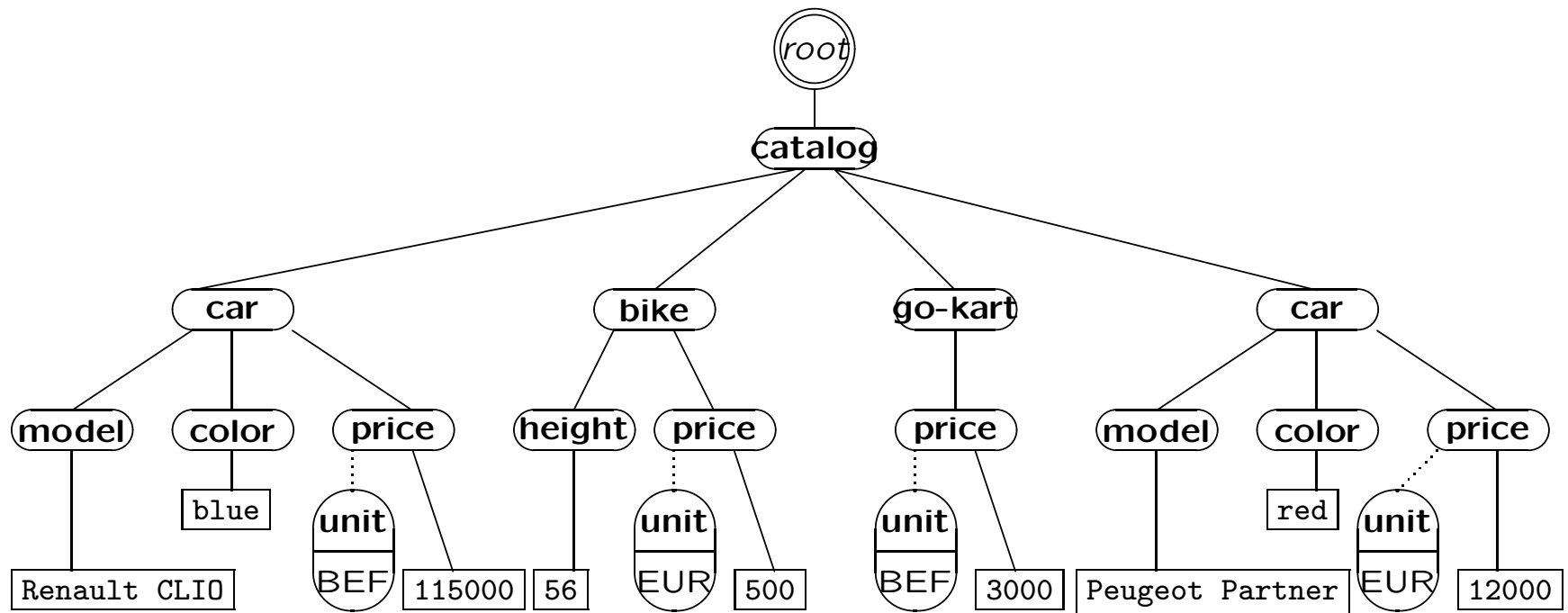


XML XPath XSLT XQuery

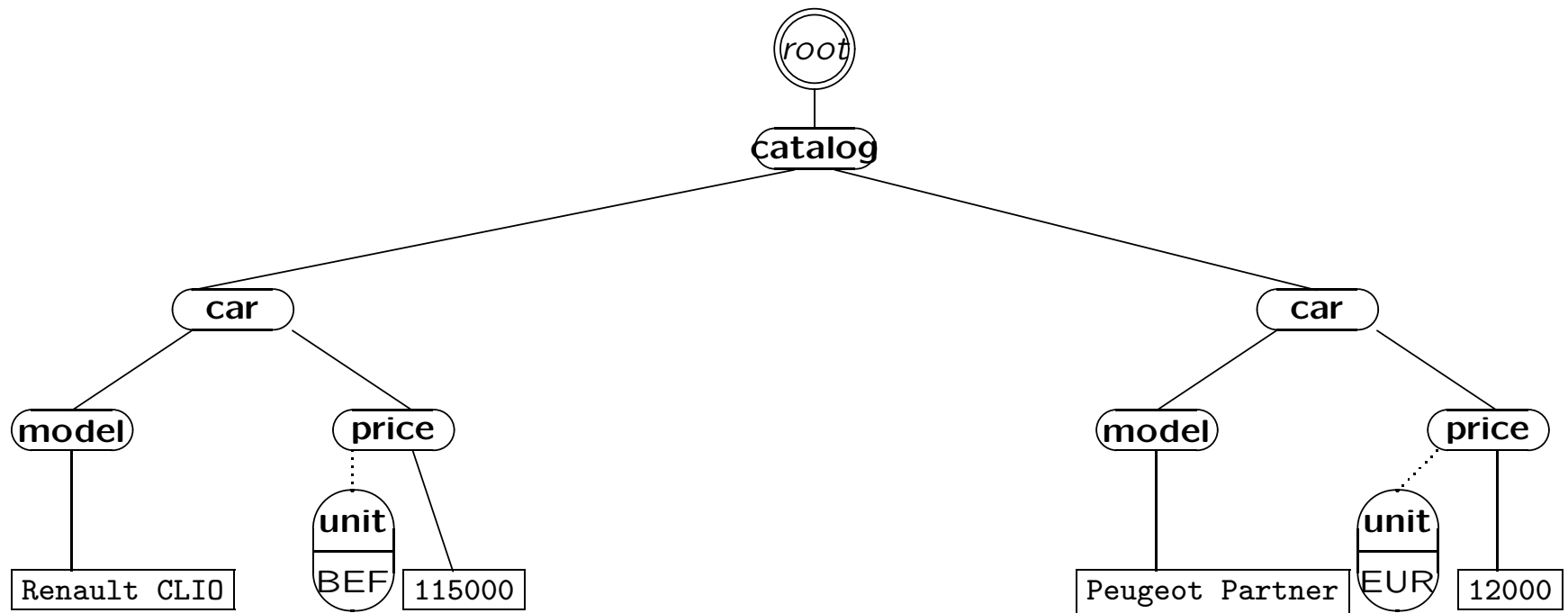
Jef Wijsen

September 16, 2025

Why XML? Semistructured data



Querying: Get models and prices of all cars.



