UNIDADE 5

CONVERSIÓN E ADAPTACIÓN DE DOCUMENTOS XML

LINGUAXES DE MARCAS E SISTEMAS
DE XESTIÓN DE INFORMACIÓN

CICLO SUPERIOR DE ASIR

Índice

1 INTRODUCIÓN

Os documentos XML son documentos de texto con etiquetas que aportan información pero que non aportan os detalles de formato ou presentación da mesma.

Nesta unidade imos a ver como se pode modificar a presentación dos documentos XML, para o que existen diferentes ferramentas que abordaremos mediante exemplos.

1.1 Aplicar estilos CSS a documentos XML

As follas de estilo CSS pódense aplicar a documentos XML, aínda que teñen moitas limitacións, pois orixinalmente foron creadas para complementar ao HTML. Do mesmo xeito que XML é unha xeneralización de HTML, o **W3C creou unha xeneralización de follas de estilo CSS chamada XSL (eXtensible Stylesheet Language)**.

Os navegadores web, por exemplo, interpretan as etiquetas que contén un documento XML, aplícalles un formato en base ao especificado nas follas CSS e preséntano ao usuario.

Segundo a recomendación un documento XML pódese ligar a unha folla de estilo mediante a instrución de procesamento:

```
<?xml-stylesheet href="..." ?>
```

Faise de forma similar a unha páxina web XHTML, onde se usa a etiqueta <link />. Nos dous casos, o atributo href inclúe o camiño absoluto ou relativo á folla de estilo CSS.

A diferenza é que a etiqueta *<link />* forma parte da cabeceira (etiqueta <head>), mentres que a instrución de procesamento *<?xml-stylesheet ... ?>* vai ao principio do documento, despois da declaración XML.

1.1.1 CSS baixo o espazo de nomes de HTML

Para que os tres navegadores recoñezan os atributos class e id, unha solución é usar o espazo de nomes html, como no seguinte exemplo.

ex1.xml

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

<?xml-stylesheet href="ex1.css"?>

<usuario xmlns="http://www.w3.org/1999/xhtml">

  <nome class="corNome">Antón Louzao Vila</nome>

  <localidade id="cidade">Compostela</localidade>

  <dataNacemento>1978</dataNacemento>

</usuario>
```

ex1.css

```
.corNome {

    color: red;

    font-size: 25px;

}



#cidade {

    font-size: 30px;

}



dataNacemento{
```

```
    font-family: Verdana, Geneva, Tahoma, sans-serif;

    font-size: 28px;

    color: rgb(6, 133, 133);

    background-color: rgb(202, 146, 146);

}
```

1.1.2 CSS usando o selector de atributo

Tamén se pode usar o selector de atributos usando a seguinte nomenclatura [attribute="value"]) na folla de estilo. Deste xeito poderíamos modificar a súa presentación tal como se amosa nos seguintes exemplo.

ex2.xml

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

<?xml-stylesheet href="ex1.css"?>

<usuario xmlns="http://www.w3.org/1999/xhtml">

  <nome class="corNome">Antón Louzao Vila</nome>

  <localidade id="cidade">Compostela</localidade>

  <dataNacemento>1978</dataNacemento>

</usuario>
```

ex2.css

```
[class=corNome] {

    color: red;

    font-size: 25px;

}



[id=cidade] {

    font-size: 30px;
```

```
}



dataNacemento{

    font-family: Verdana, Geneva, Tahoma, sans-serif;

    font-size: 28px;

    color: rgb(6, 133, 133);

    background-color: rgb(202, 146, 146);

}
```

1.1.3 Exemplo de XML con CSS

A continuación temos un exemplo completo dos conceptos vistos nesta unidade, onde se presenta un documento que representa un listado de receitas, co seu nome, ingredientes e pasos de cociñado.

receitas.xml

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

<?xml-stylesheet href="receitas.css"?>

<receitas xmlns="http://www.w3.org/1999/xhtml">

  <receita categoria="prato-principal">

    <nome>Lacón con grelos</nome>

    <ingredientes>

      <ingrediente>1 lacón</ingrediente>

      <ingrediente>500g de grelos</ingrediente>

      <ingrediente>4 patacas grandes</ingrediente>

      <ingrediente>2 chorizos galegos</ingrediente>

      <ingrediente>2 dientes de allo</ingrediente>
```

```
    <ingrediente>aceite de oliva</ingrediente>
    <ingrediente>sal ao gusto</ingrediente>
  </ingredientes>
  <modoPreparacion>
    <paso>Colocar o lacón nunha pota grande e cubrir con auga·</paso>
    <paso>Cociñar a fogo lento ata que o lacón estea tenro·</paso>
    <paso>Lavar e cortar os grelos·</paso>
    <paso>Levar os grelos a ferver nunha pota con auga e sal·</paso>
    <paso>Cociñar as patacas en auga fervendo con sal·</paso>
    <paso>Preparar un refogado de allo picado con aceite de oliva·</paso>
    <paso>Engadir os grelos escorridos e refogar·</paso>
    <paso>Desosar e cortar o lacón en tarrinas·</paso>
    <paso>Servir o lacón con grelos acompañado das patacas cocidas·</paso>
  </modoPreparacion>
</receita>
<receita categoria="aperitivo">
  <nome>Empanada de Zamburiñas</nome>
  <ingredientes>
    <ingrediente id="ingrediente1">500g de zamburiñas</ingrediente>
    <ingrediente>2 cebolas grandes</ingrediente>
    <ingrediente class="pemento-vermello">2 pementos vermellos</ingrediente>
    <ingrediente class="pemento-vermello">2 pementos verdes</ingrediente>
```

```
    <ingrediente>2 dentes de allo</ingrediente>
    <ingrediente>massa de empanada</ingrediente>
    <ingrediente>aceite de oliva</ingrediente>
    <ingrediente>sal e pementa ao gusto</ingrediente>
  </ingredientes>
  <modoPreparacion>
    <paso>Limpar e cortar as zamburiñas·</paso>
    <paso>Picar as cebolas, os pementos e o allo·</paso>
    <paso>Refogar as verduras nunha sartén con aceite de oliva·</paso>
    <paso>Engadir as zamburiñas e cocer até que estean cocidas·</paso>
    <paso>Estender a masa de empanada e colocar o recheo·</paso>
    <paso>Cubrir cunha segunda capa de masa e selar os bordos·</paso>
    <paso>Pinchar a superficie da empanada e levar ao forno preaquecido·</paso>
    <paso>Cocer ata que a masa estea dourada·</paso>
  </modoPreparacion>
</receita>
<receita id="receita3" categoria="sobremesa">
  <nome>Tarta de Santiago</nome>
  <ingredientes>
    <ingrediente>250g de améndoas sen pel e moidas</ingrediente>
    <ingrediente>250g de azucre</ingrediente>
    <ingrediente>4 ovos</ingrediente>
```

```
      <ingrediente>raspado de limón</ingrediente>
      <ingrediente>azucre en po para espolvorear</ingrediente>
    </ingredientes>
    <modoPreparacion>
      <paso>Misturar as améndoas moidas co azucre.</paso>
      <paso>Engadir os ovos batidos e o raspado de limón.</paso>
      <paso>Colocar a masa nunha forma untada e levar ao forno.</paso>
      <paso>Despois de asar, espolvorear con azucre en po.</paso>
    </modoPreparacion>
  </receita>
</receitas>
```

receitas.css

```
/* Estilo para o elemento <receitas> */
receitas {
    display: flex;
    flex-wrap: wrap;
}

/* Estilo para o elemento <receita> */
receita {
    border-radius: 5px;
    margin: 10px;
    padding: 10px;
```

```
    width: calc(20% - 20px);
    background-color: #f8f9fa;
}

/* Estilo para o atributo categoria=prato-principal */
[categoria=prato-principal] {
    border: 2px solid #c431bc;
    background-color: #a9d2fc;
}

/* Estilo para o atributo categoria=prato-principal */
[categoria=aperitivo] {
    border: 2px solid #2679a0;
    border-radius: 5px;
    background-color: #bcf0b6;
}

/* Estilo para o atributo categoria=sobremesa */
[categoria=sobremesa] {
    border: 2px solid #f1372a;
    border-radius: 5px;
    background-color: #f0e18c;
```

```
}


/* Estilo para o elemento <nome> */
nome {
    display: block;
    font-size: 24px;
    font-weight: bold;
    margin-bottom: 10px;
    color: #dc3545;
}


/* Estilo para o elemento <ingredientes> */
ingredientes {
    margin-bottom: 10px;
}


/* Estilo para o elemento <ingrediente> */
ingrediente {
    display: block;
    margin-bottom: 5px;
    margin-left: 15px;
    color: #28a745;
```

```
}


/* Estilo para o elemento <modoPreparacion> */
modoPreparacion {
    margin-bottom: 10px;
}


/* Estilo para o elemento <paso> */
paso {
    display: block;
    margin-bottom: 5px;
    margin-left: 15px;
    color: #7a0971;
}


