CSI 3540

Structures, techniques et normes du Web

Représentation des données du Web en XML

Objectifs :

- Maîtriser le langage XML 1.0
- Savoir utiliser les outils standard pour le traitement de XML

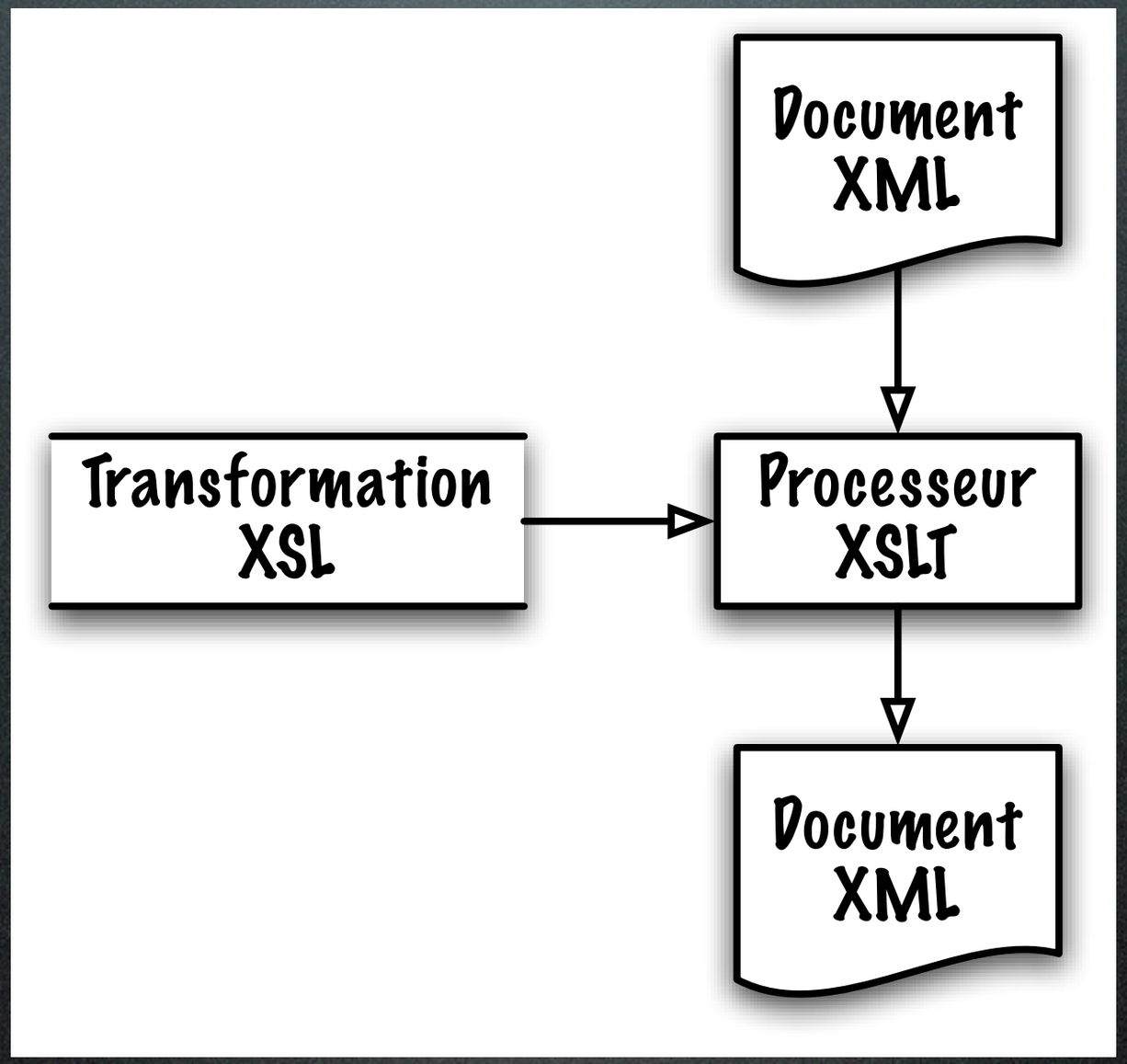
Lectures :

- Web Technologies (2007) § 7 Pages 402–418

Plan

1. Transformations
    1. XPath
    2. XSL
    3. XSLT

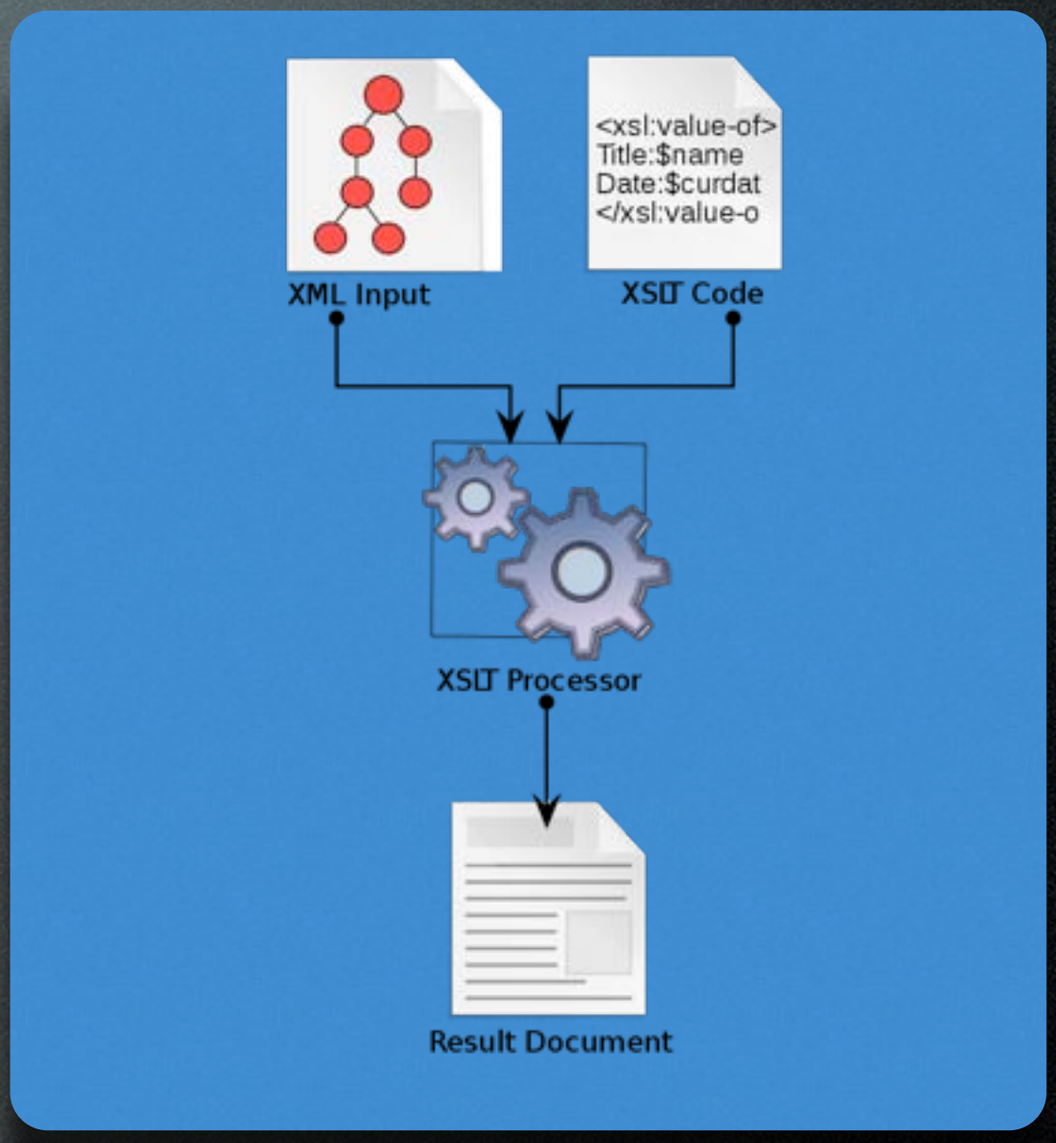
Transformations XSL (XSLT)



- **XSL** signifie « **Extensible Stylesheet Language** »
- **XSL** est un vocabulaire **XML**
- **XSL** c'est trois recommandations :
  - **XSLT** (vocabulaire **XSL**)
  - **XPath** (adressage des noeuds)
  - **XSL-FO** (« formatting objects », styles)

XSL Transformations : XML, XML, XML

- La source est une application **XML**
- Le résultat est une application **XML**
- La «transformation» est une application **XML**
- **D'où l'importance des espaces de nommage !**



- Une transformation XSL est appelée **feuille de style**
- Une feuille de style contient un ensemble de **règles modèles**
- Une règle modèle se compose d'un **filtre** (a.k.a. **motif**, **pattern**) et un **modèle**
- Le **filtre** sert à identifier des noeuds de l'arbre-source
- Le **modèle** est instantié afin de produire une partie de l'arbre-résultat

- Un **modèle** contient
  - des litéraux (éléments de l'application résultat, par exemple)
  - des instructions XSL (énoncés de contrôle)

XSLT

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

  <xsl:template match=pattern>
    contenu
  </xsl:template>

  ...
  <xsl:template match=pattern>
    contenu
  </xsl:template>

  ...
  <xsl:template name=nom>
    contenu
  </xsl:template>

</xsl:transform>
```

Règles modèle

"version" est obligatoire

Convention :

- préfixe xsl pour le vocabulaire XSLT
- espace de nommage par défault pour le vocabulaire destination

Une règle de transformation est formée d'un ou plusieurs patrons.

L'élément template possède un attribut match

```
<xsl:template match="/catalog">

  <html>
    <head>
        <title>Liste des albums</title>
    </head>
    <body>
        <ul>
          <xsl:apply-templates select="album/name"/>
        </ul>
    </body>
  </html>

</xsl:template>
```

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

XSL

- </xsl:transform> et </xsl:stylesheet> sont sémantiquement équivalent (synonymes)

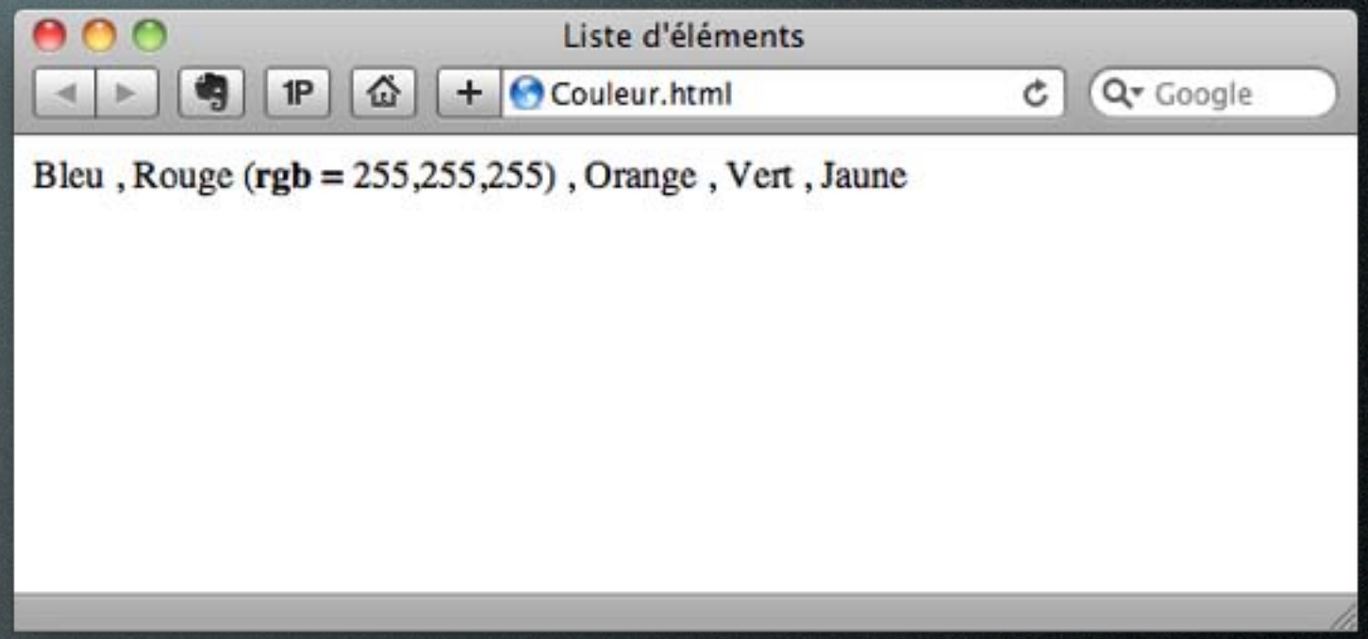
XHTML en sortie

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

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

 <xsl:output indent="yes"
        method="xhtml"
        encoding="UTF-8"
        omit-xml-declaration="yes"
        doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
        doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
        />

   ...

</xsl:transform>
```