Text $\leftrightarrow$ Tree

<A>

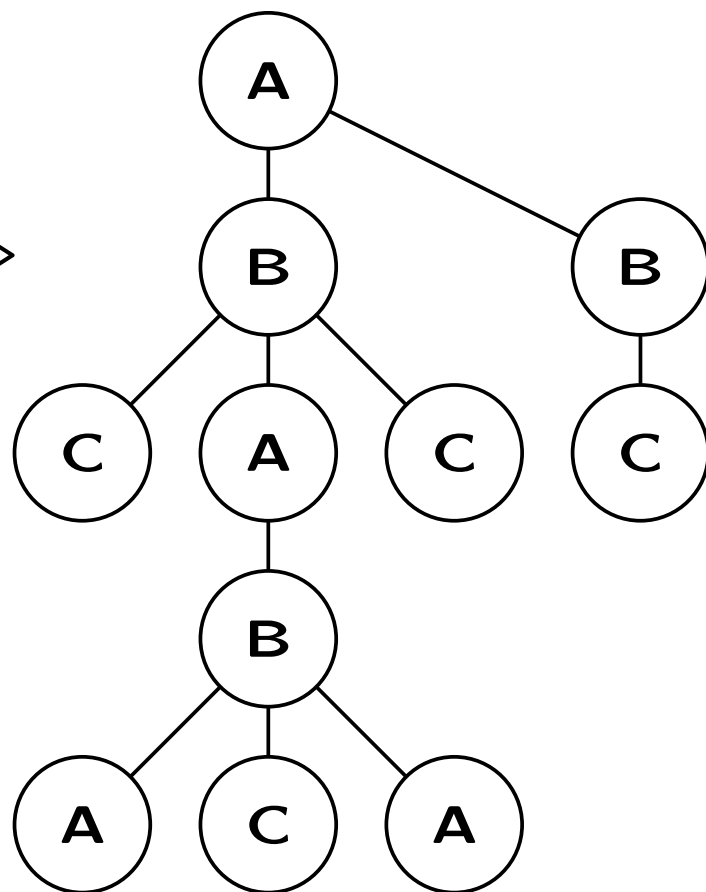
<C/>

<A>

<A/><C/><A/>

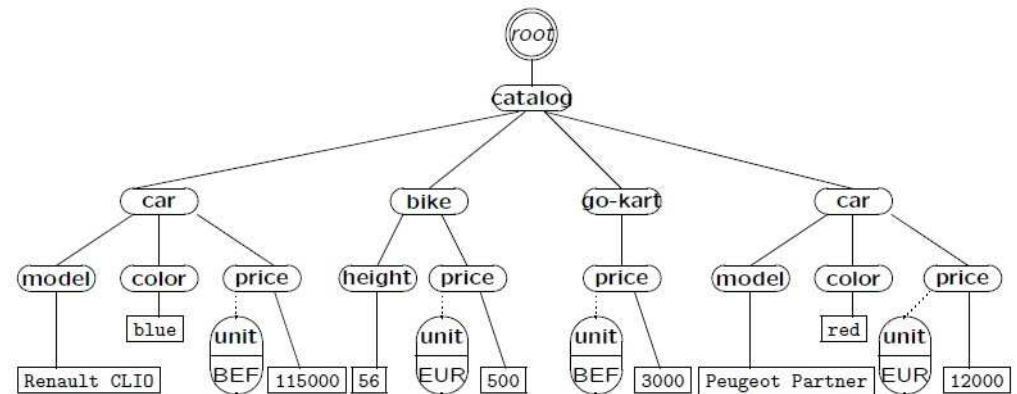
<C/>

<C/>

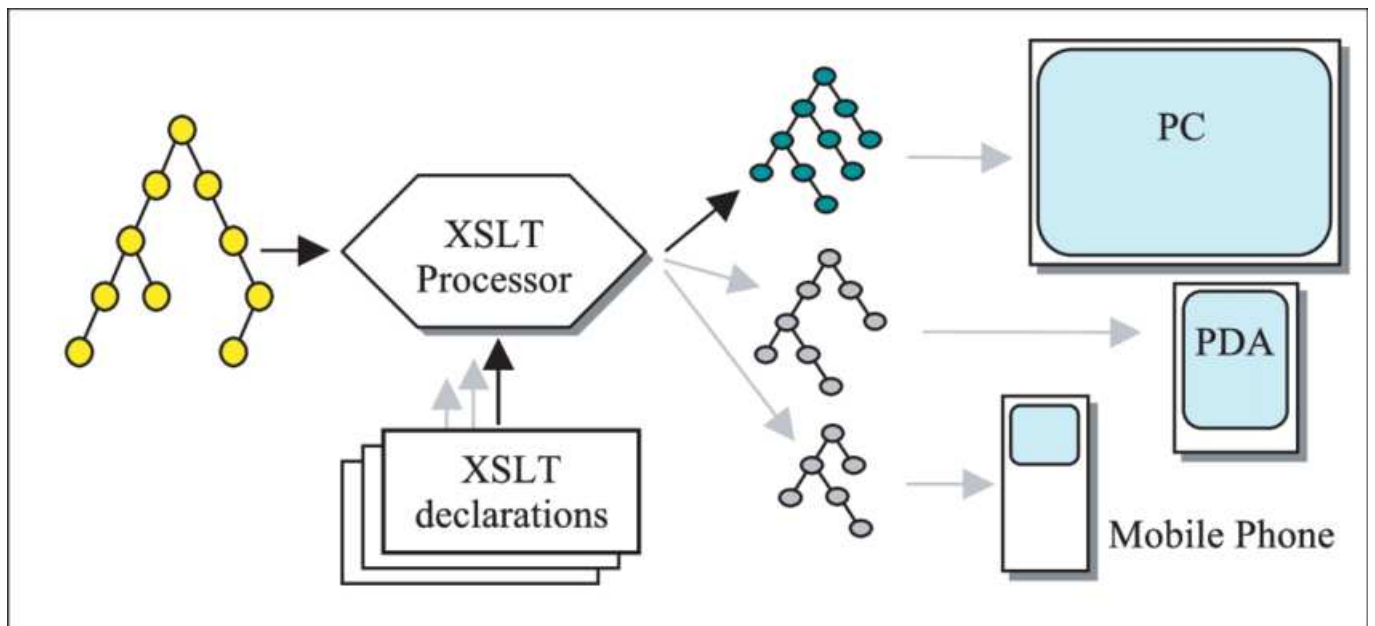


Text ↔ Tree

```
<?xml version="1.0"?>
<!DOCTYPE catalog
  SYSTEM "http://ssi.umh.ac.be/jefXML/cat.dtd">
<catalog>
  <car>
    <model>Renault CLI0</model>
    <color>blue</color>
    <price unit="BEF">115000</price>
  </car>
  <bike>
    <height>56</height>
    <price unit="EUR">500</price>
  </bike>
  <go-kart>
    <price unit="BEF">3000</price>
  </go-kart>
  <car>
    <model>Peugeot Partner</model>
    <color>red</color>
    <price unit="EUR">12000</price>
  </car>
</catalog>
```



Why XML? Separation of content and presentation



Source: Kurt Cagle: *Why XSLT and XQuery Are Coming Back.*

The Cagle Report, Published on June 19, 2020

Separation of content and presentation

```
<?xml version="1.0"?>
<!-- <!DOCTYPE catalog SYSTEM "cat.dtd" --> -->
<?xml-stylesheet type="text/xsl" href="elaborated.xsl"?>
<catalog>
  <car>
    <model>Renault CLIO</model>
    <color>blue</color>
    <price unit="BEF">115000</price>
  </car>
  <bike>
    <height>56</height>
    <price unit="EUR">500</price>
  </bike>
  <go-kart>
    <price unit="BEF">3000</price>
  </go-kart>
  <car>
    <model>Peugeot Partner</model>
    <color>red</color>
    <price unit="EUR">12000</price>
  </car>
</catalog>
```

