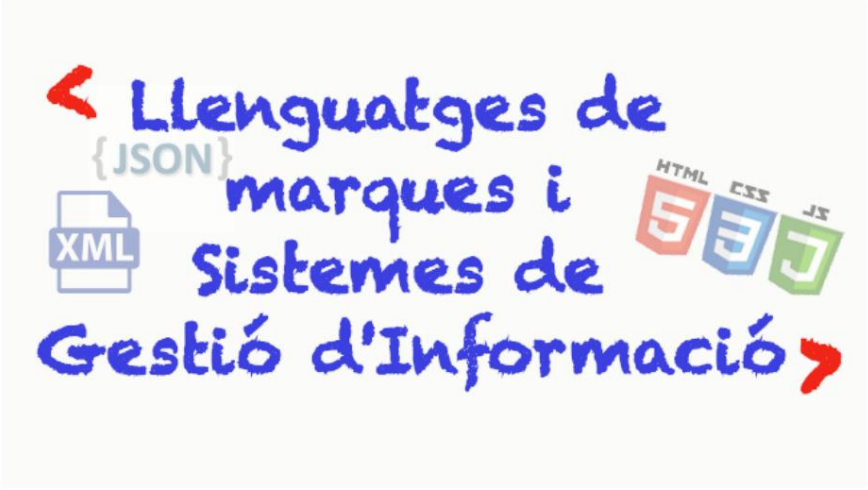
UNIDAD 6. CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS PARA EL INTERCAMBIO DE INFORMACIÓN



IES Sant Vicent Ferrer
Algemesí

Contenido

1. Introducción

A pesar de que el XML es un formato relativamente legible al ser visualizado, éste no es uno de sus objetivos principales, y menos si se tiene en cuenta que a los humanos les gusta leer los datos colocados en determinados formatos que les hagan la lectura más agradable.

En determinados casos puede ser necesario transformar los documentos XML para que sean más fáciles de visualizar, o también adaptarlos para que puedan ser leídos por programas específicos.

XML está pensado sobre todo para almacenar e intercambiar información, de modo que si es necesario representar los datos de una forma diferente para optimizar un proceso o para mejorar su visualización habrá varias posibilidades:

- Desarrollar un programa: al ser relativamente sencillo trabajar con XML, se podría desarrollar un programa que tome los datos XML y genere la salida tal y como la queremos. Esto tiene el inconveniente de que habrá que tener conocimientos de programación y que puede representar mucho trabajo aunque lo que haya que hacer sea trivial.
- Utilizar CSS: en muchos casos una solución sencilla sería utilizar CSS para representar la información de forma más amigable utilizando un navegador. Pero sólo sirve para cambiar la vista, no para cambiar el documento.
- Transformar el documento: una solución alternativa consiste en transformar el documento en otro que esté pensado para su visualización. Hay muchos formatos pensados sobre todo para ser visualizados: PDF, HTML, XHTML, etc.

1.1. Uso de CSS

En muchos casos si el objetivo es que el documento sea visualizado de forma más agradable la solución más sencilla puede ser visualizarlo mediante CSS.

CSS es interesante sobre todo cuando se quieran realizar adaptaciones sencillas. Si se tiene un documento XML como el siguiente, y se quiere representar como una tarjeta de vista, se puede utilizar un documento CSS que contenga las líneas del ejemplo:

```xml
<?xml version="1.0" ?>
<professor>
    <nom>Marcel</nom>
    <cognom>Garcia</cognom>
    <departament>Departament d'Informàtica</departament>
    <carrecs>
       <carrec>Cap de Departament</carrec>
       <carrec>Tutor</carrec>
    </carrecs>
</professor>
```

1 Ejemplo 1: Documento XML

```css
professor {
    padding: 30px;
    margin: 30px;
    border: 4px black solid;
    width: 40%;
}

nom,cognom {
    font-size:30px;
}

departament {
    padding-top:20px;
    display:block;
    font-weight:bold;
}

carrec{
    font-style: italic;
    padding-left:10px;
}

carrec:after { content:","; }
```

2 Ejemplo 1: Documento CSS

El resultado se mostrará formateado:

3 Ejemplo 1: Resultado de presentación

Sin embargo, CSS tiene muchas limitaciones a la hora de presentar la información:

- La información no puede ser reordenada como queramos. La única forma de simular el cambio de orden es utilizar posicionamientos absolutos.
- Los atributos se pueden mostrar pero existen muchas limitaciones para hacerlo.
- No se pueden añadir estructuras nuevas producto de cálculos o de proceso de los datos del documento.
- No tiene formas sencillas de formatear los datos en páginas de texto para ser impresas.

Si el objetivo final no es simplemente decorar el archivo sino transformarlo en otro documento totalmente diferente, CSS no sirve. CSS no transforma el documento sino que simplemente cambia la forma en que se visualiza.

2. Transformación de documentos

Para intentar hacer todo lo que CSS no podía hacer se creó un nuevo lenguaje de plantillas: XSL (extensible stylesheet language).

Inicialmente se concentraron en poder representar la información de forma que pudiera ser mostrada en documentos impresos y en pantalla pero después se acabó definiendo un sistema para realizar transformaciones genéricas de documentos XML en otras cosas: documentos de texto, documentos XML, páginas web, etc.

Actualmente XSL es una familia de lenguajes que sirven para definir transformaciones y presentaciones de documentos XML.

