XSLT / XPath

Agustín Cernuda del Río
Departamento de Informática
Universidad de Oviedo

Justificación de XSLT (I)

- Documentos XML
  - Documentos de texto con etiquetas
  - Contienen esencialmente información (no se entra en detalles de presentación)
  - La información se organiza jerárquicamente
  - Aunque son legibles por un humano, se supone que la información se procesará
  - Así se obtendrán formas de (re)presentación más adecuadas
  - Es posible que esas representaciones impliquen también seleccionar información
- Necesidad de algún medio para expresar la transformación de un documento XML
  - En otro documento XML (seleccionando, reordenando, calculando...)
  - En un documento que una persona pueda utilizar de manera directa (leer, imprimir...)

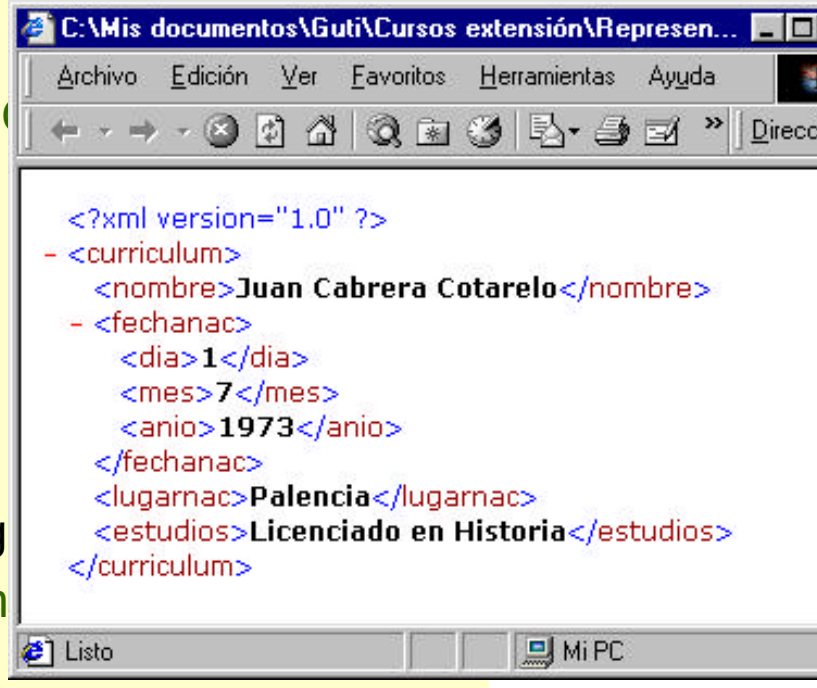
Justificación de XSLT (II)

- Pensemos en nuestra propia solución
- Fichero XML de ejemplo:

Instrucciones de procesamiento

Elemento raíz o documento

```
<?xml version="1.0"?>
<curriculum>
        <nombre>Juan Cabrera (
        <fechanac>
                <dia>1</dia>
                <mes>7</mes>
                <anio>1973</anio>
        </fechanac>
        <lugarnac>Palencia</lug
        <estudios>Licenciado en
</curriculum>
```

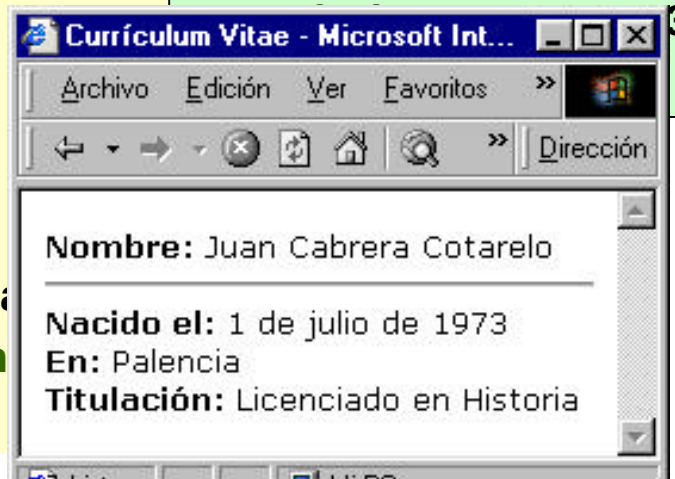


Justificación de XSLT (III)

