el orden en el que están escritos en el documento. De modo que el nodo raíz será el primer nodo. Los nodos elemento aparecerán antes de sus hijos. Por tanto, el orden del documento ordena los nodos elemento en el orden de aparición de sus etiquetas de apertura en el documento XML (tras la expansión de entidades). Los nodos atributo y los nodos espacio de nombres de un elemento aparecen antes que los hijos del elemento. Los nodos espacio de nombres aparecen por definición antes que los nodos atributo.

El nodo raíz y los nodos elemento tienen una lista ordenada de nodos hijo. Los nodos nunca comparten hijos, es decir, cada nodo tiene un único padre excepto el nodo raíz que no tiene padre.

Caso de uso:



José Antonio Molina Bautista. *Árbol de nodos de BaseX* (CC BY-NC-SA)

En la ejemplo anterior de un documento XML que almacena una agenda podemos ver el árbol de nodos que crea la aplicación BaseX. Podemos observar a la izquierda el documento XML y a la derecha el árbol creado. Esta aplicación no resalta los distintos tipos de nodos de ninguna forma especial todos los muestra con su identificador.

El **nodo raíz** está indicado con el nombre del documento XML: arbol_nodos_ejemplo.xml. Nótese que no se corresponde con el elemento raíz que en este caso sería 'agenda'. Es importante no confundir nodo raíz y elemento raíz. El primero se refiere al árbol de nodos y el segundo al documento XML.
Los **nodos elementos** son los más abundantes: agenda, contacto, nombre y **`teléfono`**.
Los **nodos texto** o contenido son los contenidos de los elementos nombre y **`teléfono`** es decir: José, 600102030, **`Pizzería Roma`** y 600302010. Estos nodos siempre serán nodos hoja. Es decir, que no tendrán nodos hijos. Nótese que los valores que toman los atributos no se recogen en nodos texto sino que aparecen junto al identificador del atributo.
Los **nodos atributos** que recogen el identificador y el valor. En nuestro ejemplo: tipo="personal" y tipo="empresa"
También podemos ver en nuestro ejemplo dos **nodos comentarios** uno al inicio y otro dentro del segundo elemento contacto: **`Ejemplo de agenda`** y añadir dirección web.

Autoevaluación

El nodo raíz de un árbol de nodos coincide con el ejemplar de su documento XML:

- ○ Sí.
- ○ No.

No es correcta porque el nodo raíz contiene el ejemplar del documento.

Efectivamente es correcto, es importante que no confundas un nodo raíz con un elemento raíz.

Solución

1. Incorrecto
2. Opción correcta

2.2.- Relaciones entre nodos

Cuando nos posicionamos en un determinado nodo dentro de un árbol de nodos se establecen distintas relaciones con otros nodos de dicho árbol.

El nodo en el que nos posicionamos para establecer una relación se le suele llamar: **nodo contexto** o nodo contextual.

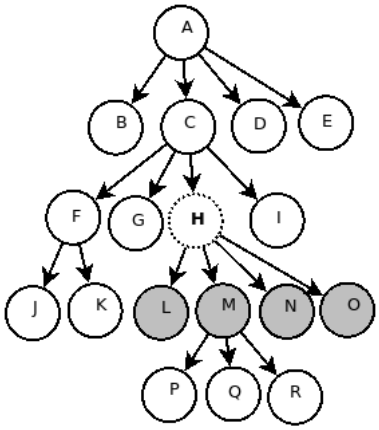
Las trece distintas relaciones que podemos tener serán utilizadas posteriormente como 'ejes' en los 'pasos de localización'.

Dado un nodo contexto marcado con borde punteado los siguientes gráficos muestran distintas relaciones de este nodo con los nodos sombreados.

self: contiene el nodo de contexto.

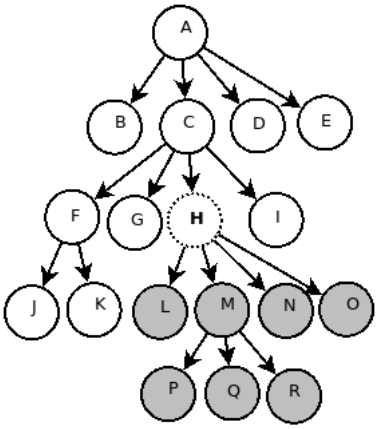
José Antonio Molina Bautista. *Eje self* (GNU/GPL)

child: contiene los hijos del nodo contexto.



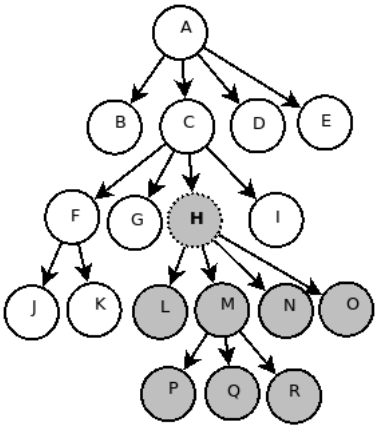
José Antonio Molina Bautista. *Eje child* (GNU/GPL)

descendant: contiene los descendientes del nodo contexto; un descendiente es un hijo o el hijo de un hijo, etc; de este modo *descendant* nunca contiene nodos atributo o espacio de nombres



José Antonio Molina Bautista. *Eje descendant* (GNU/GPL)

descendant-or-self: contiene el nodo contexto y sus descendientes.

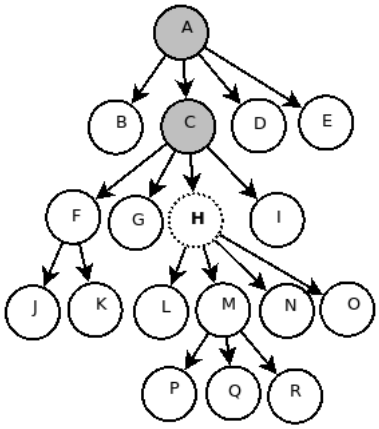


José Antonio Molina Bautista. *Eje descendant or self* (GNU/GPL)

parent: contiene el padre del nodo contexto, si lo hay.

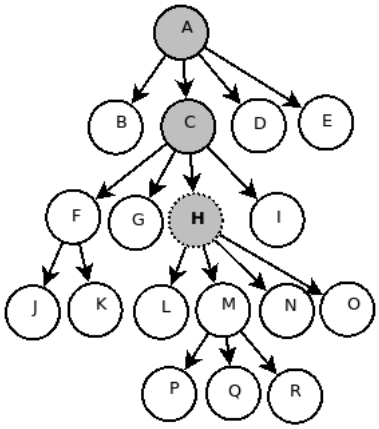
José Antonio Molina Bautista. *Eje parent* (GNU/GPL)

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



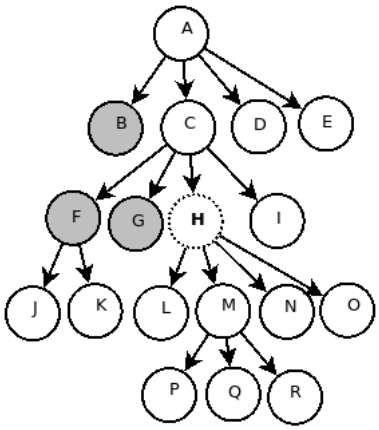
José Antonio Molina Bautista. *Eje ancestor* (GNU/GPL)

ancestor-or-self: contiene el nodo contexto y sus ancestros; así, *ancestor-or-self* siempre incluirá el nodo raíz.



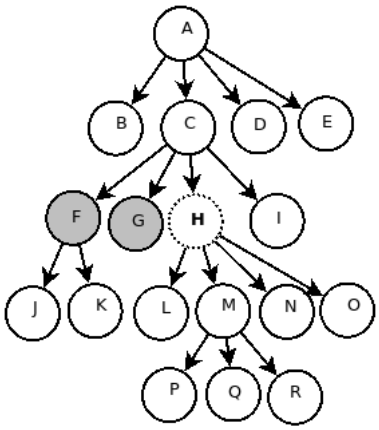
José Antonio Molina Bautista. *Eje ancestor or self* (GNU/GPL)

preceding: contiene todos los nodos del mismo documento que el nodo contexto que están antes de este según el orden del documento, excluyendo los ancestros y excluyendo nodos atributo y nodos espacio de nombres.



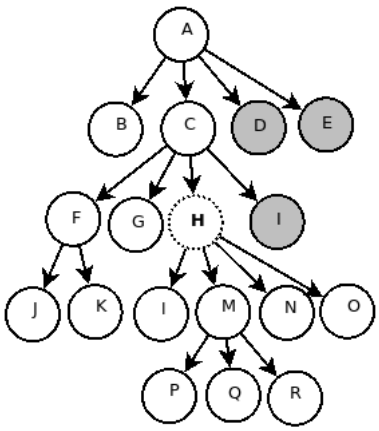
José Antonio Molina Bautista. *Eje preceding* (GNU/GPL)

preceding-sibling: contiene todos los hermanos precedentes del nodo contexto; si el nodo contexto es un nodo atributo o un nodo espacio de nombres, el eje *preceding-sibling* está vacío.



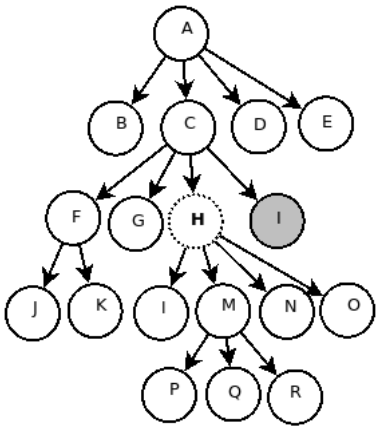
José Antonio Molina Bautista. *Eje preceding-sibling* (GNU/GPL)

following: contiene todos los nodos del mismo documento que el nodo contexto que están después de este según el orden del documento, excluyendo los descendientes y excluyendo nodos atributo y nodos espacio de nombres.



José Antonio Molina Bautista. *Eje following* (GNU/GPL)

following-sibling: contiene todos los siguientes hermanos del nodo contexto; si el nodo contexto es un nodo atributo o un nodo espacio de nombres, el eje *following-sibling* está vacío.



José Antonio Molina Bautista. *Eje following-sibling* (GNU/GPL)

Por simplificar el gráfico no se han incluido nodos atributos ni nodos de espacios de nombre. Las relaciones con esos nodos son las siguiente:

attribute: contiene los atributos del nodo contexto. Esta relación será vacía si el nodo contexto no es un nodo elemento ya que es el único que puede tener nodos atributo.

namespace: contiene los nodos espacio de nombres del nodo contexto. Esta relación será vacía si el nodo contexto no es un nodo elemento ya que es el único que puede tener nodos espacio de nombres.

Autoevaluación

¿El eje **ancestor** contienen todos los elementos del eje **parent**?

- [ ] Sí.
- [ ] No.
- [ ] Sí, pero solamente si el nodo contexto es el nodo raíz.

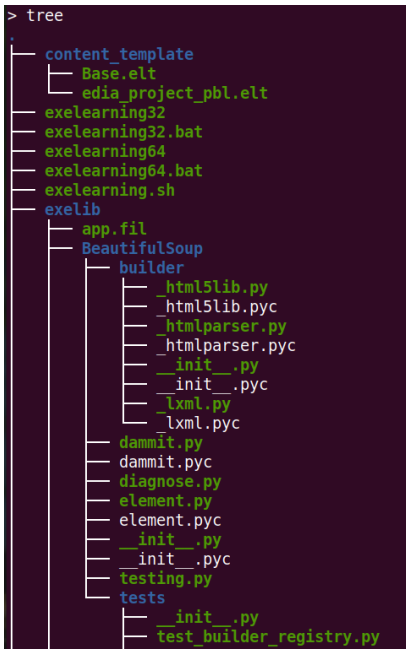
Mostrar retroalimentación

Solución

1. Correcto
2. Incorrecto
3. Incorrecto

2.3.- Sintaxis

La sintaxis de XPath no usa etiquetas por lo que no se asemeja a XML. Como su utilidad es realizar búsquedas en un árbol de nodos se utiliza una sintaxis parecida a la que se usa en los árboles de directorios y archivos de los sistemas operativos.



José Antonio Molina Bautista. *Árbol jerárquico* (GNU/GPL)

Un ejemplo de expresión XPath puede ser el siguiente '**camino de localización**' (location paths):

Versión completa:

descendant-or-self::curso[position()=1]/child::grupo[position()=2]/
child::alumno[last()]/child::nombre/child::node()

Aunque se suele utilizar, siempre que se puede, su versión simplificada. El ejemplo anterior quedaría de la siguiente forma.

Versión simplificada:

//curso[1]/grupo[2]/alumno[last()]/nombre/text()

Además en las expresiones XPath se pueden utilizar **llamadas a funciones**, **operaciones matemáticas** simples y **operaciones lógicas**.