La familia XSL está formada por tres lenguajes:

- XSL-FO (XSL formatting objects): un lenguaje para definir el formato que debe aplicarse a un documento.
- XSLT (XSL transformations): un lenguaje para transformar documentos XML.
- XPath: un lenguaje para acceder a partes de los documentos XML.

2.1. XSL-FO

XSL-FO es un lenguaje basado en XML que está pensado para dar formato a los documentos XML, pero a diferencia de otros lenguajes con objetivos similares, como CSS o XHTML, está pensado para generar salidas tanto para formatos de pantalla como para formatos paginados.

XSL-FO es un lenguaje que:

- Permite especificar con mucha precisión el contenido de un documento (paginación, organización, estilo, etc.).
- Permite crear documentos de alta calidad.
- Es ideal para generar documentos con datos que cambian a menudo.

XSL-FO se utiliza sobre todo para generar documentos en formatos pensados para ser impresos como PDF o Postcript.

Uno de los procesadores de XSL-FO más populares es el Apache FOP, que permite crear documentos en PDF a partir de documentos XSL-FO. Podemos obtener más información de este procesador en el siguiente enlace: https://xmlgraphics.apache.org/fop/.

Es corriente que la generación del documento se haga en dos fases:

1. Transformación del documento XML en un documento XSL-FO con el lenguaje de transformaciones XSLT.
2. Transformación del documento XSL-FO en el formato que queremos con un procesador XSL-FO.

2.2. Procesadores XPath o XSLT

Por lo general los procesadores XPath o XSLT se proporcionarán por medio de bibliotecas que podrán ser llamadas desde los programas.

Esto permite simplificar todo el proceso de creación de programas que utilicen XPath o XSLT, ya que cuando una aplicación se quiera aprovechar de las características de alguno de estos lenguajes no deberá implementarlo todo de nuevo sino simplemente utilizar las bibliotecas .

3. XPath

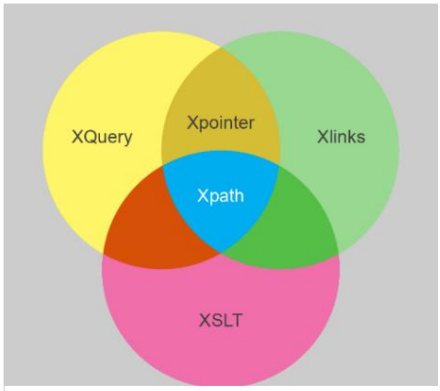
XPath (XML path language) es una forma de especificar partes de un documento XML que tiene herramientas para manipular el contenido de los datos de texto, numéricos, etc.

XPath es una recomendación de W3C (www.w3.org/TR/xpath), que aunque sirve para trabajar con XML no es un lenguaje XML. La idea es que al no estar basado en XML podrá incluirse en otros lenguajes XML sin tener que preocuparse de si el resultado está bien formado o no.

La base del funcionamiento de XPath es la evaluación de expresiones. Una expresión que se evaluará contra un documento XML y nos dará un resultado que puede ser de distintos tipos:

- Un booleano: cierto o falso
- Un número
- Una cadena de caracteres
- Un grupo de elementos

XPath está desarrollado por los comités de creación de XSL y XQuery y se ha convertido en un componente esencial para diferentes lenguajes XML como XLinks, XSLT y XQuery, como puede verse en la siguiente figura 4:



4 Relación de XPath con los lenguajes XML

La versión 2.0 de XPath está tan integrada dentro de XQuery que cualquier expresión XPath es también automáticamente una expresión XQuery correcta.

3.1. Vista de árbol

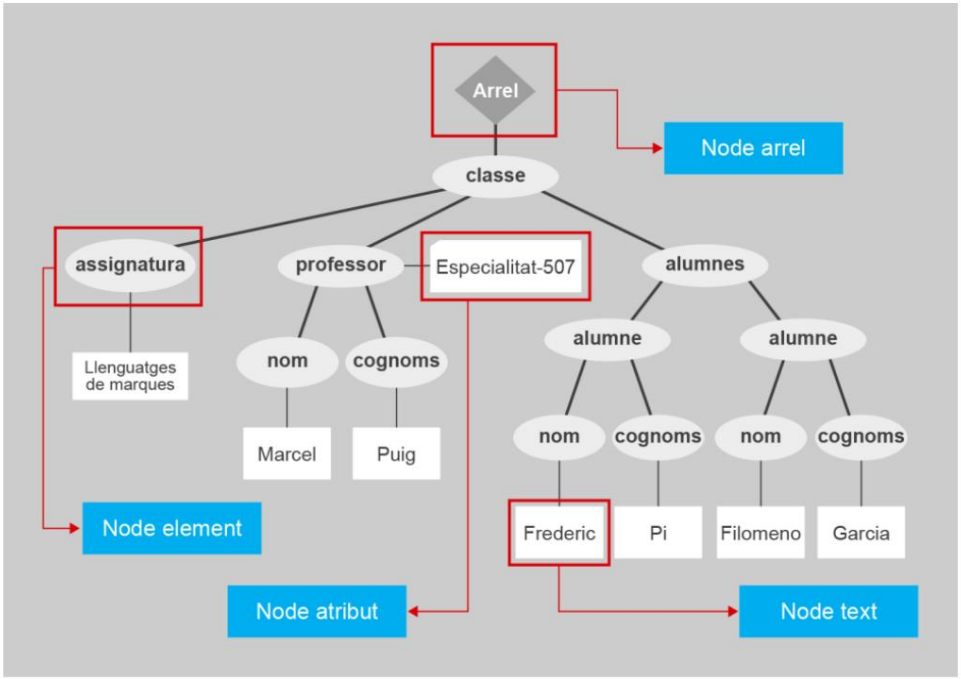
XPath trata todos los documentos XML desde el punto de vista de un árbol de nodos en el que existe una raíz que no se corresponde con la raíz del documento, sino que es el documento y se representa con el símbolo “/”.