/* Estilo para o elemento coa clase .pemento-vermello */
.pemento-vermello {
    font-style: normal;
    font-weight: 800;
    color: #e45195;
}
```

Tomando como referencia o anterior exemplo, podemos observar que neste caso unha das propiedades máis importante é *display*, que establece o modo de visualización do elemento.

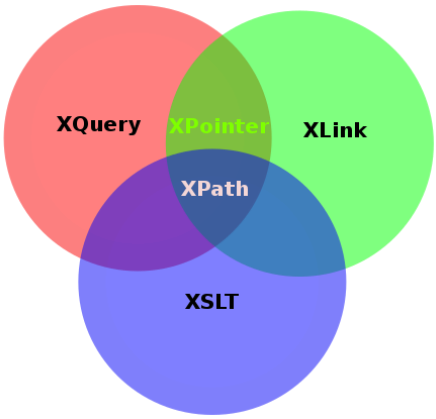
Tal como vimos na primeira avaliación, na que traballamos coas tecnoloxías HTML e CSS, esta propiedade permite varios valores entre os que destacamos os modos de visualización máis utilizados son:

- **none**: permite ocultar elementos.
- **block**: bloque é o modo de visualización dos elementos de tipo bloque, como parágrafos (<p>), títulos (<h1>, <h2>, ...), etc. O elemento ocupa toda a fiestra horizontalmente e ocupa o espazo vertical necesario para acomodar o contido do elemento.
- **inline**:en liña é o modo de visualización de etiquetas como , , , <a>, etc. O elemento só ocupa o espazo necesario para acomodar o contido do elemento. En HTML, os elementos en liña deben estar contidos dentro dos elementos de bloque.
- **list-item**: lista é o modo de visualización dos elementos da lista como <li>, <dd>, <dt>.
- **table, table-row e table-cell**: propiedades de táboa que se combinan para mostrar elementos en forma de táboa.

1.2 Linguaxes para o procesamento de documentos XML

Nas unidades anteriores traballamos con documentos XML e vimos a súa sintaxe para poder crear documentos ben formados. Tamén estudamos dúas linguaxes (DTDs e XML Schemas) que nos permiten definir gramáticas específicas que podemos empregar para validar o contido dos documentos XML.

Ao redor da linguaxe XML definíronse un conxunto de tecnoloxías que se empregan para procesar e obter información dos documentos XML.



Imaxe: Tecnoloxías para manexo de documentos XML

Estas tecnoloxías son:

- **XPath**: É unha linguaxe que permite extraer información dun documento XML.
- **XLink**: É unha linguaxe para a creación de links nun documento XML.
- **XPointer**: É unha linguaxe que permite que os links enlacen con partes específicas dun documento XML.

- **XQuery**: É unha linguaxe que permite realizar consultas a documentos XML, do mesmo xeito que SQL o fai coas bases de datos relacionais.
- **XSLT**: É unha linguaxe que permite transformar documentos XML, obtendo novos documentos XML con diferente estrutura ou documentos noutros formatos.

1.3 Procesamento dun documento XML: estrutura en árbore

Moitas aplicacións integran un parser XML. Un parser é un programa capaz de procesar un documento XML, e xeralmente o resultado obtido é a representación deste documento en forma de árbore. Por exemplo, os navegadores web integran un parser XML, e amosan a árbore obtida como resultado do procesamento que se pode ver a continuación.

Imaxe: Exemplo de documento XML visualizado no Google Chrome

O modelo máis empregado para almacenar e procesar as árbores XML é o DOM (Document Objetc Model, Modelo de Obxectos do Documento). O DOM é un estándar do W3C (World Wide Web Consortium) que tamén se emprega no procesamento dos documentos HTML que se empregan nas páxinas web.

No modelo DOM XML, unha árbore correspondente a un documento XML está composta por nodos. Estes nodos poden ser de distintos tipos. Os principais son:

- **Nodo Document (Documento)**: Representa ao documento XML enteiro. Ten un único fillo de tipo elemento (o nodo raíz).
- **Nodo Element (Elemento)**: Representa un elemento dun documento XML.
- **Nodo Attr (Atributo)**: Representa un atributo dun elemento. Aínda que se lles denomina nodos, na estrutura de árbore considérase aos atributos como unha información engadida aos nodos "Elemento" e non como fillos deles.
- **Nodo Text (Texto)**: Representa ao texto dun elemento. Contén todos os caracteres que non están dentro dalgunha etiqueta.
- **Outros nodos**: Tamén existen outros tipos de nodos, como:
  - Nodo Comment (Comentario).
  - Nodo CDATASection (Sección CData).
  - Nodo ProcessingInstruction (Instrución de Procesamento).
  - Nodo Entity (Entidade).

2 MODELO DOM DO DOCUMENTO

O Modelo de Obxecto do Documento (DOM) é unha representación en memoria dun documento XML que permite acceder, modificar e manexar os seus elementos de forma programática.

O DOM estrutura o documento XML como unha árbore de nodos, onde cada nodo pode representar diferentes partes do documento, como elementos, atributos, texto, comentarios, etc. Os elementos máis destacados do modelo DOM son:

- **Documento**: O nodo raíz da árbore DOM, que representa o documento XML completo. A partir deste nodo, pódese acceder a todos os demais nodos do documento.
- **Elemento**: Trátase dun elemento XML, como unha etiqueta ou tag no documento. Os elementos poden conter outros elementos, atributos, texto ou outros tipos de contido.
- **Atributo**: Representa un atributo dun elemento XML. Os atributos conteñen información adicional sobre os elementos, como identificadores únicos, valores ou meta-datos.
- **Texto**: Representa o contido textual dun elemento XML. Este tipo de nodo contén o texto dentro dun elemento, como unha cadea de caracteres.
- **Comentario**: Representa un comentario dentro do documento XML. Os comentarios son partes do documento que non se procesan no resultado final e serven para documentar ou anotar partes do código.
- **Procesador de instrucións**: Representa instrucións de procesamento, como as instrucións de estilo ou outras meta-instrucións que poden estar presentes no documento XML.

O Modelo de Obxecto do Documento (DOM) proporciona unha maneira poderosa de interactuar con documentos XML dende código, permitindo a creación, lectura, modificación e eliminación de elementos, así como a navegación e manipulación do documento XML como unha estrutura de datos en memoria.

2.1 Documento XML para os exemplos

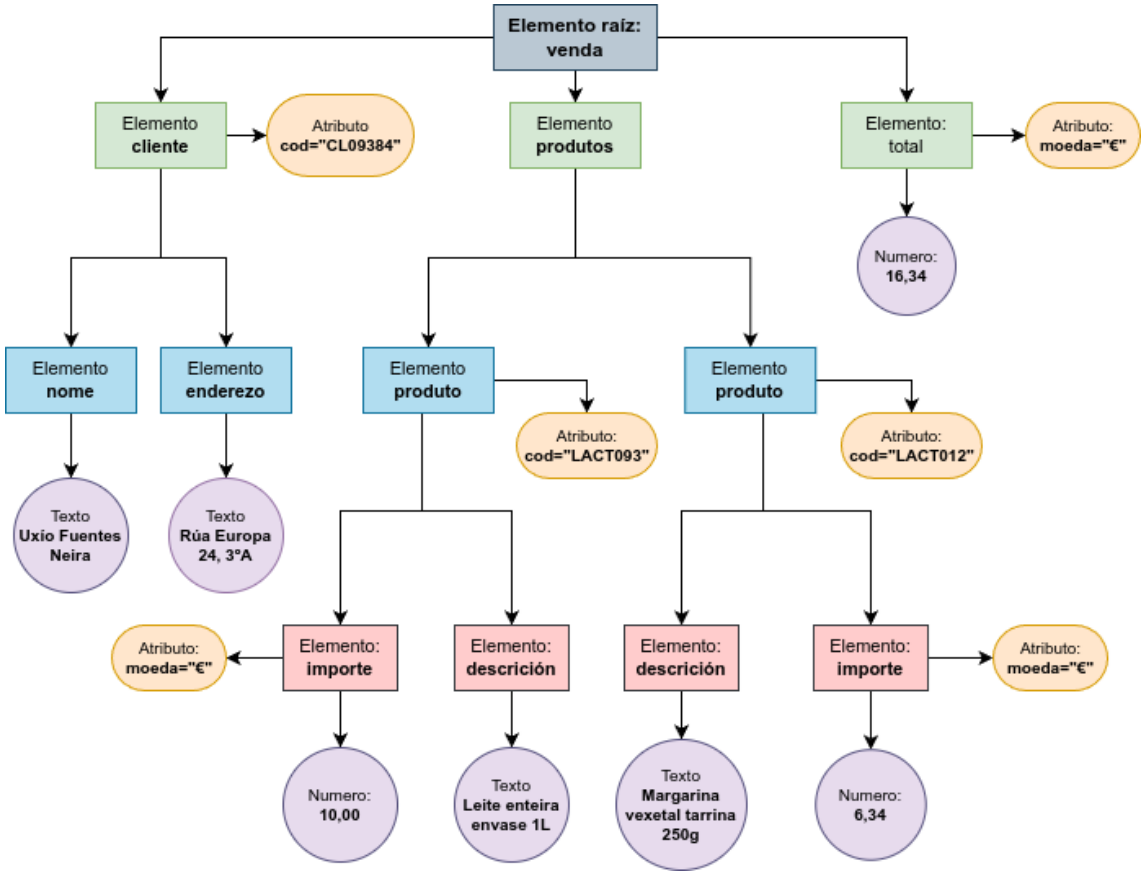
Para os diferentes exemplos que imos ver ao longo desta unidade, será común que fagamos referencia ao seguinte documento XML:

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

<venda factura="F00245" data="2023-11-30">

  <cliente cod="CL09384">

    <nome>Uxío Fuentes Neira</nome>

    <enderezo>Rúa Europa 24, 3ºA</enderezo>
```

```
  </cliente>
  <produtos>
    <produto cod="LACT093">
      <descrición>Leite enteira envase 1L</descrición>
      <importe moeda="€">10·00</importe>
    </produto>
    <produto cod="LACT012">
      <descrición>Margarina vexetal tarrina 250g</descrición>
      <importe moeda="€">6·34</importe>
    </produto>
  </produtos>
  <!--Importe total -->
  <total moeda="€">16·34</total>
</venda>
```