- Supongamos que necesitamos extraer sólo la información de nombre y fecha de nacimiento
- Supongamos que, dependiendo del caso, necesitamos que la fecha aparezca en español, o en inglés, o con el año de dos dígitos, o sólo el año, o...
- Del mismo documento pueden extraerse otros muy diversos:

```
<?xml version="1.0"?>
<curriculum>
    <nombre>Juan Cabrera Cotarelo</nombre>
    <fechanac>
        <dia>1</dia>
        <mes>7</mes>
        <anio>1973</anio>
    </fechanac>
    <lugarnac>Palencia</lugarnac>
    <estudios>Licenciado en Historia</estudios>
</curriculum>
```

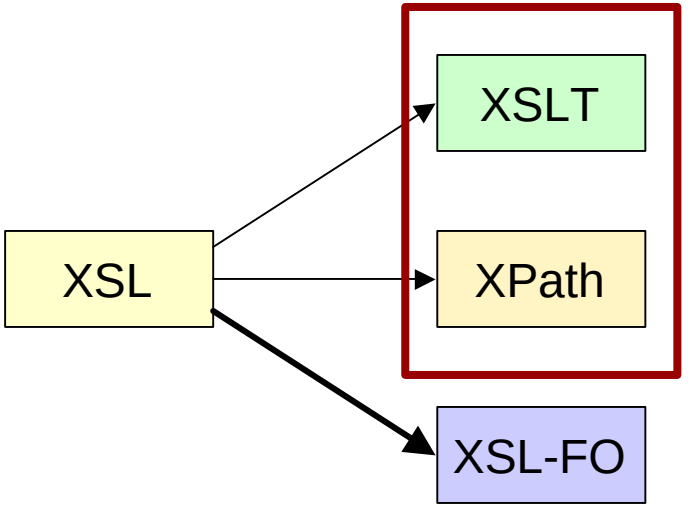
```
<?xml version="1.0"?>
<curriculum>
    <nombre>Juan Cabrera Cotarelo</nombre>
    <fechanac>1/7/1973</fechanac>
</curriculum>
```



Juan Cabrera Cotarelo, nacido el 1/7/1973

Nomenclatura de las tecnologías

- XSLT: eXtensible Stylesheet Language for Transformations. Permite definir cómo se transforma un documento XML en otro documento XML
- Además de XML, se puede generar como salida HTML y texto
- XSL-FO: eXtensible Stylesheet Language – Formatting Objects. Conversión del XML en un formato “imprimible” y legible por una persona (ej.: PDF). Inicialmente, este era el propósito del XSL
- XPath: Una sintaxis para aludir a diversas partes de un documento XML
- La parte de transformaciones ganó en importancia, y se llega a la terminología actual:



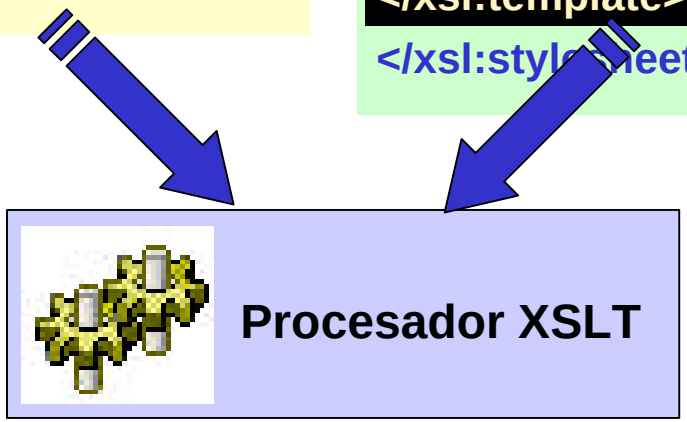
Estructura básica de una hoja XSLT (I)

- Una hoja XSLT transforma un documento XML en:
  - Otro documento XML
  - Un documento HTML
  - Texto
- La hoja XSLT consta de una serie de *reglas*
- Una hoja XSLT es también un documento XML (!)