Aparte de la raíz también existen nodos para representar los elementos, atributos , nodos de datos, comentarios , instrucciones de proceso y espacios de nombres.

El siguiente ejemplo XML:

```xml
<?xml version=”1.0” ?>
<classe>
    <assignatura>Llenguatges de marques</assignatura>
    <professor Especialitat="507">
        <nom>Marcel</nom>
        <cognoms>Puig</cognoms>
    </professor>
    <alumnes>
        <alumne>
            <nom>Frederic</nom>
            <cognoms>Pi</cognoms>
        </alumne>
        <alumne>
            <nom>Filomeno</nom>
            <cognoms>Garcia</cognoms>
        </alumne>
    </alumnes>
</classe>
```

Se representará en XPath con un árbol como el siguiente:



5 Representación en forma de árbol XPath del XML de ejemplo

En un árbol XPath, los atributos no son considerados nodos hijos sino que son “propiedades” del nodo que los contiene y los nodos de datos son nodos sin nombre que sólo contienen los datos.

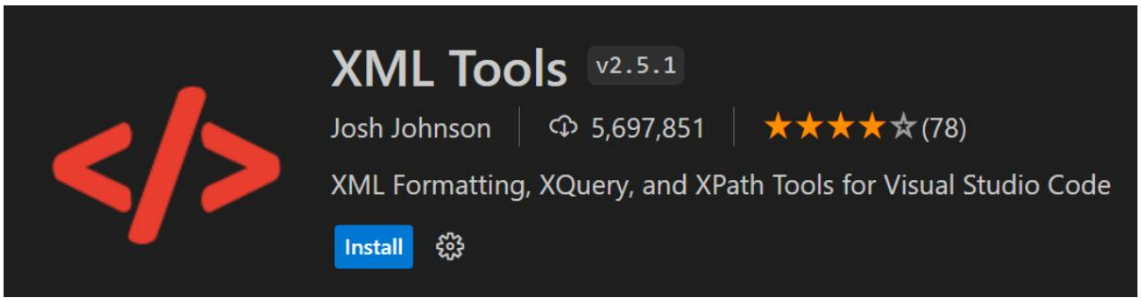
3.2. Software para evaluar XPath

Muchos programas permiten ejecutar una consulta XPath contra un documento XML. Existen programas específicos, editores XML, componentes de los navegadores web, páginas web online, etc.

3.2.1. Visual Studio Code

Podemos utilizar el editor que hemos utilizado en otras unidades de este curso, Visual Studio Code, para evaluar expresiones XPath utilizando extensiones.

Una de las extensiones más popular para llevar a cabo esta tarea es XML Tools, del autor Josh Johnson



3.2.2. XML Copy Editor

XML Copy Editor es un editor de documentos XML libre (Licencia GPL 2.0) y multiplataforma cuya página web es https://xml-copy-editor.sourceforge.io/.

La última versión disponible actualmente (abril de 2024) es la versión XML Copy Editor 1.3.1.0 (del 8 de octubre de 2022).

Algunas de sus características más destacadas son:

- Validación automática mientras escribes
- Protección de etiquetas
- Soporte para XML Schema/Relax NG/DTD
- Soporte para XSLT y XPath

Puedes consultar la sección de XML Copy Editor en mclibre donde se explica cómo instalar y configurar este editor: https://www.mclibre.org/consultar/xml/otros/xmlcopyeditor.html

3.3. Navegación

Puesto que la representación interna del documento XML para XPath será un árbol, se puede navegar especificando caminos de forma similar a como se hace en los directorios de los sistemas operativos.

Lo importante para tener en cuenta a la hora de crear una expresión XPath es saber el nodo en el que está situado el proceso (nodo de contexto), ya que es desde éste que se evaluará la expresión.
El nodo de contexto al principio está en la raíz pero se va moviendo a medida que se van evaluando las expresiones, y por tanto podemos expresar los caminos XPath de dos formas:

- Caminos absolutos
- Caminos relativos

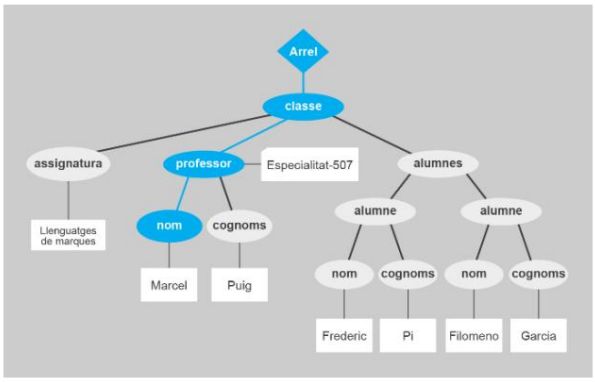
Los caminos absolutos son caminos que siempre comienzan en la raíz del árbol. Pueden identificarse porque el primer carácter de la expresión siempre será la raíz "/". No importa cuál sea el nodo de contexto si se utilizan caminos absolutos, porque el resultado siempre será el mismo.