XSLT
~

```
<HTML>
  <BODY>
    <H1>Cars</H1>
    <TABLE BORDER="3">
      <TR>
        <TH>Car Model</TH>
        <TH>Price</TH>
      </TR>
      <TR>
        <TD>Renault CLIO</TD>
        <TD>115000</TD>
      </TR>
      <TR>
        <TD>Peugeot Partner</TD>
        <TD>484079</TD>
      </TR>
    </TABLE>
    <H1>Bikes</H1>
    <TABLE BORDER="3">
      <TR>
        <TH>Frame Height</TH>
        <TH>Price</TH>
      </TR>
      <TR>
        <TD>56</TD>
        <TD>20170</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

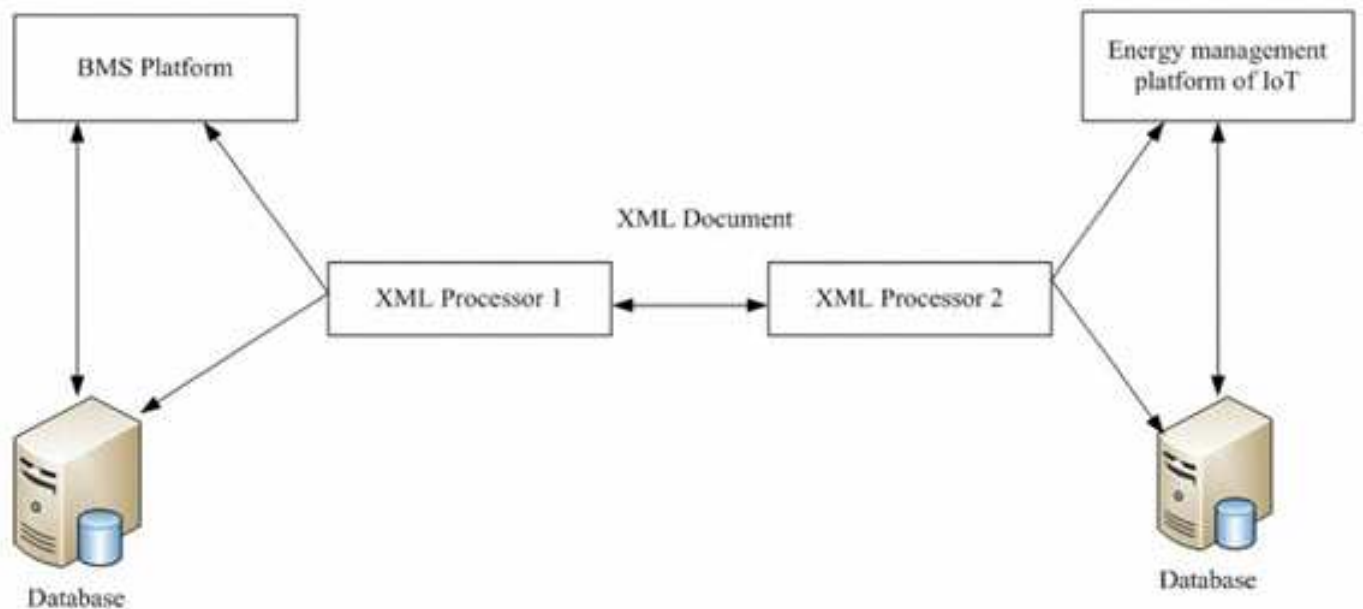
Cars

Car Model	Price
Renault CLIO	115000
Peugeot Partner	484079

Bikes

Frame Height	Price
56	20170

Why XML? Data exchange



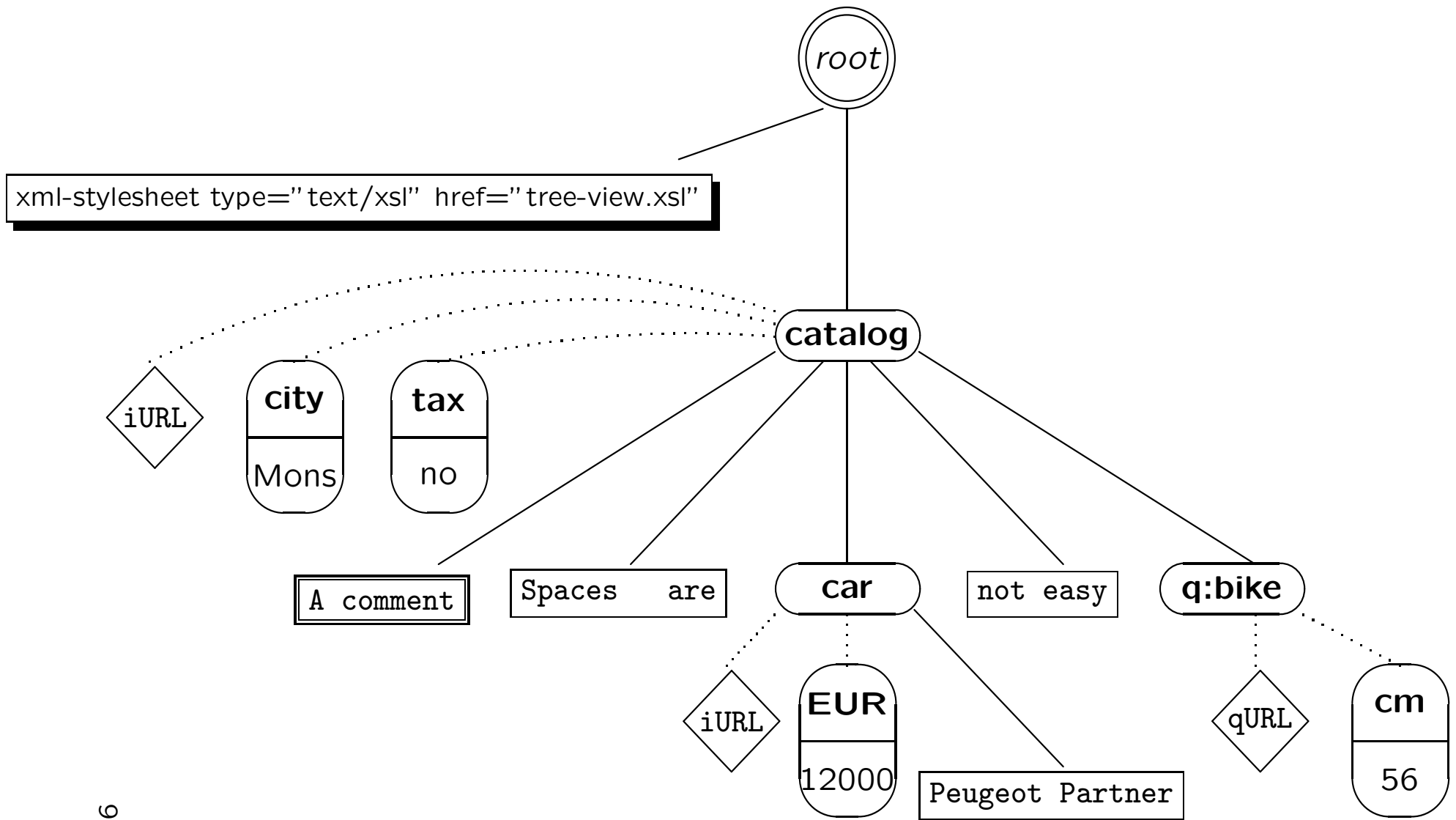
Source: Miao Yu, Guojun Yue, Jinguo Song, Xu Pang: *Research on intelligent city energy management based on Internet of things*. Clust. Comput. 22(Supplement): 8291-8300 (2019)

Well-Formed XML Document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
                  href="tree-view.xsl"?>
<catalog tax="no" city="Mons"
          xmlns="iURL" xmlns:q="qURL">
  <!--A comment-->
  Spaces   are
  <car EUR="12000">Peugeot Partner</car>
  not easy
  <q:bike cm="56"/>
</catalog>
```

XML Trees

Seven types of node: element nodes, the root node, text nodes, attribute nodes, namespace nodes, processing instruction nodes, and comment nodes.