Estructura básica de una hoja XSLT (II)

```
<?xml version="1.0"?>
<fecha>
    <dia>1</dia>
    <mes>7</mes>
    <anio>1973</anio>
</fecha>
```

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="iso-8859-1"/>
<xsl:template match="fecha">
    <cuando>
        <xsl:value-of select="anio"/>
    </cuando>
</xsl:template>
</xsl:stylesheet>
```




```
<?xml version="1.0" encoding="iso-8859-1"?>
<cuando>1973</cuando>
```

Procesadores XSLT

- Existen varias formas de realizar la transformación
  - Procesador **XSLTPROC**
  - Procesador **MSXML**
    - Un ejecutable que se limita a llamar a la biblioteca de transformación de Internet Explorer
  - Se puede invocar a la biblioteca de transformación desde un programa
  - Enlace entre el fichero XML y la hoja XSLT
    - El fichero se puede ver directamente en Internet Explorer o en otro navegador que soporte XSLT
    - Desventaja: el fichero queda "ligado" a esa vista, por lo menos si se abre directamente
    - Bajo la etiqueta <?xml...?> del fichero XML, se añade

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

Estructura básica de una hoja XSLT (III)

- La hoja XSLT contiene básicamente tres tipos de elementos:
  - Elementos de XSLT. Pertenecen al *namespace* xsl, y por tanto sus etiquetas llevan el prefijo xsl: . Son el equivalente a las palabras clave del lenguaje de programación (definidos por el estándar e interpretados por cualquier procesador de XSLT)
  - Elementos LRE (Literal Result Elements). Son elementos que no pertenecen a XSLT, sino que se repiten en la salida sin más (ejemplo: un elemento <fecha>)
  - Elementos de extensión. Son elementos no-estándar (al igual que los LRE), que son manejados por implementaciones concretas del procesador. Normalmente, no los utilizaremos

xsl:stylesheet

- Es el elemento raíz de una hoja XSL
- Se puede utilizar también xsl:transform (son prácticamente equivalentes)
- Atributos principales:
  - version: Suele ser 1.0
  - xmlns:xsl: Asigna el *namespace* xsl (las etiquetas de XSL empiezan por el prefijo xsl:). El valor para XSLT suele ser **`http://www.w3.org/1999/XSL/Transform`**
- Otros atributos:
  - extension-element-prefixes: Sirve para declarar los prefijos de elementos que deben considerarse elementos de extensión y no LRE (se incluyen separados por espacios)
  - exclude-result-prefixes: Sirve para hacer que los elementos de ciertos *namespaces* (prefijos) **no** se reproduzcan en la salida

Elementos del nivel superior

- Son elementos hijos de xsl:stylesheet
- Además, son hijos directos (tampoco pueden anidarse)
- Dos excepciones: xsl:variable y xsl:param
- No son instrucciones sobre cómo procesar elementos, sino estructuras contenedoras para instrucciones
- Son los siguientes:

xsl:include	xsl:output	xsl:attribute-set
xsl:import	xsl:key	xsl:variable
xsl:strip-space	xsl:decimal-format	xsl:param
xsl:preserve-space	xsl:namespace-alias	xsl:template

xsl:output

- Define qué tipo de salida se va a generar como resultado
- Atributos:
  - method: puede tomar los valores xml, html y text (y también un valor de extensión, con el prefijo correspondiente)
  - encoding: define la forma de representar caracteres que se adoptará en la salida. Ejemplos:
    - iso-8859-1, UTF-8, UTF-16...
    - windows-1252 (genera los caracteres acentuados con la codificación de Windows)
  - omit-xml-declaration: valores yes o no. Indica si se genera o no la declaración <?xml...?>
  - indent: valores yes o no. Si es yes, el procesador (para salidas xml o html) indentará el resultado

xsl:template (I)

- El bloque fundamental de una hoja XSLT
- Un template tiene dos elementos principales:
  - Una expresión de emparejamiento
  - Las instrucciones de procesamiento que contiene
- Si la expresión de emparejamiento coincide con un elemento del fichero XML, el template entra en acción y se ejecutan las instrucciones de procesamiento
- Las instrucciones de procesamiento pueden ser instrucciones de XSLT propiamente dichas o bien LREs
- Al procesar un template, se va construyendo el árbol resultado, en el que se incorpora el resultado de ejecutar instrucciones o bien los LREs (directamente)

xsl:template(II)

- Atributos:
  - match: Su valor es una expresión que se usa para seleccionar nodos del árbol de entrada (cuando encaja, el template entra en acción)
  - name: Además de cuando encaja, un template puede invocarse explícitamente (en ese caso se necesita que tenga un nombre)
- Ejemplo:

```
<?xml version="1.0"?>
<fecha>
    <dia>1</dia>
    <mes>7</mes>
    <anio>1973</anio>
</fecha>
```

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="iso-8859-1“
    indent=“yes”/>
    <xsl:template match=“/”>
        <raiz>He encontrado un nodo raiz</raiz>
    </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<raiz>He encontrado un nodo raiz</raiz>
```

Ejercicio

- Copiar los ficheros del ejemplo anterior, y probar el procesador XSLTPROC

  `XSLTPROC –o salida.html hoja.xsl fichero.xml`

- Probar también a enlazar directamente el XML con su hoja

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

Recapitulando...

- Elemento fundamental: xsl:stylesheet, en el que incluimos versión (1.0) y el *namespace* xsl
- Dentro de él, los elementos del nivel superior
- Utilizamos xsl:output para decir si la salida es XML, HTML o texto normal, y algunos detalles más de cómo se genera
- Utilizamos xsl:template como bloque básico
- Problemas (en este punto):
  - No hemos visto cómo escribir expresiones match para los templates
  - No hemos visto qué instrucciones podemos utilizar para generar la salida
  - Con lo visto hasta ahora, prácticamente sólo podemos procesar el nodo raíz y generar una salida constante (LRE)
- Primer problema: cómo procesar más allá del nodo raíz

xsl:apply-templates

- El procesador empareja con sus reglas por defecto (recorre todo)
- Si encuentra una regla concreta, aplica esa (**no sigue procesando los hijos de ese nodo, salvo que se lo indiquemos**)
- xsl:apply-templates se utiliza para indicar al procesador que intente emparejar templates con cierto nodo o conjunto de nodos (*nodeset*)
- Atributos:
  - select: Su valor es una expresión XPath de conjunto de nodos. El procesador intentará emparejar ese conjunto de nodos con sus templates respectivos
- Ejemplo: apply-templates1
- xsl:apply-templates permite realizar un tratamiento recursivo de todos los elementos del árbol fuente

Ejercicio

- Escribir una hoja para el fichero “fecha” que a la salida genere un fichero XML en el cual:
  - Cuando se encuentre una fecha en el raíz, creará un elemento de tipo fecha con el texto “Una fecha”
  - Dentro del mismo, aplicará los templates del mes, el día y el año
  - El template del mes se limitará a crear un elemento mes con el texto “Un mes”
  - Los templates de día y año actuarán de manera similar

xsl:value-of

- Resuelto cómo procesar más allá del nodo raíz
- Pero hasta ahora sólo hemos generado salidas *constantes* (LREs)
- ¿Cómo averiguar el contenido de un nodo?
- xsl:value-of
  - Permite evaluar una expresión XPath
  - El contenido del nodo actual viene dado por la expresión **“.”**
- Ejercicio: modificar el anterior para que escriba los valores reales de día, mes y año
- Resumen:
  - xsl:stylesheet
    - xsl:output
    - xsl:template match=...
      - xsl:value-of select=...
      - xsl:apply-templates select=...
- Esos match y select requieren de XPath

Introducción a XPath

- XPath es una especificación del W3C (aprobada el mismo día que XSLT)
- Define cómo acceder a partes de un documento XML
- Se basa en relaciones de “parentesco” entre nodos
- Su estilo de notación es similar a las rutas de los ficheros, pero se refiere a nodos en un documento XML
  - Ejemplo: /fecha/dia
- XPath se usa en XSLT, pero también en XSL-FO, XPointer, XLink, y otros
- En XSLT, XPath se utiliza en los valores de atributos (tales como match o select)
- Frecuentemente, como expresión de emparejamiento