2.2 Modelo DOM do documento

Para o anterior documento XML, co que se traballará ao longo desta unidade, mostramos a representación do seu modelo DOM (Document Object Model):



Imaxe: Representación do modelo DOM para o documento XML.

3 LINGUAXES XSL

XSL (acrónimo de eXtensible Stylesheet Language) é unha familia de linguaxes desenvolvida polo W3C (World Wide Web Consortium) que permite describir como se debe presentar a información contida nun documento XML.

XSL comezou como unha proposta de Microsoft, Inso e Arbortext ao W3C en 1997. A linguaxe SGML, a partir da que se creou XML, xa tiña o seu propio estándar para a representación dos seus documentos que se denominaba DSSSL (Document Style Semantics and Specification Language). Polo tanto, a idea era crear unha linguaxe baseada nesta para a transformación e representación de documentos XML.

As linguaxes de follas de estilo extensible (XSL) permítelle dar formato e reorganizar documentos XML existentes noutro formato e consta de tres recomendacións da W3C:

- **XPath (XML Path Language)**: ofrece a posibilidade de acceder a diversos compoñentes que conforman o documento XML (estándar 3.1 da W3C).
- **XSLT (XSL Transformations)**: Úsase para definir o xeito no que se transforma un documento XML dunha sintaxe a outra. Trátase dunha linguaxe XML que se emprega xunto con Xpath, para converter un documento XML nun documento HTML ou XHTML, nun documento de texto, ou mesmo noutro documento XML. (estándar da W3C).
- **XSL-FO (XSL Formatting Objects)**: É outra linguaxe XML, pensada para dar formato aos datos contidos nun documento XML. Traballa con áreas, bloques, rexións,

anchuras e alturas das páxinas, das marxes, etc. Trátase dunha tecnoloxía bastante utilizada para xerar arquivos PDF a partir de documentos XML.

Malia que debemos coñecer a existencia destas tres tecnoloxías XSL, nesta unidade centrarémonos en traballar con XPath e con follas de estilos XSLT.

4 XPATH

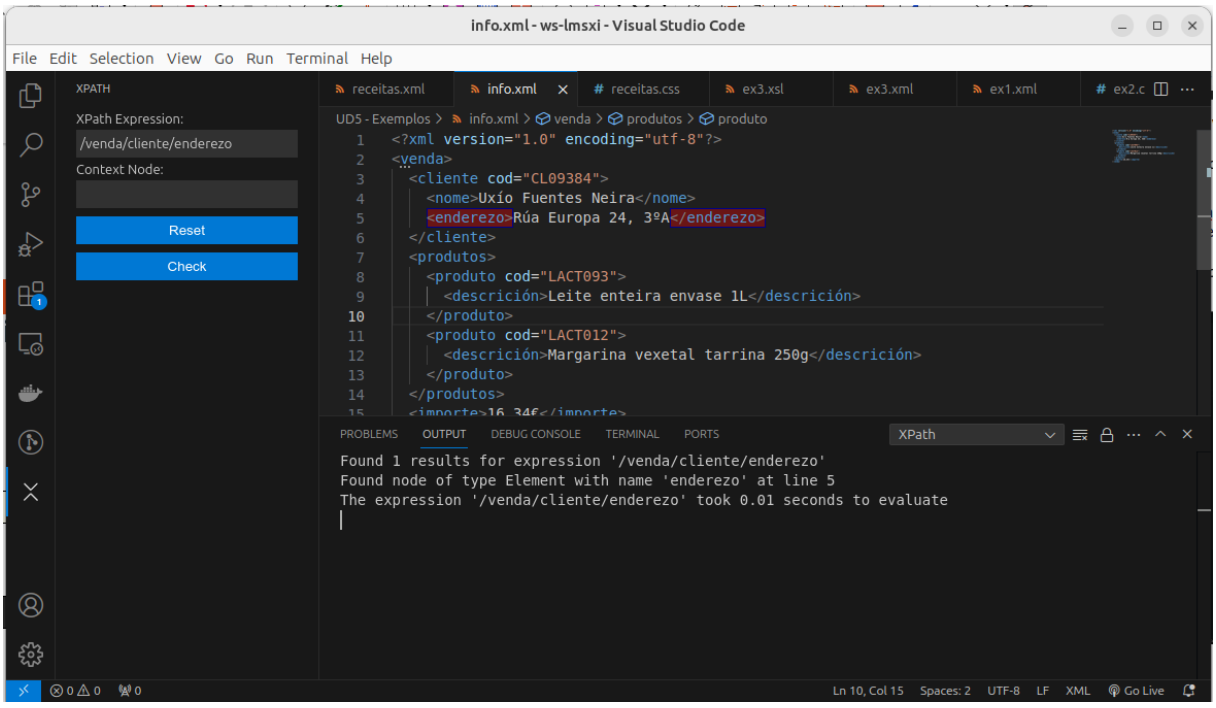
XPath (XML Path) é unha linguaxe para acceder ás distintas partes dun documento XML. Empregando XPath podemos seleccionar e facer referencia a texto, elementos, atributos e calquera outra información contida dentro dun documento XML. Non é unha linguaxe XML; ten a súa propia sintaxe.

Por exemplo, no documento XML anterior poderíamos empregar a seguinte expresión para obter o enderezo do cliente:

```
/venda/cliente/enderezo
```

Existen tres versións de XPath. As dúas primeiras son recomendacións (versións finais) desenvolvidas polo W3C, e a terceira é unha versión candidata. XPath 1.0 (novembro 1999) aínda segue a ser con diferenza a versión máis empregada polo que nos imos a centrar nela.

Existen múltiples ferramentas e páxinas web que permiten a evaluación e xeración de expresións Xpath a partir de documentos XML. No noso caso imos usar a extensión de VS Code coñecida como XPath Tester. Unha vez instalada, veremos no panel de navegación vertical á esquerda, un novo símbolo en forma de X. Na seguinte captura pódese visualizar o que se indica.



Como podemos observar devolve o nodo atopado e ademais de indicar o número de liña na saída, o propio márcao cun sombreado.

Ollo! Para que a extensión XPath Tester funcione correctamente, temos que ter o cursor sobre o documento XML ao que queiramos aplicar a expresión XPath avaliada.

En XPath 1.0, as expresións poden empregar e devolver os seguintes tipos de datos:

- Números en punto flotante.
- Valores booleanos (verdadeiro ou falso).
- Cadeas de caracteres codificadas en Unicode.
- Conxuntos de nodos, que poden conter cero, un ou máis nodos.

4.1 Rutas de localización

O tipo máis común de expresión en XPath é a ruta de localización. Unha ruta de localización permite a selección dun conxunto de nodos, partindo dun nodo contexto no que se avaliará a expresión. O resultado de avaliar unha ruta de localización é sempre un conxunto de nodos (de distintos tipos, non soamente nodos "Elemento").

As rutas de localización poden ser absolutas ou relativas. Se empregamos XPath como parte doutra linguaxe como XSLT, podemos avaliar expresións cuxo contexto sexa un conxunto de nodos dun documento XML (ruta relativa). Estas rutas non comezan cunha barra "/".

Por exemplo, poderíamos avaliar unha expresión tendo como contexto:

```
<produto cod="LACTO12">

  <descrición>Margarina vexetal tarrina 250g</descrición>

</produto>
```

De xeito que a seguinte expresión obteña o nodo "descrición" do produto:

```
produto/descrición
```

Se empregamos XPath sobre un documento XML, as expresións comezarán cunha barra "/" (ruta absoluta), o que indica que o contexto da expresión é o nodo "Documento" (o documento XML enteiro).