Por el contrario, los caminos relativos parten desde el nodo en el que estamos situados.

Por ejemplo, se puede obtener el nodo <nombre> del profesor del ejemplo que hemos especificado anteriormente utilizando la expresión XPath siguiente:

/clase/profesor/nombre

Podemos ver cómo se evalúa la expresión en el árbol XPath en la siguiente figura:



6 Evaluación de la expresión /clase/profesor/nombre

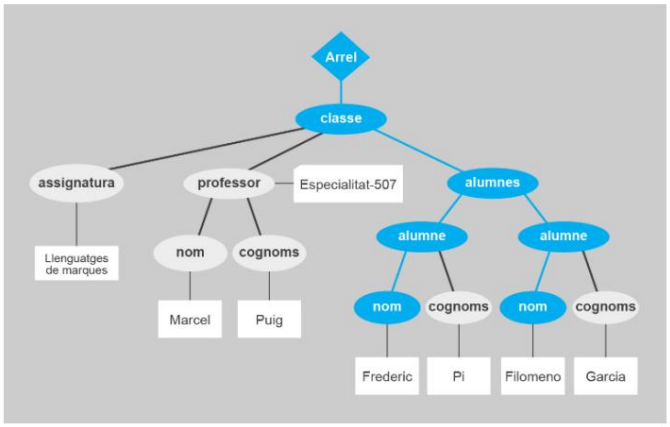
El resultado de esta expresión no es sólo el contenido del elemento sino todo el elemento <nombre>:

<nom>Marcel</nom>

La expresión siempre intenta conseguir el número máximo de caminos correctos y, por tanto, no necesariamente debe devolver sólo un solo valor. Por ejemplo, si la expresión para evaluar fuese la siguiente:

/clase/alumnos/alumno/nombre

XPath lo evaluaría intentando conseguir todos los caminos que cuadran con la expresión, tal y como puede ver en la siguiente figura:



7 Evaluación de la expresión /clase/alumnos/alumno/nombre

El resultado serán los dos resultados posibles:

<nom>Frederic</nom>

<nom>Filomeno</nom>

XPath puede devolver cualquier tipo de elemento como resultado, no es necesario que sean nodos finales. La siguiente expresión:

/clase/alumnos

devolverá el siguiente resultado:

```
<alumnes>
    <alumne>
        <nom>Frederic</nom>
        <cognom>Pi</cognom>
    </alumne>
    <alumne>
        <nom>Filomeno</nom>
        <cognom>Garcia</cognom>
    </alumne>
</alumnes>
```

Si se sabe que una expresión devolverá varios resultados pero sólo se quiere uno específico se puede utilizar un número rodeado por corchetes cuadrados ”[]” para indicar cuál es lo que se quiere conseguir. Para devolver sólo el primer alumno puede hacer lo siguiente:

/clase/alumnos/alumno[1]

De los dos nodos disponibles como hijos de <alumnos>, sólo se seleccionará el primero:

```
<alumne>
   <nom>Frederic</nom>
   <cognom>Pi</cognom>
</alumne>
```

Obtener los atributos de un elemento

Los valores de los atributos se pueden conseguir especificando el símbolo @ delante del nombre una vuelta se haya llegado al elemento que lo contiene.

Por ejemplo, la expresión XPath:

/clase/profesor/@especialidad

devolverá:

507

Hay que tener en cuenta que a diferencia de lo que ocurre con los elementos, al obtener un atributo no tendremos un elemento sino sólo su valor.

Obtener el contenido de un elemento

Para aquellos casos en los que sólo queramos el contenido del elemento, se ha definido la función text() para obtener ese contenido. Esto se ha hecho así porque de otra forma, dado que los nodos de texto no tienen nombre, no se podría acceder a su contenido.

De modo que si a un elemento que tenga contenido de datos se le añade texto:

/clase/profesor/nombre/texto()

devolverá el contenido del nodo sin las etiquetas:

Marcel

Comodines

Al igual que en los sistemas operativos, se pueden utilizar comodines diversos en las expresiones XPath.

Comodín	Significado
*	El asterisco se utiliza para indicar todos los elementos de un determinado nivel.
.	Como en los directorios de los sistemas operativos, el punto sirve para indicar el nodo actual.
..	Se utiliza para indicar el padre del nodo en el que estamos.
//	Las dobles barras indican que cuadrará con cualquier cosa desde el nodo en el que estamos. Puede ser un solo elemento o árbol de nodos.

Ejes XPath

Para evaluar las expresiones XPath se explora un árbol, por lo que también se proporcionan una serie de elementos para hacer referencia a partes del árbol. Estos elementos se llaman ejes XPath.

Eje	Significado
self::	El nodo en el que está el proceso (hace lo mismo que el punto)
child::	Hijo del elemento actual
pariente::	El padre del elemento actual (idéntico a utilizar ..)
attribute::	Se utiliza para obtener un atributo del elemento actual (@)

descendante:: Todos los descendientes del nodo actual

desdendant-or-self:: El nodo actual y sus descendientes

ancestor::	Los ascendientes del nodo
ancestor-or-self::	El nodo actual y sus ascendientes
predeciendo::	Todos los elementos precedentes en el nodo actual
preceding-sibling::	Todos los hermanos precedentes
following::	Elementos que siguen el nodo actual
following-sibling::	Hermanos posteriores al nodo actual

namespace:: Contiene el espacio de nombres del nodo actual

Condiciones

Un apartado interesante de las expresiones XPath es poder añadir condiciones para la selección de nodos. A cualquier expresión XPath se le pueden añadir condiciones para obtener sólo los nodos que cumplan la condición especificada.