Términos básicos en XPath

- Nodo actual (current node)
  - Es un nodo que *está seleccionado* cuando se va a evaluar una expresión XPath
  - Constituye el punto de partida al evaluar la expresión
- Nodo contexto (context node)
  - Para evaluar una expresión, se van evaluando *subexpresiones* parciales
  - Cada vez que se evalúa una subexpresión se obtiene un nuevo conjunto de nodos (*node-set*) que es el nuevo *contexto* para evaluar la siguiente subexpresión
- Tamaño del contexto (context size)
  - El número de nodos que se están evaluando en un momento dado en la expresión XPath

Expresiones XPath

- Una expresión XPath arroja (tras ser evaluada) una expresión de 4 tipos posibles: conjunto de nodos (node-set), booleano, número, cadena
- Tokens válidos en una expresión XPath
  - Paréntesis y similares: ( ) { } [ ]
  - Elemento actual **.** y elemento padre **..**
  - Atributo @, elemento ***** y separador **::**
  - La coma **,**
  - El nombre de un elemento
  - Tipo de nodo (comment, text, processing instruction, node)
  - Operadores: and, or, mod, div, *, /, //, |, +, -, =, !=, <, <=, >, >=
  - Nombres de función
  - Nombre de eje (*axis*): ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self
  - Literales, entre comillas dobles o simples (se pueden anidar alternadas)
  - Números
  - Referencias a variables ($nombreVariable)

Expresiones XPath

- Hay que considerar una expresión XPath como un “predicado”, que devuelve todo lo que encaja con dicho predicado
- Lo que devuelve es procesado por la regla XSL
- Las expresiones XPath se usan sobre todo en los atributos match, select y test

Node-set

- Node-set
  - Grupo de nodos (no ordenado) resultado de evaluar una expresión XPath
  - Los nodos pueden ser de 7 tipos
    - Elemento
    - Atributo
    - Texto
    - Espacio de nombres
    - Instrucción de procesamiento
    - Comentario
    - Raíz
  - Los elementos de un node-set son siempre hermanos (da igual lo que fuesen originalmente)
  - Sus hijos originales no están incluidos (no hablamos de “subárboles”), pero se puede acceder a ellos

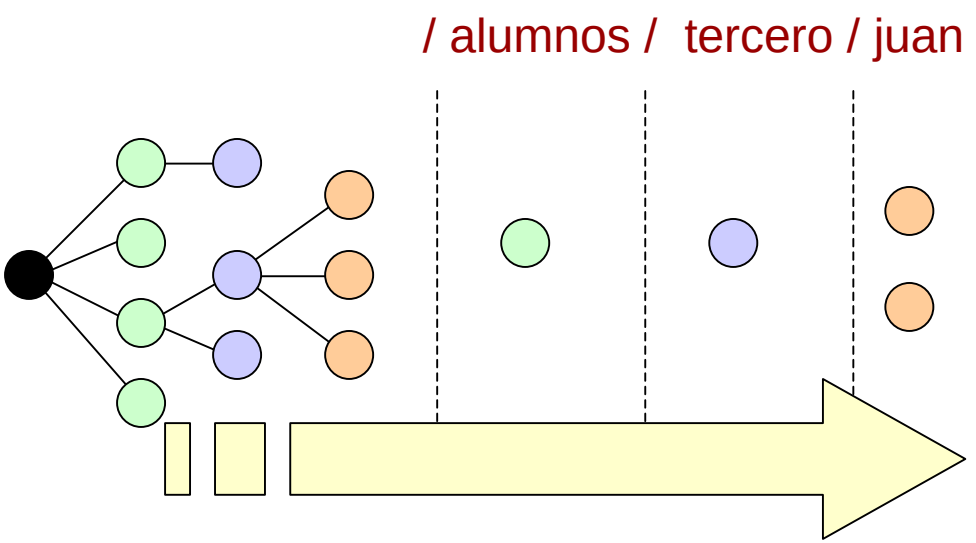
Location path (ruta de localización)