XML en sortie

- **indent="yes"** facilite la lecture du code XHTML généré
- **method="xhtml"** force le processeur XSLT à produire la balise **<meta http-equiv="Content-Type" .../>** qui semble essentielle pour la reconnaissance de l'encodage
- un document **XHTML** n'a pas de déclaration **xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<list>
  <elem>Bleu</elem>
  <elem rgb="255,255,255">Rouge</elem>
  <elem>Orange</elem>
  <elem>Vert</elem>
  <elem>Jaune</elem>
</list>
```




```xml
<xsl:template match="/list">
 <html>
  <head>
    <title>Liste d'éléments</title>
  </head>
  <body>
    <xsl:apply-templates select="elem"/>
  </body>
 </html>
</xsl:template>

<xsl:template match="elem[position()=1]">
 <xsl:value-of select="."/>
 <xsl:apply-templates select="@rgb"/>
</xsl:template>

<xsl:template match="elem">
 , <xsl:value-of select="."/>
   <xsl:apply-templates select="@rgb"/>
</xsl:template>

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

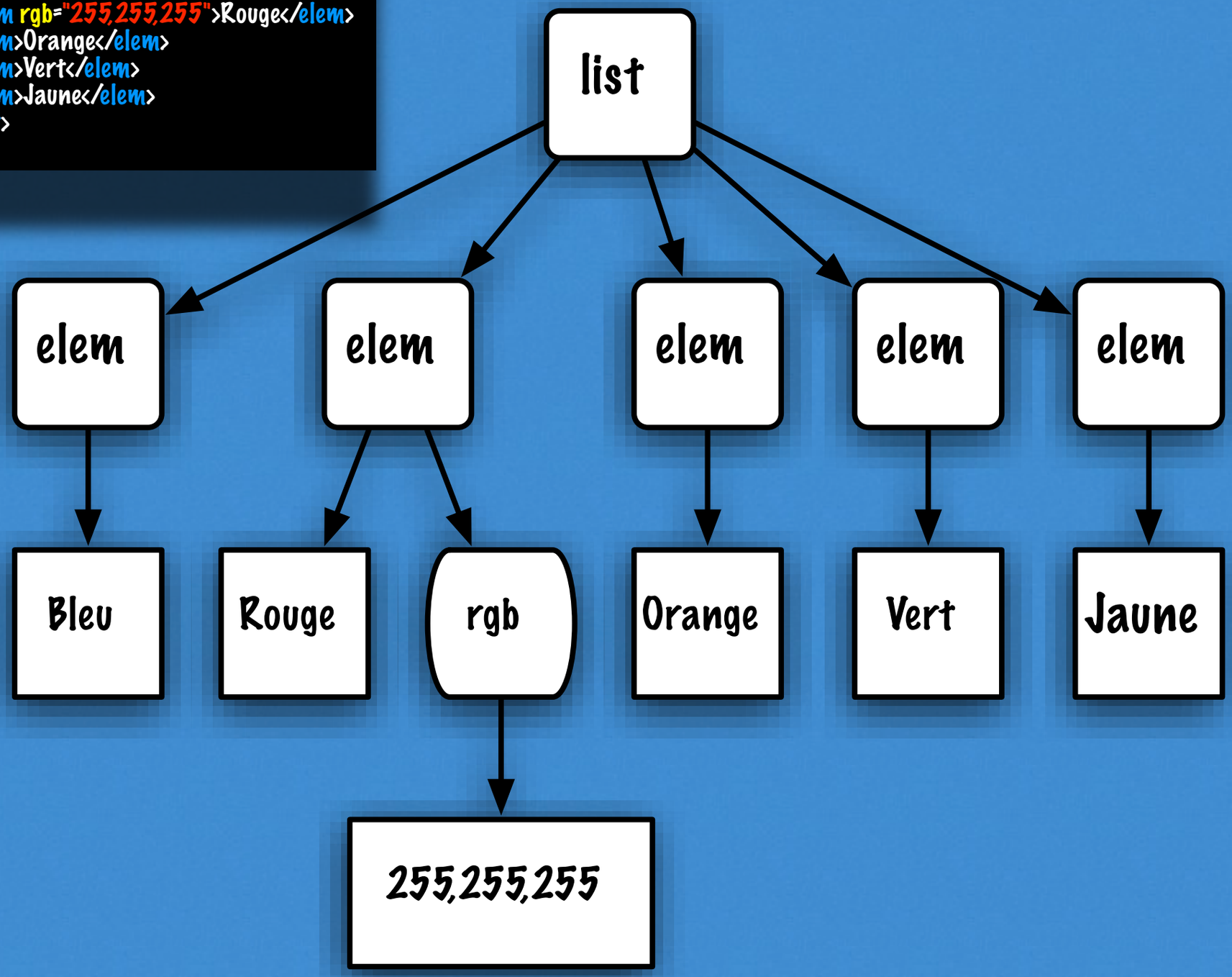
Cet exemple démontre :

- la conception de filtres ayant un prédicat

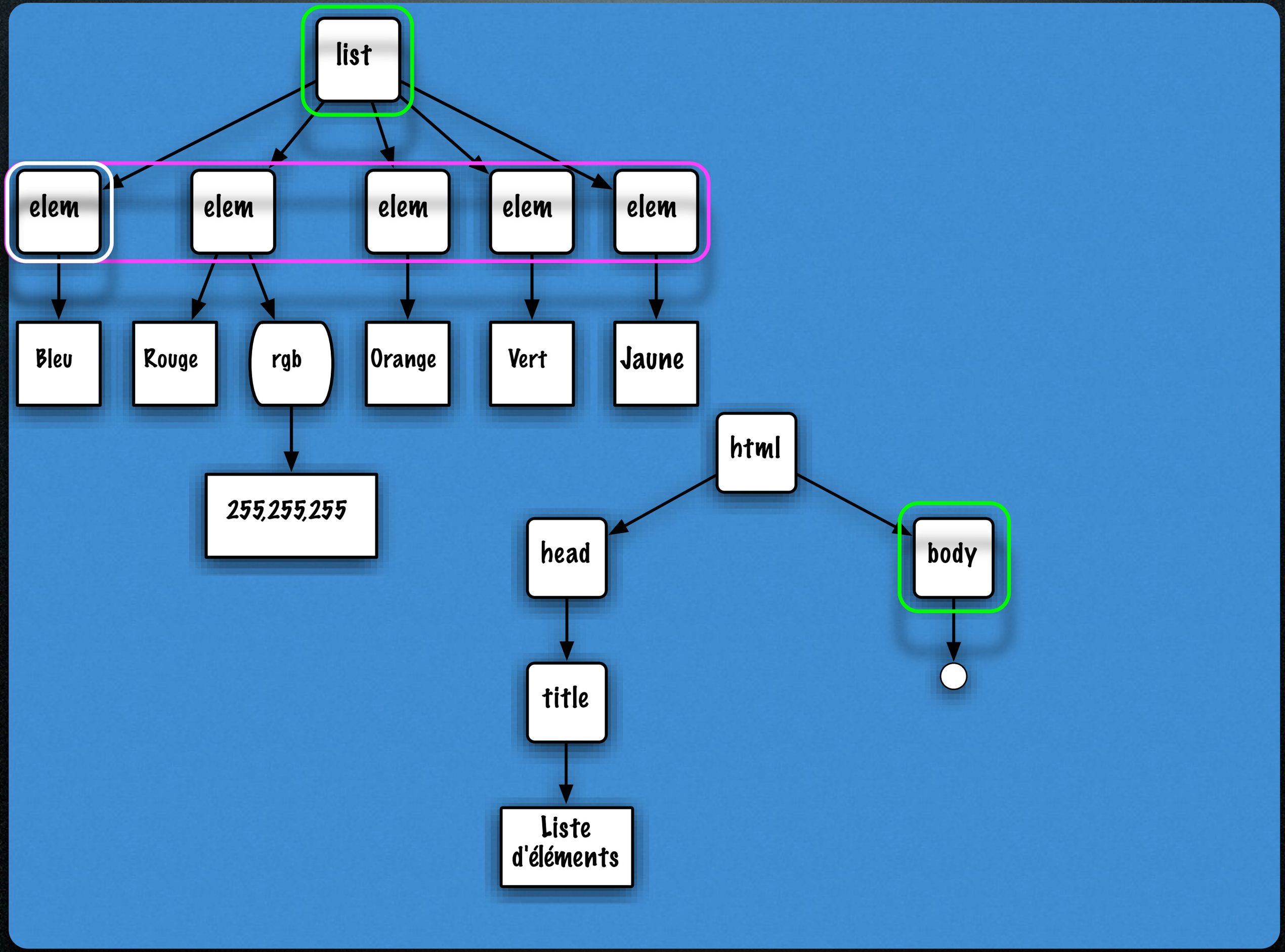
- la règle modèle la plus spécifique s'applique