La selección de nodos se realiza especificando un predicado XPath dentro de corchetes.

Por ejemplo, esta expresión selecciona sólo a los profesores que tengan un elemento <nombre> como hijo de <profesor>:

```
/clase/profesor[nombre]
```

Si se aplica la expresión al ejemplo que hemos utilizado para realizar la vista de árbol, el resultado será el nodo <profesor> que tiene en su interior <nombre>.

```
<professor especialitat="507">
  <nom>Marcel</nom>
  <cognoms>Puig</cognoms>
</professor>
```

En el valor de la expresión se especifican caminos relativos desde el nodo que tenga la condición. Utilizando condiciones se puede realizar una expresión que sólo devuelva el profesor si tiene alumnos.

```
/clase/profesor[../alumnos/alumno]
```

Normalmente la complejidad de las condiciones va más allá de comprobar si el nodo existe, y se utilizan para comprobar si un nodo tiene un valor determinado. Por ejemplo, para obtener los profesores que se llamen “Marcel”:

```
/clase/profesor[nombre="Marcel"]
```

Las condiciones se pueden poner en cualquier lugar del camino y puede haber tantas como sea necesario. Para obtener el apellido del profesor llamado “Marcel” se puede utilizar una expresión como esta.

```
/clase/profesor[nombre="Marcel"]/apellidos
```

Que dará de resultado:

```
<apellidos>Puig</apellidos>
```

3.4. Secuencias

Una secuencia es una expresión XPath que devuelve más de un elemento. Se parecen bastante a las listas de otros lenguajes:

- Tienen orden
- Permiten duplicados
- Pueden contener valores de tipo diferente en cada término

Es fácil crear secuencias, ya que basta con cerrarlas entre paréntesis y separar cada uno de los términos con comas. La siguiente expresión aplicada a cualquier documento:

```
(1,2,3,4)
```

Devuelve la secuencia de números de uno en uno:

1

2

3

4

También se pueden crear secuencias a partir de expresiones XPath. En este caso se evaluará primero la primera expresión, después la segunda, etc.

(//nombre/text(), //apellidos/text())

Aplicado a nuestro ejemplo devolverá primero todos los nombres y después todos los apellidos:

Marcel

Federico

Filomeno

Puig

Pino

García

Unión, intersección y disyunción

También puede operarse con las secuencias de elementos. Una forma sería utilizar los operadores de unión (union), intersección (intersec) o disyunción (excepto).

Por ejemplo, la siguiente expresión nos devolvería los apellidos de los alumnos que coincidan con los de un profesor:

(//alumno/nombre) intersect (//profesor/nombre)

Con la unión se pueden unir las listas de forma que quede una sola sin duplicados:

(//alumno/nombre) union (//profesor/nombre)

Y con la disyunción obtenemos los nombres de la primera secuencia que no salen en la segunda:

(//alumno/nombre) excepto (//profesor/nombre)

3.5. Funciones

XPath ofrece una gran cantidad de funciones destinadas a manipular los resultados obtenidos.

Las funciones que ofrece XPath se clasifican en distintos grupos:

- Para manipular conjuntos de nodos: se puede obtener el nombre de los nodos, trabajar con los posiciones, contarlos, etc.
- Para trabajar con cadenas de caracteres: permiten extraer caracteres, concatenar, comparar… las cadenas de caracteres. • Para realizar

operaciones numéricas: se pueden convertir los resultados a valores numéricos, contar los nodos, realizar operaciones, etc.

- Funciones boleanas: permiten realizar operaciones boleanas con los nodos.

- Funciones por fechas: permiten realizar operaciones diversas con fechas y horas.

Es imposible especificarlas todas aquí, por lo que lo mejor es consultar la especificación XPath (http://www.w3.org/TR/xpath-functions) para ver qué funciones se pueden utilizar.

Con las funciones se podrán realizar peticiones que devuelvan valores numéricos. Por ejemplo, "¿cuántos alumnos tenemos?":

```
count(/clase/alumnos/alumno)
```

Que devolverá el número de alumnos del archivo:

```
2
```

O devolver cadenas de caracteres. Por ejemplo, unir todos los nombres separándolos por una coma:

```
string-join(//nombre,",")
```

Que devolverá en un único resultado los nombres separados por una coma:

```
Marcel,Frederic,Filomeno
```

También pueden ponerse las funciones en los predicados. Por ejemplo, esta expresión nos devolverá a los alumnos con apellido que empiece por p.

```
/clase/alumnos/alumno[starts-with(apellidos,"P")]
```

En esta expresión queremos obtener el segundo alumno de la lista:

```
/clase/alumnos/alumno[position()=2]
```

4. XSLT

XSLT (extensible stylesheet language for transformations) es un lenguaje de plantillas basado en XML que permite convertir la estructura de los elementos XML en otros documentos.

Con XSLT se pueden realizar transformaciones que cambien totalmente la estructura del documento, puede reordenar la información del documento XML, añadir información nueva donde sea, tomar decisiones en función de la información que se encuentre, realizar cálculos, etc.

XSLT se apoya en otras tecnologías XML para funcionar:

- Utiliza XPath para determinar las plantillas a aplicar en cada momento (y por tanto se integra en XQuery).
- Soporta XML Schemas para definir los tipos de datos.