Cuando un programa evalúa un *camino de localización* XPath devolverá el resultado en uno de los siguientes cuatro tipos básicos:

Conjunto de nodos (Node-Set): una colección desordenada de nodos sin duplicados. Nótese que este conjunto de nodos puede ser vacío y no contener ningún nodo o también contener un único nodo.
Un valor lógico (booleano): verdadero o falso.
Un valor numérico: un número en punto flotante.
Una cadena de caracteres.

Las palabras reservadas del lenguaje XPath son:

Ejes: ancestor, ancestor-or-self-atribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self
Selectores de nodos: node(), text(), comment(), procesing-instruction()
Operaciones lógicas: and, or, not()
Operaciones matemáticas: div, mod

Además los símbolos siguientes tienen funcionalidades definidas:

Agrupación de operaciones con paréntesis, ( )
Predicados con corchetes, [ ]
Abreviación de Nodo actual .
Abreviación de Nodo padre ..
Abreviación de Atributo @
Todos los tipos de nodos, *
Separador eje-selector ::
Separador de pasos de localización /
Abreviación del paso de localización *descendant-or-self::node()* //
Referencia a una variable $
Unión de conjuntos de nodos |
Coma, ,
Operaciones lógicas: =, !=, <, >, <=, >=
Operaciones matemáticas: + , - , *

El lenguaje también usa:

Nombres cualificados de los identificadores del documento XML.
Los **nombres de las funciones** que se desean utilizar.
Nombres de las referencias a variables anteponiendo el símbolo $.
Números y literales. Estos últimos entrecomillados con comillas simples o dobles. Pueden anidarse alternando el tipo de comillas.

Ejercicio Resuelto

Caso de uso: colegio.xml

Utilizaremos este documento XML para desarrollar ejemplos. En él guardamos información de un colegio donde hay una estructura de **/colegio/curso/grupo/alumno**.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE colegio SYSTEM "colegio.dtd">
<!-- Documento colegio.xml  -->
<colegio>
    <!-- Datos del colegio -->
    <nombre>Cervantes</nombre>
    <telefono>900102030</telefono>
    <curso nivel="1" etapa="ESO">
        <grupo orden="A">
            <alumno codigo="342">
                <nombre>Ana</nombre>
                <apellidos>Abad Álvarez</apellidos>
                <anno_nac>2018</anno_nac>
                <nota_media>6.5</nota_media>
            </alumno>
            <alumno>
            <alumno>
        </grupo>
        <!-- Grupo B -->
        <grupo orden="B">
            <alumno>
            <alumno>
            <alumno>
        </grupo>
    </curso>
    <curso nivel="2" etapa="ESO">
        <grupo orden="A">
            <alumno>
            <alumno>
            <alumno>
        </grupo>
        <grupo orden="B">
            <alumno>
            <alumno>
            <alumno>
        </grupo>
    </curso>
</colegio>
```

Documento colegio.xml (xml - 3.4 KB) utilizado en el ejemplo.

a) ¿Cómo localizamos al alumnado de la clase de 2 ESO A?

b) ¿Cómo localizamos a todas las alumnas llamadas *Ana*?

c) ¿Cómo se llama el/la alumno/a con la nota más alta?

Mostrar retroalimentación

a) ¿Cómo localizamos al alumnado de la clase de 2 ESO A?

```
/child::colegio/child::curso[@nivel="2"]/child::grupo[@orden="A"]/child::alumno
```

```
/colegio/curso[@nivel="2"]/grupo[@orden="A"]/alumno
```

```
<alumno codigo="534">
  <nombre>Alejando</nombre>
  <apellidos>Álvarez Amate</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>5.25</nota_media>
</alumno>
<alumno codigo="234">
  <nombre>Benito</nombre>
  <apellidos>Benítez Bellido</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>6.5</nota_media>
</alumno>
<alumno codigo="211">
  <nombre>Carmen</nombre>
  <apellidos>Casado Carmona</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>4</nota_media>
</alumno>
```

b) ¿Cómo localizamos a todas las alumnas llamadas *Ana*?

```
descendant-or-self::alumno[nombre="Ana"]<br />
```

```
//alumno[nombre="Ana"]
```

```
<alumno codigo="342">
  <nombre>Ana</nombre>
  <apellidos>Abad Álvarez</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>6.5</nota_media>
</alumno>
<alumno codigo="563">
  <nombre>Ana</nombre>
  <apellidos>Amate Antunez</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>7.5</nota_media>
</alumno>
<alumno codigo="342">
  <nombre>Ana</nombre>
  <apellidos>Antunez Amaya</apellidos>
  <anno_nac>2017</anno_nac>
  <nota_media>2.5</nota_media>
</alumno>
```

c) ¿Cómo se llama el/la alumno/a con la nota más alta?

```
descendant-or-self::alumno[child::nota_media=max(/descendant-or-self::nota_media)]/child::nombre
```

```
//alumno[nota_media=max(//nota_media)]/nombre
```

```
<nombre>Carmen</nombre>
```

2.3.1.- Caminos de localización

Aunque los *caminos de localización* no son la construcción gramatical más general en el lenguaje XPath sí son la construcción más importante.

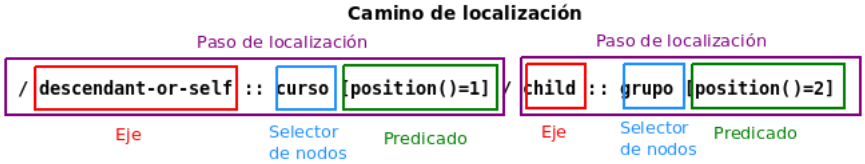
Todo *camino de localización* se puede expresar utilizando una sintaxis completa aunque algo extensa. Existen también ciertas abreviaturas sintácticas que permiten expresar los casos más frecuentes de forma más concisa.

Siguiendo con el ejemplo anterior:

> /descendant-or-self::curso[position()=1]/child::grupo[position()=2]

A esta expresiones se le llama '**camino de localización**' (location paths). Estos caminos están compuestos por trozos separados por barras (/). A estos trozos se les llama '**paso de localización'** (location steps).

Cada *paso de localización,* a su vez, está separado en dos partes por dos puntos dobles (::). A la primera parte se le conoce como '**eje**' (axe) y a la segunda parte como '**selector de nodos**' (node test).



José Antonio Molina Bautista. *Camino de localización* (GNU/GPL)

En algunos *pasos de localización* hay instrucciones entre corchetes [ ] después del selector *de nodos*. Estas instrucciones entre corchetes son '**predicados**'.

Además en los caminos de localización se pueden utilizar **llamadas a funciones**, **operaciones matemáticas** simples y **operaciones lógicas**.

> sum(//nota_media) div count(//nota_media)

> substring(/child::colegio/child::comment(),12,4)

Los caminos de localización pueden ser relativos o absolutos:

Un **camino de localización es absoluto** si comienza por el nodo raíz. Esto se puede reconocer fácilmente ya que el camino de localización comenzará con una barra " / ".

> /child::colegio/child::curso/child::grupo/attribute::*

Si no comienza por el nodo raíz será un **camino de localización relativo**, es decir relativo al nodo contexto que esté posicionado en un determinado lugar. En el siguiente ejemplo el nodo contexto estaría posicionado en un elemento grupo

> child::grupo/attribute::*

Se puede usar el operador | (**unión**) para unir el resultado de dos caminos de localización que devuelvan conjuntos de nodos.

> descendant-or-self::apellidos | descendant-or-self::nota_media

Autoevaluación

¿Los caminos de localización son las únicas expresiones que se pueden realizar en XPath?

○ Verdadero ○ Falso

Falso

2.3.2.- Pasos de localización

Estos *caminos de localización* pueden estar compuestos por uno o más ***pasos de localización***. Cada paso de localización irá refinando la búsqueda de la información que deseemos localizar. Los pasos de localización están separados unos de otros por una barra (***/***).

Un paso de localización tiene tres partes:

Un **eje**, que especifica la relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual,
Un **selector de nodos** o prueba de nodos, que especifica el tipo de nodo de los nodos seleccionados por el paso de localización
Cero o más **predicados**, que usan expresiones lógicas para refinar aún más el conjunto de nodos seleccionado por el paso de localización.

La sintaxis del paso de localización es el nombre de eje y el selector de nodos separados por dos puntos dobles, seguido de cero o más predicados, cada uno entre corchetes.

Por ejemplo, en el siguiente paso de localización:

child::grupo[position()=2]

child es el nombre del eje, grupo es la selector de nodos y [position()=2] es un predicado.

El conjunto de nodos seleccionado por un paso de localización es el que resulta de generar un conjunto de nodos inicial a partir del eje y el selector de nodos, y a continuación filtrar dicho conjunto por cada uno de los predicados sucesivamente. El conjunto de nodos final es el conjunto de nodos seleccionado por el paso de localización.

Autoevaluación

Cada paso de localización ...

- ○ Debe tener obligatoriamente un predicado.
- ○ Debe tener al menos un predicado.
- ○ Puede tener dos predicados

No es cierto.

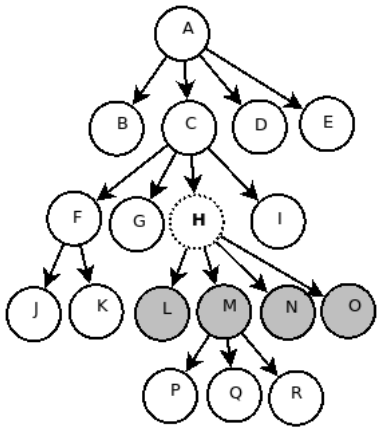
No es obligatorio

Correcto, puede uno, varios o ninguno.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

2.3.3.- Ejes



José Antonio Molina Bautista. *Árbol de nodos* (GNU/GPL)

Los ejes se corresponden con las relaciones entre nodos de un árbol de nodos que vimos en los apartados anteriores.

Estos trece ejes son:

- **self**: nodo contexto.
- **child**: hijos del nodo contexto.
- **descendant**: descendientes del nodo contexto.
- **descendant-or-self**: nodo contexto y sus descendientes
- **parent**: padre.
- **ancestor**: ancestros del nodo contexto.
- **ancestor-or-self**: nodo contexto y sus ancestros.
- **preceding**: nodos anteriores.
- **preceding-sibling**: hermanos anteriores.
- **following**: nodos posteriores
- **following-sibling**: hemanos posteriores
- **attribute**: atributos
- **namespace**: espacio de nombres. En XPath 2.0 se dejó de utilizar.

Recordar que para seleccionar atributos o espacios de nombres hay que utilizar de forma explícita sus ejes correspondientes. Por ejemplo, el eje child de un nodo elemento no contendrá sus atributos ni sus espacios de nombres.

2.3.4.- Selectores de nodos

Los selectores de nodos utilizan los distintos tipos de nodos que vimos al estudiar los árboles de nodos. Cada eje tiene un tipo principal de nodos. Si un eje puede contener elementos, entonces el tipo principal de nodo es elemento; en otro caso, será el tipo de los nodos que el eje contiene. Así,

Para el eje attribute, el tipo de nodo principal son los **atributos**.
Para el eje namespace, el tipo de nodo principal son los **espacios de nombres**.
Para los demás ejes, el tipo de nodo principal son los **elementos**.

Los selectores de nodos son los siguientes:

Nombre cualificado: nombre del objeto que se está buscando.
Todos (*)
text()
comment()
processing-instruction()
node()

En versiones posteriores de XPath se añadieron los siguientes selectores:

element()
attribute()
document-node()

janjf93 (Pixabay License)

Un selector de nodos que sea un **nombre cualificado** (QName) es cierto si los tipos de nodos se corresponden y tiene un identificador igual al especificado por el nombre cualificado. Por ejemplo, child::alumno selecciona los elementos alumno hijos del nodo contexto; si el nodo contexto no tiene ningún hijo alumno, seleccionará un conjunto de nodos vacío. Otro ejemplo: attribute::nivel selecciona el atributo nivel del nodo contexto; si el nodo contexto no tiene atributo nivel, seleccionará un conjunto de nodos vacío.

Un selector de nodos ***** es cierto para cualquier nodo del tipo principal de nodo. Por ejemplo, child::* seleccionará todo elemento hijo del nodo contexto y attribute::* seleccionará todos los atributos del nodo contexto.

El selector de nodos text() es cierto para cualquier nodo de texto. Por ejemplo, child::text() seleccionará los nodos de texto hijos del nodo contexto.

Análogamente, el selector de nodos comment() es cierto para cualquier nodo comentario, y el selector de nodos processing-instruction() es cierto para cualquier instrucción de procesamiento. La prueba processing-instruction() puede tener un argumento que sea literal; en este caso, será verdadera para cualquier instrucción de procesamiento que tenga un nombre igual al valor del literal.

Un selector de nodos node() es cierto para cualquier nodo de cualquier tipo que sea.

Ejemplos:

Selector nombre cualificado: /descendant-or-self::alumno/child::nombre

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Alejando</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carmen</nombre>
```

Selector todos: /descendant-or-self::grupo/attribute::*

```
orden="A"
orden="B"
orden="A"
orden="B"
```

Selector text() : /descendant-or-self::apellidos/text()

```
Abad Álvarez
Benitez Bermúdez
Carmona Casado
Amate Antunez
Bellido Bravo
Cuesta Camacho
Álvarez Amate
Benítez Bellido
```

```
Casado Carmona
Antunez Amaya
Bravo Benítez
Camacho Camacho
```

Selector comment() /descendant-or-self::colegio/child::comment()

```
<!-- Datos del colegio -->
```

Selector node(): /descendant-or-self::grupo/child::node()

```
<alumno codigo="342">
  <nombre>Ana</nombre>
  <apellidos>Abad Álvarez</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>6.5</nota_media>
</alumno>
<alumno codigo="353">
  <nombre>Beatriz</nombre>
   ...
   ...
```

Autoevaluación

¿Qué selector de nodos selecciona todos los nodos?