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
  <elem>Bleu</elem>
  <elem rgb="255,255,255">Rouge</elem>
  <elem>Orange</elem>
  <elem>Vert</elem>
  <elem>Jaune</elem>
</list>
```

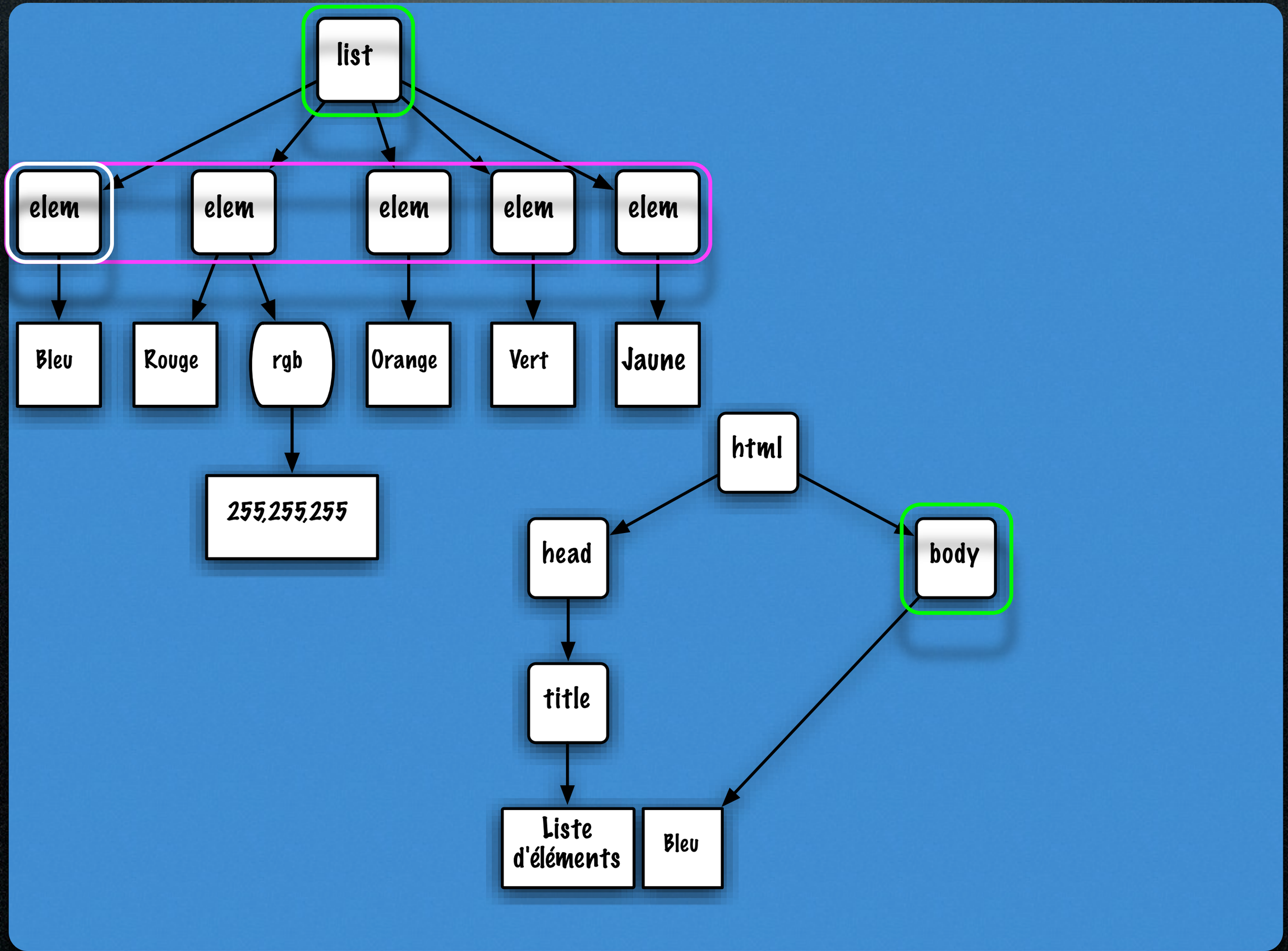
```xml
<?xml version="1.0" encoding="UTF-8"?>
<list>
  <elem>Bleu</elem>
  <elem rgb="255,255,255">Rouge</elem>
  <elem>Orange</elem>
  <elem>Vert</elem>
  <elem>Jaune</elem>
</list>
```



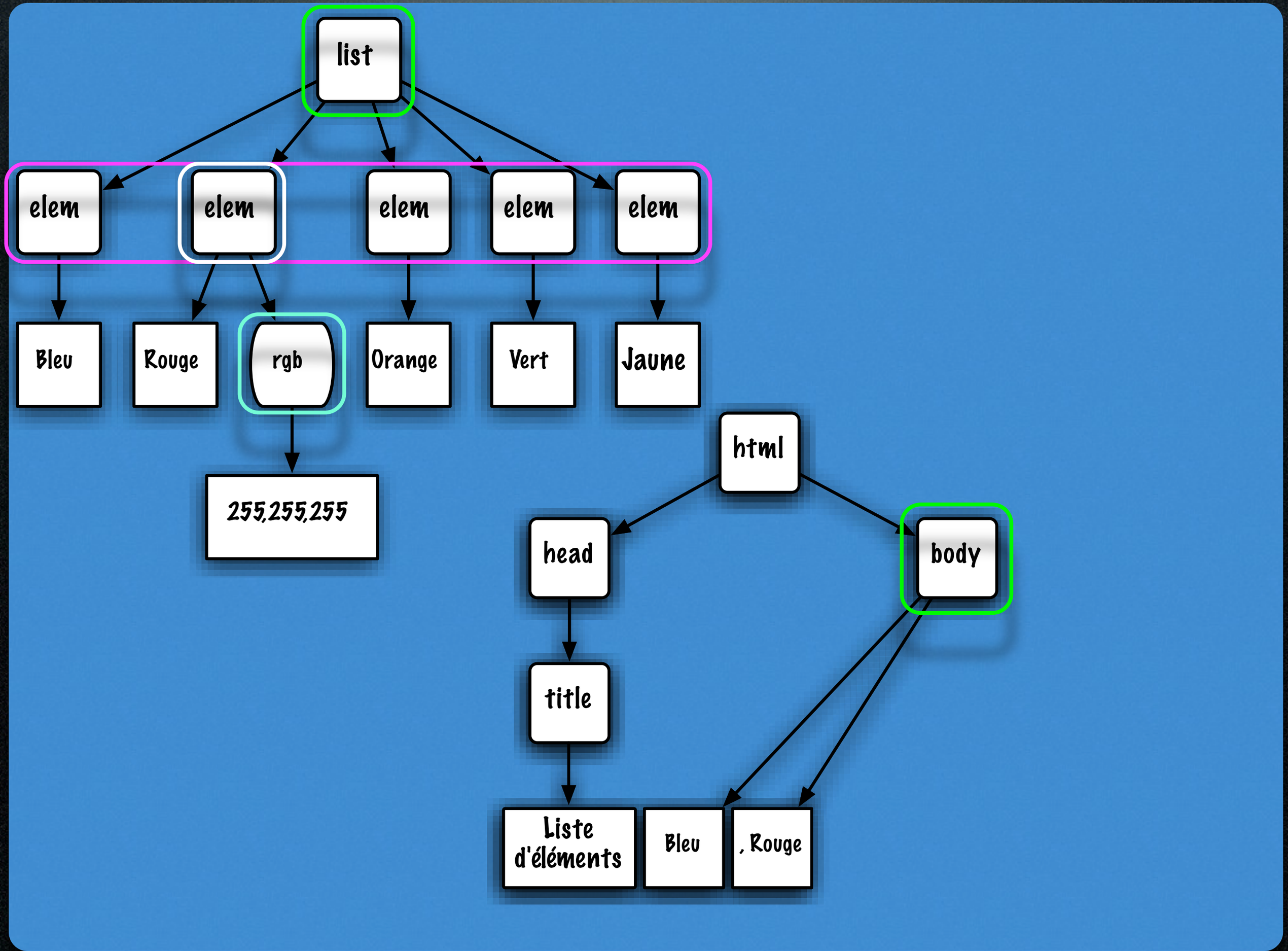
```
<xsl:template match="/list">
 <html>
  <head>
   <title>Liste d'éléments</title>
  </head>
  <body>
   <xsl:apply-templates select="elem" />
  </body>
 </html>
</xsl:template>

<xsl:template match="elem[position()=11]">
 <xsl:value-of select="." />
 <xsl:apply-templates select="@rgb" />
</xsl:template>

<xsl:template match="elem">
 , <xsl:value-of select="." />
 <xsl:apply-templates select="@rgb" />
</xsl:template>

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


list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments

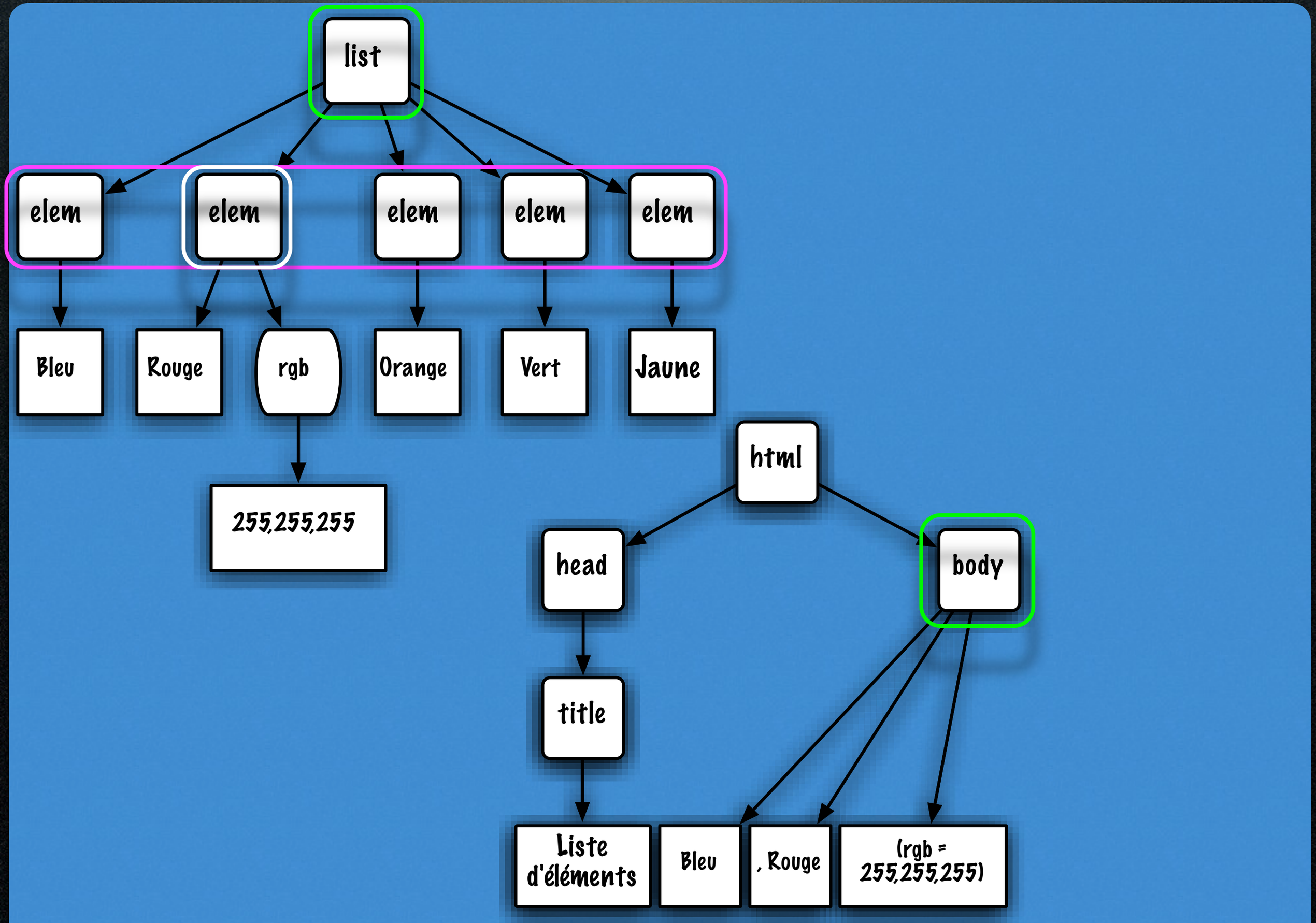
```
<xsl:template match="/list">
 <html>
  <head>
   <title>Liste d'éléments</title>
  </head>
  <body>
   <xsl:apply-templates select="elem" />
  </body>
 </html>
</xsl:template>

<xsl:template match="elem[position()=1]">
 <xsl:value-of select="." />
 <xsl:apply-templates select="@rgb" />
</xsl:template>

<xsl:template match="elem">
 , <xsl:value-of select="." />
 <xsl:apply-templates select="@rgb" />
</xsl:template>

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

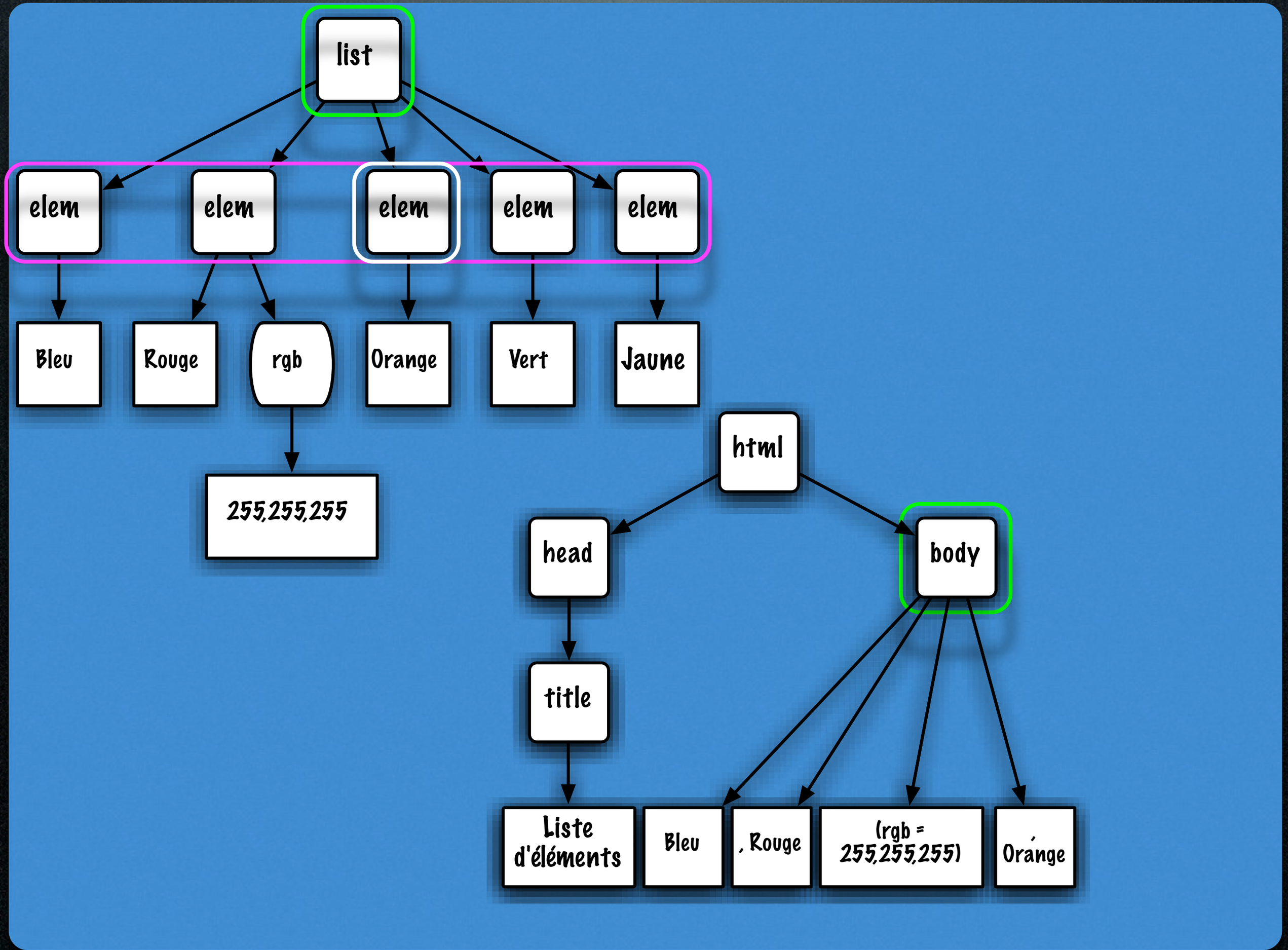

list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu

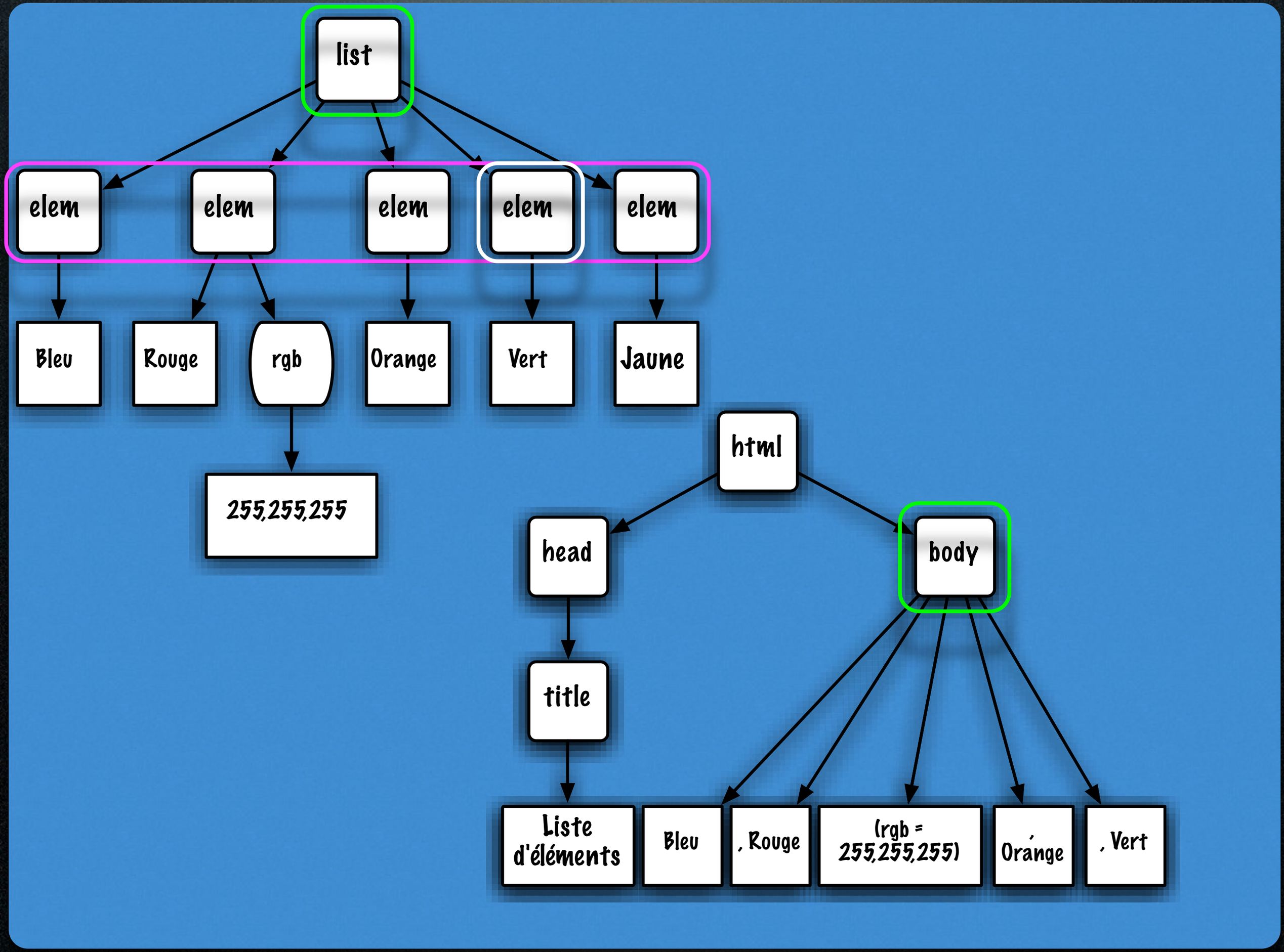
```
<xsl:template match="/list">
  <html>
    <head>
     <title>Liste d'élément</title>
    </head>
    <body>
     <xsl:apply-templates select="elem"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="elem[position()=1]">
  <xsl:value-of select="."/>
  <xsl:apply-templates select="@rgb"/>
</xsl:template>