Hint 1 The XSLT and CSS stylesheets

`tree-view.xsl`

`tree-view.css`

available at

<http://skew.org/xml/stylesheets/treeview/html/>

transform any XML document into an HTML document showing the tree representation of the XML document. Simply download both stylesheets and add to your XML document a processing instruction

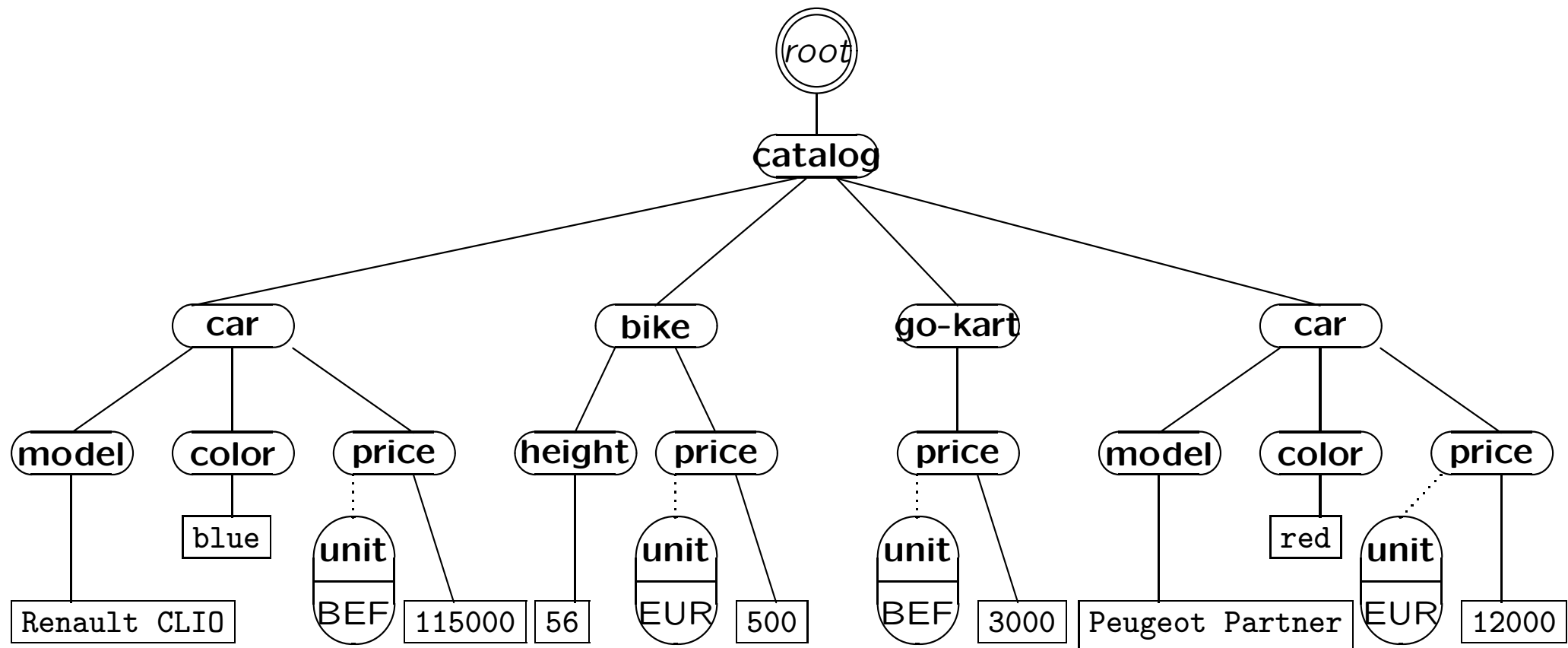
```
<?xml-stylesheet type="text/xsl"
                href="tree-view.xsl"?>
```

to ask Microsoft Internet Explorer to do the transformation.

Valid XML Document

```
<!ELEMENT catalog (car|bike|go-kart)*>
<!ELEMENT car (model,color?,price)>
<!ELEMENT bike (height,price)>
<!ELEMENT go-kart (price)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ATTLIST price unit (EUR|BEF) #REQUIRED>
```

```
<?xml version="1.0"?>
<!DOCTYPE catalog
  SYSTEM "http://ssi.umh.ac.be/jefXML/cat.dtd">
<catalog>
  <car>
    <model>Renault CLI0</model>
    <color>blue</color>
    <price unit="BEF">115000</price>
  </car>
  <bike>
    <height>56</height>
    <price unit="EUR">500</price>
  </bike>
  <go-kart>
    <price unit="BEF">3000</price>
  </go-kart>
  <car>
    <model>Peugeot Partner</model>
    <color>red</color>
    <price unit="EUR">12000</price>
  </car>
</catalog>
```



Location Step

A *location step* selects nodes relative to the *context node*; it is of the form:

*axis::node-test[predicate]**

For example,

```
child::price[attribute::unit="EUR"]
```

selects the `price` element children of the context node that have a `unit` attribute with value `EUR`.

Location Path

A sequence of one or more *location steps* separated by */*, and optionally preceded by */*.

For example,

```
child::* / child::color
```

selects all `color` grandchildren of the context node.

Axes

The thirteen axes are:

- self
- parent and child
- attribute
- ancestor and descendant
- ancestor-or-self and descendant-or-self
- preceding and following
- preceding-sibling and following-sibling
- namespace

Node Tests

```
node() mylabel *  
text() comment()  
processing-instruction()
```

Attribute Nodes and Namespace Nodes Precede Their Element Node

Because the standard says:

[...] if the context node is an attribute node or namespace node, the preceding-sibling axis is empty.

Different Versions of XPath

- XPath 1.0 (1999)
<http://www.w3.org/TR/1999/REC-xpath-19991116>
- XPath 2.0, Second Edition (2010)
- XPath 3.0 (2014)
- XPath 3.1 (2017)
<https://www.w3.org/TR/xpath-3/>