- ○ element()
- ○ all()
- ○ node()

Incorrecto, no selecciona los nodos atributo.

Incorrecto, ese selector no existe.

Correcto.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

2.3.5.- Sintaxis abreviada

La abreviatura más importante es que child:: puede ser omitida en los pasos de localización. A efectos prácticos, child es el eje por defecto. Por ejemplo, un camino de localización /colegio/curso es abreviatura de child::colegio/child::curso.

Hay también una abreviatura para atributos: attribute:: puede abreviarse como @. Por ejemplo, un camino de localización grupo[@orden="A"] es abreviatura de child::grupo[attribute::orden="A"] y por tanto selecciona hijos grupo con un atributo orden con valor igual a A.

// es abreviatura de descendant-or-self::node() . Por ejemplo,//alumno es abreviatura de /descendant-or-self::node()/child::alumno y por tanto seleccionará cualquier elemento alumno en el documento; curso//alumno es abreviatura de child::curso/descendant-or-self::node()/child::alumno y por tanto seleccionará todos los descendientes alumno de hijos curso.

Un paso de localización . es abreviatura de self::node(). Esto es particularmente útil en conjunción con //. Por ejemplo, el camino de localización.//grupo es abreviatura de

```
self::node()/descendant-or-self::node()/child::grupo
```

y por tanto seleccionará todos los descendientes elementos grupo del nodo contexto.

Análogamente, un paso de localización .. es abreviatura de parent::node(). Por ejemplo, ../apellidos es abreviatura de parent::node()/child::apellidos y por tanto seleccionará los hijos apellidos del padre del nodo contexto.

En resumen:

Si se omite el eje: estamos usando el eje por defecto child.
@ : estamos usando el eje attribute.
// : paso de localización descendant-or-self::node().
.. : paso de localización parent::node().
. : paso de localización self::node().

Ejemplos:

descendant-or-self::node()/child::nota_media se abrevia en //nota_media

```
<nota_media>6.5</nota_media>
<nota_media>8.25</nota_media>
<nota_media>4.75</nota_media>
<nota_media>7.5</nota_media>
<nota_media>2.25</nota_media>
<nota_media>8.75</nota_media>
<nota_media>5.25</nota_media>
<nota_media>6.5</nota_media>
<nota_media>4</nota_media>
<nota_media>2.5</nota_media>
<nota_media>7.75</nota_media>
<nota_media>10</nota_media>
```

child::colegio/child::curso/attribute::etapa se abrevia en colegio/curso/@etapa (camino relativo)

```
etapa="ESO"
etapa="ESO"
```

descendant-or-self::alumno/parent::node()/attribute::* se abrevia en //alumno/../@*

```
orden="A"
orden="B"
orden="A"
orden="B"
```

self::node()/descendant::telefono se abrevia en ./descendant::telefono (camino relativo)

```
<telefono>900102030</telefono>
```

Autoevaluación

¿La abreviación del eje atributo es @ y del eje padre #?

○ Verdadero ○ Falso

False

2.3.6.- Predicados

Un predicado filtra un conjunto de nodos con respecto a un eje y un selector de nodos para producir un nuevo conjunto de nodos. El resultado de la expresión contenida en el predicado será de tipo booleano. Los predicados se escriben dentro de unos corchetes. Un paso de localización puede tener cero, uno o más predicados en cascada. Si existen más de un predicado estos se comprobarán de izquierda a derecha.

Si estamos evaluando un valor numérico el predicado será cierto si el número es igual a la posición contextual y se convertirá en falso en otro caso. Así un camino de localización grupo[2] es equivalente a grupo[position()=2] .

Las operaciones que ponemos usar en los predicados son las siguientes:

- Operadores de comparación: =, !=, <, >, <=, >=
- Operadores lógicos: and, or
- Operadores matemáticos: +, -, *, div, mod

AsoyID (Pixabay License)

Ejemplos:

a) Primer alumno de 2 ESO A.

```
//curso[@nivel="2"]/grupo[@orden="A"]/alumno[1]
```

```
<alumno codigo="534">
  <nombre>Alejando</nombre>
  <apellidos>Álvarez Amate</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>5.25</nota_media>
</alumno>
```

b) Apellidos de los alumnos con nota superior a 8.

```
<span>//alumno[nota_media>8]/apellidos</span>
```

```
<apellidos>Benitez Bermúdez</apellidos>
<apellidos>Cuesta Camacho</apellidos>
<apellidos>Camacho Camacho</apellidos>
```

c) Nombre de los alumnos que no nacieron en el 2017 y tienen media superior a 7.

```
<span>//alumno[anno_nac!=2017][nota_media>7]/nombre</span>
```

```
<span>//alumno[anno_nac!=2017 and nota_media>7]/nombre</span><span></span>
```

```
<nombre>Beatriz</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carmen</nombre>
```

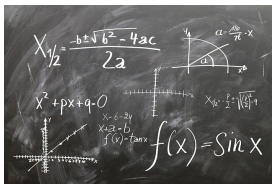
d) Apellidos de los alumnos que nacieron en el 2019 y que están suspensos.

```
<span>//alumno[anno_nac=2019 or nota_media<5]/apellidos</span><span></span>
```

```
<apellidos>Carmona Casado</apellidos>
<apellidos>Bellido Bravo</apellidos>
<apellidos>Casado Carmona</apellidos>
<apellidos>Antunez Amaya</apellidos>
<apellidos>Camacho Camacho</apellidos>
```

2.3.7.- Funciones de conjuntos de nodos

Todas las implementaciones del lenguaje XPath 1.0 deben incluir las siguientes funciones que se podrán usar para evaluar expresiones.



geralt (Pixabay Licese)

Vamos a especificar cada función con un prototipo de función, que da el tipo devuelto, el nombre de la función y el tipo de los argumentos: número, string (cadena de caracteres), booleano o node-set (conjunto de nodos). Si un tipo de argumento es seguido por un signo de interrogación, entonces el argumento es opcional; en otro caso, el argumento es obligatorio.

Estas funciones las podemos clasificar en cuatro grupos:

Funciones de conjuntos de nodos
Funciones de cadenas de caracteres
Funciones lógicas
Funciones numéricas

Funciones de conjuntos de nodos:

last(): devuelve el número total de nodos del contexto seleccionado.

número last()

position(): devuelve la posición del nodo actual dentro de los nodos del contexto seleccionado. Esta posición se enumera desde el 1 no desde el 0 como en otros lenguajes de programación.

número position()

count(): devuelve el número de nodos del conjunto de nodos pasado como argumento.

número count(nodo-set)

name(): devuelve una cadena de caracteres con el nombre cualificado de un nodo del conjunto de nodos pasado como argumento. Formado por el URI del espacio de nombres y el nombre local. Si no se pasa el argumento tomará los nodos contexto como argumentos. Si no se declaran espacios de nombres dará el mismo resultado que la función local-name().

string name(nodo-set?)

local-name(): devuelve el nombre local, sin URI del espacio de nombres de un nodo del conjunto de nodos pasados como argumento. Si no se pasa el argumento tomará los nodos contexto como argumentos. Si no se declaran espacios de nombres dará el mismo resultado que la función name().

string local-name(node-set?)

namespace-uri(): devuelve el URI del espacio de nombres, sin el nombre local, de un nodo del conjunto de nodos pasados como argumento. Si no se pasa el argumento tomará los nodos contexto como argumentos.

string namespace-uri(nodo-set?)

id(): selecciona elementos mediante su identificador único. Los nodos deben estar declarados con el tipo ID en el DTD o XML Schema correspondiente.

node-set id(object)

Ejemplos:

a) Nombre de los primeros alumnos/as de cada grupo.

```
//alumno[position()=1]/nombre
```

```
//alumno[1]/nombre
```

```
<nombre>Ana</nombre>
<nombre>Ana</nombre>
<nombre>Alejando</nombre>
<nombre>Ana</nombre>
```

b) Nombre y apellidos del/a último/a alumno/a de cada grupo de 2 ESO

```
//curso[@nivel="2"]//alumno[last()]/(nombre|apellidos)
```

```
<nombre>Carmen</nombre>
<apellidos>Casado Carmona</apellidos>
<nombre>Carmen</nombre>
<apellidos>Camacho Camacho</apellidos>
```

c) Número total de alumnos/as nacidos/s en 2018

```
count(//alumno[anno_nac=2018])
```

8

d) Número total de alumnos/as de primer curso

count(//curso[@nivel="1"]/grupo/alumno)

6

e) Números de alumnos/as de cada grupo de segundo curso

//curso[@nivel="2"]/grupo/count(alumno)

3
3

Autoevaluación

¿Qué función o expresión nos indica la posición del último nodo del conjunto de nodos actual?

- ○ position(end)
- ○ last()
- ○ first()
- ○ end()

Incorrecto, la palabra end no se usa en el lenguaje XSLT.

Correcto.

Incorrecto, está función no existe en el lenguaje XSLT.

Está función no existe en el lenguaje XSLT. Respuesta incorrecta.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

2.3.8.- Funciones de cadenas de caracteres

Funciones de cadenas de caracteres:

string(): convierte un objeto en cadena de caracteres. Si no se incluye argumentos toma un conjunto de nodos con el nodo contexto como único miembro.

string string(object?)

concat(): devuelve la concatenación de argumentos

string concat(string, string, string*)

starts-with(): devuelve verdadero si la primera cadena argumento empieza con la segunda cadena argumento, devuelve falso en caso contrario.

boolean starts-with(string, string)

contains(): devuelve verdadero si la primera cadena argumento contiene a la segunda cadena argumento, devuelve falso en caso contrario.

boolean contains(string, string)

substring-before(): devuelve la subcadena de la primera cadena argumento que precede a la primera aparición de la segunda cadena argumento en la primera cadena argumento, o la cadena vacía si la primera cadena argumento no contiene a la segunda cadena argumento. Si el segundo argumento es la cadena vacía, entonces se devuelve la cadena vacía.

string substring-before(string, string)

substring-after(): La función substring-after devuelve la subcadena de la primera cadena argumento que sigue a la primera aparición de la segunda cadena argumento en la primera cadena argumento, o la cadena vacía si la primera cadena argumento no contiene a la segunda cadena argumento. Si el segundo argumento es la cadena vacía, entonces se devuelve la primera cadena argumento.

string substring-before(string, string)

substring(): La función substring devuelve la subcadena del primer argumento que comienza en la posición especificada en el segundo argumento y tiene la longitud especificada en el tercer argumento. Si no se especifica el tercer argumento, devuelve la subcadena que comienza en la posición especificada en el segundo argumento y continúa hasta el final de la cadena. Se considera que cada caracter en la cadena tiene una posición numérica: la posición del primer carácter es 1, la posición del segundo carácter es 2 y así sucesivamente.

string substring(string, number, number?)

string-length(): devuelve el número de caracteres en la cadena. Si se omite el argumento, toma por defecto el nodo contexto convertido en cadena de caracteres.

number string-length(string?)

normalize-space(): devuelve la cadena argumento con el espacio en blanco normalizado mediante la eliminación del que se encuentra al principio y al final y la substitución de secuencias de caracteres de espacio en blanco por un solo espacio. Si se omite el argumento, toma por defecto el nodo contexto convertido en cadena de caracteres.

string normalize-space(string?)

translate(): devuelve la cadena primer argumento con las apariciones de caracteres del segundo argumento substituidas por los caracteres en las posiciones correspondientes de la tercera cadena argumento.

string translate(string, string, string)

Ejemplos:

a) Alumnos que su nombre comiencen por 'Al'

```
//alumno[starts-with(nombre,"Al")]
```

```
<alumno codigo="534">
  <nombre>Alejando</nombre>
  <apellidos>Álvarez Amate</apellidos>
  <anno_nac>2018</anno_nac>
  <nota_media>5.25</nota_media>
</alumno>
```

b) Nombre y apellidos de los alumnos que se apelliden 'Carmona' tanto de primero como de segundo apellido.

```
//alumno[contains(apellidos,"Carmona")]/(nombre|apellidos)/text()
```

```
Carlos
Carmona Casado
Carmen
Casado Carmona
```

c) Del texto "En un lugar de la Mancha" extrae todo lo que hay delante de 'de'.