- Se corresponde con la idea intuitiva de “ruta de directorio”
- Un *location path* siempre devuelve un node-set
- Tipos de rutas de localización
  - Patrones (patterns): sólo permiten el uso de los ejes child y attribute (se verá después)
  - Absolutas: parten de la raíz
  - Relativas: no parten de la raíz (depende del nodo de contexto, context node). Este cambia con cada /, que actúa como separador de los pasos de localización. En cada paso se selecciona un nuevo node-set que pasa a ser el nodo de contexto
- Ejercicio: Dado el fichero horario.xml, generar una hoja HTML que muestre los días que aparecen (simplemente el número de día) – horario1.xsl

Pasos de localización

- Paso de localización: cada paso de una ruta de localización (separados por /)
- Un paso de localización consta de:
  - Eje (axis). Es la relación entre el nodo de contexto y el paso
  - Prueba de nodo (node test). Es el “nombre de directorio”
  - Predicado (predicate). Expresión XPath entre corchetes.
- El eje a veces está implícito (no se pone). El predicado es opcional

```
eje::pruebanodo[predicado]
```



Prueba de nodo (node test)

- La forma más simple es escribir simplemente el nombre del nodo (su etiqueta)
- También se puede utilizar el asterisco ***** que simboliza cualquier nombre
- Ejemplos:
  - /universidad/euitio/alumnos/alumno
    Encaja con cualquier nodo “alumno” que sea hijo de un nodo “alumnos” que sea hijo de un nodo “euitio” que sea hijo del nodo “universidad” que será el nodo raíz
  - /universidad/*****
    Encaja con cualquier nodo que sea hijo del nodo “universidad” que será el nodo raíz
  - universidad/*****
    Encaja con cualquier nodo que sea hijo de un nodo “universidad” que sea hijo del nodo de contexto
- **IMPORTANTE**: ***//*** indica “que sea hijo de cualquiera”

Ejes (axis)

- El eje denota la relación de un paso de localización con su nodo de contexto
- Hay una serie de ejes posibles: ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self
- El eje y la prueba de nodo se separan mediante el operador **::**
- Equivale a "**que es un**", pero sus argumentos se leen de derecha a izquierda
- child está implícito y casi nunca se pone. Para el nodo raíz, está implícito self (self denota al nodo de contexto)
- Ejemplos:
  - /universidad/euitio
    Equivale de manera implícita a /self::universidad/child::euitio
  - /universidad/euitio/following-sibling::*
    Todos los nodos que son "hermanos después de" euitio (en el orden del documento) que es hijo de universidad

Predicado

- Añade un nivel de verificación al paso de localización
- Expresión booleana
- Dada la prueba de nodo, y dado el eje, del conjunto de nodos resultante quedan sólo los que cumplan el predicado
- En el predicado pueden intervenir funciones XPath (ojo con las expresiones, > → >)
- Ejercicio: crear una hoja para el horario en la que sólo salgan las tareas después del miércoles (día 3 en adelante) horario2.xsl

Funciones XPath

- Hay una gran variedad de funciones
  - boolean(): convierte a booleano. Aplicada a un conjunto de nodos, devuelve true si no es vacío. not(), true()
  - count(): Devuelve el número de nodos en un conjunto de nodos
  - name(): Devuelve el nombre de un nodo (su etiqueta). local-name(), namespace-uri()
  - position(): Devuelve la posición de un nodo en su contexto (empieza en 1) last()
  - Biblioteca de strings. normalize-space(), string(), concat(), string-length()
  - sum()
- Ejercicio: Escribir una hoja que muestre en HTML todos los nodos de un documento, como listas no numeradas, indicando el número de orden de cada nodo y el número de hijos que contiene (cada elemento irá, además, numerado) – horario3.xsl

Acceso a atributos

- Se puede acceder a un elemento atributo gracias al eje attribute::
- Contiene todos los nodos atributo del nodo contexto
- Una abreviatura de esto es la arroba @
- Ejemplo:
  - <xsl:value-of select="individuo/@edad"/>
    Nodos de nombre "edad" que son atributos de nodos "individuo"
- Ejercicio: Generar una versión del horario que para cada día muestra la lista de tareas (sus nombres) y su prioridad, y también la hora de inicio y fin – horario4.xsl

Acceso a elementos de otro documento XML

- Muy importante: se puede acceder a datos de otro fichero XML
- Uso de la función document()
- Ejercicio: usando el fichero “literales.xml” generar una versión del horario que en vez de “Día 1” muestre “Lunes” y así sucesivamente (NOTA: usar concat(), current() y normalize-space()) - horario5.xsl

Recapitulando...