<xsl:template match="elem">
  , <xsl:value-of select="."/>
    <xsl:apply-templates select="@rgb"/>
</xsl:template>

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

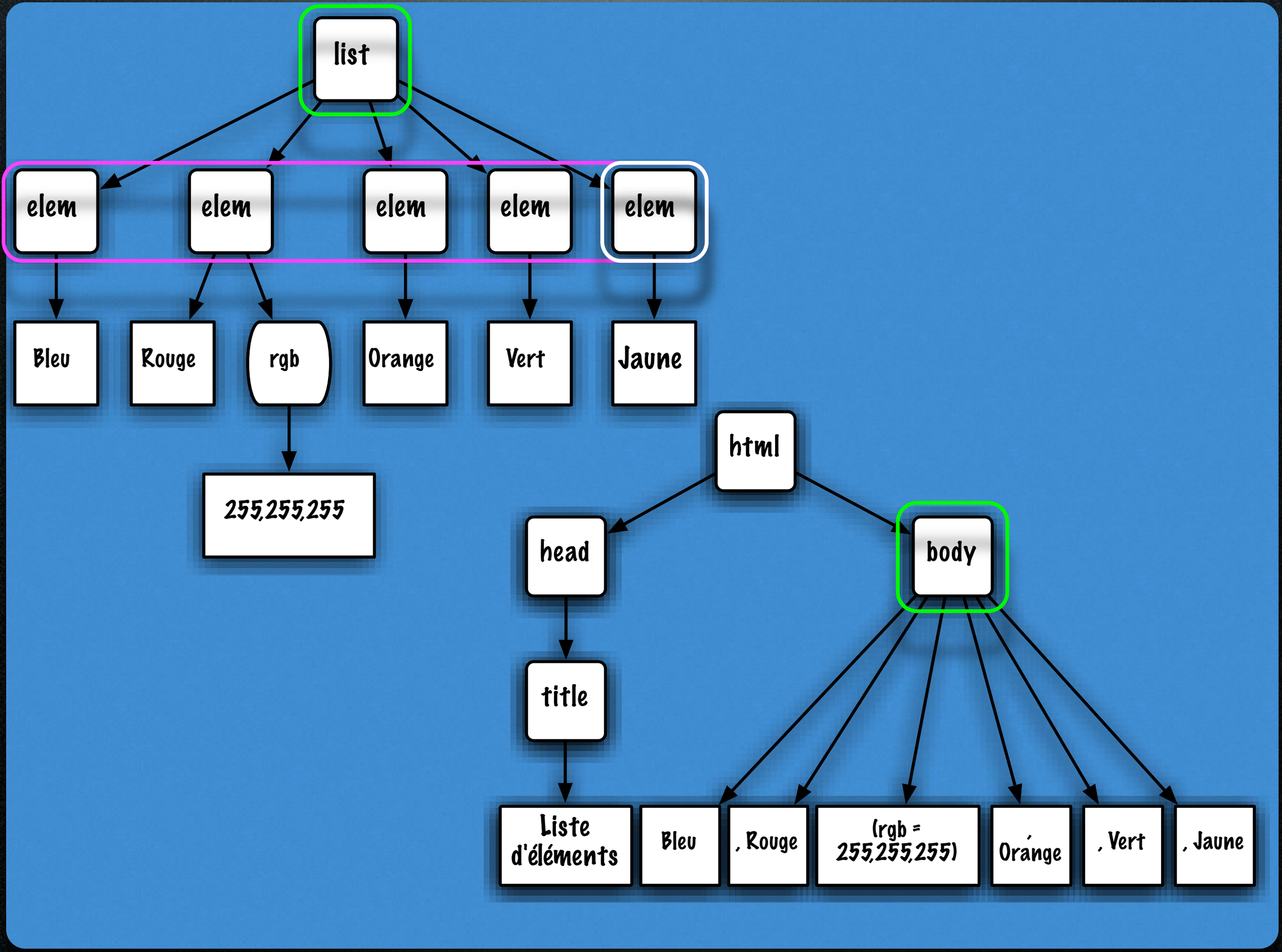

list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu
, Rouge

```
<xsl:template match="/list">
  <html>
    <head>
     <title>Liste d'élément</title>
    </head>
    <body>
     <xsl:apply-templates select="elem"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="elem[position()=1]">
  <xsl:value-of select="."/>
  <xsl:apply-templates select="@rgb"/>
</xsl:template>

<xsl:template match="elem">
  , <xsl:value-of select="."/>
   <xsl:apply-templates select="@rgb"/>
</xsl:template>

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


list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu
, Rouge
(rgb = 255,255,255)


list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu
, Rouge
(rgb = 255,255,255)
, Orange


list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu
, Rouge
(rgb = 255,255,255)
, Orange
, Vert

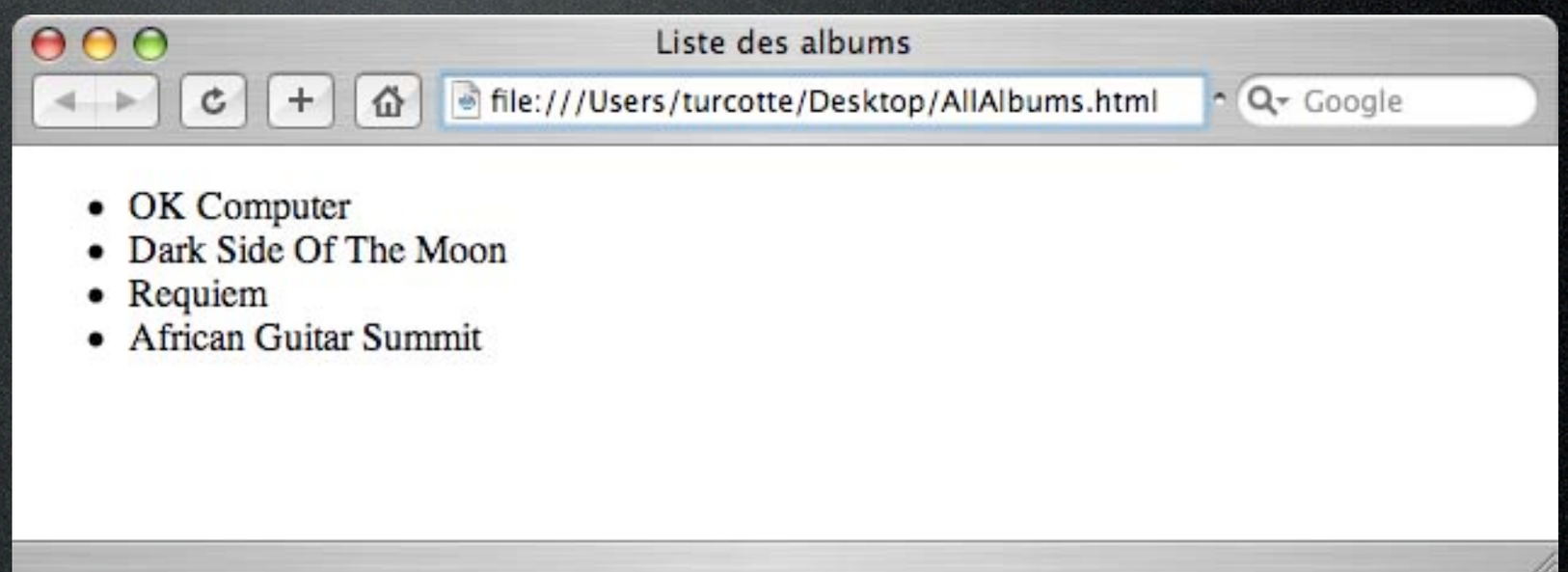

list
elem
elem
elem
elem
elem
Bleu
Rouge
rgb
Orange
Vert
Jaune
255,255,255
html
head
body
title
Liste d'éléments
Bleu
, Rouge
(rgb = 255,255,255)
, Orange
, Vert
, Jaune