As expresións XPath empregan a barra "/" para separar as diferentes partes das que se compoñen. Estas partes chámanse pasos de localización. Cada un dos pasos de localización vai refinando a procura de datos a través dos nodos da árbore.

Por exemplo, a ruta de localización:

```
/venda/cliente/enderezo
```

Está composta de tres pasos de localización. O primeiro fai referencia ao nodo raíz "venda", que colga do nodo documento "/"; o seguinte fai referencia ao nodo "cliente" que é fillo de "venda"; e o mesmo para o nodo "enderezo".

4.2 Pasos de localización

Os pasos de localización (location steps) son cada un dos elementos dunha ruta de localización. Co resultado obtido de avaliar un paso de localización, obtense o contexto do seguinte paso de localización. Constan dun eixo (axis), un test de nodo (node test) e opcionalmente dun predicado. A sintaxe é a seguinte:

```
eixo::test-nodo[predicado]
```

4.3 Eixos

Os eixos indican, con respecto ao nodo contexto, o conxunto de nodos sobre os cales se avaliará o test de nodo e o predicado se existe. Basicamente trátase de realizar un primeiro filtro de nodos da árbore para obter o resultado cos datos buscados.

Os eixos que podemos empregar en XPath 1.0 son:

EIXO	DESCRICIÓN
self	O propio nodo contexto.
child	Os fillos do nodo contexto.
parent	O pai do nodo contexto.
ancestor	Os antepasados do nodo contexto.
ancestor-or-self	O nodo contexto e os seus antepasados.
descendant	Os descendentes do nodo contexto.
descendant-or-self	O nodo contexto e os seus descendentes.
following	Os nodos seguintes ao nodo contexto, sen descendentes.
preceding	Os nodos anteriores ao nodo contexto, sen antepasados.
attribute	Os nodos atributo do nodo contexto.
namespace	Os nodos de espazo de nomes do nodo contexto.

Outros aspectos que debemos ter en conta son:

- ✔ O eixo por defecto é "child", que será o que use no caso de non indicar ningún.
- ✔ O eixo "attribute" pódese substituír polo símbolo "@".

Tomando como exemplo isto que se indica, podemos afirmar que as dúas seguintes expresións son equivalentes:

```
/venda/cliente/@cod

/child::venda/cliente/attribute::cod
```

4.4 Tests de nodo

O test de nodo serve para, unha vez identificado un conxunto de nodos co eixo adecuado, especificar exactamente que nodos dese conxunto son os que queremos. Os tests de nodo poden ser:

TEST DE NODO	DESCRICIÓN
nome_dun_nodo	Selecciona todos os nodos co nome indicado.
*	Selecciona todos os elementos e atributos.
@*	Selecciona todos os atributos.
node()	Selecciona todos os nodos (de calquera tipo).
text()	Selecciona os nodos de texto.
comment()	Selecciona os nodos de comentario.
processing-instructions()	Selecciona os nodos de procesamento de instrucións.

A continuación podemos ver varios exemplos:

- Se quixéramos obter o conxunto de tódolos atributos do documento, poderíamos facer:

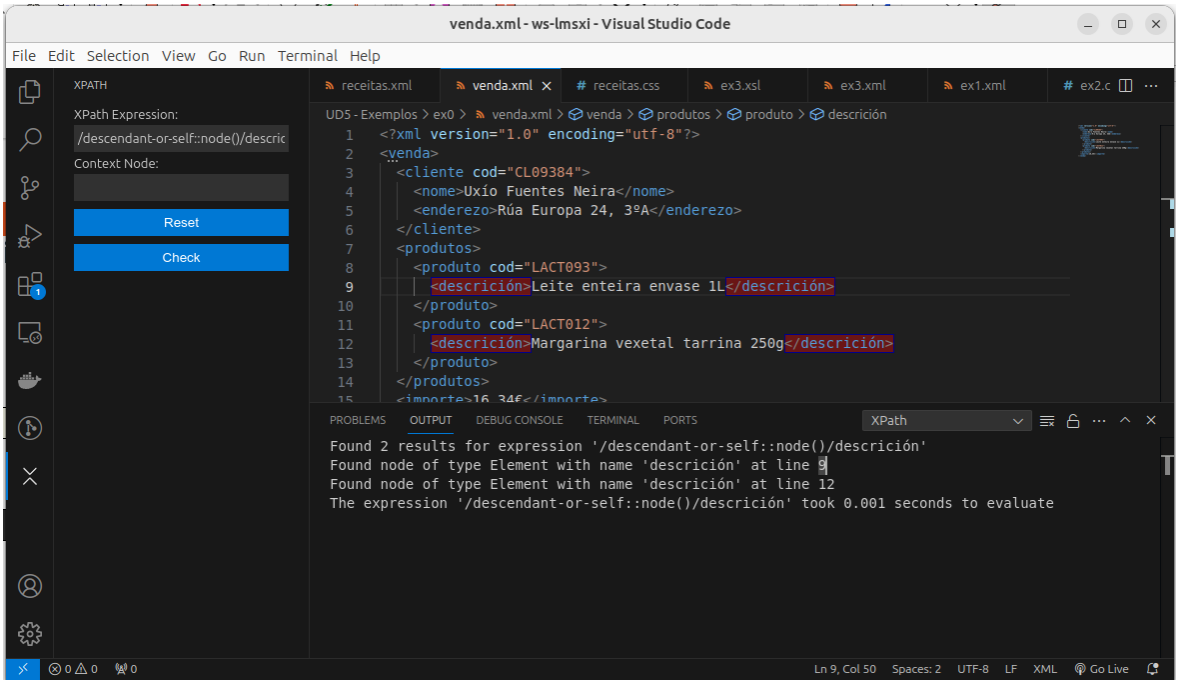
```
/descendant-or-self::node()/@*
```

Isto é, seleccionamos tódolos nodos do documento, e despois quedamos cos seus atributos.

- Podemos obter como resultado un conxunto de nodos sen filtrar, coa seguinte expresión:

```
/descendant-or-self::node()/descrición
```

Tal como se pode apreciar na seguinte captura:



➔ Se quixeramos soamente os atributo de nome "cod", poderiamos facer:

```
➔    /descendant-or-self::node()/@cod
```

➔ Para obter os textos das descricións dos produtos, calquera das seguintes formas é válida. Esta é unha opción:

```
➔    /venda/produtos/produto/descrición/text()
```

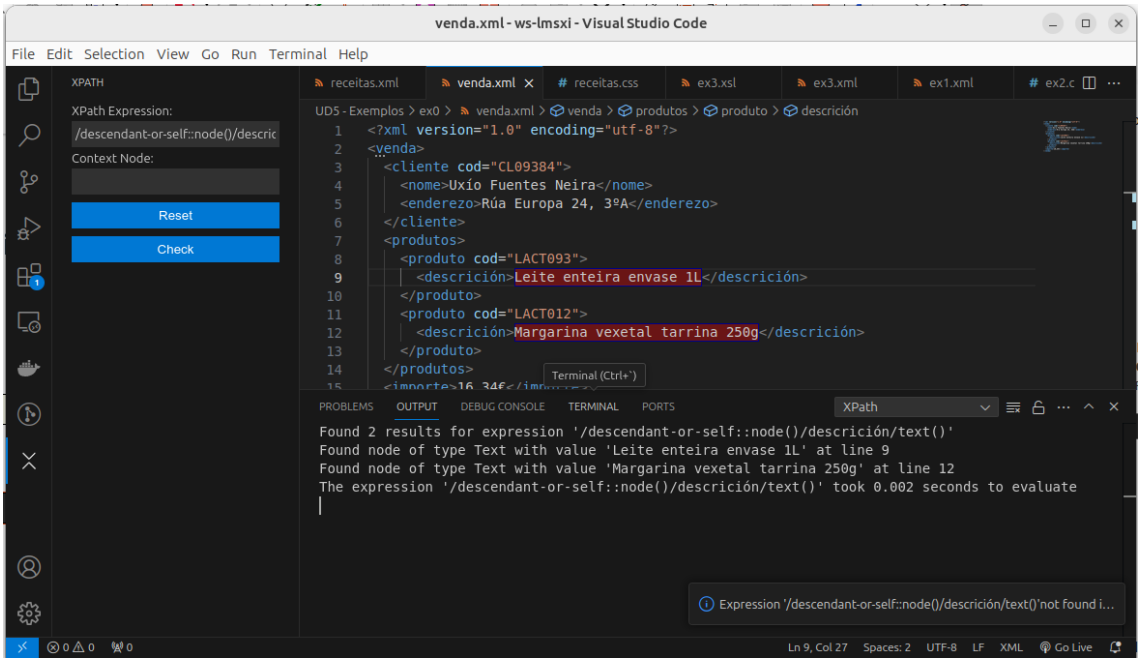
E tamén temos estoutra:

```
➔    /descendant-or-self::node()/descrición/text()
```

➔ Tamén é posible indicar que queremos obter soamente os nodos de tipo "Texto" coa seguinte expresión:

```
/descendant-or-self::node()/descrición/text()
```

Tal como se pode apreciar na imaxe:



4.5 Abreviaturas

Debido ao seu uso frecuente, existen unha serie de abreviaturas para algunhas rutas de localización:

OPERADOR	DESCRICIÓN
//	equivale a /descendant-or-self::node()
.	equivale a /self::node()
..	equivale a /parent::node()

Polo tanto, as seguintes expresións son equivalentes:

```
/descendant-or-self::node()/descrición/text()

//descrición/text()
```

4.6 Predicados

Os predicados son condicións opcionais nun paso de localización, e introdúcense entre corchetes despois do test de nodo. Axúdannos a axustar a busca no conxunto de nodos que nos interesan.