<A>

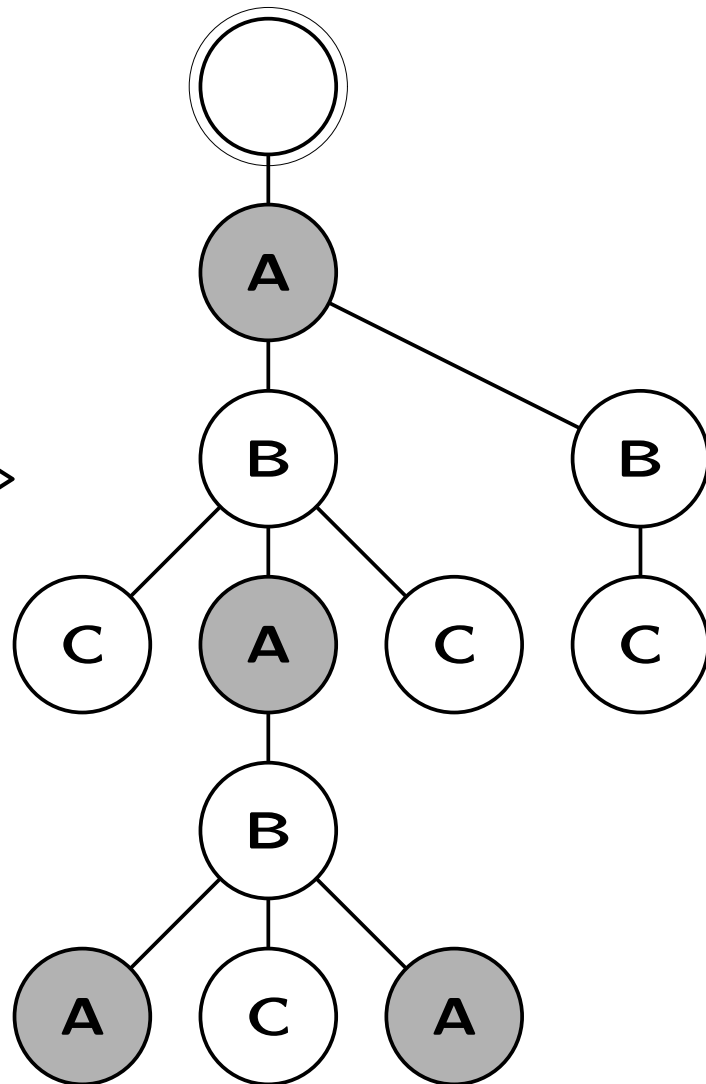
<C/>

<A>

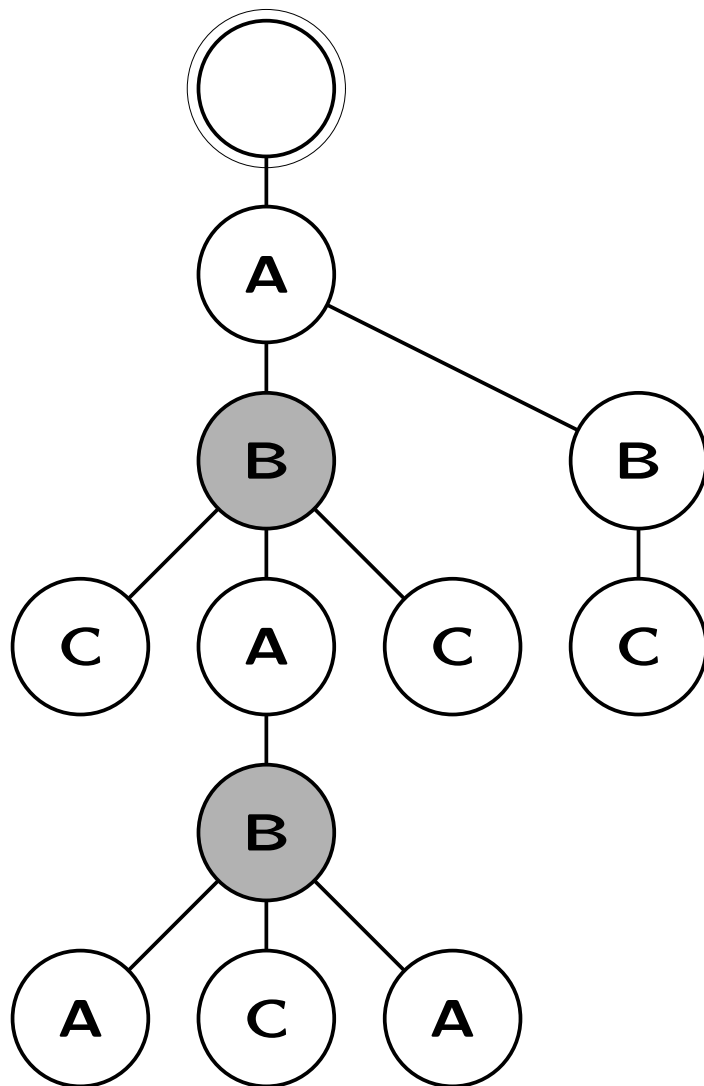
<A/><C/><A/>

<C/>

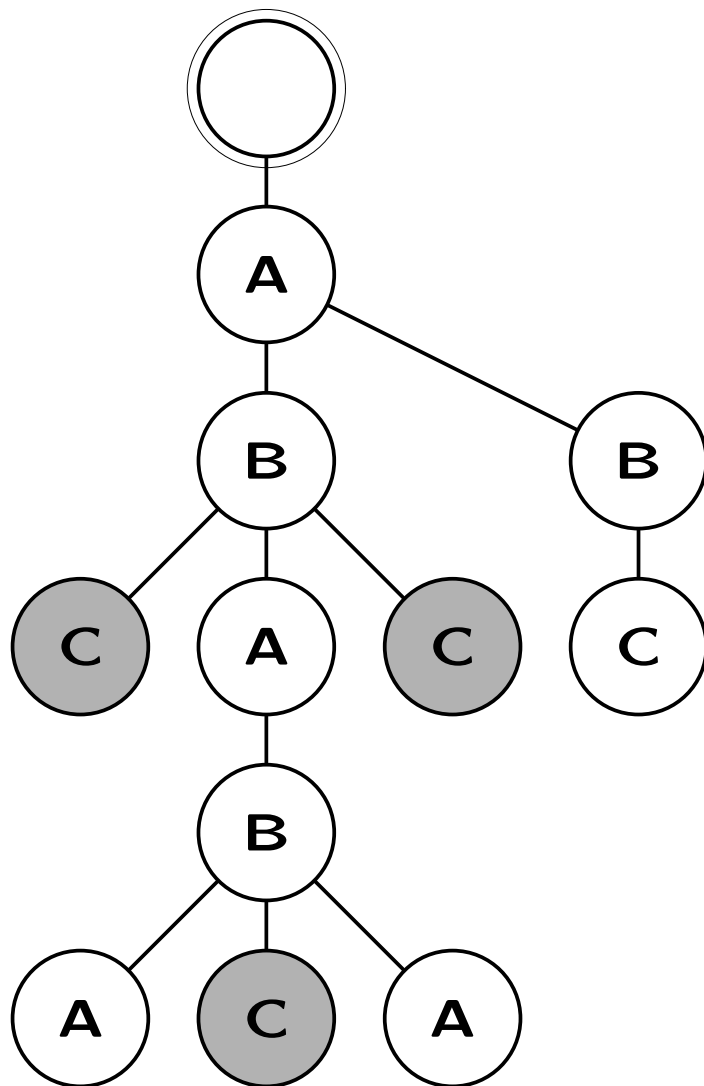
<C/>



/descendant::A



/descendant::A/parent::B



/descendant::A/parent::B/child::C

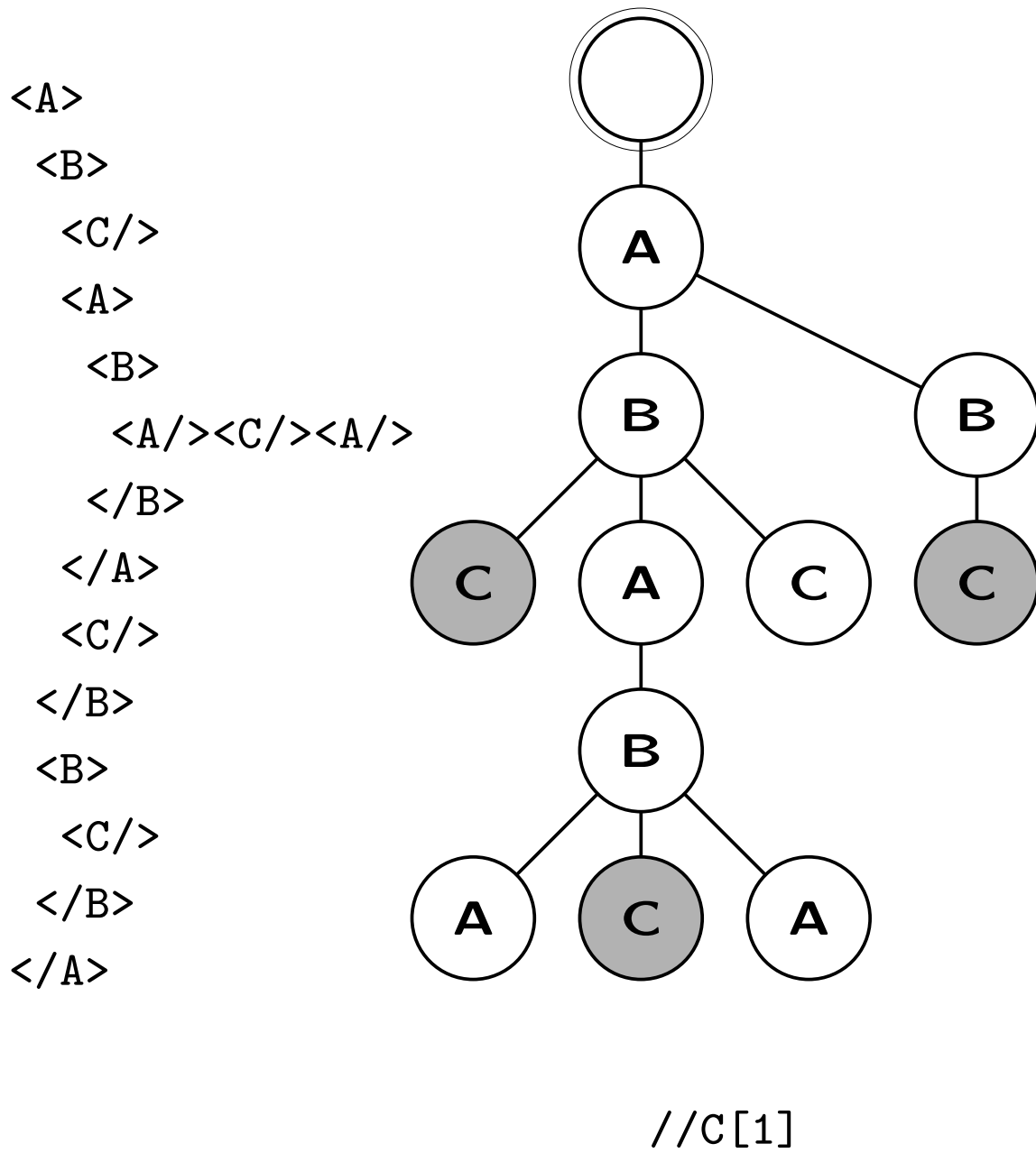
Predicates

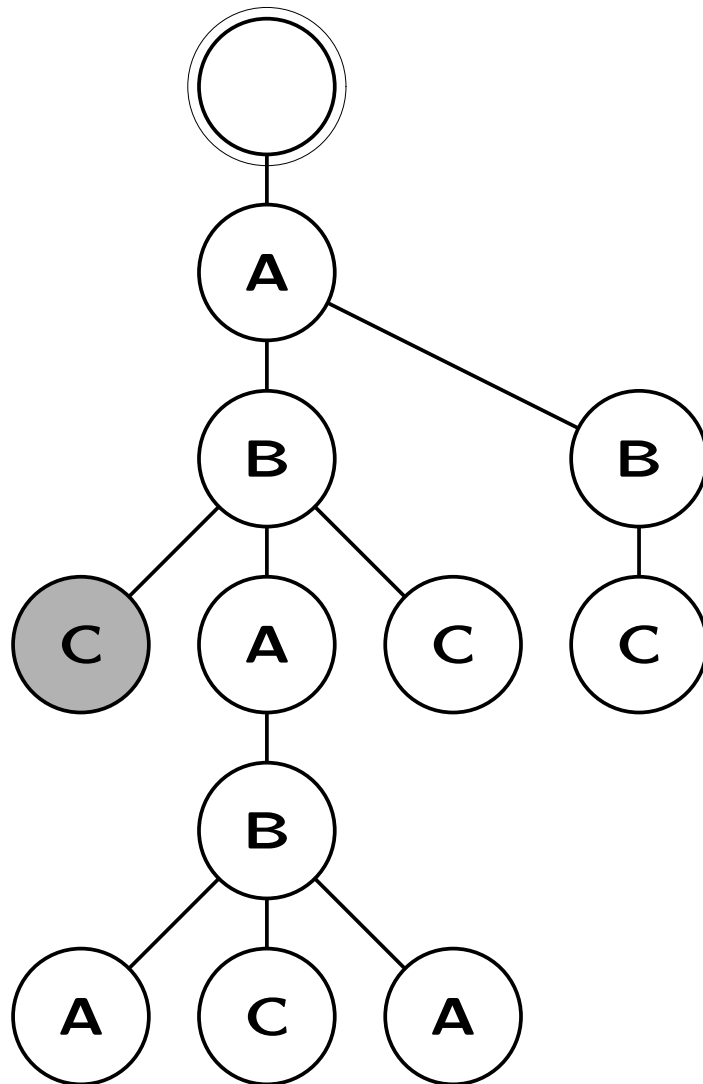
- `child::car[descendant::color]`
- `car[color='blue']`
- `car[count(color)>1]`
- `car[not(starts-with(model,'R'))]`
- `descendant::car[position()=2]` can be abbreviated as `descendant::car[2]`
- `descendant::car[position()=last()-1]`