substring-before("En un lugar de la Mancha","de")

```
En un lugar
```

d) Del texto "En un lugar de la Mancha" extrae todo lo que hay detrás de 'de'.

substring-after("En un lugar de la Mancha","de")

```
la Mancha
```

e) Del texto "En un lugar de la Mancha" extrae todo lo que hay detrás de 'de'.

substring-after("En un lugar de la Mancha","de")

```
la Mancha
```

f) Del texto "En un lugar de la Mancha" extrae todo lo que hay a partir del carácter número 9.

substring("En un lugar de la Mancha",9)

```
gar de la Mancha
```

g) Del texto "En un lugar de la Mancha" calcula el número de caracteres que tienes.

string-length("En un lugar de la Mancha")

```
24
```

h) Normaliza un texto, quitando los espacios iniciales y finales y poniendo un único espacio entre palabras.

normalize-space(" Esto es una prueba ")

```
Esto es una prueba
```

g) Sustituir los caracteres en el primer texto que aparecen en el segundo texto por los del tercero.

translate("abcdefghi","bdg","BDG")

```
aBcDefGhi
```

2.3.9.- Funciones lógicas y numéricas

Funciones lógicas

boolean(): convierte su argumento en booleano.

boolean boolean(object)

not(): devuelve verdadero si su argumento es falso, y falso en otro caso.

boolean not(boolean)

true(): devuelve verdadero.

boolean true()

false(): devuelve falso.

boolean false()

lang(): devuelve verdadero o falso dependiendo de si el lenguaje del nodo contextual tal como se especifica por los atributos xml:lang es el mismo que, o es un sublenguaje de, el lenguaje especificado por la cadena argumento.

boolean lang(string)

Funciones numéricas

number(): convierte su argumento en un número. Si se omite el argumento, toma por defecto un conjunto de nodos con el nodo contexto como único miembro.

número number(object?)

sum(): devuelve la suma, a lo largo de todos los nodos del conjunto de nodos argumento, del resultado de convertir los valores de las cadena de caracteres de los distintos nodos en números.

número sum(node-set)

floor(): redondea hacia abajo. Devuelve el mayor número entero que sea menor o igual al argumento.

número floor(number

ceiling(): redondea hacia arriba. Devuelve el menor número entero que sea mayor o igual al argumento.

número ceiling(number)

round(): redondea. Devuelve el número que esté más próximo al argumento y que sea entero.

número round(number)

Ejemplos:

a) Alumnos que no se llamen 'Carmen'

```
//alumno[not(nombre="Carmen")]/nombre
```

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Alejando</nombre>
<nombre>Benito</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
```

b) Suma de las notas de los/as alumnos/as de 1 ESO A

```
sum(//curso[@nivel="1"]/grupo[@orden="A"]/alumno/nota_media)
```

```
19.5
```

c) Número de notas de los/as alumnos/as de 1 ESO A

```
count(//curso[@nivel="1"]/grupo[@orden="A"]/alumno/nota_media)
```

```
3
```

d) Media de notas de los/as alumnos/as de 1 ESO A. Usando los dos cálculos anteriores.

```
( sum(//curso[@nivel="1"]/grupo[@orden="A"]/alumno/nota_media)<br /> div<br /> count(//curso[@nivel="1"]/grupo[@orden="A"]/alumno/nota_media) )
```

```
6.5
```

En XPath 2.0 tenemos la función avg() que nos hace la media de una forma directa.

```
avg(//curso[@nivel="1"]/grupo[@orden="A"]/alumno/nota_media)
```

```
6.5
```

e) Redondea la nota 4.75 a un valor entero.

```
round(4.75)
```

```
5
```

f) Redondea 'por encima' la nota 4.75 a un valor entero.

```
ceiling(4.75)
```

```
5
```

g) Redondea 'por debajo' la nota 4.75 a un valor entero.

```
floor(4.75)
```

```
4
```

2.4.- Estrategias de uso

¿Dónde posicionar los predicados?

Queremos saber el **grupo del alumno que se apellida 'Bellido Bravo'**. Primero buscamos en la estructura de nuestro documento XML la posición del atributo y del elemento que se indican en el enunciado.

colegio/curso/grupo/alumno/apellidos
colegio/curso/grupo/@orden

A continuación sabemos que el resultado debe ser el número de orden, es decir, nuestro camino de localización debe acabar en este atributo. Ese será nuestro punto de partida. Podemos probar esa consulta colegio/curso/grupo/@orden antes de filtrar el el apellido dado.

```
orden="A"
orden="B"
orden="A"
orden="B"
```

Wikimedia, *Estrategia* (Dominio público)

Para posicionar el predicado podemos utilizar varias estrategias. Una de ellas puede ser poner le predicado en el primer elemento padre común. En nuestro ejemplo sería grupo.

colegio/curso/grupo[alumno/apellidos="Bellido Bravo"]

Si probamos esta consulta obtendremos el grupo que buscábamos. Pero nos devuelve todo el grupo no solamente su orden.

Si mezclamos ambos caminos de localización en el elemento común que habíamos determinado antes obtendremos lo buscado. Es decir usaremos el camino de localización del atributo que queremos mostrar con el predicado que situaremos en el elemento común

colegio/curso/grupo/@orden
grupo[alumno/apellidos="Bellido Bravo"]
colegio/curso/grupo[alumno/apellidos="Bellido Bravo"]/@orden

Y simplificando con doble barra la parte inicial:

//grupo[alumno/apellidos="Bellido Bravo"]/@orden

También sería válida la siguiente consulta:

//alumno[apellidos="Bellido Bravo"]/../@orden

```
orden="B"
```

Cuando trabajamos con información en distintos niveles de nuestra estructura es importante no utilizar la doble barra en la mitad de los caminos de localización ya que la búsqueda la reiniciará desde el elemento raíz y no con el predicado ya establecido. Si probáis la siguiente sentencia veréis la diferencia.

//grupo[//apellidos="Bellido Bravo"]/@orden

```
orden="A"
orden="B"
orden="A"
orden="B"
```

Vemos también la utilidad de usar el . (nodo actual) cuando necesitamos bajar niveles de jerarquía y .. (nodo padre) cuando queremos subir niveles.

Consultas anidadas

En ocasiones necesitamos hacer varias consultas anidadas, es decir, que necesitamos el resultado de una consulta para realizar una segunda consulta.

Por ejemplo, si queremos saber la nota media de los/as alumnos/as que tienen una nota media menor que la del/a alumno/a que se apellida 'Abad Álvarez'.

Tendremos que hacer dos consultas:

La primera para saber la nota media de 'Abad Álvarez' y la otra para saber los/s alumnos/as que tienen una nota media inferior a un número cualquier, por ejemplo 5.

```
//alumno[apellidos="Abad Álvarez"]/nota_media
```

```
<nota_media>6.5</nota_media>
```

```
//alumno[nota_media < 5]/nota_media
```

```
<nota_media>4.75</nota_media>
<nota_media>2.25</nota_media>
<nota_media>4</nota_media>
<nota_media>2.5</nota_media>
```

Ahora tendremos que mezclar ambas consultas y en lugar de el número 5 de antes poner la subconsulta que nos muestra la nota de 'Abad Álvarez'.

```
//alumno[number(nota_media) < <span style="text-decoration: underline;">//alumno[apellidos="Abad Álvarez"]/nota_media</span> ]/nota_media
```

```
<nota_media>4.75</nota_media>
<nota_media>2.25</nota_media>
<nota_media>5.25</nota_media>
<nota_media>4</nota_media>
<nota_media>2.5</nota_media>
```

* Se usa la función number() para evitar la comparación de orden alfabético lo que haría que 10 sea menor que 6.5. Ya que alfabéticamente la letra '6' esta después (es mayor) de la letra '1'.

Expresión para evitar elementos repetidos

En ocasiones queremos mostrar todos los elementos de un determinado tipo pero sin que salgan repetidos. Gracias al eje preceding podemos hacer lo una forma sencilla. Recordad que este eje no tiene abreviación por lo que su notación es la completa.

Mostramos todos los nombres de los alumnos

```
<strong><span style="text-decoration: underline;">//alumno/nombre</span></strong>
```

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Alejando</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carmen</nombre>
```

Y ahora los nombres de los alumnos sin repetir

```
<strong><span style="text-decoration: underline;">//alumno[not(nombre=preceding::nombre)]/nombre</span></strong>
```

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Alejando</nombre>
```

Autoevaluación

XPath no permite consultas anidadas:

○ Verdadero ○ Falso

False

3.- XSL Transformations (XSLT)

Caso práctico

María ya tiene claro que es posible seleccionar distintos elementos de los documentos XML con los que trabajan en su empresa. Pero el formato de salida no es nada legible. Los usuarios están acostumbrados a ver la información en formatos más visuales como HTML.

Félix está interesado en mostrar la información en la web pero no quiere realizar trabajo extra.

Juan les explica que esto es posible. El lenguaje XSLT permite crear documentos HTML con la información aportada por documentos XML. Que en dichos documentos HTML se puede seleccionar y filtrar la información que se desee y presentar en todos los formatos que HTML permita.

Además gracias a las plantillas se puede estandarizar las distintas páginas web y crear un estilo homogéneo y más profesional.

El lenguaje de Transformaciones XSL (Extensible Stylesheet Language Transformations - XSLT) es una recomendación aprobada por el consorcio W3C. Su versión 1.0 fue aprobada el 16 de noviembre de 1999. El mismo día que la recomendación XPath 1.0.

Gracias a este lenguaje los procesadores XSLT pueden transformar un documento XML en otros documentos XML, HTML o de texto plano con estructuras y contenidos distintos de los originales. Esta transformación se realiza gracias al uso de hojas de estilos (xsl:stylesheet). El lenguaje XSLT establece la sintaxis y la semántica para construir dichas hojas de estilos.

El lenguaje XSLT es un lenguaje derivado de XML por lo que podremos comprobar si un documento XSLT está bien formado (well-formed) y si es válido frente a un vocabulario.

Una transformación expresada en XSLT describe reglas para transformar un árbol de nodos origen en un árbol de nodos resultado.

La transformación se logra asociando patrones definidos en la plantilla (xsl:template). Un patrón, expresado mediante XPath, se compara con los elementos del árbol de origen. Si cumplen con alguna de las reglas de la plantilla estos pasan a formar parte del árbol de nodos resultados. Es importante resaltar que el árbol de nodos resultado es distinto del árbol de nodos origen. Además la estructura del árbol de resultado puede ser completamente diferente de la estructura del árbol de origen. Al construir el árbol de resultado los elementos del árbol de origen se pueden filtrar y reordenar, y se puede agregar una estructura arbitraria.

Para saber más

Puedes ver la recomendación XSLT 1.0 completa en la web del consorcio W3C.

Si quieres ampliar información de las distintas versiones de XSLT puedes buscar en el apartado correspondiente de la web del consorcio W3C.

Autoevaluación

¿Cuáles de los siguientes formatos de salida permite el lenguaje XSLT?

- [ ] HTML
- [ ] PDF
- [ ] Texto plano
- [ ] Microsoft Word

Mostrar retroalimentación

Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Incorrecto

3.1.- Hojas de estilos XSLT

A la hora de ir creando el árbol de nodos destino que generará el documento de salida podemos usar los siguientes elementos:

Elementos del espacio de nombres asociado al URI:
http://www.w3.org/1999/XSL/Transform. Se suele usar el prefijo xsl. Son las instrucciones que usa el lenguaje XSLT. Ejemplos: xsl:stylesheet, xsl:template, xsl:value-of, xsl:for-each,...
Elementos de extensión. Se utilizan como mecanismos implementados por los distintos desarrolladores para aportar funcionalidades extra. Cada desarrollador determinará su uso. No los usaremos en esta unidad.
Elementos de resultado literal (LRE, literal result element). Son elementos que se añaden al árbol de nodos resultado que no pertenecen al espacio de nombre xsl. Por ejemplo cuando incluimos elementos HTML o texto entre la información obtenida del documento XML.

El lenguaje XSLT no especifica cómo se asociar una hoja de estilos XSLT a un documento XML. Para eso se utiliza un mecanismo propio del estándar XML, las instrucciones de procesamiento. Debe tenerse en cuenta que algunos navegadores por seguridad bloquean estas instrucciones de procesamiento. Un ejemplo de documento XML asociado a una hoja de estilos XSLT podría ser el siguiente:

En este ejemplo utilizamos la instrucción de procesamiento xml-stylesheet indicando en el atributo type el tipo de archivo con la nomenclatura MIME (text/xml o **`text/xsl`**) y en el atributo href la ubicación de la hoja de estilos a utilizar.

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

Esta instrucción de procesamiento no es obligatoria, ya que al ejecutar el procesador XSLT se le puede indicar como parámetros los documentos XML y XSL sobre los que queremos realizar la transformación.

Para crear una hoja de estilos utilizamos la instrucción xsl:stylesheet. Para usar esta instrucción necesitamos declarar el espacio de nombres del lenguaje XSLT que se corresponde con la URI: http://www.w3.org/1999/XSL/Transform. Se suele asociar dicho espacio de nombres al prefijo xsl:. Además esta instrucción tiene un atributo obligatorio, version, en el que indicaremos la versión del lenguaje XSLT en la que hemos codificado nuestra hoja de estilos.

Normalmente cada hoja de estilos se creará en un archivo de texto plano con extensión .xsl. Como dijimos anteriormente el lenguaje XSLT es derivado de XML por lo que será necesario incluir en la primera línea el prólogo XML. Posteriormente declararemos nuestra hoja de estilos como en el siguiente ejemplo:

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

También podemos usar la instrucción xsl:transform para crear hojas de estilos ya que funciona como un sinónimo de la instrucción xsl:stylesheet.

Para saber más

Puedes ampliar la información sobre cómo asociar hojas de estilo a documentos XML en el siguiente enlace.

https://www.w3.org/TR/xml-stylesheet/

Autoevaluación

¿Con qué instrucción asociamos un documento XSLT de transformaciones con un documento XML?

- ○
- ○
- ○
- ○

Incorrecto, esa instrucción la utilizamos para indicar el prólogo XML.

Incorrecto, esa instrucción se utiliza para indicar el vocabulario DTD que validará nuestro documento XML.

Correcto.

Incorrecto, esa instrucción es del lenguaje XSLT y se utiliza para declarar hojas de estilos.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

3.1.1.- Simplificación de hojas de estilos usando elementos de contenido literal

XSLT permite una sintaxis simplificada para las hojas de estilo que consten de una plantilla para el nodo raíz. La sintaxis simplificada permite omitir la estructura de xsl:stylesheet cuando se trate de un elemento de contenido literal. Además en lugar de declarar una plantilla xsl:template esta se sustituirá directamente por dicho elemento de contenido literal. Desde este contenido literal se podrá acceder a las instrucciones del espacio de nombres xsl: como si se hubiera seleccionado como patrón el nodo raíz / del árbol de nodos origen.

En este caso de simplificación dicho elemento literal debe tener el atributo xsl:version. Para utilizar dicho atributo y si se quiere utiliza además alguna instrucción del espacio de nombres xsl: se deberá añadir dicho espacio de nombres al citado elemento literal. Este elemento que actúa de plantilla no podrá contener instrucciones de nivel superior.

Con nuestro documento de ejemplo colegio.xml podemos realizar la siguiente transformación:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Colegio Cervantes</title>
      </head>
      <body>
        <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
        <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
        <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Y que podríamos simplificar en:

```
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <head>
    <title>Colegio Cervantes</title>
  </head>
  <body>
    <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
    <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
    <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
  </body>
</html>
```

En esta simplificación observamos que el elemento de contenido literal html contiene la declaración del espacio de nombres xmlns del lenguaje XSLT y el atributo obligatorio version. Posteriormente haciendo uso del prefijo xsl: podemos utilizar instrucciones que no sean de nivel superior como xsl:value-of.

3.2.- Elementos XSLT de nivel superior.

Si consideramos un documento XSLT como un documento XML el elemento raíz será un elemento xsl:stylesheet o xsl:transform. Los elementos de nivel superior son aquellos del espacio de nombres xsl: que son hijos directos de alguno de estos dos elementos. Tienen un tratamiento especial ya que su ámbito de aplicación es toda la hoja de estilos que estamos declarando.

Los elementos de nivel superior más utilizados y que veremos con más detalle son:

xsl:template
xsl:variable y xsl:param
xsl:output

Otros elementos de nivel superior son:

xsl:import y xsl:include sirven para añadir contenido a las hoja de estilo que estamos desarrollando. Los contenido añadidos mediante xsl:import tienen menor preferencia que los desarrollados en nuestra hoja de estilos. La preferencia de los contenidos añadidos mediante xsl:include es la misma que los desarrollados en nuestra hoja de estilos. Ambos elementos tienen un atributo href que indica dónde se encuentra los contenidos a añadir.
xsl:strip-space y xsl:preserve-space: sirven para indicar al procesador XSLT como debe actuar al normalizar los espacios en blanco.
xsl:key: sirve para declarar un elemento como clave de modo que pueda usarse como los tipos de datos ID, IDREF y IDREFS. Declara una clave con nombre que se puede usar en cualquier otro lugar de la hoja de estilos con la función key().
xsl:decimal-format: define el formato que se utilizará para convertir números en cadenas de caracteres.
xsl:namespace-alias: sirve para definir un prefijo alternativo para un espacio de nombres.
xsl:attribute-set: se utiliza para crear grupos de atributos que se pueden reutilizar en los elementos xsl:element y `xsl:copy` mediante el atributo use-attribute-sets.

Estos elementos de nivel superior pueden aparecer varias veces o no aparecer. El orden de aparición no es relevante excepto para xsl:import que debe aparecer en primer lugar. Además estos elementos de nivel superior no pueden usarse dentro de otros elementos excepto xsl:variable y xsl:param.

Autoevaluación

Relaciona los siguientes elementos con sus funcionalidades.

Ejercicio de relacionar.

Elemento	Relación	Funcionalidad
xsl:strip-space	0	1. Añade contenido desde otros archivos.
xsl:import	0	2. Normaliza el espaciado del contenido de los elementos.
xsl:key	0	3. Define el formato de conversión de números a cadena de caracteres.
xsl:decimal-format	0	4. Estable un elemento como clave.

Enviar

3.2.1.- xsl:template

Las plantillas (xsl:template) constan de dos partes: un patrón, expresado mediante XPath en el atributo match, que se compara con los nodos del árbol de origen y unas instrucciones que se aplicarán a los nodos seleccionados. Con la aplicación de esas instrucciones y la incorporación de elementos de contenido literal podremos ir construyen los nodos que necesitemos en el árbol de nodos resultado. Esto permite que una hoja de estilo sea aplicable a distintos documentos XML siempre que tengan la misma estructuras.

El atributo match es obligatorio a no ser que exista un atributo name. El atributo name sirve para hacer referencia a una plantilla mediante la instrucción xsl:call-template. El valor del atributo match no puede incluir el uso de variables.

El contenido del elemento xsl:template serán las reglas que se ejecutarán cuando la plantilla sea utilizada.

En una hoja de estilos podemos tener más de una plantilla. La ejecución de plantillas tiene una serie de abreviaciones que si no se tienen en cuentas pueden llegar a confundir. Cada plantilla se puede hacer coincidir con un determinado nodo de nuestra estructura. Veamos unos ejemplos de distintas construcciones de plantillas:

Plantillas vacías: Si tenemos una plantilla que se corresponde con un nodo de nuestra estructura y esta plantilla no tiene contenido entonces al recorrer esos nodos no mostrará nada.
Nodos sin plantillas asociadas: Si de un determinado nodo de nuestra estructura no creamos plantilla por simplificación mostrará el contenido textual de ese determinado nodo. Pero no mostrará el valor de los atributos.

En el siguiente ejemplo tenemos una plantilla asociada a los nodos de tipo 'alumno' pero está vacía es decir cuando recorra esos nodos mostrará el contenido de esa plantilla que es nada. Además en nuestra estructura tenemos otros nodos que no tienen correspondencia con el atributo match de ninguna plantilla (colegio, nombre, teléfono, curso y grupo) por lo que por simplificación mostrará el contenido textual de esos nodos sin plantilla.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml"/>
  <xsl:template match="//alumno">
    <!-- No hacer nada -->
  </xsl:template>
</xsl:stylesheet>
```

El resultado será

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

    Cervantes
    900102030
```

En ocasiones cuando tenemos más de una plantilla vemos que no funciona tal como esperábamos. Por ejemplo, si tenemos una plantilla para 'alumno' que nos muestra el 'nombre' y otra plantilla para 'apellidos' que nos vuestra el propio elemento '.' vemos que no funciona correctamente:

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

  <xsl:template match="//alumno">
    <xsl:value-of select="nombre"/>
  </xsl:template>

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

</xsl:stylesheet>
```

El resultado es solamente lo que muestra la primera plantilla:

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

    Cervantes
    900102030

            Ana
            Beatriz
            Carlos

            Ana
            Benito
            Carmen
```

Pero esto a qué es debido. El motivo es el recorrido del árbol de nodos origen. Cuando recorremos en la primera plantilla los elementos 'alumno' podemos mostrar perfectamente su nombre. Pasamos al siguiente 'alumno' y volvemos a mostrar su 'nombre' y así hasta el último 'alumno'. Cuando vamos a aplicar la segunda plantilla ya hemos terminado de recorrer todo el árbol de nodos origen y no podemos volver hacia detrás. Ya veremos que esto puede funcionar pero necesitamos de la instrucción xsl:apply-templates.

Sabiendo esto comprendemos porque el siguiente ejemplo sí muestra ambos valores: nombre y apellidos, ya que los recorre cuando nos encontramos en el nodo 'alumno'.

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

  <xsl:template match="//alumno">
    <xsl:value-of select="nombre"/>
    <xsl:value-of select="apellidos"/>
  </xsl:template>

</xsl:stylesheet>
```

Una forma habitual de simplificar el uso de las plantillas es usar una única plantilla que acceda al elemento raíz. Dentro de esta única plantilla podremos utilizar expresiones XPath para seleccionar las partes de documento XML que queramos e instrucciones de recorrido xsl:for-each y selección xsl:if para construir la estructura que deseemos. Por ejemplo esta estructura con llamadas anidadas mediante el uso de plantillas:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" />
  <xsl:strip-space elements="*"/>
  <xsl:template match="/">
    <xsl:apply-templates select="//alumno"/>
  </xsl:template>
  <xsl:template match="//alumno">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="//anno_nac">
    <xsl:text>privado</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Se puede simplificar a una sola plantilla con expresiones XPath e instrucciones xsl:for-each y **`xsl:if`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" />
  <xsl:template match="/">
        <xsl:for-each select="//alumno">
          <xsl:value-of select="nombre"/>
          <xsl:value-of select="apellidos"/>
          <xsl:text>privado</xsl:text>
          <xsl:value-of select="nota_media"/>
        </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

El resultado es el mismo:

```
AnaAbad Álvarezprivado6.5BeatrizBenitez Bermúdezprivado8.25CarlosCarmona Casadoprivado4.75AnaAmate Antunezprivado7.5Be ...
```

3.2.2.- xsl:variable y xsl:param

Una variable es un nombre que se vincula a un valor. El valor de la variable puede ser de los distintos tipos que devuelven las expresiones y funciones XPath. Hay dos elementos que utilizan esta vinculación de valores xsl:variable y xsl:param.

Tanto xsl:variable como xsl:param tienen un atributo name obligatorio. En el momento de utilizar el contenido de la variable hay que especificar el nombre del atributo name con un simbolo $ delante.

El otro atributo de estos elementos es select que si se le especifica una expresión XPath almacenará su resultado en la variable.

La diferencia es que el valor de xsl:param es un valor predeterminado que se usa normalmente pero puede ser sobreescrito cuando se invoca a una plantilla o una hoja de estilos mediante un elemento xsl:with-param que tenga el mismo nombre (name) pero distinto valor que el predeterminado.