Cada predicado pode conter unha ruta de localización relativa ao nodo actual. Unha forma habitual que se emprega nos predicados fai uso dunha característica especial de XPath: calquera conxunto de nodos non baleiro é tratado de forma booleana como "verdadeiro",

mentres que a un conxunto de nodos baleiro asígnaselle o valor booleano "falso". Desta forma, na expresión:

```
//produto[child::descrición]
```

O predicado filtra os produtos obtendo soamente aqueles que teñen un nodo elemento fillo de nome "descrición". O mesmo poderíase facer da seguinte forma abreviada:

```
//produto[descrición]
```

Tamén podemos poñer condicións ao resultado obtido pola ruta de localización indicada no predicado. Por exemplo, a expresión:

```
//produto[@cod="LACT012"]
```

obtería os nodos "produto" cun atributo "cod" con valor "LACT012". E para obter os clientes cun nome concreto, faríamos:

```
//cliente[nome/text()="Uxío Fuentes Neira"]
```

Un mesmo paso de localización pode conter cero, un ou varios predicados; neste último caso, poranse un a continuación do outro, cadanseu cos seus propios corchetes:

```
//produto[descrición][@cod="LACT012"]
```

4.7 Operadores

Nos predicados, ademais do operador "=", podemos utilizar outros operadores e funcións XPath para filtrar o conxunto de nodos en función do valor dalgunha estrutura do documento. XPath 1.0 define os seguintes operadores que se poden empregar nas expresións.

OPERADOR	TIPO	DESCRICIÓN	EXEMPLO
-	Numérico	Devolve a resta dos operandos.	count(//produto) - 1
!=	Booleano	Devolve verdadeiro se o valor dos dous operandos non coincide, falso en caso contrario.	count(//produto) != 3
*	Numérico	Devolve o produto dos operandos.	count(//produto) * 2
+	Numérico	Devolve a suma dos operandos.	count(//produto) + 1
<	Booleano	Devolve verdadeiro se o valor do primeiro operando é menor que o valor do segundo, falso en caso contrario.	count(//produto) < 3
<=	Booleano	Devolve verdadeiro se o valor do primeiro operando é menor ou igual que o valor do segundo, falso en caso contrario.	count(//produto) <= 3
=	Booleano	Devolve verdadeiro se o valor dos dous operandos coincide, falso en caso contrario.	count(//produto) = 3

>	Booleano	Devolve verdadeiro se o valor do primeiro operando é maior que o valor do segundo, falso en caso contrario.	count(//produto) > 3
>=	Booleano	Devolve verdadeiro se o valor do primeiro operando é maior ou igual que o valor do segundo, falso en caso contrario.	count(//produto) >= 3
|	Pipe (Unión de conxunto de nodos)	Permíte combinar os resultados de dúas ou máis expresións XPath nun conxunto único de nodos, sen duplicados. No exemplo indica que obterá todos os produtos co id indicado e os clientes co nome “Xoel”.	//produto [id=3] | //cliente[nome="Xoel"]
and	Booleano	Devolve verdadeiro se o valor de ambos operandos é verdadeiro, falso en caso contrario.	count(//produto) > 3 and count(//produto) < 7
div	Numérico	Devolve a división dos operandos.	count(//produto) div 2
mod	Numérico	Devolve o resto da división enteira dos operandos.	count(//produto) mod 2
or	Booleano	Devolve falso se o valor de ambos operandos é falso, verdadeiro en caso contrario.	count(//produto) < 3 or count(//produto) > 7

4.8 Funcións

A linguaxe XPath 1.0 define as seguintes funcións que se poden empregar nas expresións.

FUNCIÓN	PARÁMETROS	VALOR DEVOLTO	DESCRICIÓN	EXEMPLO
boolean()	Conxunto de nodos, booleano, numérico ou cadea de caracteres	Booleano	Converte o parámetro a un valor booleano.	boolean(//produto)
ceiling()	Numérico	Numérico	Devolve o primeiro enteiro maior que o valor do parámetro.	ceiling(8 div 3)
concat()	Varias cadeas de caracteres	Cadea de caracteres	Concatena nunha cadea tódalas que se lle pasan como parámetros.	concat("Don ", //nome/text())
contains()	Dúas cadeas de caracteres	Booleano	Devolve verdadeiro se a primeira cadea contén á segunda, falso en caso contrario.	contains(//nome/text(),"Uxío")
count()	Conxunto de nodos	Numérico	Devolve o número de nodos do conxunto de nodos.	count(//produto)
false()		Booleano	Devolve falso.	false()

floor()	Numérico	Numérico	Devolve o primeiro enteiro menor que o valor do parámetro.	floor(8 div 3)
id()	Cadea de caracteres	Conxunto de nodos	Devolve o nodo do elemento co ID especificado como parámetro.	id("G0097763")
lang()	Cadea de caracteres	Booleano	Devolve verdadeiro se a linguaxe definida con xml:lang coincide coa especificada no parámetro, falso en caso contrario.	lang("es")
last()		Conxunto de nodos	Devolve o número de nodos no contexto actual. Pódese empregar para acceder ao último nodo do contexto.	//produto[last()]
local-name()	Conxunto de nodos	Cadea de caracteres	Devolve o nome local (non o nome cualificado) do primeiro nodo no conxunto de nodos que se lle pasa como parámetro.	local-name(//nome)
name()	Conxunto de nodos	Cadea de caracteres	Devolve o nome cualificado do primeiro nodo no conxunto de nodos que se lle pasa como parámetro.	name(//nome)
namespace-uri()	Conxunto de nodos	Cadea de caracteres	Devolve o URI do espazo de nomes do primeiro nodo no conxunto de nodos que se lle pasa como parámetro.	namespace-uri(//produto)
normalize-space()	Cadea de caracteres	Cadea de caracteres	Devolve unha cadea como a que se lle pasa como parámetro, quitando os espazos ao comezo, ao final, e os duplicados.	normalize-space(//nome/text())
not()	Conxunto de nodos, booleano, numérico ou cadea de caracteres	Booleano	Devolve verdadeiro se o valor do operando é falso, verdadeiro en caso contrario.	not (count(//produto) < 3)
position()		Numérico	Devolve a posición (comezando con 1) do nodo contexto no	//produto[position()=2]

			conxunto de nodos do contexto actual.	
round()	Numérico	Numérico	Devolve o enteiro máis próximo ao valor do parámetro.	round(8 div 3)
starts-with()	Dúas cadeas de caracteres	Booleano	Devolve verdadeiro se a primeira cadea comeza coa segunda, falso en caso contrario.	starts-with(//nome/text(),"Uxío")
string-length()	Cadea de caracteres	Numérico	Devolve o número de caracteres da cadea.	string-length(//nome/text())
string()	Conxunto de nodos, booleano, numérico ou cadea de caracteres	Cadea de caracteres	Devolve o parámetro convertido a unha cadea de caracteres.	string(//nome)
substring-after()	Dúas cadeas de caracteres	Cadea de caracteres	Devolve a cadea do primeiro parámetro a partir da primeira ocorrencia do segundo parámetro.	substring-after(//nome/text()," ")
substring-before()	Dúas cadeas de caracteres	Cadea de caracteres	Devolve a cadea do primeiro parámetro anterior á primeira ocorrencia do segundo parámetro.	substring-before(//nome/text()," ")
substring()	1º: Cadea de caracteres 2º e 3º: Numérico	Cadea de caracteres	Da cadea que recibe como primeiro parámetro, devolve tantos caracteres como indique o terceiro parámetro, contando a partir da posición que indique o segundo parámetro.	substring(//nome/text(), 6, 7)
sum()	Conxunto de nodos	Numérico	Devolve a suma dos valores dos nodos que se pasan como parámetros.	sum(/venda/produtos/produto/importe)
translate()	Tres cadeas de caracteres	Cadea de caracteres	Devolve a cadea do primeiro parámetro, substituíndo tódalas ocorrencias dos caracteres do segundo parámetro polos caracteres do terceiro parámetro.	translate(//enderezo/text(),",","-")
true()		Booleano	Devolve verdadeiro.	true()