- `descendant::car[(price*40.3399>500000 and
price/@unit='EUR')
or (price>500000 and
price/@unit='BEF')]`
- `descendant::car[(price[@unit='BEF']>500000)
or (40.3399*price[@unit='EUR']>500000)]`
- `catalog[sum(*price)>999999]`

Abbreviations

Shorthand	Unabbreviated syntax
can be omitted	<code>child::</code>
<code>//</code>	<code>/descendant-or-self::node()/</code>
<code>@</code>	<code>attribute::</code>
<code>.</code>	<code>self::node()</code>
<code>..</code>	<code>parent::node()</code>

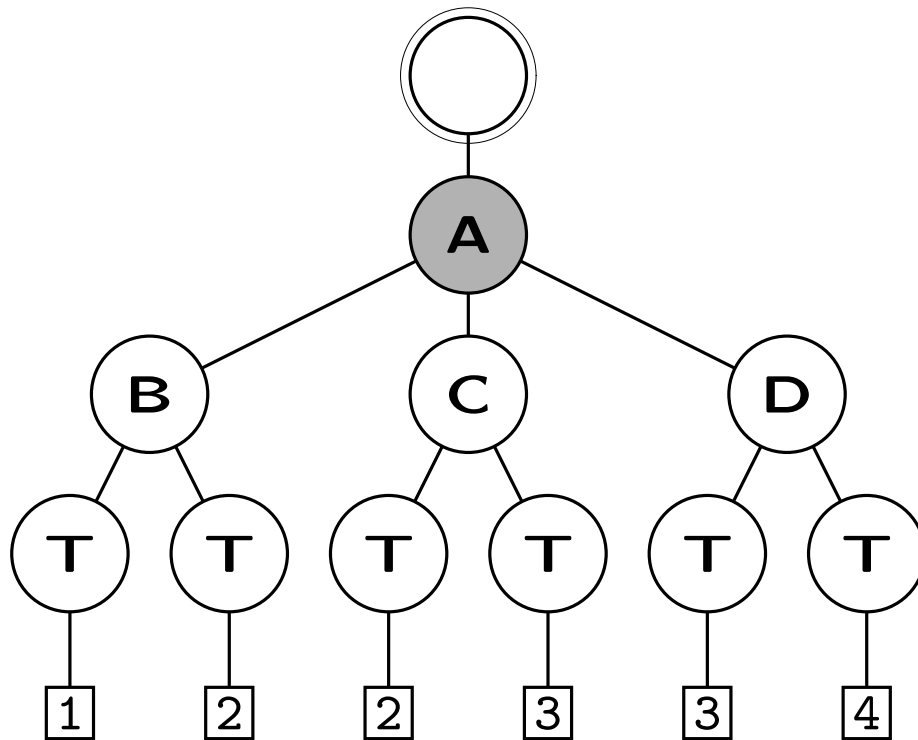
Pitfall



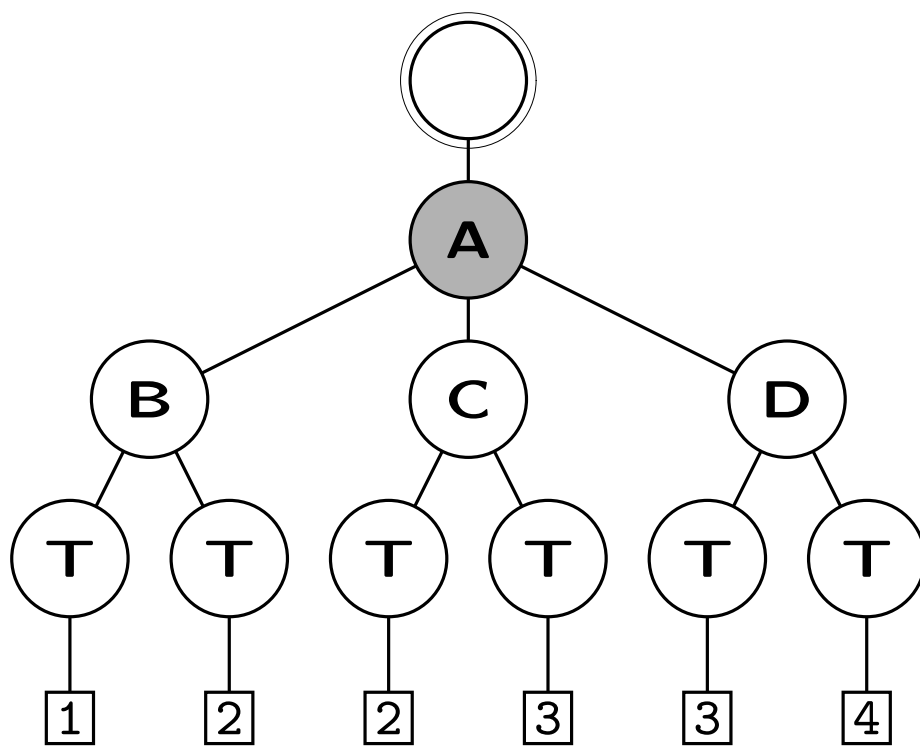


(//C) [1]