```
<xsl:template match="/catalog">

 <html>
  <head>
        <title>Liste des albums</title>
  </head>
  <body>
        <ul>
          <xsl:apply-templates select="album/name"/>
        </ul>
  </body>
 </html>

</xsl:template>

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

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Liste des albums</title>
  </head>
  <body>
    <ul>
      <li>OK Computer</li>
      <li>Dark Side Of The Moon</li>
      <li>Requiem</li>
      <li>African Guitar Summit</li>
    </ul>
  </body>
</html>
```

```
> java -jar saxon.jar -t MusicLibrary.xml AllAlbums.xsl > AllAlbums.html
```

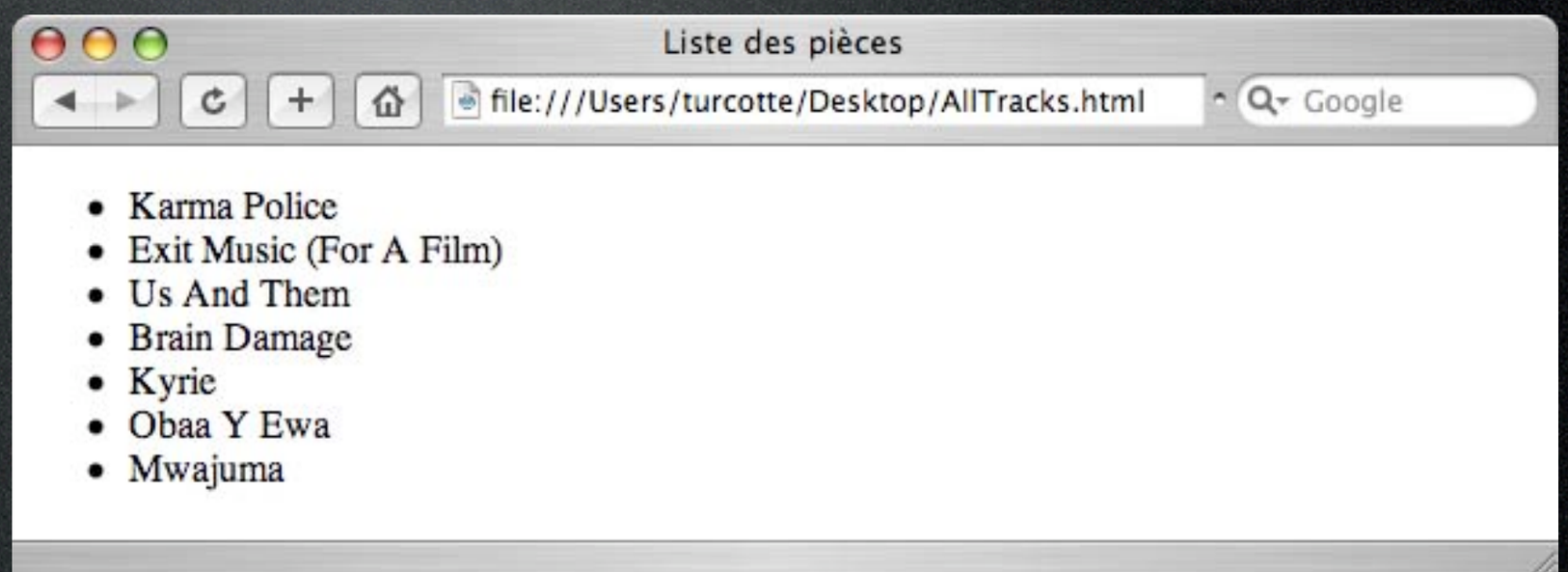
```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Liste des albums</title>
   </head>
   <body>
      <ul>
         <li>OK Computer</li>
         <li>Dark Side Of The Moon</li>
         <li>Requiem</li>
         <li>African Guitar Summit</li>
      </ul>
   </body>
</html>
```



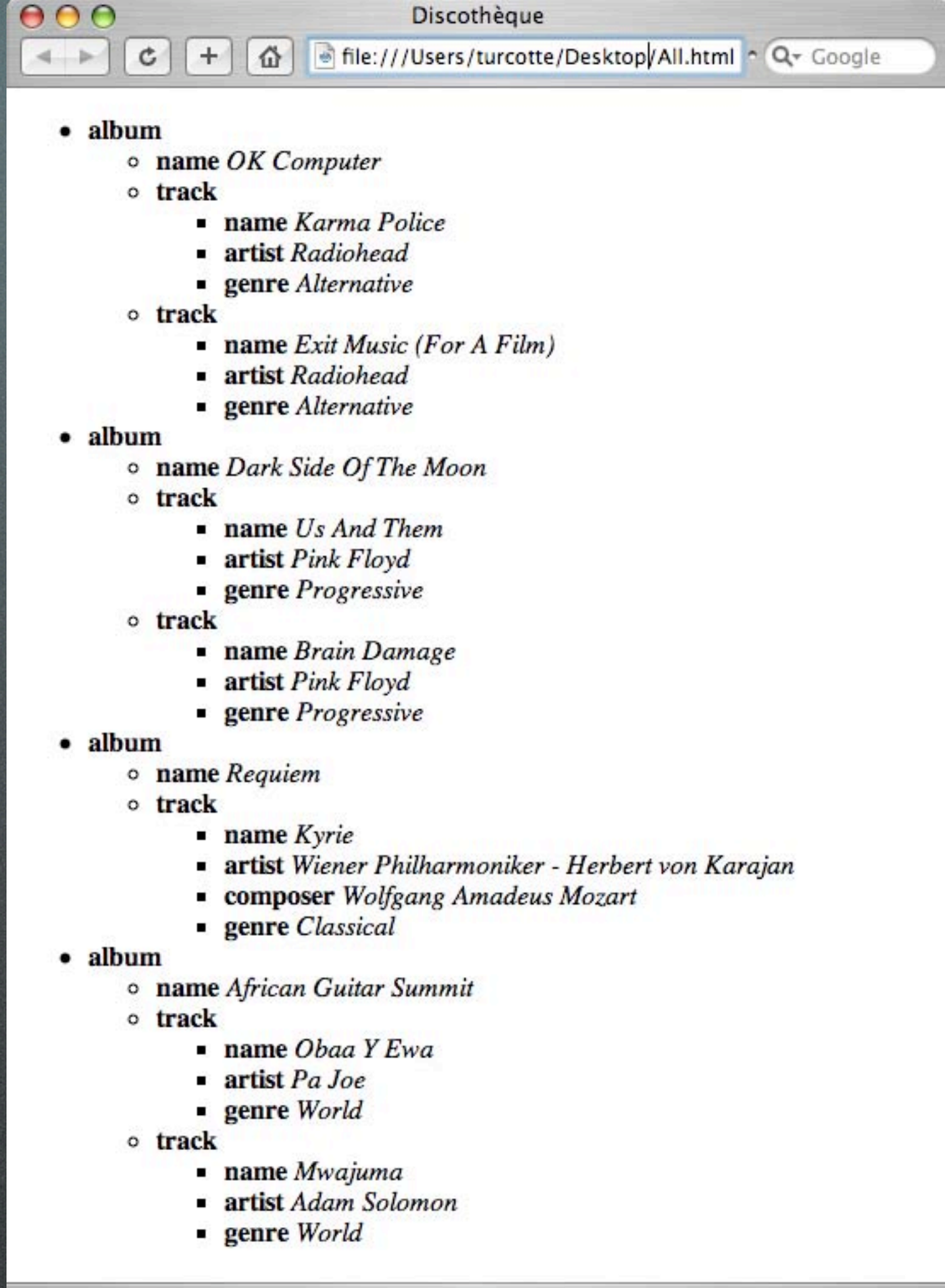
```
<xsl:template match="/catalog">

  <html>
    <head>
        <title>Liste des albums pièces</title>
    </head>
    <body>
        <ul>
          <xsl:apply-templates select="album/name"  select="album/track/name"/>
        </ul>
    </body>
  </html>

</xsl:template>

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

```
> java -jar saxon.jar -t MusicLibrary.xml AllTracks.xsl > AllTracks.html
```

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Liste des pièces</title>
   </head>
   <body>
      <ul>
         <li>Karma Police</li>
         <li>Exit Music (For A Film)</li>
         <li>Us And Them</li>
         <li>Brain Damage</li>
         <li>Kyrie</li>
         <li>Obaa Y Ewa</li>
         <li>Mwajuma</li>
      </ul>
   </body>
</html>
```



```
<xsl:template match="/catalog">
  <html>
    <head>
      <title>Discothèque</title>
    </head>
    <body>
      <ul>
        <xsl:apply-templates/>
      </ul>
    </body>
  </html>
</xsl:template>

<xsl:template match="*">
  <li>
    <b><xsl:value-of select="local-name()"/></b>
    <xsl:choose>
      <xsl:when test="count(*)=0">
        <xsl:text> </xsl:text>
        <em><xsl:value-of select="."/></em>
      </xsl:when>
      <xsl:otherwise>
        <ul>
          <xsl:apply-templates/>
        </ul>
      </xsl:otherwise>
    </xsl:choose>
  </li>
</xsl:template>
```



```
<xsl:template match="/catalog">
...
</xsl:template>

<xsl:template match="*">
  <li>
    <b><xsl:value-of select="local-name()"/></b>
    <xsl:apply-templates select="@*"/>
    <xsl:choose>
      <xsl:when test="count(*)=0">
        <xsl:text> </xsl:text>
         <em><xsl:value-of select="."/></em>
      </xsl:when>
      <xsl:otherwise>
        <ul>
          <xsl:apply-templates/>
        </ul>
      </xsl:otherwise>
    </xsl:choose>
  </li>
 </xsl:template>

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

Discothèque

file:///Users/turcotte/Desktop/All.html

Google

- **album**
  - **name** *OK Computer*
  - **track** (id = 1)
    - **name** *Karma Police*
    - **artist** *Radiohead*
    - **genre** *Alternative*
  - **track** (id = 2)
    - **name** *Exit Music (For A Film)*
    - **artist** *Radiohead*
    - **genre** *Alternative*
- **album**
  - **name** *Dark Side Of The Moon*
  - **track** (id = 3)
    - **name** *Us And Them*
    - **artist** *Pink Floyd*
    - **genre** *Progressive*
  - **track** (id = 4)
    - **name** *Brain Damage*
    - **artist** *Pink Floyd*
    - **genre** *Progressive*
- **album**
  - **name** *Requiem*
  - **track** (id = 5)
    - **name** *Kyrie*
    - **artist** *Wiener Philharmoniker - Herbert von Karajan*
    - **composer** *Wolfgang Amadeus Mozart*
    - **genre** *Classical*
- **album**
  - **name** *African Guitar Summit*
  - **track** (id = 6)
    - **name** *Obaa Y Ewa*
    - **artist** *Pa Joe*
    - **genre** *World*
  - **track** (id = 7)
    - **name** *Mwajuma*
    - **artist** *Adam Solomon*
    - **genre** *World*

Principaux éléments

- Définition d'une règle modèle à l'aide de **<xsl:template match="XPath Expr">**

  …

  </xs:template>

- Relance l'évaluation des règles modèles pour tous les fils du noeud courant, ou ceux sélectionnés par le filtre

  <xsl:apply-templates select="XPath Expr"/>

Principaux éléments

- Insère le contenu de l'élément désigné par l'expression

  <xsl:value-of select="XPath Expr"/>

- Insère une copie du sous arbre désigné par l'expression

  <xsl:copy-of select="XPath Expr"/>

“Une liste de noeuds sources est traitée pour créer un fragment de l'arbre résultat. **L'arbre résultat est construit par traitement d'une liste contenant juste le noeud racine.** Une liste de noeuds sources est traitée par rajout successif des arbres résultants du traitement successif de chaque noeud de la liste. **Un noeud est traité en trouvant toutes les règles modèle dont les motifs [filtres] correspondent au noeud**, et en choisissant le meilleur parmi elles; **la règle modèle retenue est alors instanciée** en considérant le noeud comme noeud courant et la liste de noeuds source comme liste courante de noeuds.”