4.9 Outras expresións

Aínda que a ruta de localización é o tipo máis común de expresión en XPath, podemos empregar os operadores e funcións anteriores para crear diversos tipos de expresións como por exemplo:

- Contar o número de produtos dunha venda (devolve un número):

```
count(//produto)
```

- Comprobar se o número de produtos dunha venda cumpre ou non certas condicións (devolve un valor booleano):

```
count(//produto) > 3 and count(//produto) < 7
```

- Obter os datos dun produto e os do cliente (devolve un conxunto de nodos):

```
//produto[2] | //cliente
```

- Comprobar o valor do código dun produto determinado (devolve un valor booleano):

```
//produto[2]/@cod = "LACTO12"
```

- Comprobar se existe algún produto cun código determinado (devolve un valor booleano):

```
//produto/@cod = "LACTO12"
```

- Obter o importe total da venda, cambiando a coma por un punto (devolve unha cadea de texto):

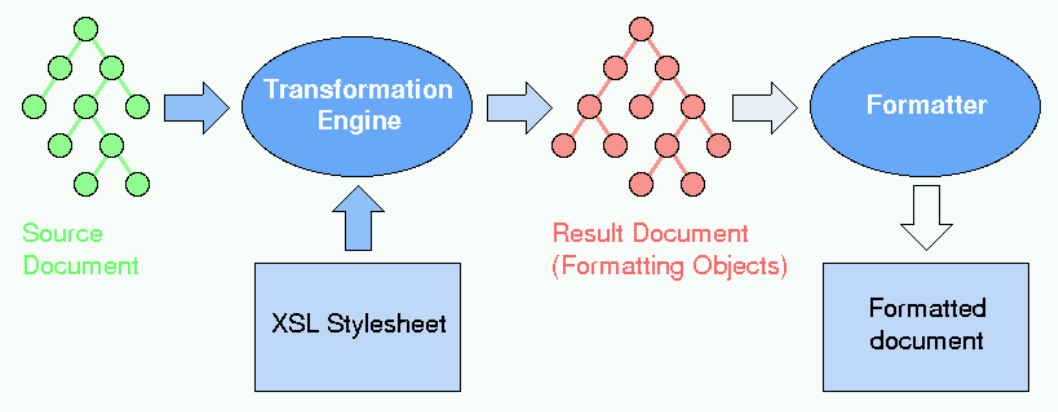
```
substring(translate(/venda/total,",","·"),1,5)
```

5 XSLT

Podemos usar XSL para dar formato a un documento XML, co obxectivo de especificar como se deben presentar os datos na web, e para iso usaremos follas de estilo XSLT.

Tamén se poden usar follas de estilo XSLT para reorganizar un documento XML, para eliminar ou engadir elementos, ou para cambialo a outro documento XML.

XSLT (Transformacións XSL) é unha linguaxe de programación declarativa que permite transformar documentos XML noutros documentos, en formato XML ou incluso noutros (como HTML).



Imaxe: Arquitectura de XSL para dar formato a un documento XML

Para realizar este proceso imos a definir para que serve cada un dos recursos que necesitamos:

- ✔ **O documento XML**: é o documento inicial a partir do cal se xerará o resultado.
- ✔ **A folla de estilo XSLT**: é o documento que contén o código fonte do programa, isto é, as regras de transformación que se aplicarán ao documento inicial.
- ✔ **O procesador XSLT**: é unha aplicación informática encargada de aplicar ao documento inicial o conxunto de regras de transformación incluídas na folla de estilo XSLT, xerando como resultado un documento final. Para probar os exercicios con XLS, usaremos o navegador web.
- ✔ **O resultado da execución do programa**: é un novo documento (que pode ser un documento XML ou non).

5.1 Definición de follas de estilo

As follas de estilo en XSLT (eXtensible Stylesheet Language for Transformations), tamén coñecidas como follas de estilo de transformación, son documentos XML especiais que conteñen instrucións para transformar documentos XML noutros formatos, como pode ser: HTML, XHTML, XML ou texto simple.

Unha folla de estilo XSLT define as regras de transformación que especifican como debe ser procesado cada elemento e atributo do documento XML de entrada. Estas regras indican como os datos deben ser extraídos, modificados e organizados para producir o documento de saída desexado.

Porén, as follas de estilo XSLT son poderosas ferramentas para a transformación de datos XML, permitindo a conversión de documentos XML dun formato a outro, así como a presentación da información de forma flexible e personalizada.

XSLT define unha serie de instrucións que podemos utilizar, o documento XML será procesado polo motor de transformación XSLT, que aplicará a folla de estilo especificada para xerar un novo documento no formato desexado. A folla de estilo XSLT controlará a aparencia e a estrutura do documento resultante en función das regras de transformación especificadas nela mediante o uso de instrucións.

A continuación facemos unha descrición detallada das principais instrucións soportadas pola

linguaxe nunha folla de estilos XSLT.

5.1.1 Declaración do XML

A primeira liña do documento XML debe conter a instrución de procesamento XML, que especifica a versión de XML utilizada e a codificación do documento. Por exemplo:

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

5.1.2 Declaración da folla de estilos: xsl:stylesheet

Para declarar que imos a definir unha folla de estilos XSLT para un documento XML, despois da declaración temos que definir como primeiro primeiro elemento da nosa folla *xsl:stylesheet*, que representa a definición do *elemento raíz dunha folla de estilos XSLT.*

Contén varios atributos, como a versión de XSLT utilizada e o espazo de nomes XML asociado. a de vinculación da folla de estilos XSLT. Por exemplo:

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

    <!-- Instrucións para a definición da folla de estilos XSLT-->

</xsl:stylesheet>
```

Esta instrución contén os seguintes atributos:

- **xmlns:xsl**: O atributo "xmlns:xsl" serve para asociar un prefixo (neste caso, "xsl") a un namespace específico de XML. Neste contexto, o namespace XML especificado é "http://www.w3.org/1999/XSL/Transform", o reservado para a linguaxe de transformación XSLT.
- **version**: atributo que indica a versión de XSL que estamos a usar.

Debemos ter en conta, que se usa o espazo de nomes xsl por convenio, mas podería utilizarse outro (aínda que non é recomendable).

5.1.3 Formato saída: *xsl:output*

O elemento de alto nivel <xsl:output> serve para indicar o tipo de saída producida despois de aplicar as transformacións da folla de estilos XLST a un documento XML.

No seguinte exemplo vemos o seu uso básico:

```
<?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:template>

</xsl:stylesheet>
```

Permite os seguintes atributos:

- **method**: Este atributo especifica o método de saída utilizado para a transformación. Algunhos valores comúns son:
  - *xml*: Indica que o resultado é un documento XML.
  - *html*: Indica que o resultado é un documento HTML.
  - *text*: Indica que o resultado é un documento de texto plano.
- **indent**: Este atributo controla se o resultado debe ser formatado co sangrado adecuado para facelo máis lexíbel. Os valores posíbeis son:
  - *yes*: Indica que o resultado debe ter sangría.
  - *no*: Indica que o resultado non debe ter sangría.
- **encoding**: Este atributo especifica o esquema de codificación de caracteres utilizado para a saída. Algunhas opcións comúns son:
  - *UTF-8*: Codificación Unicode UTF-8.
  - *ISO-8859-1*: Codificación ISO 8859-1 (latin1).
  - *UTF-16*: Codificación Unicode UTF-16.
- **doctype-system**: Este atributo define o sistema de doctype do documento de saída. Utilízase para asociar un documento DTD ao documento XML de saída.
- **doctype-public**: Este atributo define o identificador público do doctype do documento de saída. Utilízase para asociar un documento DTD ao documento XML de saída.

5.2 Tipos de elementos

Existen unha serie de elementos que poden formar parte dos modelos. Estes elementos son instrucións que indican como se debe levar a cabo a transformación.

5.2.1 Modelos: xsl:template

Este elemento define un modelo de coincidencia para un nodo específico no documento XML de entrada e especifica as accións que se deben realizar cando se atopa un nodo que coincide con ese modelo.

- O atributo de coincidencia indica os elementos afectados polo modelo e contén unha expresión XPath.
- O contido da instrución define a transformación a aplicar (se a instrución non contén nada, como no exemplo anterior, substituirá o nodo por nada, é dicir, borrará o nodo, aínda que conservará o texto contido na instrución). elemento).

Os templates son fundamentais para a transformación de documentos, xa que definen como

se debe procesar cada parte do documento de entrada.

A continuación indicamos unha serie de exemplos que usan o documento XML de vendas da unidade.

➔ Para imprimir o número de factura:

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Factura: <xsl:value-of select="/venda/@factura"/></h2>
    </body>
  </html>
</xsl:template>
```

➔ Para obter e imprimir o nome do cliente:

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Nome do cliente: <xsl:value-of select="/venda/cliente/nome"/></h2>
    </body>
  </html>
</xsl:template>
```

➔ Para listar todos os produtos e os seus importes:

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Produtos:</h2>
      <ul>
```

```
        <xsl:for-each select="/venda/produtos/produto">
          <li>
            <xsl:value-of select="descrición"/>: <xsl:value-of select="importe"/>
          </li>
        </xsl:for-each>
      </ul>
    </body>
  </html>
</xsl:template>
```

➔ Para calcular e imprimir o importe total:

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Importe total:</h2>
      <xsl:variable name="total" select="/venda/total"/>
      <xsl:value-of select="concat($total, ' ', $total/@moeda)"/>
    </body>
  </html>
</xsl:template>
```

5.2.2 Texto: <xsl:text>

Emprégase para incluír texto no documento resultante. Tamén se pode escribir o texto directamente no modelo. A principal vantaxe da utilización deste elemento é o control sobre os espazos e saltos de liña que se xerarán.

Os seguintes modelos son similares.

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

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">


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


     <xsl:template match="//produtos/produto">
    <div>
      <xsl:text>Atopado o produto!</xsl:text>
    </div>
     </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:template match="/">
        <html>
            <body>
```

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

        </body>

    </html>

</xsl:template>



<xsl:template match="//produtos/produto">

        <div>Atopado o produto!</div>

</xsl:template>

</xsl:stylesheet>
```

5.2.3 Comentarios: <xsl:comment>

Tamén podemos engadir comentarios ao noso documento xerado, que terán o formato

```
<!-- Isto é un comentario -->
```

Por exemplo:

```
<xsl:comment>Neste exemplo imos mostrar o importe total</xsl:comment>
```

5.2.4 Elementos: <xsl:element>

Crea un novo elemento no documento resultante. O atributo name indica o nome do novo elemento. Este atributo pódese compoñer como combinación de texto, partes do documento orixinal, variables, valores retornados por funcións, etc.

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- Modelo principal -->

  <xsl:template match="/">

      <!-- Engadir o elemento "venda" -->

      <xsl:element name="venda">

      <!-- Engadir contido -->
```

```
      <xsl:text>Contido da venda</xsl:text>

    </xsl:element>

  </xsl:template>

</xsl:stylesheet>
```

5.2.5 Atributos: <xsl:attribute>

Crea un novo atributo no documento resultante. O valor do atributo name, que se pode xerar como no caso anterior, será o nome do novo atributo.

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">



  <!-- Modelo principal -->

  <xsl:template match="/">

    <!-- Crear elemento venda -->

    <xsl:element name="venda">

      <!-- Engadir atributo factura -->

      <xsl:attribute name="factura">F00245</xsl:attribute>

      <!-- Engadir atributo data -->

      <xsl:attribute name="data">2023-11-30</xsl:attribute>

      <!-- Engadir contido -->

      <xsl:text>Contido da venda</xsl:text>

    </xsl:element>

  </xsl:template>

</xsl:stylesheet>
```

O documento resultante da transformación anterior é o seguinte:

```
<venda factura="F00245" data="2023-11-30">

    Contido da venda

</venda>
```

5.2.6 Conxuntos de atributos: <xsl:attribute-set>

Tamén é posible definir un conxunto de atributos para despois empregalo en un ou varios elementos. A continuación deixamos un exemplo do uso de *xsl:attribute-set*:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">



  <!-- Definición do conxunto de atributos -->

  <xsl:attribute-set name="venda-atributos">

    <xsl:attribute name="factura">F00245</xsl:attribute>

    <xsl:attribute name="data">2023-11-30</xsl:attribute>

  </xsl:attribute-set>



  <!-- Modelo principal -->

  <xsl:template match="/">

    <!-- Crear elemento venda e aplicar os atributos -->

    <xsl:element name="venda" use-attribute-sets="venda-atributos">

      <!-- Engadir contido -->

      <xsl:text>Contido da venda</xsl:text>

    </xsl:element>

  </xsl:template>
```

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

5.2.7 Valor dun nodo: xsl:value-of

Este elemento extrae o valor dun nodo no documento XML de entrada e inclúeo no documento de saída. Pode ser útil para incluír contidos de elementos ou atributos nunha parte específica do documento de saída, tal como se viu nos exemplos anteriores.

A continuación deixamos un exemplo do uso de *xsl:value-of*:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Importe total da venda:</h2>
        <p>
          O importe total é:
          <xsl:value-of select="/venda/total"/>
          <xsl:value-of select="/venda/total/@moeda"/>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

5.2.8 Variables: xsl:variable

Trátase dun elemento utilizado en XSLT para declarar e asignar un valor a unha variable local dentro do ámbito dun modelo. Unha vez declarada, o valor da variable pode ser utilizado en calquera parte do mesmo ámbito, pero non pode ser modificado despois de ser asignado.

5.2.9 Parámetros: xsl:param

Representa un elemento utilizado en XSLT para declarar parámetros que se poden pasar ao proceso de transformación XSLT dende o exterior. Un parámetro é semellante a unha variable, pero o seu valor pode ser proporcionado externamente cando se inicia a transformación. Os parámetros permiten que o comportamento da transformación sexa personalizado sen necesidade de modificar o código XSLT en si mesmo

A continuación vemos un exemplo de uso de parámetros (e tamén de variables):

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">



  <!-- Declaración do parámetro moeda -->

  <xsl:param name="moeda" select="'€'"/>



  <!-- Declaración dun parámetro co mesmo nome que o elemento do XML -->

  <xsl:param name="total" select="/venda/total"/>



  <!-- Declaración dunha variable -->

  <xsl:variable name="cliente" select="/venda/cliente"/>



  <!-- Template principal -->

  <xsl:template match="/">

    <html>
```

```
    <body>
      <h2>Información do cliente:</h2>
      <p>Nome: <xsl:value-of select="$cliente/nome"/></p>
      <p>Enderezo: <xsl:value-of select="$cliente/enderezo"/></p>
      <p>Moeda: <xsl:value-of select="$moeda"/></p>
      <p>Prezo con IVE: <xsl:value-of select="$total * 1.21"/></p>
    </body>
  </html>
 </xsl:template>


</xsl:stylesheet>
```

5.2.10 Elementos condicionais

Estes elementos permiten a execución condicional de accións durante a transformación, permitindo que se realicen diferentes accións en función de certas condicións nos datos de entrada.

5.2.10.1 xsl:if

Con este elemento podemos realizar unha condición e executar un bloque de código se esta é verdadeira.

Un exemplo de uso básico sería:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
   <xsl:if test="/venda/cliente/nome = 'Uxío Fuentes Neira'">
     <!-- Se o nome do cliente é 'Uxío Fuentes Neira' -->
     <mensaje>O cliente é Uxío Fuentes Neira</mensaje>
   </xsl:if>
```

```
  </xsl:template>

</xsl:stylesheet>
```

5.2.10.2 xsl:choose

Este elemento permite realizar múltiples condicións e executar o bloque de código da primeira condición que sexa verdadeira. Usarase en combinación cos elementos *xsl:when* e *xsl:otherwise.*

A continuación temos un exemplo de uso:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">