Ejemplo:

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

    <xsl:variable name="mensaje" >Hola mundo</xsl:variable>
    <xsl:value-of select="$mensaje" />

    <xsl:text>&#10;</xsl:text>

    <xsl:variable name="nombre_cole" select="colegio/nombre" />
    <xsl:value-of select="$nombre_cole"/>

  </xsl:template>
</xsl:stylesheet>


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

    <xsl:param name="mensaje">Hola mundo</xsl:param>
    <xsl:value-of select="$mensaje" />

    <xsl:text>&#10;</xsl:text>

    <xsl:param name="nombre_cole" select="colegio/nombre" />
    <xsl:value-of select="$nombre_cole"/>

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

3.2.3.- xsl:output

Con esta instrucción especificamos el formato de salida deseado. Esta instrucción se puede omitir ya que el procesador XSLT devolverá el resultado como secuencias de caracteres. El uso de este elemento nos facilita poder especificar características de la salida. Los atributos más usados son:

method que especifica el formato de salida. Tiene los siguiente valores predeterminados: xml, **`html`** y text.
version con el que podemos especificar la versión del método de salida.
indent para especificar si se deben añadir espacios en blanco adicionales. Los posibles valores son: yes o **`no`**.
encoding para especificar la codificación.
standalone para especificar si el documento tendrá relación con otros documentos. Los posibles valores son: yes o **`no`**.
Otros atributos son: media-type, doctype-system, doctype-public, omit-xml-declaration, cdata-section-elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" encoding="UTF-8" indent="yes" />
  ....
</xsl:stylesheet>
```

Con el siguiente ejemplo probamos a cambiar el atributo indent entre los valores yes y no.

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

</xsl:text>
   <xsl:element name="ciclo_superior">
     <xsl:element name="primer_curso">
       <xsl:element name="modulo" >LMSGI</xsl:element>
       <xsl:element name="modulo" >SI</xsl:element>
     </xsl:element>
     <xsl:element name="segundo_curso" >
       <xsl:element name="modulo" >SAD</xsl:element>
       <xsl:element name="modulo" >SI</xsl:element>
     </xsl:element>
   </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Los resultados son los siguientes:

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

<ciclo_superior>
    <primer_curso>
        <modulo>LMSGI</modulo>
        <modulo>SI</modulo>
    </primer_curso>
    <segundo_curso>
        <modulo>SAD</modulo>
        <modulo>SI</modulo>
    </segundo_curso>
</ciclo_superior>
```

y

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

<ciclo_superior><primer_curso><modulo>LMSGI</modulo><modulo>SI</modulo></primer_curso><segundo_curso><modulo>SAD</modulo><modulo>SI</modulo></
```

3.3.- Resto de elementos XSLT

El resto de elementos del lenguaje XSLT más utilizados y que veremos a continuación podemos clasificarlos en las siguientes categoría:

Instrucciones de manipulación de plantillas:

xsl:apply-templates.
xsl:call-template.
xsl:with-param.

Instrucciones de control:

xsl:for-each
xsl:if
xsl:choose
xsl:when
xsl:otherwise
xsl:sort

Instrucciones de salida

xsl:value-of
xsl:number
xsl:element
xsl:attribute
xsl:text
xsl:comment
xsl:processing-instruction

Otras instrucciones XSLT son:

xsl:copy: con esta instrucción podemos añadir el nodo actual al árbol de nodos destino. Al copiar un elemento también se copia sus nodos espacio de nombre pero no se copian sus nodos atributos ni sus elementos hijos. Este elemento puede usar su atributo use-attribute-sets para añadir conjuntos de atributos declarados anteriormente. Sólo se puede usar esta instrucción con nodos que puedan tener atributos o elementos hijos, es decir, nodo raíz y nodos elemento.
xsl:copy-of: con esta instrucción podemos añadir un fragmento completo del árbol de nodos origen en el árbol de nodos destino. No es necesario convertir ese fragmento a cadenas de caracteres. Tiene un atributo obligatoria select donde debe indicarse una expresión XPath con el fragmento de árbol a copiar. Cuando se copia un elemento se copia el propio elemento y también sus nodos atributo, sus nodos espacio de nombres y sus elementos hijos.
xsl:apply-imports: Sirve para indicar cuándo realizar una importación para sobreescribir una regla de plantillas.
xsl:message: Se utiliza para enviar mensajes al procesador XSLT. Tiene un atributo terminate que puede tomar los valores: yes o `no` que se utilizar para indicar al procesador XSLT que interrumpa el procesamiento de la hoja de estilos.
xsl:fallback: Su función es establecer una secuencia ordenada de ejecución.

Autoevaluación

¿Qué diferencia hay entre las instrucciones xsl:copy y `xsl:copy-of`?

- ○ xsl:copy Copia el nodo actual, sus atributos, sus espacios de nombres y sus elementos hijos en el árbol de nodos destino.
- ○ xsl:copy-of Copia el nodo actual, sus atributos, sus espacios de nombres y sus elementos hijos en el árbol de nodos destino.
- ○ xsl:copy-of Copia solamente el nodo actual en el árbol de nodos destino.

Incorrecto, xsl:copy sólo copia el nodo actual.

Correcto.

Incorrecto, xsl:copy-of copia el nodo actual, sus atributos, sus espacios de nombres y sus elementos hijos en el árbol de nodos destino.

Solución

1. Incorrecto

2. Opción correcta
3. Incorrecto

3.3.1.- Instrucciones de manipulación de plantillas

xsl:apply-templates

Como vimos con la instrucción xsl:template cuando comparamos el árbol de nodos origen con las expresiones match, por defecto, no se continua buscando más plantillas recorriendo los elementos hijo. Es decir se pasa al siguiente elemento hermano que coincide con la expresión match.

Con la instrucción xsl:apply-templates podemos indicar que sí queremos que se continúe la búsqueda de plantillas en los elementos hijos. Esta instrucción tiene un atributo select. En dicho atributo podemos indicar, mediante una expresión XPath, el conjunto de nodos hijos que queremos que se continúe recorriendo y buscando plantillas que se correspondan con ellos. Si no usamos en atributo select estaremos indicando que se recorran todos los elementos hijos en busca de plantillas.

Como contenido de la instrucción xsl:apply-template puede aparecer la instrucción xsl:sort que nos ordenará los resultados antes de aplicar la plantilla que corresponda. También podemos encontrar la instrucción xsl:with-param que nos permite pasar valores a la plantilla. Estos valores sobreescribirán los valores por defecto indicados por una instrucción xsl:param.

En el siguiente ejemplo vemos el uso de xsl:apply-template. Vemos como se van invocando a los elementos desde el elementos raíz hacia los elementos hijo. En unos casos usando el atributo select y en otros no. En la segunda plantilla al no indicar select aplica a todos los elementos hijo (nombre, apellidos, anno_nac y nota_media). Estos elementos al no tener plantillas, por defecto, mostrarán su contenido textual. Excepto el elemento anno_nac que sí tiene plantilla específica que en lugar de mostrar su contenido muestra el texto 'privado'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" />
  <xsl:strip-space elements="*"/>

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

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

  <xsl:template match="//anno_nac">
    <xsl:text>privado</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Una forma habitual de simplificar el uso de las plantillas es usar una única plantilla que acceda al elemento raíz. Dentro de esta única plantilla podremos utilizar expresiones XPath para seleccionar las partes de documento XML que queramos e instrucciones de recorrido xsl:for-each y selección xsl:if para construir la estructura que deseemos. Por ejemplo la estructura anterior con llamadas anidadas a distintas plantillas se puede simplificar a una sola plantilla con expresiones XPath e instrucciones xsl:for-each y **`xsl:if`**.

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

  <xsl:template match="/">
        <xsl:for-each select="//alumno">
          <xsl:value-of select="nombre"/>
          <xsl:value-of select="apellidos"/>
          <xsl:text>privado</xsl:text>
          <xsl:value-of select="nota_media"/>
        </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

El resultado es el mismo:

```
AnaAbad Álvarezprivado6.5BeatrizBenitez Bermúdezprivado8.25CarlosCarmona Casadoprivado4.75AnaAmate Antunezprivado7.5Be ...
```

Otras instrucciones de manipulación de plantillas son:

xsl:call-template: Permite invocar a una plantilla que esté declarada con un atributo name.
xsl:with-param: Permite pasar valores a una plantilla que sobreescribirán los indicados por el elemento xsl:param que tengan el mismo atributo name.

3.3.2.- Instrucciones de control

xsl:for-each

Con esta instrucción podemos hacer un bloque repetitivo que recorra de uno en uno todos los nodos indicados mediante una expresión XPath en el atributo select. Dicho atributo es obligatorio. El valor indicado deben ser un conjunto de nodos. El recorrido de los nodos se hace en el orden en el que son procesados a no ser que se utilice la instrucción xsl:sort.

xsl:if

Con esta instrucción podemos hacer un bloque condicional. De modo que sólo ejecutemos una parte de la plantilla si se cumple una determinada condición. Esta condición debe indicarse en el atributo test. La expresión de la condición se evaluará y se convertirá al tipo de dato boolean. Si el resultado es true entonces se ejecutará el contenido del elemento xsl:if. Si el resultado no es true se pasará a la siguiente instrucción después del elemento xsl:if.

Vamos a realizar una transformación en la que vamos a ordenar los alumnos por orden de apellido descendente y de cada uno de ellos mostraremos su nombre y su fecha de nacimiento o su nota media dependiendo de si su posición es par o impar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:for-each select="//alumno">
            <xsl:sort select="apellidos" order="descending"/>
            <li>
              <xsl:if test="((position() mod 2)=1)">
                <xsl:value-of select="nombre"/>
                <xsl:text>. Fecha de nacimiento: </xsl:text>
                <xsl:value-of select="anno_nac"/>
              </xsl:if>
              <xsl:if test="((position() mod 2)=0)">
                <xsl:value-of select="nombre"/>
                <xsl:text>. Nota media: </xsl:text>
                <xsl:value-of select="nota_media"/>
              </xsl:if>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Esta misma transformación también se podría hacer sustituyen el xsl:for-each por una llamada recursiva a otra plantilla mediante xsl:apply-templates.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:apply-templates select="//alumno">
            <xsl:sort select="apellidos" order="descending"/>
          </xsl:apply-templates>
        </ol>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="alumno">
    <li>
      <xsl:if test="((position() mod 2)=1)">
        <xsl:value-of select="nombre"/>
```

```
        <xsl:text>. Fecha de nacimiento: </xsl:text>
        <xsl:value-of select="anno_nac"/>
      </xsl:if>
      <xsl:if test="((position() mod 2)=0)">
        <xsl:value-of select="nombre"/>
        <xsl:text>. Nota media: </xsl:text>
        <xsl:value-of select="nota_media"/>
      </xsl:if>
    </li>
  </xsl:template>

</xsl:stylesheet>
```

El resultado de ambas transformaciones sería;

```
<html>
    <body>
        <ol>
            <li>Carmen. Fecha de nacimiento: 2017</li>
            <li>Carmen. Nota media: 4</li>
            <li>Carlos. Fecha de nacimiento: 2017</li>
            <li>Carmen. Nota media: 10</li>
            <li>Beatriz. Fecha de nacimiento: 2018</li>
            <li>Beatriz. Nota media: 8.25</li>
            <li>Benito. Fecha de nacimiento: 2018</li>
            <li>Benito. Nota media: 2.25</li>
            <li>Ana. Fecha de nacimiento: 2017</li>
            <li>Ana. Nota media: 7.5</li>
            <li>Alejando. Fecha de nacimiento: 2018</li>
            <li>Ana. Nota media: 6.5</li>
        </ol>
    </body>
</html>
```

xsl:choose, xsl:when y xsl:otherwise

Esta instrucción es parecida a la instrucción xsl:if pero permite evaluar más de un caso posible. Consiste en una secuencia de elementos xsl:when seguidos de un elemento xsl:otherwise opcional. Cada elemento xsl:when tiene un único atributo test, que especifica una expresión. Cuando se procesa un elemento xsl:choose, cada uno de los elementos xsl:when es evaluado en orden. Sólo se ejecutará el primer y solo el primer elemento xsl:when que haya devuelto un resultado true . Si no hay ningún xsl:when con resultado true se ejecutará el contenido del elemento xsl:otherwise. Si ningún elemento xsl:when es verdadero y no se ha indicado ningún elemento xsl:otherwise no se ejecutará nada.

xsl:sort

Con esta instrucción se pueden ordenar los elementos seleccionados por las instrucciones xsl:apply-templates o **`xsl:for-each`**. Debemos agregar xsl:sort como elemento hijo de alguna de las instrucciones anteriores. Podemos agregar más de un elemento xsl:sort. Si se agregan más de uno la ordenación se realizará en el orden de aparición. Si xsl:sort se aplica a un bucle xsl:for-each deberá aparecer como primer hijo de ese elemento. En el atributo select se indicará una expresión que devuelva una cadena de caracteres que servirá como clave de ordenación del conjunto de nodos seleccionados. El valor por defecto de este atributo es el nodo actual (.)

Otros atributos son:

order: especifica el orden, puede tomar los valores: ascending o descending.
lang: especifica el lenguaje de la clave de ordenación (alfabeto).
data-type: especifica el tipo de dato. Puede tomar los valores: text, number o un nombre cualificado.
case-order Especifica casos especiales. Puede tomar los valores: upper-first, lower-first para los data-type="text".