Algorithme : survol

- Transforme un **arbre-source** en un **arbre-résultat**
- La transformation consiste en une série d'associations entre **filtres** et instantiation de **modèles**
- Les filtres sont appliqués aux éléments de l'**abre source**
- Les modèles sont instanciés afin de créer une partie de l'**arbre résultat**

```
<xsl:template match="/catalog">

  <html>
   <head>
        <title>Liste des albums</title>
   </head>
   <body>
        <ul>
          <xsl:apply-templates select="album/name"  select="album/track/name"/>
        </ul>
   </body>
  </html>

</xsl:template>

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

Algorithme : principes généraux

- L'instanciation d'un modèle se fait par rapport à
  - **un noeud courant**
  - **une liste de noeuds courante**
- La liste de noeuds courante contient toujours le noeud courant

- Durant l'instanciation :
  - la liste de noeuds courante est transformée afin de produire **une nouvelle liste de noeuds courante**
  - chaque noeud de la nouvelle liste, à son tour, devient le **noeud courant**
  - **à la fin de l'instanciation**, le noeud courant et la liste de noeuds courante redeviennent ce qu'ils étaient avant l'instanciation

Algorithme : initialisation

1. Créer un arbre-résultat vide
2. Initialiser le noeud courant à la valeur du noeud racine de l'arbre-source
3. Initialiser la liste de noeuds courante pour ne contenir que le noeud courant
4. Faire un appel à <xsl:apply-templates/>

Algorithme : principal

1. Trouver toutes les règles modèles correspondant au noeud courant de l'arbre-source
2. « Choisir la meilleure parmi elles »
3. Instancier son modèle afin de produire une partie de l'arbre-résultat

- Note : bien que plusieurs règles modèles puissent correspondre à un élément donné, une seule règle modèle sera appliquée (voir résolution de conflits)

Filtre

- L'attribut match spécifie le **filtre** d'une règle modèle
- La syntaxe des filtres est un **sous-ensemble** des expressions XPath
- Entre autres, ces expressions **retournent toujours un ensemble de noeuds**

Filtre

- Un **filtre** est un ensemble de chemins de localisation séparés par «|»
- Un **filtre** est un chemin de localisation dont les étapes n'utilisent que les axes **child** ou **attribut** ; ou l'opérateur **//** (mis pour **descendant-or-self**) ; ou commence par un appel de fonction id ou key

- “Un motif [filtre] est conçu pour **concorder** avec un noeud si et seulement s[i]
  - il existe un contexte tel que
  - lorsque le motif [filtre] est évalué comme une expression dans ce contexte,
    - le noeud appartient à l’ensemble de noeuds résultat”
- “[L]es contextes possibles ont un noeud contextuel qui est le noeud en train d’être mis en correspondance ou un de ses ancêtres”

Filtre

- «track» correspond à n'importe quel élément track
- «artist|composer» correspond à l'un ou l'autre de artist ou composer
- «track/name» correspond à un élément name dont le parent est track
- «/» correspond à la racine

Filtre

- «text()» correspond à n'importe quel noeud textuel
- «id(parall)» correspond à l'élément ayant l'identifiant unique parall
- «@code» correspond à un attribut code
- «@*» correspond à n'importe quel attribut
- «*» correspond à n'importe quel élément

Application des règles modèles

- Sans attribut **select**, **xsl:apply-templates** traite tous les fils du noeud courant
- Y compris les noeuds textuels

```
<xsl:template match="album">
    Les pièces de cet album sont :
    <xsl:apply-templates/>
</xsl:template>
```

Produit possiblement plusieurs espaces et lignes blanches

Application des règles modèles

- Avec l'attribut **select=expr**, **xsl:apply-templates** traite tous les noeuds retournés par l'expression
- Pour cet exemple, les fils **artist**, **composer** et **genre** ne seront pas traités

```
<xsl:template match="track">
   Le nom  de cette pièce est
   <xsl:apply-templates select="name"/>
</xsl:template>
```

Donc pas d'appel récursif pour artist, composer et genre

Application des règles modèles

- Sélectionner tous les descendants de type heading

```
<xsl:template match="book">
    <xsl:apply-templates select=".//heading"/>
</xsl:template>
```

```
<xsl:template match="/catalog">
  <html>
    <head>
      <title>Liste des pièces</title>
    </head>
    <body>
      <ul>
        <xsl:apply-templates select="album/track/name"/>
      </ul>
    </body>
  </html>
</xsl:template>

<xsl:template match="track/name">
  <li>
    <xsl:value-of select="."/>
    <xsl:apply-templates select="ancestor::track/ancestor::album/name"/>
  </li>
</xsl:template>

<xsl:template match="album/name">
    <br/>
    ( <em> <xsl:value-of select="."/> </em> )
</xsl:template>
```

Attention aux boucles infinies !

```
<xsl:template match="/catalog">
  <html>
    <head>
      <title>Liste des pièces</title>
    </head>
    <body>
      <ul>
        <xsl:apply-templates select="album/tra
      </ul>
    </body>
  </html>
</xsl:template>

<xsl:template match="track/name">
  <li>
    <xsl:value-of select="."/>
    <xsl:apply-templates select="ancestor::tr
  </li>
</xsl:template>

<xsl:template match="album/name">
    <br/>
    ( <em> <xsl:value-of select="."/> </em> )
</xsl:template>
```

1. l'application de la règle interne "*|/" force un appel à <xsl:apply-templates> pour tous les fils de la racine (ici, c'est catalog)

2. application de la règle "/catalog" (début de la construction de l'arbre résultat)

3. appel à nouveau à <xsl:apply-templates> pour tous les élements "album/track/name"

3.1 pour chaque "track/name" produire le nom de l'album ; appel à <xsl:apply-templates> sélectionnant l'élément ancetre ancestor::track/ancestor::album/name

3.2 application de la règle "album/track"

```
<xsl:template match="/catalog">
  <html>
    <head>
      <title>Liste des pièces</title>
    </head>
    <body>
      <table border="1" cellpadding="5">
        <tr>
          <th>Id.</th>
          <th>Nom</th>
        </tr>
        <xsl:apply-templates select="album/track"/>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="track">
  <tr>
    <td><xsl:apply-templates select="@id"/></td>
    <td><xsl:apply-templates select="name"/></td>
  </tr>
</xsl:template>

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

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

Id.	Nom
1	Karma Police
2	Exit Music (For A Film)
3	Us And Them
4	Brain Damage
5	Kyrie
6	Obaa Y Ewa
7	Mwajuma

Deux appels récursifs par règle modèle

Règles modèle internes

- Que se passe-t-il si l'on applique une transformation vide à un document ?

```
> java -jar saxon.jar -t MusicLibrary.xml Tfo-00.xsl > AllAlbums.html
```

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

</xsl:transform>
```

Règles modèle internes

- On obtient ce qui suit (à l'exception des quelques lignes vides supprimées) :

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

    OK Computer
    Karma Police
    Radiohead
    Alternative

    Exit Music (For A Film)
    Radiohead
    Alternative

Dark Side Of The Moon

    Us And Them
    ...
```

Règles modèle internes

- Il y a donc un ensemble de règles modèles implicites (internes)

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

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

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

Règles modèle internes

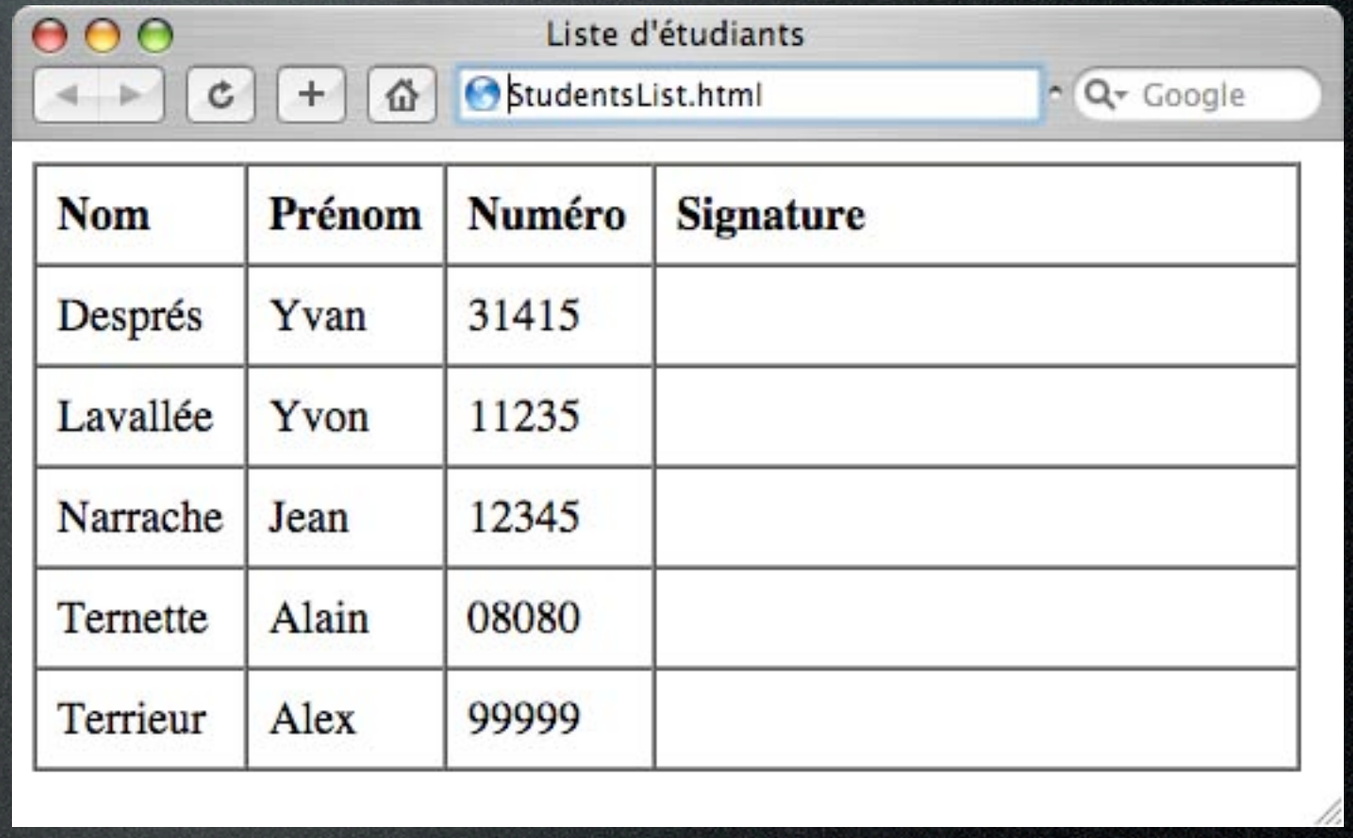
- Pourquoi est-ce important ?
- Parce que si vos filtres sont mal définis, il se peut que le processeur juge l'une des règles internes comme étant « meilleure », et l'applique !
- Ce qui produira un résultat différent de celui escompté

Résolution de conflits

- Voir section 5.5 de la recommandation :
  - Les filtres généraux, tels que « * » et «@*», ont une faible priorité (-0.5)
  - Les filtres constitués d'un type spécifique de noeud, par exemple «track», ont une priorité neutre (0.0)
  - Les filtres plus spécifiques ont une priorité de 0.5
  - L'usager peut aussi spécifier une valeur à l'aide de l'attribut priority

Répétition

```
<group>
 <student>
  <name>
    <first>Jean</first>
    <last>Narrache</last>
  </name>
  <id>12345</id>
  <midterm>40</midterm>
  <final>30</final>
  <project>20</project>
  <grade>30</grade>
 </student>
 ...
 <student>
  <name>
    <first>Alain</first>
    <last>Ternette</last>
  </name>
  <id>08080</id>
  <midterm>100</midterm>
  <final>100</final>
  <project>100</project>
  <grade>100</grade>
 </student>
</group
```



Nom	Prénom	Numéro	Signature
Després	Yvan	31415	
Lavallée	Yvon	11235	
Narrache	Jean	12345	
Ternette	Alain	08080	
Terrieur	Alex	99999	

```
<xsl:template match="/group">
  <html>
    <head>
      <title>Liste d'étudiants</title>
    </head>
    <body style="font-size:large">
      <table border="1" cellpadding="8" cellspacing="0">
        <tr>
          <th align="left">Nom</th>
          <th align="left">Prénom</th>
          <th align="left">Numéro</th>
          <th align="left" width="240">Signature</th>
        </tr>
        <xsl:for-each select="student">
          <xsl:sort select="name/last" order="ascending"/>
          <tr>
            <td align="left"><xsl:value-of select="name/last"/></td>
            <td align="left"><xsl:value-of select="name/first"/></td>
            <td align="left"><xsl:value-of select="id"/></td>
            <td></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
```

xsl:for-each seul n'est pas très intéressant puisqu'on peut obtenir les mêmes résultats à l'aide d'appels récursifs.