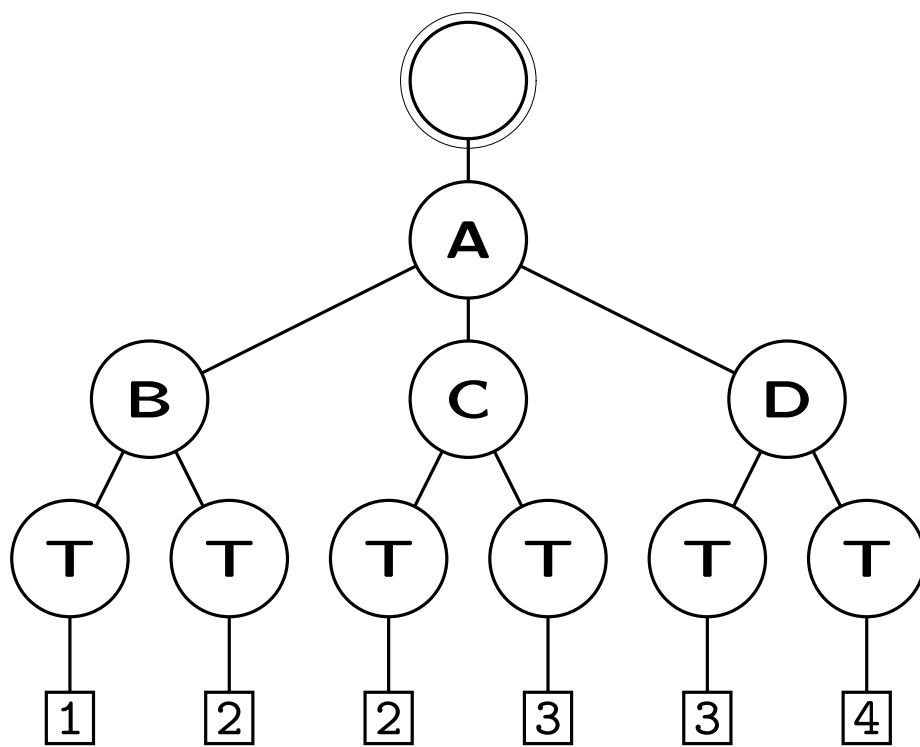
Equality



$/A[B/T=C/T]$



/A[C/T=D/T]



/A[B/T=D/T]

Exercice films.xml

```
<?xml version="1.0"?>
<!DOCTYPE filmotheque SYSTEM "films.dtd">
<filmotheque>
  <ACTEURS>
    <acteur id="IC" genre="M" naissance="1949" mort="1990">Ian Charleson
      </acteur>
    <acteur id="CC" genre="F" naissance="1949">Cheryl Campbell</acteur>
    <acteur id="NH" genre="M" naissance="1949">Nigel Havers</acteur>
    <acteur id="BM" genre="M" naissance="1950">Bill Murray</acteur>
    <acteur id="MR" genre="F" naissance="1959">Miranda Richardson</acteur>
    <acteur id="JM" genre="F" naissance="1960">Julianne Moore</acteur>
  </ACTEURS>
  <FILMS>
    <film annee="1981">
      <titre >Chariots of Fire</titre>
      <directeur naissance="1936">Hugh Hudson</directeur>
      <cast>
        <acteur id="IC"/><acteur id="CC"/><acteur id="NH"/>
      </cast>
    </film>
    <film annee="1980">
      <titre>McVicar</titre>
      <directeur naissance="1936">Tom Clegg</directeur>
      <cast>
        <acteur id="CC"/><acteur id="BM"/>
      </cast>
    </film>
    <film annee="1987">
      <titre >Empire of the Sun</titre>
      <directeur naissance="1946">Steven Spielberg</directeur>
      <cast>
        <acteur id="MR"/><acteur id="NH"/>
      </cast>
    </film>
  </FILMS>
</filmotheque>
```

```
<!-- This file is called films.dtd -->
<!ELEMENT filmotheque (ACTEURS, FILMS)>
<!ELEMENT ACTEURS (acteur)*>
<!ELEMENT FILMS (film)*>
<!ELEMENT film (titre, directeur, cast)>
    <!ATTLIST film annee CDATA #REQUIRED>
<!ELEMENT cast (acteur)*>
<!ELEMENT acteur (#PCDATA)>
    <!ATTLIST acteur id CDATA #REQUIRED>
    <!ATTLIST acteur naissance CDATA #IMPLIED>
    <!ATTLIST acteur mort CDATA #IMPLIED>
    <!ATTLIST acteur gendre CDATA #IMPLIED>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT directeur (#PCDATA)>
    <!ATTLIST directeur naissance CDATA #REQUIRED>
    <!ATTLIST directeur mort CDATA #IMPLIED>
```

Question 1 Écrivez une expression **XPath** qui rend les titres des films dont le réalisateur est né en 1936. Il y en a deux :

```
<titre>Chariots of Fire</titre>
```

```
<titre>McVicar</titre>
```

Question 2 Écrivez une expression **XPath** qui rend les titres des films dans lesquels a joué Nigel Havers. L'expression doit rester valide si on change l'identifiant de Nigel Havers (par exemple, si on remplaçait NH par NiHa). Il y en a deux :

```
<titre>Chariots of Fire</titre>
```

```
<titre>Empire of the Sun</titre>
```

Question 3 Traduisez l'expression XPath suivante en français simple.

```
//film[cast/acteur/@id=/filmothèque/ACTEURS/acteur[@mort]/@id]/titre
```

Question 4 Écrivez un programme **XSLT** qui rend tous les noms d'acteur et, pour chaque acteur, les titres de tous les films dans lesquels il a joué. L'output est formaté comme un document XML, comme suit :

```
<ACTEURS>
<acteur>
  <nom>Ian Charleson</nom>
  <FILMS>
    <film>Chariots of Fire</film>
  </FILMS>
</acteur>
<acteur>
  <nom>Cheryl Campbell</nom>
  <FILMS>
    <film>Chariots of Fire</film>
    <film>McVicar</film>
  </FILMS>
</acteur>
<acteur>
  <nom>Nigel Havers</nom>
  <FILMS>
    <film>Chariots of Fire</film>
    <film>Empire of the Sun</film>
  </FILMS>
</acteur>

:

<acteur>
  <nom>Julianne Moore</nom>
  <FILMS></FILMS>
</acteur>
</ACTEURS>
```

Exercice ListOfLists

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

<!DOCTYPE ListOfLists [
  <!ELEMENT ListOfLists (list)*>
  <!ELEMENT list (e)*>
  <!ELEMENT e (#PCDATA)>
  <!-- ATTLIST e a CDATA #REQUIRED -->
]>

<ListOfLists>
  <list><e a="1"/><e a="2"/><e a="3"/></list>
  <list><e a="1"/><e a="2"/><e a="4"/></list>
  <list><e a="22"/><e a="11"/><e a="44"/></list>
  <list><e a="1"/><e a="1"/><e a="1"/><e a="7"/></list>
</ListOfLists>
```

Ce document XML encode:

$((1, 2, 3), (1, 2, 4), (22, 11, 44), (1, 1, 1, 7))$

Écrivez des expressions XPath pour les problèmes suivants:

- Renvoyer les listes qui ont le plus d'éléments.

~>