La versión 2.0 de XSLT añade una serie de características que aún hacen más potente XSLT:

- Soporta los tipos de datos de XML Schemas
- Incluye nuevos elementos que permiten agrupar resultados, etc.
- Puede generar múltiples salidas en una sola transformación
- Añade un grupo de funciones nuevas además de las de XPath 2.0
- Puede procesar archivos que no sean XML

Una de las limitaciones de XSLT es que el documento de entrada debe ser siempre XML, a pesar de que la salida no tiene esa limitación y puede acabar siendo en cualquier formato: texto, HTML, XML, etc.

4.1. Software XSLT

Uno de los programas de uso corriente que tienen incorporadas bibliotecas XSLT son los navegadores web (Firefox, Internet Explorer, Google Chrome, y otros). Si se carga un documento XML que tiene asociada una transformación XSLT en un navegador web se transformará automáticamente y se mostrará el resultado transformado.

Las plantillas XSLT son documentos XML que se pueden crear utilizando editores de texto o editores especializados en XML. En la creación de plantillas XSLT, las ventajas que ofrecen los editores XML son tan importantes (depuradores de expresiones, ayudas…) que les convierten en una herramienta muy valiosa para trabajar profesionalmente.

Tal y como ya se ha visto en la sección de software de XPath, podemos utilizar tanto Visual Studio Code (extensión XML Tools) como XML Copy Editor para definir y ejecutar transformaciones XSLT.

4.2. El proceso de transformación

Una transformación XSLT consiste en pasar un documento XML y una plantilla por un procesador XSLT, el cual aplicará las reglas que encuentre en la plantilla en el documento para generar un documento nuevo. El archivo de resultado de la transformación puede ser tanto un documento XML como en cualquier otro formato.

Para asociar un documento XML a una plantilla XSLT específica se utiliza la etiqueta <?xml-stylesheet ?>.

Por ejemplo si queremos asociar la plantilla XSLT alumnos.xsl a un documento es necesario editarlo y añadir la etiqueta <?xml-stylesheet ?> tras la declaración xml:

```
<?xml-stylesheet href="alumnos.xsl" type="texto/xsl" ?>
```

La raíz del documento XSLT

Las plantillas XSLT son documentos XML, y por tanto, deben cumplir las reglas de los documentos XML. Esto significa que deben tener un elemento raíz.

La raíz de las plantillas XSLT es el elemento <stylesheet>.

El elemento raíz <stylesheet> debe tener obligatoriamente dos atributos:

- El atributo version, que especificará cuál es la versión de XSLT utilizada para crear la plantilla y que se utilizará para realizar la transformación.
- El espacio de nombres de XSLT, que normalmente se asocia con el sobrenombre 'xsl'.

Por ejemplo, para la versión 2.0 de XSLT la definición de la raíz sería algo así:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="2.0">
</xsl:stylesheet>
```

Dentro de la raíz se definirán las plantillas que determinarán cuál es la transformación que se desea realizar en el documento original.

Crear una plantilla

El proceso de transformación se realizará intentando aplicar alguna plantilla a los nodos del documento original.

Cualquier elemento que no pueda procesarse con una plantilla se transformará con el funcionamiento por defecto:

- Si el nodo tiene contenido se devolverá el contenido del nodo
- Si el nodo no tiene contenido se devolverá sin escribir nada

Por tanto el elemento básico para hacer las transformaciones son las plantillas. Las plantillas se definen con el elemento <xsl:template>.

El elemento de plantilla puede tener varios atributos, pero lo más corriente es match. El atributo match sirve para determinar a qué nodos debe aplicarse esta plantilla por medio de una expresión XPath.

En el contenido del elemento de plantilla se especificará cuál es la transformación a aplicar a los elementos. La transformación más sencilla consiste en escribir un texto literal. Por ejemplo, en la siguiente plantilla se está definiendo que al llegar algún elemento <nombre> se guarde en el archivo de destino la letra A:

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

Por tanto si se aplica la plantilla anterior a este documento:

```
<persona>
    <nom>Pere Garcia</nom>
</persona>
```

El procesador irá analizando los nodos del documento original y primero intentará buscar una plantilla para el elemento <persona>, y como no encontrará ninguna recurrirá al comportamiento por defecto y escribirá una línea en blanco.

Después intentará encontrar una plantilla para el elemento <nombre> que, según la plantilla, debe transformarse en una letra A.

Como no hay más nodos el resultado será éste:

```
—
    En
```

Hay que tener en cuenta que una vez un elemento ha sido procesado por una plantilla se procesa todo su contenido con él (y, por tanto, también los elementos que pueda contener). Si cambiamos la plantilla anterior por la siguiente:

```
<xsl:template match="persona">
    Persona
</xsl:template>
<xsl:template match="nombre">
    En
</xsl:template>
```

Al aplicarla al mismo XML primero procesará el elemento <persona> y lo transformará en “Persona” en el documento de salida, pero como <persona> contiene todos los demás elementos, la transformación se terminará.