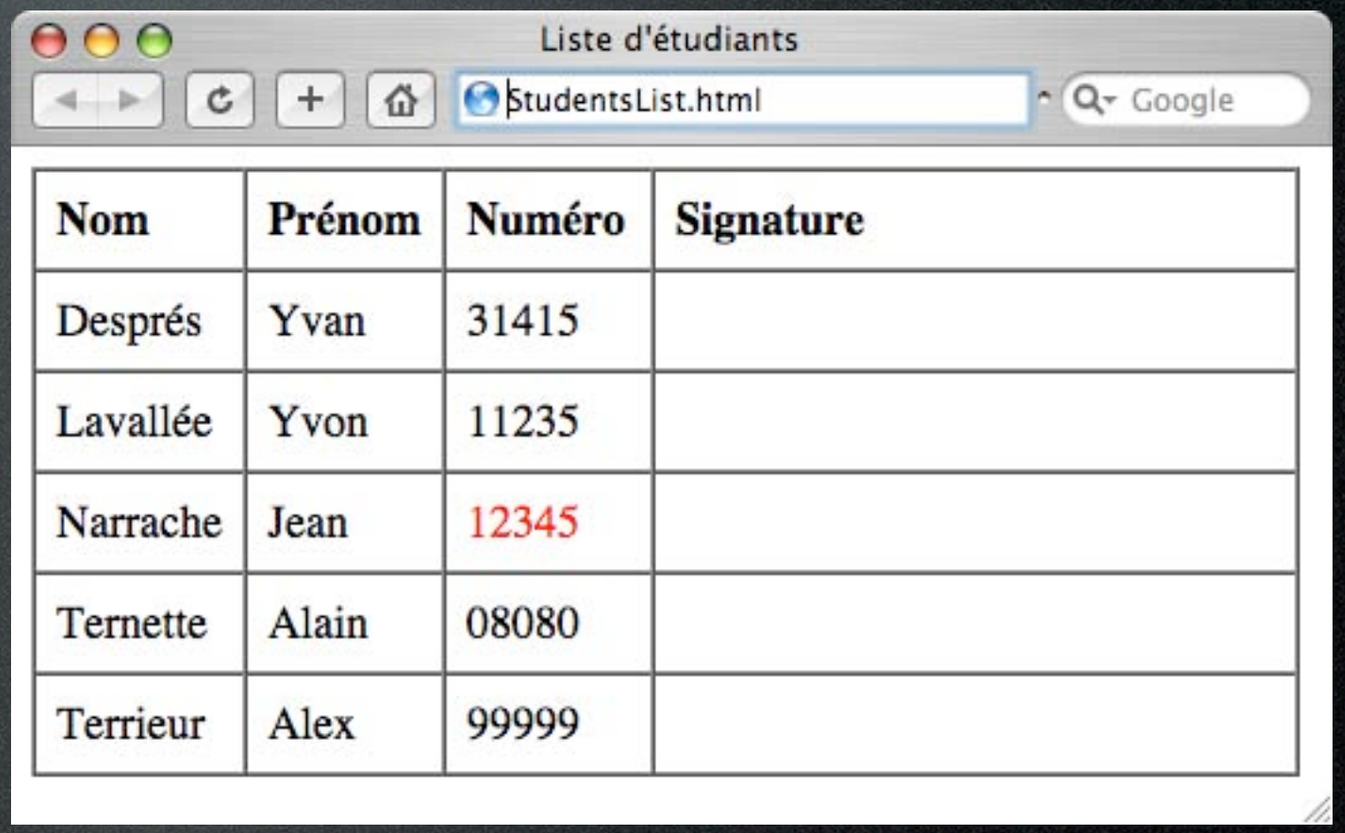
C'est lorsqu'on le combine avec le tri que l'on voit l'intérêt.

Conditionnel

```
<xsl:for-each select="student">
  <xsl:sort select="name/last" order="ascending"/>
  <tr>
    <td align="left"><xsl:value-of select="name/last"/></td>
    <td align="left"><xsl:value-of select="name/first"/></td>

    <td align="left">
      <xsl:if test="grade &lt; 50">
        <xsl:attribute name="style">
          <xsl:text>color:red</xsl:text>
        </xsl:attribute>
      </xsl:if>
      <xsl:value-of select="id"/>
    </td>
    <td></td>
  </tr>
</xsl:for-each>
```

C'est exemple démontre :
- L'utilisation d'un énoncé conditionnel
- La construction d'un attribut

Notez aussi l'utilisation du < obligatoire



Nom	Prénom	Numéro	Signature
Després	Yvan	31415	
Lavallée	Yvon	11235	
Narrache	Jean	12345	
Ternette	Alain	08080	
Terrieur	Alex	99999	

Conditionnel

```
<xsl:choose>
    <xsl:when test=expression-booléenne>
        ...
    </xsl:when>
    <xsl:when test=expression-booléenne>
        ...
    </xsl:when>
    <xsl:otherwise>
        ...
    </xsl:otherwise>
</xsl:choose>
```

<xsl:copy-of ...>

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

<xsl:transform version="1.0"
               xmlns:xsl="http://www                orm">

 <xsl:output indent="yes"
             method="xml"
             encoding="UTF-8"/>

 <xsl:template match="/group">

  <group>
   <xsl:for-each select="student">
    <xsl:sort select="id" data-type="number" order="descending"/>
    <xsl:copy-of select="."/>
   </xsl:for-each>
  </group>

 </xsl:template>

</xsl:transform>
```

Remarquez que le tri est
- effectué sur des nombres
- l'ordre est descending

xsl:copy-of copie un sous-emsemble de noeuds de l'arbre-source vers l'arbre-résultat sans les convertir en chaîne

xsl:copy-of copie un sous-emsemble de noeuds de l'arbre-source vers l'arbre-résultat sans les convertir en chaîne

Quel est le résultat de cette transformation?

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

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

 <xsl:output indent="yes"
             method="xml"
             encoding="UTF-8"/>

 <xsl:template match="/group">
  <group>
   <xsl:apply-templates select="student">
    <xsl:sort select="id" order="descending"/>
   </xsl:apply-templates>
  </group>

 </xsl:template>

 <xsl:template match="student">
  <xsl:copy-of select="."/>
 <xsl:template>

</xsl:transform>
```

- Remplacer une boucle “xsl:for-each” par un traitement récursif
- xsl:sort s’applique aussi à xsl:apply-templates

Modèles nommés

- Une règle modèle peut avoir un attribut **name**
- Généralement, l'instanciation de cette règle se fait à l'aide d'un appel de fonction **xsl:call-template**
- Généralement, un modèle nommé possède un ou plusieurs paramètres

```
<xsl:template match="/group">

  <html>
   <head>
     <title>Liste d'étudiants</title>
   </head>
   <body style="font-size:large">
     <table border="1" cellpadding="8" cellspacing="0">
       <tr>
         <th align="left">Nom</th>
         <th align="left">Prénom</th>
         <th align="left">Numéro</th>
         <th>Note</th>
       </tr>
      <xsl:for-each select="student">
       <xsl:sort select="name/last" order="ascending"/>
       <tr>
         <td align="left"><xsl:value-of select="name/last"/></td>
         <td align="left"><xsl:value-of select="name/first"/></td>
         <td align="left"><xsl:value-of select="id"/></td>
         <td>
           <xsl:call-template name="toLetter">
             <xsl:with-param name="mark" select="grade"/>
           </xsl:call-template>
         </td>
       </tr>
      </xsl:for-each>
     </table>
   </body>
  </html>

</xsl:template>
```

```
<xsl:template name="toLetter">
 <xsl:param name="mark"/>
 <xsl:choose>
   <xsl:when test="$mark >= 90">A+</xsl:when>
   <xsl:when test="$mark >= 85">A</xsl:when>
   <xsl:when test="$mark >= 80">A-</xsl:when>
   <xsl:when test="$mark >= 75">B+</xsl:when>
   <xsl:when test="$mark >= 70">B</xsl:when>
   <xsl:when test="$mark >= 66">C+</xsl:when>
   <xsl:when test="$mark >= 60">C</xsl:when>
   <xsl:when test="$mark >= 55">D+</xsl:when>
   <xsl:when test="$mark >= 50">D</xsl:when>
   <xsl:when test="$mark >= 40">E</xsl:when>
   <xsl:otherwise>F</xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Liste d'étudiants

StudentsGrades.html

Google

Nom	Prénom	Numéro	Note
Després	Yvan	31415	A
Lavallée	Yvon	11235	B
Narrache	Jean	12345	E
Ternette	Alain	08080	A+
Terrieur	Alex	99999	A-

Processeurs XSLT

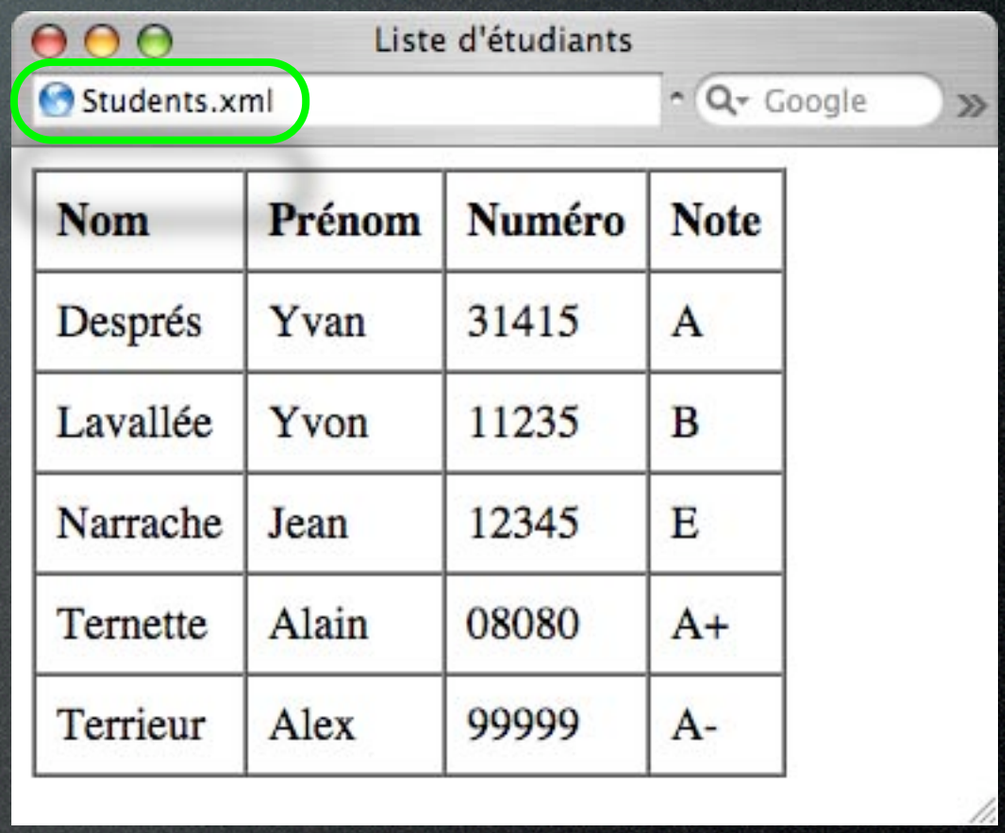
- JAXP (TransformerFactory.newInstance().newTransformer(tfo))
- Xalan-Java
  [http://xml.apache.org/xalan-j/]
  XSLT 1.0
- Saxon
  [http://saxon.sourceforge.net]
  XSLT 2.0

Processeurs XSLT

Favorise la séparation contenu/ présentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="StudentsGrades.xsl"?>
<group>
 <student>
  <name>
    <first>Jean</first>
    <last>Narrache</last>
  </name>
  <id>12345</id>
  <midterm>40</midterm>
  <final>30</final>
  <project>20</project>
  <grade>30</grade>
 </student>
 ...
 <student>
   ...
 </student>
</group
```

- Tout simplement à travers votre fureteur (Safari, IE)



Nom	Prénom	Numéro	Note
Després	Yvan	31415	A
Lavallée	Yvon	11235	B
Narrache	Jean	12345	E
Ternette	Alain	08080	A+
Terrieur	Alex	99999	A-

- Attention Firefox n'aime pas method="xhtml" dans xsl:output ...