    <xsl:choose>

      <xsl:when test="/venda/total > 10">

        <!-- Se o importe dun produto é maior de 10 -->

        <aviso>O importe é maior de 10</aviso>

      </xsl:when>

      <xsl:when test="/venda/total &lt;= 10">

        <!-- Se o importe dun produto é menor ou igual de 10 -->

        <aviso>O importe é menor ou igual de 10</aviso>

      </xsl:when>

      <xsl:otherwise>

        <!-- Se ningunha das condicións anteriores é verdadeira -->

        <aviso>O importe non está definido ou non é válido</aviso>

      </xsl:otherwise>

    </xsl:choose>

  </xsl:template>
```

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

Ollo! Fíxate que precisamos usar na expresión *test* que algo "<" (menor), como este é un carácter especial da linguaxe, debemos usar a secuencia de escapa que representa a ese carácter "<"

5.2.11 Percorrer un conxunto de nodos: xsl:for-each

Este elemento permite iterar sobre un conxunto de nodos no documento XML de entrada e aplicar accións a cada un deles. É útil cando se necesita procesar repetidamente varios elementos dun mesmo tipo.

O seguinte sería un exemplo de uso:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">

    <!-- Inicio do documento de saída -->

    <html>

      <body>

        <!-- Inicio dun bucle para iterar sobre cada produto -->

        <xsl:for-each select="venda/produtos/produto">

          <!-- Template para cada produto -->

          <xsl:apply-templates select="." />

        </xsl:for-each>

      </body>

    </html>

  </xsl:template>



  <!-- Template para os produtos -->

  <xsl:template match="produto">

    <!-- Saída de información para cada produto -->
```

```
    <div>
      <h2>
        <xsl:value-of select="descrición" />
      </h2>
      <p>Importe: <xsl:value-of select="importe" /> <xsl:value-of select="../total/@moeda" /></p>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

5.2.12 Aplicar modelos: xsl:apply-templates

Este elemento indica que se debe aplicar un modelo (template) específico a un ou máis nodos do documento XML de entrada. Permite a aplicación regular de regras de transformación a diferentes partes do documento. Deste xeito podemos facer que no procesamento dun elemento se procesen tamén os modelos (templates) correspondentes aos seus fillos.

5.2.12.1 Exemplo 1

Por exemplo, para o documento XML de vendas co que estamos traballando poderíamos definir a seguinte folla de estilos XLST:

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


  <!-- Aplica o modelo a todos os elementos -->
  <xsl:template match="/">
    <html>
      <body>
        <h2>Detalles da Venda</h2>
        <xsl:apply-templates />
```

```
    </body>
  </html>
</xsl:template>


<!-- Modelo para os elementos cliente -->
<xsl:template match="cliente">
  <div>
    <p>
      <strong>Nome do Cliente:</strong>
      <xsl:value-of select="nome" />
    </p>
    <p>
      <strong>Enderezo:</strong>
      <xsl:value-of select="enderezo" />
    </p>
  </div>
</xsl:template>


<!-- Modelo para os elementos produtos -->
<xsl:template match="produtos">
  <div>
    <h3>Produtos:</h3>
```

```
    <xsl:apply-templates />
  </div>
</xsl:template>

<!-- Modelo para os elementos produto -->
<xsl:template match="produto">
  <p>
    <strong>Código do Produto:</strong>
    <xsl:value-of select="@cod" />
  </p>
  <p>
    <strong>Descrición:</strong>
    <xsl:value-of select="descrición" />
  </p>
  <p><strong>Importe:</strong> <xsl:value-of select="importe" /> €</p>
  <hr />
</xsl:template>

<!-- Modelo para o elemento total -->
<xsl:template match="total">
  <div>
    <h3>Total:</h3>
```

```
      <p>
        <strong>Importe Total:</strong>
        <xsl:value-of select="." />
        <xsl:value-of select="@moeda" />
      </p>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

5.2.12.2 Exemplo 2

Por exemplo se a este documento:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="modulos.xsl"?>
<ciclo>
  <modulo sesions="5" horas="133">
    <nome>Linguaxes de marcas</nome>
    <profesor>Xaime Louzán</profesor>
  </modulo>
  <modulo sesions="9" horas="239">
    <nome>Programación</nome>
    <profesor>Sira Rego</profesor>
  </modulo>
</ciclo>
```

Se aplicásemos a seguinte transformación ao documento anterior:

```
<xsl:stylesheet xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">


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


    <xsl:template match="/ciclo">
        <xsl:for-each select="/ciclo/modulo">
            <div style="background-color: lightblue;">
                <!-- Crear un novo elemento "modulo-ciclo" -->
                <xsl:element name="modulo-ciclo">
                    <h1>
                        <!-- Engadir o contido de modulo ao elemento -->
                        <xsl:value-of select="nome" />
                    </h1>
                </xsl:element>
                <!-- Crear un novo elemento "docente" -->
```

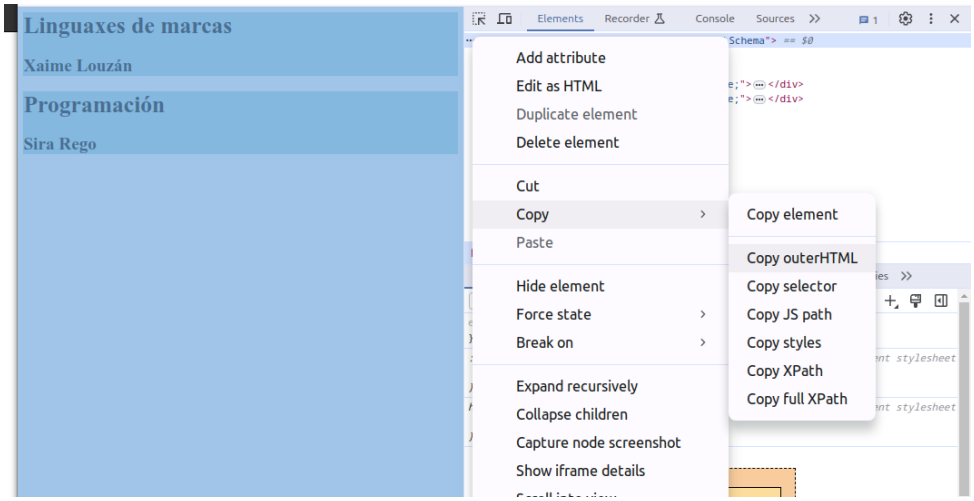
```
            <xsl:element name="docente">
                <h2>
                    <!-- Engadir o contido de modulo ao elemento -->
                    <xsl:value-of select="profesor" />
                </h2>
            </xsl:element>
        </div>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Neste caso, ao procesar o elemento "<modulo>" que está contido dentro do elemento raíz "curso", define:

- *Un modelo para o raíz, para o que define o esqueleto dun HTML, e a posteriori aplica o template ou modelo definido para "ciclo", que percorre os módulos (/ciclo/modulo).*
- *Para cada un deses módulos que percorre , obtén:*
  - *Un novo elemento chamado <modulo-ciclo> que contén cada un dos nomes do módulo, envolto nunha cabeceira HTML de tipo h1.*
  - *Un novo elemento chamado <docente> que contén cada un dos nomes do módulo, envolto nunha cabeceira HTML de tipo h2.*

Podemos comprobar o resultado da transformación indo ao documento XML a través do servidor estático de VS Code, e unha vez dentro, mostraranos a páxina. Se lle damos a ver código, mostraranos o XML vinculado á folla de estilos XSLT.

Para poder obter o resultado obtido, que neste caso é un HTML que carga o propio navegador, temos que abrir o inspector de código e seguir a indicación da seguinte captura.



Imaxe: Ver o código xerado no Chrome a través do inspector de DevTools

O resultado que se obtén é este documento:

```
<html xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <head></head>
  <body>
    <div style="background-color: lightblue;">
      <modulo-ciclo>
        <h1>Linguaxes de marcas</h1>
      </modulo-ciclo>
      <docente>
        <h2>Xaime Louzán</h2>
      </docente>
    </div>
    <div style="background-color: lightblue;">
      <modulo-ciclo>
        <h1>Programación</h1>
      </modulo-ciclo>
```

```
      <docente>
        <h2>Sira Rego</h2>
      </docente>
    </div>
  </body>
</html>
```

É moi habitual atopar documentos XSLT coa seguinte estrutura:

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

    <xsl:template match="nodos1">
            ...
            <xsl:apply-templates />
            ...
    </xsl:template>

    <xsl:template match="nodos2">
            ...
```

```
            <xsl:apply-templates />

            ...

    </xsl:template>

</xsl:stylesheet>
```

5.3 Aplicar estilos XSLT a un documento XML

5.3.1 Declaración do XML

A primeira liña do documento XML debe conter a instrución de procesamento XML, que especifica a versión de XML utilizada e a codificación do documento. Por exemplo:

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

5.3.2 Vincular XSLT: xml-stylesheet

Para vincular un documento XML cunha folla de estilo XSLT comezamos con esta palabra clave: *xml-stylesheet.* Despois da primeira instrución de procesamento XML, engádese esta, que é a de vinculación da folla de estilos XSLT. Por exemplo:

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

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

Son necesarias certas instrucións que se inclúen no propio documento XML e que basicamente indican ao procesador XML cal é a folla de estilo que debe aplicar durante a transformación do documento XML noutro formato, como pode ser HTML ou texto plano. Para facer iso, contén os seguintes atributos:

- **type**: atributo opcional que especifica o tipo de contido da folla de estilo: text/css, text/xsl.
- **href**: Atributo que contén o URL ou a ruta relativa da folla de estilo XSLT.

5.3.3 Resto do documento XML

No resto do documento XML, despois das instrucións de procesamento, o documento XML continúa co seu contido, e polo tanto a definición dun nodo ou elemento principal, do que colgará toda a súa estrutura (sub-elementos, atributos e contido de texto).

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

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

<root>
```

```
    <!-- Resto do documento XML -->

</root>
```

5.3.4 Abrir nun navegador web

Cando se abre unha páxina XML vinculada a unha folla de estilos XSLT nun navegador, este visualiza o resultado da transformación. Non se mostra o código fonte obtido como resultado, senón que se interpreta ese código fonte xerado (XML, HTML ,...).

Ollo! Google Chrome non mostra documentos XML que se vinculan con follas de estilo XSLT como ficheiros locais (file://...).

Para evitar ese problema de Chrome, pódese utilizar un servidor de contido estático. No módulo imos traballar coa extensión "Live Server" de VS Code.

5.4 Material de referencia

A a elaboración desta unidade fíxose consultando a documentación oficial das diferentes aplicacións e linguaxes. **Outra fonte** importante para elaborar este material, **foi o material realizado por Xaime Louzán**, **ao que agradecemos o seu traballo**.