- Con XPath podemos
  - Seleccionar los nodos para la aplicación de templates
  - Obtener valores (bastante elaborados)
  - La selección de nodos puede basarse en similitud de nombres, en el eje y/o en ciertas condiciones (predicado)

Instrucciones XSL

- No son elementos de nivel superior; son las instrucciones contenidas dentro de los templates
- Indican cómo realizar el procesamiento
- xsl:value-of es un caso simple
- Otras instrucciones permiten realizar tratamientos condicionales, iteraciones, construcción de elementos en el árbol resultado, etc.

Ordenar: xsl:sort

- xsl:sort
  - Se especifica dentro de xsl:apply-templates o xsl:for-each
  - ¿Podría haber sido un atributo?
  - Su atributo es select
  - Indica cómo se establece el orden
- Ejercicio: hacer que el horario salga en orden – horario6.xsl

Condicional: xsl:if

- xsl:if
  - Atributo: test
  - El valor del atributo es una expresión booleana
  - Las instrucciones que contiene se ejecutan sólo si la condición se cumple
- Ejercicio: hacer que en el horario no salga la “Prioridad” si realmente el elemento no tiene tal atributo – horario7.xsl

Condicional: xsl:choose

- xsl:choose
- Contiene elementos xsl:when
  - Atributo: test (similar al de xsl:if)
  - Son los diferentes “casos” de una sentencia CASE
- Caso por defecto: xsl:otherwise (sin atributos)

Iteración: xsl:for-each

- xsl:for-each
  - Atributo: select
  - Aplica las instrucciones de su interior para todos y cada uno de los nodos del conjunto de nodos dado por select
- Ejercicio: Al final del horario, sacar una lista de todas las tareas, indicando si la tarea en cuestión es por la mañana (acaba ANTES de las 12), por la tarde (empieza DESPUES de las 12), o al mediodía (toca a las 12 de cualquier manera) – horario8.xsl
- NOTA: Usar // para recorrer las tareas

Construcción de elementos en el árbol resultado (I)

- ¿Cómo generar un elemento con cierta etiqueta y “construir” sus atributos?
- A veces la sintaxis no nos lo permite directamente. Posible ejemplo:

```
<BODY BGCOLOR=“<xsl:value-of select=“color-elegido”/>”>
```

- Se pueden utilizar los llamados AVT (Attribute Value Template): las expresiones entre llaves se evalúan como si hubiera un value-of
  - Para poner llaves "de verdad", poner cada una dos veces
- Se pueden necesitar instrucciones para “construir” dichos elementos
- xsl:element
  - Construcción de un elemento en el árbol resultado
  - Atributos: name
- xsl:attribute
  - Añadir un atributo al elemento en cuestión
  - Atributos: name
  - El valor está encerrado como texto libre dentro de xsl:attribute

Construcción de elementos en el árbol resultado (II)

- Ejemplo: código XSLT equivalente

```
<BODY BGCOLOR="#00FFFF">
    <P>Esto es una prueba</P>
</BODY>
```

```
<xsl:element name="BODY">
    <xsl:attribute name="BGCOLOR">
        #00FFFF
    </xsl:attribute>
    <xsl:element name="P">
        Esto es una prueba
    </xsl:element>
</xsl:element>
```

Y ahora...

- Mejorar la hoja XSL para el horario: generar mejor HTML (título, por ejemplo)
- Hacer que el horario salga en forma de tabla
- Hacer que salgan todos los días, aunque en el documento XML no estén
- Hacer que cada tarea salga en la casilla que ocupa en el horario
- Utilizar también una hoja CSS