```
<list><e a="1"/><e a="1"/><e a="1"/><e a="7"/></list>
```

- en utilisant count et max;
- en utilisant count mais sans utiliser max.

- Renvoyer les listes qui ont le plus d'éléments *distincts*, sans utiliser distinct-values.

~>

```
<list><e a="1"/><e a="2"/><e a="3"/></list>  
<list><e a="1"/><e a="2"/><e a="4"/></list>  
<list><e a="22"/><e a="11"/><e a="44"/></list>
```

- Renvoyer les listes qui n'ont aucun élément en commun avec une autre liste.

~>

```
<list><e a="22"/><e a="11"/><e a="44"/></list>
```

- Renvoyer les listes qui ne sont pas triées par ordre croissant.

~>

```
<list><e a="22"/><e a="11"/><e a="44"/></list>
```

Exercice Catalog

- Expliquez la différence entre les trois requêtes suivantes:

```
/catalog/car[.="blue"]
```

```
/catalog/car[text()="blue"]
```

```
/catalog/car[descendant::text()="blue"]
```

- Quel est le résultat des requêtes suivantes sur le catalog?

```
/*[descendant::color != descendant::color]
```

et

```
/*[descendant::bike != descendant::bike]
```

XSL Stylesheet

A *template rule* has the form:

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

An XSL *stylesheet* is a family of such template rules.

XSLT Processing Model

XML document + XSL stylesheet $\xrightarrow{\text{xslt}}$ result

The xslt program can be understood as follows:

```
program xslt
  procedure process(aNode)
  begin
    find the template rule with pattern
      that best matches aNode
    execute the template
      with aNode as the current node
  end
begin
  process(theRootNode)
end
```

Matching

The *match pattern* is a restricted location path.

A pattern p is defined to *match* a node n if and only if there is a possible context node c such that when the pattern p is evaluated relative to c , the node n is a member of the resulting node-set.

	match=				
	"node()"	"*"	"@*"	"/"	"text()"
root	—	—	—	+	—
element	+	+	—	—	—
text	+	—	—	—	+
comment	+	—	—	—	—
processing -instruction	+	—	—	—	—
attribute	—	—	+	—	—
namespace	—	—	—	—	—

Executing the Template

Typically the template contains one or more of the following elements:

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

for each node n in *node-set*, `process( $n$ )` ,

- `<xsl:copy-of select="node-set" />`

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

Create a text node in the result tree. The text node is obtained by converting *node-set* into a string.

Built-In Template Rules

```
<xsl:template match="/|*">  
  <xsl:apply-templates/>  
</xsl:template>
```

```
<xsl:template match="text()|@"*>  
  <xsl:value-of select="."/>  
</xsl:template>
```

```
<xsl:template  
  match="processing-instruction()|comment()"/>
```

Illustration of Built-In Template Rules

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

~→

Renault CLI0blue115000565003000Peugeot Partnerred12000

Priority Processing

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <xsl:apply-templates select="//car"/>
    </HTML>
  </xsl:template>
  <xsl:template match="car[price>99999]"
                priority="2">
    <FONT COLOR="red">
      <xsl:value-of select="model"/>
    </FONT>
  </xsl:template>
  <xsl:template match="car[price>10000]"
                priority="1">
    <FONT COLOR="green">
      <xsl:value-of select="model"/>
    </FONT>
  </xsl:template>
  <xsl:template match="car"/>
</xsl:stylesheet>
```

Multiple Processing

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <xsl:apply-templates select="//car"
                             mode="theblues"/>
      <xsl:apply-templates select="//car"
                             mode="blackout"/>
    </HTML>
  </xsl:template>
  <xsl:template match="car" mode="theblues">
    <FONT COLOR="blue">
      <xsl:value-of select="."/>
    </FONT>
  </xsl:template>
  <xsl:template match="car" mode="blackout">
    <FONT COLOR="black">
      <xsl:value-of select="."/>
    </FONT>
  </xsl:template>
</xsl:stylesheet>
```

Context and Current Node

Context node: `self::node()`

Current node: `current()`, new in XSLT

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

  <xsl:template match="/">
    <xsl:apply-templates
      select="//price/@unit"/>
  </xsl:template>

  <xsl:template match="@unit">
    <BR/>
    <xsl:value-of select=
      "count(/catalog/*[price/@unit=current()])"/>
    vehicle prices are expressed in
    <xsl:value-of select="current()"/>
  </xsl:template>

</xsl:stylesheet>
```



2 vehicle prices are expressed in BEF
2 vehicle prices are expressed in EUR
2 vehicle prices are expressed in BEF
2 vehicle prices are expressed in EUR

Eliminating Duplicates

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

  <xsl:template match="/">
    <xsl:apply-templates
      select=
"//price/@unit[not(.=preceding::*/@unit)]"/>
    </xsl:template>

    ...

</xsl:stylesheet>
```

Variables

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

<xsl:template match="catalog">
  <xsl:variable name="eurtobef" select=
    "round(40.3399*sum(../price[@unit='EUR']))"/>

  <xsl:variable name="bef" select=
    "sum(../price[@unit='BEF'])"/>

  <xsl:value-of select="$eurtobef+$bef"/>
</xsl:template>

</xsl:stylesheet>
```

Creating Element and Attribute Nodes

The desired result:

```
<LIST-OF-CARS>  
  <Renault-CLIO prix="115000"/>  
  <Peugeot-Partner prix="12000"/>  
</LIST-OF-CARS>
```

The stylesheet:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:element name="LIST-OF-CARS">
    <xsl:apply-templates select="//car"/>
  </xsl:element>
</xsl:template>

<xsl:template match="car">
  <xsl:variable
    name="var"
    select="translate(./model,' ','-')"/>
  <xsl:element name="$var">
    <xsl:attribute name="prix">
      <xsl:value-of select="price"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Elaborated Example

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <HTML>
    <BODY>
    <H1>Cars</H1>
    <TABLE BORDER="3">
    <TR><TH>Car Model</TH>
        <TH>Price</TH></TR>
    <xsl:apply-templates select="//car"/>
    </TABLE>
    <H1>Bikes</H1>
    <TABLE BORDER="3">
    <TR><TH>Frame Height</TH>
        <TH>Price</TH></TR>
    <xsl:apply-templates select="//bike"/>
    </TABLE>
    </BODY>
    </HTML>
</xsl:template>
```

```

<xsl:template match="car">
    <TR><TD>
        <xsl:value-of select="model"/>
    </TD><TD>
        <xsl:apply-templates select="price"/>
    </TD></TR>
</xsl:template>

<xsl:template match="bike">
    <TR><TD>
        <xsl:value-of select="height"/>
    </TD><TD>
        <xsl:apply-templates select="price"/>
    </TD></TR>
</xsl:template>

<xsl:template match="price[@unit='BEF']">
    <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="price[@unit='EUR']">
    <xsl:value-of select="round(40.3399*.)"/>
</xsl:template>
</xsl:stylesheet>

```



```
<HTML>
  <BODY>
    <H1>Cars</H1>
    <TABLE BORDER="3">
      <TR>
        <TH>Car Model</TH>
        <TH>Price</TH>
      </TR>
      <TR>
        <TD>Renault CLIO</TD>
        <TD>115000</TD>
      </TR>
      <TR>
        <TD>Peugeot Partner</TD>
        <TD>484079</TD>
      </TR>
    </TABLE>
    <H1>Bikes</H1>
    <TABLE BORDER="3">
      <TR>
        <TH>Frame Height</TH>
        <TH>Price</TH>
      </TR>
      <TR>
        <TD>56</TD>
        <TD>20170</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```