Vamos a realizar una transformación que muestre una tabla con los alumnos ordenados por apellidos con las filas de distintos colores según las notas medias obtenidas. Recordando un poco de HTML+CSS, en este ejemplo queremos asignar dinámicamente con un xsl:choose el valor del atributo class del elemento tr para dar uno de los tres estilos creados en la cabecera.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <head>
        <style>
          .aprobado {color:green;}
          .suspenso {color:red;}
          .sobresaliente {color:blue}
        </style>
      </head>
      <body>
        <table border="0">
```

```
          <thead style="background-color:gray; color:white;">
            <th>NOMBRE</th>
            <th>APELLIDOS</th>
            <th>NOTA MEDIA</th>
          </thead>
          <tbody style="background-color:white;">
          <xsl:for-each select="//alumno">
            <xsl:sort select="apellidos"/>
            <xsl:element name="tr">
              <xsl:attribute name="class">
                <xsl:choose>
                  <xsl:when test="nota_media < 5">suspenso</xsl:when>
                  <xsl:when test="nota_media < 9">aprobado</xsl:when>
                  <xsl:otherwise>sobresaliente</xsl:otherwise>
                </xsl:choose>
              </xsl:attribute>
              <td><xsl:value-of select="nombre"/></td>
              <td><xsl:value-of select="apellidos"/></td>
              <td style="text-align: center;"><xsl:value-of select="nota_media"/></td>
            </xsl:element>
          </xsl:for-each>
          </tbody>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

El resultado será

```
<html>
    <head>
        <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <style>
          .aprobado {color:green;}
          .suspenso {color:red;}
          .sobresaliente {color:blue}
        </style>
    </head>
    <body>
        <table border="0">
            <thead style="background-color:gray; color:white;">
                <th>NOMBRE</th><th>APELLIDOS</th><th>NOTA MEDIA</th>
            </thead>
            <tbody style="background-color:white;">
                <tr class="aprobado">
                    <td>Ana</td><td>Abad &Aacute;lvarez</td><td style="text-align: center;">6.5</td>
                </tr>
                <tr class="aprobado">
                    <td>Alejando</td><td>&Aacute;lvarez Amate</td><td style="text-align: center;">5.25</td>
                </tr>
                <tr class="aprobado">
                    <td>Ana</td><td>Amate Antunez</td><td style="text-align: center;">7.5</td>
                </tr>
                <tr class="suspenso">
                    <td>Ana</td><td>Antunez Amaya</td><td style="text-align: center;">2.5</td>
                </tr>
                <tr class="suspenso">
                    <td>Benito</td><td>Bellido Bravo</td><td style="text-align: center;">2.25</td>
                </tr>
                <tr class="aprobado">
                    <td>Benito</td><td>Ben&iacute;tez Bellido</td><td style="text-align: center;">6.5</td>
                </tr>
                <tr class="aprobado">
                    <td>Beatriz</td><td>Benitez Berm&uacute;dez</td><td style="text-align: center;">8.25</td>
                </tr>
                <tr class="aprobado">
                    <td>Beatriz</td><td>Bravo Ben&iacute;tez</td><td style="text-align: center;">7.75</td>
                </tr>
                <tr class="sobresaliente">
                    <td>Carmen</td><td>Camacho Camacho</td><td style="text-align: center;">10</td>
                </tr>
                <tr class="suspenso">
                    <td>Carlos</td><td>Carmona Casado</td><td style="text-align: center;">4.75</td>
                </tr>
                <tr class="suspenso">
```

```
                <td>Carmen</td><td>Casado Carmona</td><td style="text-align: center;">4</td>
            </tr>
            <tr class="aprobado">
                <td>Carmen</td><td>Cuesta Camacho</td><td style="text-align: center;">8.75</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

3.3.3.- Instrucciones de salida

xsl:value-of

Este elemento se utiliza para generar texto ya sea extraído del árbol de nodos origen, generado por una expresión XPath o insertando el valor de una variable. El elemento xsl:value-of creará un nodo text en el árbol de nodos destino. Tiene un atributo obligatorio select en el que se indicará una expresión que será evaluada y su resultado se convertirá a una cadena de caracteres. Si queremos añadir un conjunto de nodos, es decir, elementos con sus elementos hijos al árbol de nodos destino deberíamos utilizar xsl:copy-of en su lugar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes" />
  <xsl:template match="/">
    <xsl:text>
    </xsl:text>
    <xsl:comment>Esto es un saludo</xsl:comment>
    <xsl:text>
    </xsl:text>
    <xsl:element name="saludo">
      <xsl:attribute name="idioma">español</xsl:attribute>
      <xsl:text>Hola, bienvenido al colegio </xsl:text>
      <xsl:value-of select="colegio/nombre" />
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
    <!--Esto es un saludo-->
    <saludo idioma="español">Hola, bienvenido al colegio Cervantes</saludo>
```

xsl:number

Este elemento se utiliza para insertar un número formateado en el árbol de nodos resultado. El número que se va a insertar se debe especificar en el atributo value mediante una expresión. La expresión se evaluará y el objeto resultante se convertirá a un número. El número se redondeará a un entero y después se convertirá a una cadena de caracteres. Si no se especifica el atributo value el número añadido estará basado en la posición del nodo contexto. Los siguientes atributos controlarán como se realizará la numeración:

level: especificará que niveles del árbol de nodos origen se utilizarán. Sus valores pueden ser single, multiple o any. Por defecto será single.
count: será un patrón que especifique qué nodos serán contados en esos niveles. Por defecto se contarán los nodos semejantes al nodo contexto.
from: es un patrón que especifica dónde se empieza a contar.

Para especificar el formato se utilizar el atributo opcional format, al que se le indicará un patrón con una combinación de los siguientes caracteres:

Un token que acabe en 1: Formará una secuencia numérica. Por ejemplo 1 formará la secuencia 1, 2, 3, ... y 001 formará la secuencia 001, 002, 003, ...
Letra A: formará la secuencia A, B, C, ..., AA, AB, ...
Letra a: formará la secuencia a, b, c, ..., aa, ab, ...
Letra I: formará la secuencia I, II, III, IV, ...
Letra i: formará la secuencia i, ii, iii, iv, ...
Otras numeraciones especificadas en unicode: ア イ ๏ א ১ α a

El patrón por defecto es 1.

xsl:element

Este elemento permite crear un elemento con el nombre indicado por el atributo obligatorio name. También se le puede indicar un espacio de nombres de forma opcional con el atributo namespace. El contenido de este elemento podrá ser, por ejemplo, elementos xsl:attribute, otros elementos xsl:element hijos, comentarios xsl:comment, etc.

xsl:attribute

Este elemento puede ser usado para añadir atributos a los elementos creados en una plantilla mediante el elemento xsl:element o directamente escribiendo las etiquetas de apertura y cierre (LRE). El nombre del atributo será indicado mediante el atributo obligatorio name. También se le puede indicar un espacio de nombres de forma opcional con el atributo namespace. El contenido de este elemento será el valor del atributo.

Al igual que un elemento se puede crear mediante (LRE), los atributos también pueden construirse mediante el uso de llaves {}, a esto se le llama attribute value templates. Consiste en escribir el atributo dentro de su correspondiente etiqueta y cuando se quiere introducir el valor indicarlo mediante una expresión entre llaves {}.

Si un elemento tiene varios atributos o un atributo que se quiere reutilizar se puede utilizar la instrucción attribute-set.

xsl:text

Este elemento se utiliza para crear nodos texto en el árbol de nodos resultado. Los nodos de texto adyacentes se simplificarán en uno solo. Este elemento aplicará lo especificado en los elementos de nivel superior xsl:strip-space y xsl:preserve-space en lo referente al manejo de espacios en blanco.

xsl:comment

Este elemento creará un nodo comentario en el árbol de nodos destino. Su contenido será una cadena de texto.

xsl:processing-instruction

Este elemento creará un nodo instrucción de procesamiento en el árbol de nodos destino. En el atributo obligatorio name se deberá especificar el nombre o clase de la instrucción de procesamiento.

A partir de nuestro documento colegio.xml vamos a crear otro documento XML que contenga un listado de alumnos ordenado por notas cuyo nombre comience por 'A'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" version="1.0" indent="yes" />
  <xsl:template match="/">
<xsl:text>
</xsl:text>
    <xsl:comment>Documento de estilos CSS vinculado al archivo XML resultante</xsl:comment>
<xsl:text>
</xsl:text>
    <xsl:processing-instruction name="xml-stylesheet">href="style.css" type="text/css"</xsl:processing-instruction>
<xsl:text>
</xsl:text>
    <xsl:element name="listado">
    <xsl:comment>Listado de alumnos ordenados por notas cuyo nombre empieza por A</xsl:comment>
      <xsl:for-each select="//alumno[starts-with(nombre,'A')]">
        <xsl:sort select="nota_media" order="descending" data-type="number"/>
        <xsl:element name="persona">
          <xsl:attribute name="calificacion">
            <xsl:value-of select="nota_media"/>
          </xsl:attribute>
          <xsl:attribute name="num_orden">
            <xsl:number value="position()" format="&#x30A2;"/>
          </xsl:attribute>
          <xsl:value-of select="nombre"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="apellidos"/>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

El resultado será el siguiente

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Documento de estilos CSS vinculado al archivo XML resultante-->
<?xml-stylesheet href="style.css" type="text/css"?>
<listado>
    <!--Listado de alumnos ordenados por notas cuyo nombre empieza por A-->
    <persona calificacion="7.5" num_orden="ア">Ana Amate Antunez</persona>
    <persona calificacion="6.5" num_orden="イ">Ana Abad Álvarez</persona>
    <persona calificacion="5.25" num_orden="ウ">Alejando Álvarez Amate</persona>
    <persona calificacion="2.5" num_orden="エ">Ana Antunez Amaya</persona>
</listado>
```

El número de orden no está mal, es que hemos utilizado el formato Unicode para la numeración japonesa *katakana*.

3.4.- Funciones propias de XSLT

Funciones propias XSLT

document(): Permite el acceso a otros documentos XML distintos del original. Tiene dos parámetros el primero indicará la URI o lugar donde estará el documento a añadir. Si no se indica este parámetro por defecto se tomará el documento actual. El segundo parámetro opcional nos permite seleccionar parte de ese nuevo documento mediante una expresión XPath. Esta función devuelve el conjunto de nodos.

node-set document(object, node-set?)

key(): esta función es similar a id() de XPath. El primer parámetro es el nombre de la clave que debe ser un nombre cualificado indicado en un elemento xsl:key. El segundo argumento debe ser el valor para dicha clave. Devuelve el conjunto de nodos del documento que cumplen estas condiciones.

node-set key(string, object)

format-number(): Convierte el primer parámetro a una cadena de caracteres siguiendo el patrón indicado en el segundo parámetro. Si existen un tercer argumento indicará el nombre del xsl:decimal-format que se utilizará. Si no se indica este tercer argumento se cogerá el valor por defecto. Los patrones del segundo parámetro son los de la clase DecimalFormat de JDK.

string format-number(number, string, string?)

current(): devuelve el valor del nodo contexto. Normalmente se sustituye por su versión abreviada, <xsl:value-of select="current()"/> equivale a <xsl:value-of select="."/>.

node-set current()

generate-id(): genera un identificador único para el conjunto de nodos pasado como parámetro. si no se indica este parámetro se tomará el nodo contexto actual. Devuelve una cadena de caracteres con dicha identificación.

string generate-id(node-set?)

Otras funciones XSLT son: unparsed-entity-uri(), unparsed-entity-uri(), system-property(), element-available() y function-available().

Vamos a realizar una transformación para calcular la media de notas de los alumnos agrupados por nombre.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <xsl:key name="nombre_clave" match="//alumno/nombre" use="."/>
    <html>
      <body>
        <ol>
          <xsl:for-each select="//alumno/nombre[generate-id() = generate-id(key('nombre_clave',.)[1])]">
            <xsl:sort select="." order="ascending" data-type="text"/>
            <xsl:variable name="nombre_actual" select="."/>
            <xsl:variable name="media" select="sum(//alumno[nombre=$nombre_actual]/nota_media) div count(//alumno[nombre=$nombre_actual]/nota_
            <li>
                <xsl:value-of select="$nombre_actual"/>
                <xsl:text>. Media de notas: </xsl:text>
                <xsl:value-of select="$media"/>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

En este caso generamos una clave por nombre de alumno, recorremos los nombres de alumnos que al generar de nuevo la clave es igual a la primera ya creada. Realizamos la media agrupando por el nombre que tenemos seleccionado. Otra forma podría haber sido utilizar el eje preceding para evitar elementos repetidos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:for-each select="//alumno[not(nombre=preceding::nombre)]">
            <xsl:sort select="nombre" order="ascending" data-type="text"/>
            <xsl:variable name="nombre_actual" select="nombre"/>
```

```
            <xsl:variable name="media" select="sum(//alumno[nombre=$nombre_actual]/nota_media) div count(//alumno[nombre=$nombre_actual]/nota_
            <li>
                <xsl:value-of select="$nombre_actual"/>
                <xsl:text>. Media de notas: </xsl:text>
                <xsl:value-of select="$media"/>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Autoevaluación

Indica cuál de estas afirmaciones es correcta:

- ○ Para aplicar una ruta de XPath a un documento diferente de aquel con el que se trabaja en un momento dado el lenguaje nos proporciona la función document().
- ○ La función document() devuelve únicamente una rama de un nodo dado.
- ○ No existe ningún modo de acceder a los elementos de varios documentos.
- ○ XPath no proporciona un modo de acceder a datos de varios documentos simultáneamente.

Correcto. document() se utilizar para ese fin.

No es correcto, puede devolver todo el documento XML.

Incorrecto. Sí existe, la función document().

Podemos acceder con la función document() por lo que no es correcto.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

3.5.- Procesadores XSLT

Un procesador XSLT es una aplicación que realiza una transformación partiendo de un documento XML y siguiendo las reglas indicadas en otro documento XSLT. El resultado será un documento en uno de los formatos permitidos; XML, HTML y texto plano.

Como ya hemos comentado a lo largo de este unidad el procesador XSLT cuando lee el archivo XML inicial y crea una árbol de nodos origen. A continuación va procesando las instrucciones de la hoja de estilos recogida en el documento XSLT. Esta hoja de estilos contendrá una o varias plantillas. Cada plantilla tendrá una expresión de emparejamiento. Estas expresiones serán buscada en el archivo XML original y si encuentra elementos compatibles se ejecutan las instrucciones oportunas. Estas instrucciones irán generando un árbol de nodos destino. Una vez recorridas todas las plantillas y generado todo el árbol de nodos destino este será escrito en un archivo de salida.

También hay que tener en cuenta que el consorcio W3C publica sus recomendaciones sobre el lenguaje XSLT pero no se encarga de implementar los procesadores XSLT. Existen varias empresas software que se dedican a ello.

Algunas de las empresas de software que tienen procesadores XSLT son:

Saxon: desarrolla su procesador XSLT llamado saxon con las recomendaciones W3C más recientes: XSLT 3.0, XPath 3.1. Se distribuye como librerías de programación para varios lenguajes: Java, .NET, C/C++, PHP, Python y JavaScript aunque también se puede utilizar en la línea de comandos gracias al empaquetado de aplicaciones java .jar. El fundador de esta empresa es Michael Kay una persona que participa en las recomendaciones del consorcio W3C.
Gnome: tiene una implementación de las recomentaciones XSLT 1.0 y XPath 1.0 en una librería llamada libxslt. Esta librería tiene una utilidad de línea de comandos llamada xsltproc. Gnome tiene otra librería relacionada con XML llamada libxml2 que permite trabajar y validar documentos XML. Libxml2 tiene una utilidad de líneas de comandos llamada xmllint.
Apache: desarrolló un procesador XSLT muy utilizado llamado Xalan. Este procesador XSLT implementabla las recomendaciones XSLT 1.0 y XPath 1.0. Se distribuía como librerías para los lenguajes Java y C++. Su última versión estable la 2.7.1 es de noviembre de 2007. Dentro de la empresa Apache existe un proyecto hermano Xerces que permite trabajar y validar documentos XML.
Microsoft: desarrolló una librería con funciones XML llamada MSXML (Microsoft XML Core Services) que implementaba XML 1.0, XML Schema 1.0 (2ª edición), XPath 1.0 y XSLT 1.0. Su última versión es la 6.

La forma de utilizar estos procesadores XSLT varían según las necesidades.

Podemos encontrar procesadores XSLT que se ejecutan en la línea de comandos de nuestro sistema operativo (xsltproc, saxonb-xslt, xalan), como librerías de distintos lenguajes de programación, en páginas web accesibles con nuestro navegador (http://xsltransform.net/, https://xslttest.appspot.com/) o integrados en editores específicos de XML (XML Copy Editor, Oxygen XML Editor, Altova XMLSpy, Liquid XML Studio, Stylus Studio) o generales (Netbeans, Visual Studio, IntelliJ IDEA, Eclipse).

Vamos a realizar una transformación desde la línea de comandos con la librería saxon. En este caso utilizaremos la última versión con mantenimiento la 10.8 en su edición Home (HE) empaquetada como librería java. Esta edición 'Home Edition' es open source y puede descargarse gratis. Para poder ejecutar esta librería es necesario tener instalado al menos la versión Java SE 8 o el JDK 1.8.

Para comprobar que tenemos instalada la máquina virtual de java ejecutaremos en nuestra línea de comandos:

```
java --versión
```

En Windows un solo guión.

Una vez descargada la librería SaxonHE10-8J.zip la descomprimiremos en una carpeta. En la línea de comandos nos posicionaremos en dicha carpeta y probaremos que la librería es accesible con:

```
java -jar saxon-he-10.8.jar -t
```

con -t indicaremos a saxon que nos muestra información sobre su versión

```
Saxon-HE 10.8J from Saxonica
Java version 11.0.15
No source file name
Usage: see http://www.saxonica.com/documentation/index.html#!using-xsl/commandline
Format: net.sf.saxon.Transform options params
Options available: -? -a -catalog -config -cr -diag -dtd -ea -expand -explain -export -ext -im -init -it -jit -l -lib -license -m -nogo -now -
Use -XYZ:? for details of option XYZ
Params:
  param=value           Set stylesheet string parameter
  +param=filename       Set stylesheet document parameter
  ?param=expression     Set stylesheet parameter using XPath
  !param=value          Set serialization parameter
```

Por simplificar la configuración en esa carpeta también pondremos los archivos para crear nuestra transformación, por ejemplo colegio.xsl y **`colegio.xml`**. Y ejecutaremos la transformación con:

```
java -jar saxon-he-10.8.jar -xsl:colegio.xsl -s:colegio.xml -o:colegio.html
```

Si no hay errores, en nuestro directorio tendremos nuestro nuevo archivo con la transformación realizada.

En algunos sistemas operativos está disponible el paquete saxonb-xslt y podríamos realizar la transformación de forma similar al ejemplo anterior:

```
saxonb-xslt -xsl:colegio.xsl -s:colegio.xml -o:colegio.html
```

Otro procesador de línea de comandos es xsltproc de la librería Gnome LibXSLT. Una vez instalado el paquete xsltproc, podemos realizar la transformación de la siguiente forma:

```
xsltproc colegio.xml colegio.xsl -o colegio.html
```

De igual forma instalando el paquete xalan podemos realizar nuestra transformación con el comando:

```
xalan -xsl colegio.xsl -in colegio.xml -out colegio.html
```

Para saber más

En los siguientes enlaces puedes encontrar algunos de los procesadores de XSLT:

Saxon

LibXSLT

Xalan

MSXML

Autoevaluación

El único modo de generar un documento a partir de una transformación XSLT de otro es utilizando un editor XML que tenga incorporado un procesador XSLT:

- ○ No.
- ○ Sí.

Muy bien, has captado la idea.

Incorrecta, podemos utilizar un procesador en modo comando, sin necesidad de editor, o un procesador integrado en un navegador.

Solución

1. Opción correcta
2. Incorrecto

3.6.- Depuración de documentos XSLT

Algunos editores tienen la funcionalidad de depuración de código. Esta funcionalidad permite ejecutar paso a paso las instrucciones de forma que facilita enormemente encontrar y corregir errores. Además suelen permitir mostrar los valores que van tomando las distintas variables o estructuras de datos. También permiten establecer puntos de parada (breakpoint) o ejecución hasta el cursor de modo que podemos ejecutar el código de forma normal hasta encontrar dichos puntos de parada o la posición de nuestro cursor en el editor donde el depurador de código cambiará a ejecución paso a paso. Otra funcionalidad muy útil es mostrar paso a paso los resultados que va generando nuestro código.

Esta funcionalidad de depuración puede estar incluida o no en los distintos editores. Normalmente en los editores gratuitos no suele estar presente.

Entre los editores genéricos con versión open source y que tengan esta funcionalidad de depuración tenemos:

IntelliJ IDEA + Plugin XSLT Debugger
Eclipse + Plugin EclipseXML Editors and Tools
NetBeans + Plugin netbeans-xslt-debugger (obsoleto)

Otro editor genérico de pago con esta funcionalidad es:

Visual Studio Profesional o Enterprise

Dentro de los editores especializados en XML no tenemos ninguno con versión gratuita pero si con periodos de evaluación de 15 a 30 días:

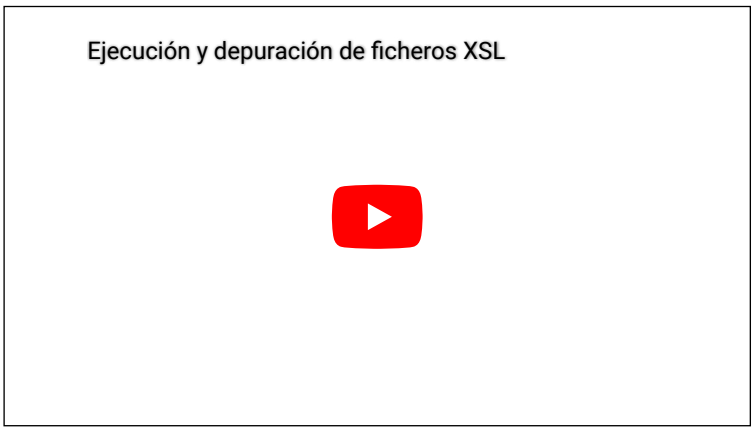
Oxygen XML Editor
Liquid XML Studio
Editix
Stylus Studio
Altova XMLSpy

Para saber más

En el siguiente vídeo se muestra cómo ejecutar y depurar una transformación XSLT utilizando el editor Oxygen XML Editor.

Ejecución y depuración de ficheros XSLT

https://www.youtube.com/embed/gwYGEY80vi4



Resumen textual alternativo

Autoevaluación

La depuración facilita la labor de ...

- ○ Compilación del código.
- ○ Localización de errores en el código.
- ○ Ejecución del código.
- ○ Documentación del código.

Incorrecto

Opción correcta

Incorrecto

Incorrecto

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

Condiciones y términos de uso de los materiales

Historial de actualizaciones

Versión: 02.00.01	Fecha de actualización: 24/06/22
Actualización de materiales y correcciones menores.	

Versión: 02.00.00	Fecha de actualización: 23/06/22	Autoría: José Antonio Molina Bautista
Ubicación: Toda la unidad **Mejora (tipo 3):** Unidad 5: XPath: Los contenidos son unos listados de palabras del lenguajes y poca o ninguna explicación de la utilidad de las distintas estructuras del lenguaje. No se desarrollan adecuadamente los contenidos. Se habla de los nodos y en ningún momento se indica el árbol del documento. Hay conceptos que se repiten en distintos apartados como los token del apartado 2.2. Existe un apartado, el 2.6, solamente para indicar una abrevación. Aunque dicha abreviación ya se enuncia en el apartado 2.2 Los ejes se simplifican directamente y no se relacionan con las rutas ni sus abreviaciones. Hay funciones repartidas en varios apartados. Incluso un apartado, el 2.7, para explicar una única función. Solamente existe un ejemplo y no está explicado ni comentado. Faltan ejemplos de xpath donde se apliquen condiciones o funciones a distintos niveles de la ruta. XSLT: Los contenidos de esta parte son una mera lista de elementos que podemos usar y un ejemplo. No se desarrolla cómo se van creando los árboles, qué datos se van procesando, qué datos se van omitiendo, qué pasa con las plantillas vacías, cómo se anidan las llamadas a las plantillas, qué ocurre si no se encuentra una plantilla o si hay dos plantillas para los mismos nodos. Falta todo el desarrollo de las instrucciones: value-of, for-each, sort, if, choose. Tampoco revisa el uso de expresiones Xpath ni de sus funciones. No se explica la creación de elementos y atributos mediante element y attribute que son de gran utilidad al generar HTML. También habría que actualizar el mapa conceptual y las orientaciones al alumnado. **Ubicación:** El tema entero **Mejora (tipo 1):** El tema está un poco obsoleto, ya casi no hay ni programas para tratar este lenguaje. **Ubicación:** No especificada. **Mejora (tipo 2):** Falta añadir una descripción más extensa y mejor sobre las transformaciones XSLT y XPath, con muchos más ejemplos y ejercicios resueltos. He elaborado bastante documentación al respecto en el siguiente enlace de Google Drive: https://drive.google.com/open?id=0ByKos1mgJL8geW53YURmbGdXNzQ&authuser=0 Puede interesar el documento sobre XSLT: https://drive.google.com/open?id=0ByKos1mgJL8gWFcwOUliZ256VzA&authuser=0 Y el documento sobre XPath: https://drive.google.com/open?id=0ByKos1mgJL8gdjdqYWQ3ZkVBSWc&authuser=0 **Ubicación:** Toda la unidad **Mejora (Mapa conceptual):** Mapa conceptual afectadas por el un cambio de tipo 3. Quitar la relación ambigua XSL y XSLT. Quitar XSL-FO que no se trabaja en la unidad. **Ubicación:** Toda la unidad **Mejora (Orientaciones del alumnado):** Orientaciones del alumnado afectadas por el un cambio de tipo 3.		

Versión: 01.01.00	Fecha de actualización: 09/02/15	Autoría: Víctor Javier Gómez Arques
Acrónimos, ampliación de ejemplos, referencias, videos explicativos y demostrativos y ejercicios de repaso.		

Versión: 01.00.00	Fecha de actualización: 28/04/14
Versión inicial de los materiales.	