```
    Persona
```

Añadir elementos

El hecho de poder hacer transformaciones utilizando texto literal hace que también se pueda utilizar el mismo sistema para crear nuevas etiquetas. Si se modifica la plantilla anterior por una como la siguiente:

```
<xsl:template match="nombre">
    <señor>Manel</señor>
</xsl:template>
```

El resultado será que lo transformará en éste:

```
<señor>Manel</señor>
```

Se debe tener cuidado cuando se especifican etiquetas literales, ya que no se pueden crear etiquetas que dejen la plantilla mal formada. La siguiente plantilla que sólo abre la etiqueta <señor> no se puede utilizar, porque deja la plantilla mal formada:

```
<xsl:template match="nombre">
    <señor>Manel
</xsl:template>
```

Una forma alternativa de crear elementos en una transformación es utilizar <xs:elemento>. Entre los atributos que puede tener, el único obligatorio es name, que define el nombre que tendrá la etiqueta.

Por ejemplo, con la siguiente plantilla creamos un elemento <señor> que siempre tendrá de contenido “Pere” para cada elemento <nombre> que se encuentre en el documento.

```
<xsl:template match="nombre">
    <xs:elemento name="señor">Pere</xs:elemento>
</xsl:template>
```

O sea, que la salida de cada elemento <nombre> será:

```
<señor>Pere</señor>
```

Añadir atributos

Los atributos se pueden definir dentro de un nuevo elemento con <xsl:attribute> y al igual que con los nuevos elementos, el único atributo obligatorio es name.

Una transformación que tuviera esto:

```
<xs:elemento name="persona">
    <xsl:attribute name="home">Si</xsl:attribute>
    Marcel
</xsl:elemento>
```

Generaría lo siguiente:

```
<persona home="Si">Marcel</persona>
```

Control de las salidas de texto

En lugar de utilizar literales también se pueden hacer transformaciones en texto utilizando el elemento <xsl:text>. Este elemento es importante cuando se desea controlar mejor los espacios y los saltos de línea que habrá en las salidas.

Podemos hacer que salgan 5 espacios detrás de la letra A definiendo la plantilla de la siguiente forma:

```
<xsl:template match="nombre">
    <xsl:text>A     </xsl:text>
</xsl:template>
```

Si no lo hicimos con <xsl:texto> los espacios de detrás de la A se perderían.

Obtener valores

Una tarea habitual a la hora de realizar una transformación suele ser obtener los valores de los elementos del documento de origen para ponerlos en el elemento de destino. Se puede obtener el valor de los elementos de origen utilizando <xsl:value-of>.

Este elemento evalúa la expresión XPath del atributo select y genera una salida con su contenido.

Por ejemplo, a partir del siguiente XML:

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

Añadirá el valor del elemento <nombre> en el archivo de salida:

```
Pedro García
```

Pero debe tenerse en cuenta que sólo evalúa el primer elemento que cumpla la condición, o sea, que si el fichero de muestra fuera el siguiente:

```
<?xml version="1.0" ?>
<persona>
    <nombre tipo="Profesor">Pere Garcia</nom>
    <nom>Frederic Pi</nom>
    <nom>Manel Puig</nom>
</persona>
```

…la salida de aplicar la misma plantilla sólo sería el primero de los nombres y no ambos, porque tomaría sólo el primer <nombre> de <persona>:

Pedro García

Por tanto es muy importante elegir bien cuáles son las expresiones XPath de las plantillas para evitar este comportamiento. En este caso la solución podría ser elegir <nombre> en vez de <persona>:

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

Como resultado del elemento <xsl:value-of> se puede poner cualquier expresión XPath. Por tanto, también se pueden obtener los valores de los atributos añadiendo ”@” delante del nombre.

```
<xsl:template match="nombre">
    <xsl:value-of select="."/> ( <xsl:value-of select="@tipus" /> )
</xsl:template>
```

Que generará:

Pere Garcia (Profesor)

Federico Pi ( )

Manel Puig ( )

También se pueden utilizar las funciones XPath. Por tanto, se puede generar una salida con el total de las personas del archivo:

```
<xsl:template match="/">
    Total: <xsl:value-of select="count(//nombre)"/>
</xsl:template>
```

Que dará de resultado:

Total: 3

Reenviar nodos a otra plantilla

Otro uso habitual de las plantillas es el de utilizarlas para llamar a otras cuando se quieren procesar algunos de los nodos obtenidos por la plantilla cuando ésta obtiene varios.

Las llamadas a plantillas se pueden realizar con el elemento <xsl:apply-templates>. Con este elemento se puede hacer que los distintos resultados de una expresión XPath se vayan procesando uno a uno, buscando una plantilla donde aplicarse.

Si partimos de este código XML:

```
<clase>
    <nom>Frederic Pi</nom>
    <nom>Filomeno Garcia</nom>
    <nom>Manel Puigdevall</nom>
</clase>
```

Y la siguiente hoja de plantillas:

```
<xsl:template match="/">
    <xsl:apply-templates select="clase/nombre"/>
</xsl:template>
<xsl:template match="nombre">
    <xsl:value-of select="."/>
</xsl:template>
```

Al ejecutarse el apply-templates se evalúa la expresión XPath que contiene, clase/nombre, que devuelve tres resultados:

```
<nom>Frederic Pi</nom>
<nom>Filomeno Garcia</nom>
<nom>Manel Puigdevall</nom>
```

Cada uno de estos resultados individualmente intentará encontrar una plantilla en la que aplicarse. De modo que se procesará tres veces el segundo <xsl:template>, ya que es quien captura nombre, y dará como resultado:

```
Federico Pi
Filomeno Garcia
Manel Puigdevall
```

Una de las características interesantes es que <apply-templates> también se puede utilizar para reordenar el contenido. Por ejemplo, con el siguiente XML:

```
<clase>
    <profesores>
        <nom>Marcel Puig</nom>
        <nom>Maria Sabartés</nom>
```

```
        </profesores>
        <alumnos>
            <nom>Frederic Pi</nom>
            <nom>Filomeno Garcia</nom>
            <nom>Manel Puigdevall</nom>
        </alumnos>
    </clase>
```

Se pueden obtener todos los nodos con una plantilla que capture la raíz y que primero muestre los nombres de los alumnos y después los de los profesores, de la siguiente manera:

```
<xsl:template match="clase">
    Alumnos
    ----------
    <xsl:apply-templates select="alumnos/nombre"/>
    Profesores
    ----------
    <xsl:apply-templates select="profesores/nombre"/>
</xsl:template>
<xsl:template match="nombre">
    <xsl:value-of select="."/>
</xsl:template>
```

Que generará:

```
Alumnos
----------
Federico Pi
Filomeno Garcia
Manel Puigdevall
Profesores
----------
Marcel Puig
Maria Sabartés
```

Puede verse que uno de los usos de este elemento puede ser reordenar los resultados. En el ejemplo, los alumnos en el documento original estaban detrás de los profesores, y en cambio en lo que hemos obtenido están enfrente.

Procesar los nodos para realizar tareas diferentes

A veces puede interesar procesar los mismos nodos varias veces para obtener distintos resultados. Para ello se pueden definir las plantillas que tengan el atributo modo.

Si se define una plantilla con el atributo modo para activar la plantilla no será suficiente que cuadre con el atributo match, sino que también será necesario que se pase el atributo modo.

```
<template match="nombre" modo="resultado">
    ...
</template>
```

Las plantillas con el atributo modo deben ser llamadas específicamente. Por ejemplo, con <apply-templates>:

```
<xsl:apply-templates select="nombre" modo="resultado"/>
```

Esto permite hacer dos cosas diferentes con el elemento <alumnos> del ejercicio anterior: contar a los alumnos y hacer una lista.

```
<xsl:template match="clase">
    <xsl:apply-templates select="//alumnos" mode="resultado"/
>
    <xsl:apply-templates select="//alumnos"/>
</xsl:template>
<xsl:template match="alumnos">
    <xsl:apply-templates select="nombre" />
</xsl:template>
<xsl:template match="alumnos" mode="resultado">
    Total: <xsl:value-of select="count(nombre)"/>
</xsl:template>
<xsl:template match="nombre">
    <xsl:value-of select="."/>
</xsl:template>
```

El resultado será:

```
Total: 3
Federico Pi
Filomeno Garcia
Manel Puigdevall
```

Forzar el tipo de salida

XSLT está pensado para generar salidas en texto, XML, HTML y XHTML. Por defecto considera que la salida será otro documento XML y, por tanto, si se quiere generar un documento XML no es necesario especificar nada.

Podemos forzar que la salida sea otra con <xsl:output>. Por ejemplo, podemos hacer que la salida sea interpretada como texto plano con:

```
<xsl:output type="texto">
```

Si el atributo type no es “xml”, no aparecerá la cabecera XML en el resultado final.

Pero aparte de type existen otros atributos que permiten controlar cómo se generarán los resultados. Por ejemplo, indent sirve para formatear las salidas y encoding para definir la codificación.

Instrucciones de control

XSLT proporciona una forma de procesar los resultados muy parecida a cómo lo hacen los lenguajes de programación. Como muchos lenguajes de programación, incorpora instrucciones para procesar los resultados.

Expresiones condicionales

<xsl:if> permite añadir los resultados al archivo sólo si se cumple una determinada condición.

Por ejemplo, delante de un documento como el siguiente:

```
<alumnos>
    <alumno nota="5">Pere Garcia</nota>
    <alumno nota="3">Manel Puigdevall</nota>
</alumnos>
```

Con <xsl:if> se puede conseguir poner algún texto sólo en los alumnos que tengan una nota superior a 5.

```
<xsl:if test="@nota>=5">
   (aprobado)
</xsl>
```

Existen otras funciones para expresar alternativas, como <xsl:choose>, que permite explicitar múltiples alternativas.

```
<xsl:choose>
    <xsl:when select="@nota=10"> (excelente) </xsl:when>
    <xsl:when select="@nota>=5"> (aprobado)                    </xsl:when>
    <xsl:otherwise> (suspendido) </xsl:otherwise>
</xsl:choose>
```

Iteraciones

<xsl:for-each> sirve para procesar una secuencia de nodos uno por uno para realizar alguna acción en cada uno:

```
<xsl:for-each select="/clase/alumnos/nombre">
    <xsl:value-of select="."/>
</xsl:for-each>
```

5. Enlaces de interés

XSLT en w3schools: https://www.w3schools.com/xml/xsl_intro.asp

Especificación XSLT 1.0 a w3: www.w3.org/TR/xslt

Especificación XSLT 2.0 a w3: www.w3.org/TR/xslt20/

Xpath en w3schools: https://www.w3schools.com/xml/xpath_intro.asp

Hojas de estilo CSS en XML, mclibre: https://www.mclibre.org/consultar/xml/lecciones/xml- css.html

Apache FOP: https://xmlgraphics.apache.org/fop/

6. Bibliografía

Sala, Javier. (2023) Lenguajes de marcas y sistemas de gestión de información.

Instituto Abierto de Cataluña