Génération d'attributs

- La construction suivante sert à créer un attribut **href** pour l'élément englobant **a**.

```
...
    <li>
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="link"/>
        </xsl:attribute>
        <xsl:value-of select="name"/>
      </a>
    </li>
...
```

Génération d'attributs

- La construction suivante sert à créer un attribut **href** pour l'élément englobant **a**. Ici, le contenu des parenthèses est une expression XPath

```
...
    <li>
      <a href= "{ link }"><xsl:value-of select="name"/></a>
    </li>
...
```

XSLT

- Transformation d'un document XML
- En général, le résultat est aussi un document XML
- L'arbre-résultat peut avoir une structure différente de l'arbre-source
- Le résultat contient un sous-ensemble ou la totalité des informations du document source

XML

Côté-client

JavaScript et XSL

- Voici un exemple où une **transformation XSL** est exécutée à partir d'un programme **JavaScript**
  - **index.html** (JavaScript)
  - **MusicLibrary.xml**
  - **TracksAndAlbum.xsl**

MusicLibrary.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <album>
    <name>OK Computer</name>
    <track>
      <name>Karma Police</name>
      <artist>Radiohead</artist>
      <genre>Alternative</genre>
    </track>
    <track>
      <name>Exit Music (For A Film)</name>
      <artist>Radiohead</artist>
      <genre>Alternative</genre>
    </track>
  </album>
  <album>
    <name>Dark Side Of The Moon</name>
    <track>
      <name>Us And Them</name>
      <artist>Pink Floyd</artist>
      <genre>Progressive</genre>
    </track>
    <track>
      <name>Brain Damage</name>
      <artist>Pink Floyd</artist>
      <genre>Progressive</genre>
    </track>
  </album>
  ...
  <album>
    <name>Requiem</name>
    <track>
      <name>Kyrie</name>
      <artist>Wiener Philharmoniker - Herbert von Karajan</artist>
      <composer>Wolfgang Amadeus Mozart</composer>
      <genre>Classical</genre>
    </track>
  </album>
  <album>
    <name>African Guitar Summit</name>
    <track>
      <name>Obaa Y Ewa</name>
      <artist>Pa Joe</artist>
      <genre>World</genre>
    </track>
    <track>
      <name>Mwajuma</name>
      <artist>Adam Solomon</artist>
      <genre>World</genre>
    </track>
  </album>
</catalog>
```

TracksAndAlbum.xsl

```
<xsl:template match="/catalog">
  <html>
    <head>
      <title>Liste des pièces</title>
    </head>
    <body>
      <ul>
        <xsl:apply-templates select="album/track/name"/>
      </ul>
    </body>
  </html>
</xsl:template>

<xsl:template match="track/name">
  <li>
    <xsl:value-of select="."/>
    <xsl:apply-templates select="ancestor::track/ancestor::album/name"/>
  </li>
</xsl:template>

<xsl:template match="album/name">
    <br/>
    ( <em> <xsl:value-of select="."/> </em> )
</xsl:template>
```

index.html

```
<html>
 <head>
  <script>
   function loadXMLDoc( fname ) {
    var doc;
    doc = document.implementation.createDocument( "", null, null );
    doc.async=false;
    doc.load( fname );
    return( doc );
   }
   function displayResult() {
    xml=loadXMLDoc( "MusicLibrary.xml" );
    xsl=loadXMLDoc( "TracksAndAlbum.xsl" );

    xsltProcessor = new XSLTProcessor();
    xsltProcessor.importStylesheet( xsl );

    resultDocument = xsltProcessor.transformToFragment( xml, document );

    document.getElementById( "example" ).appendChild( resultDocument );
   }
  </script>
 </head>
 <body id="example" onLoad="displayResult()">
 </body>
</html>
```

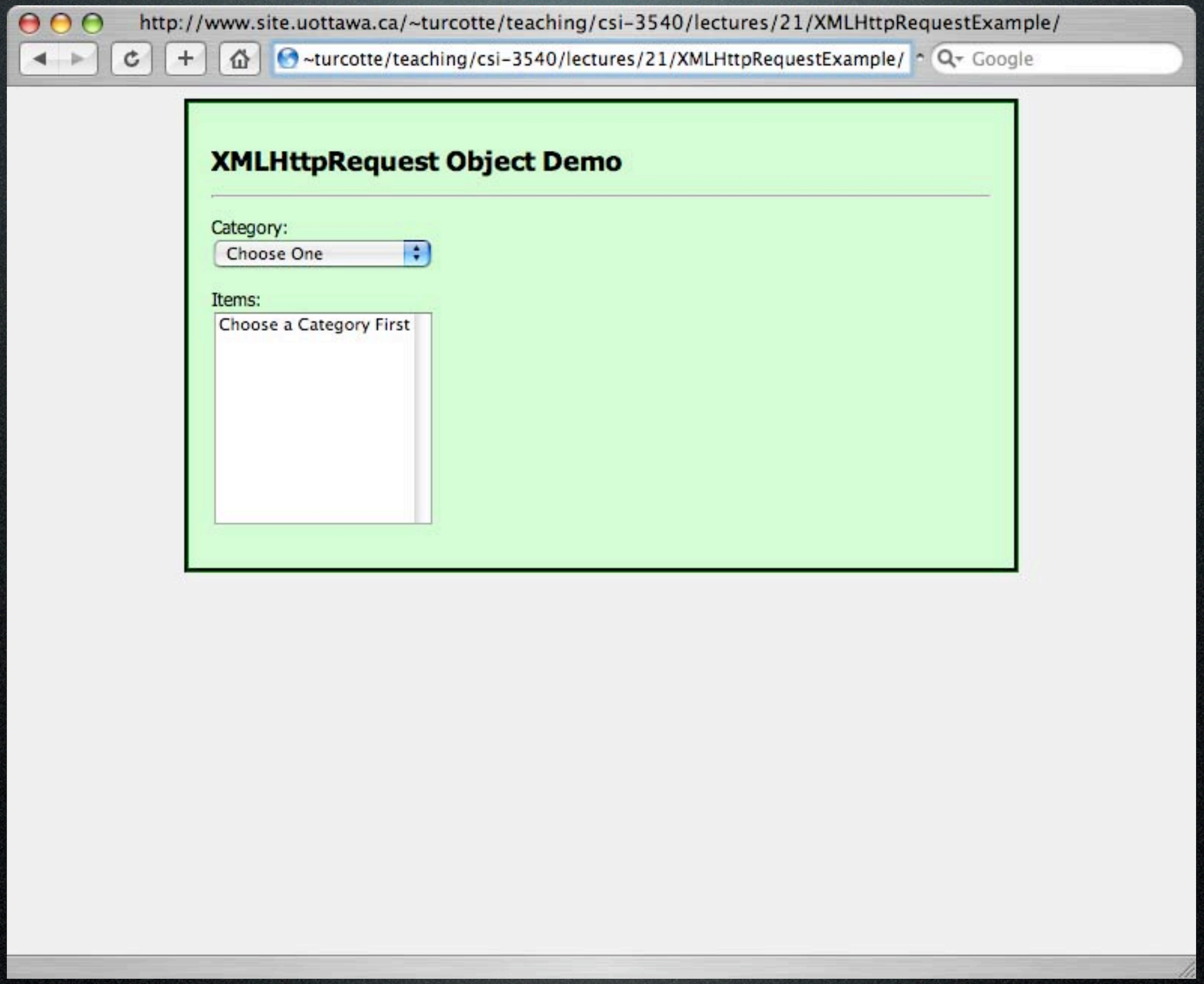
JavaScript et XSL

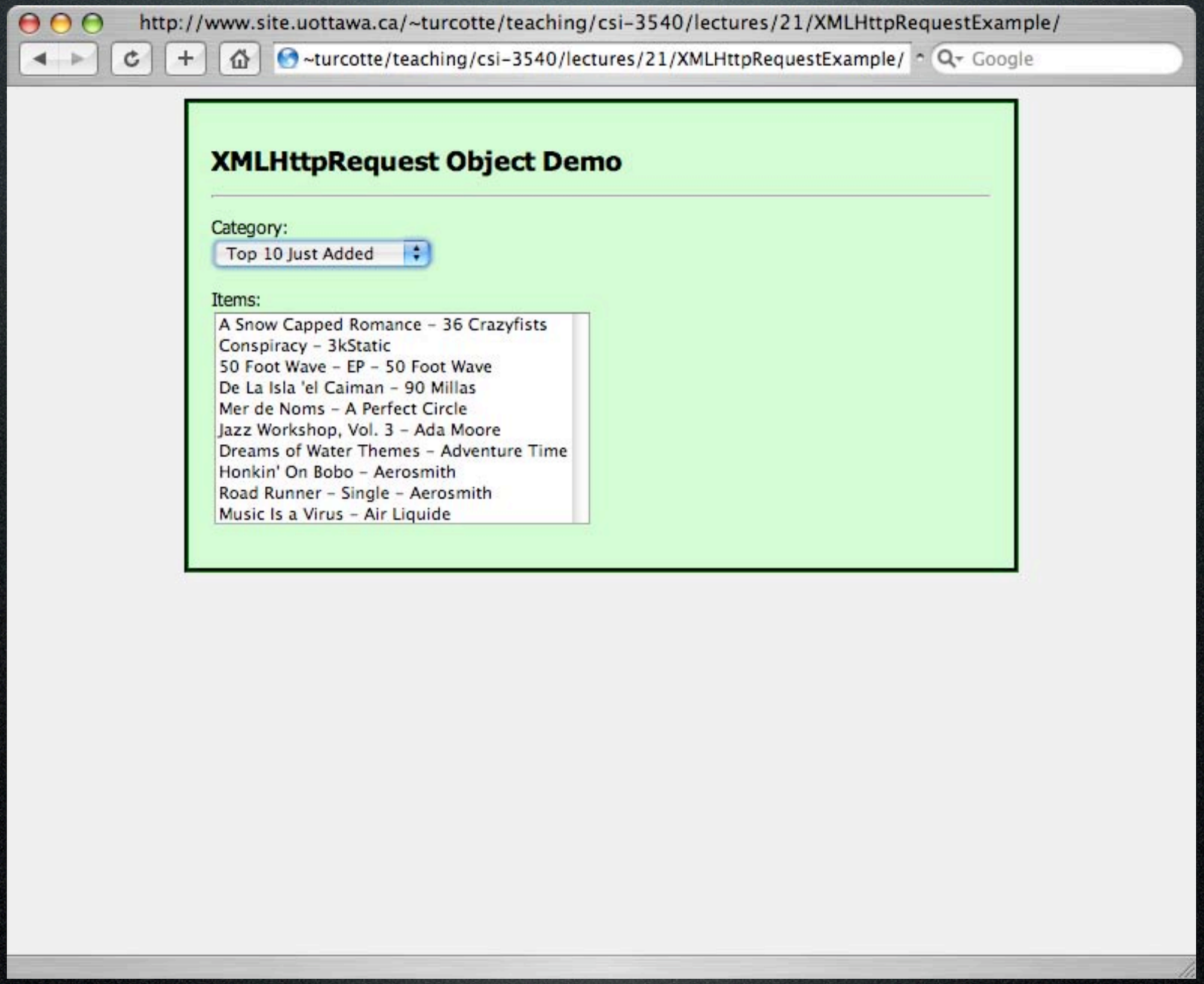
Applications

- Serveur transmet les données à l'aide d'un document XML (liste de villes)
- Le client applique une transformation afin de créer des éléments XHTML (les items d'un menu)
- ...

XMLHttpRequestExample

developer.apple.com/internet/webcontent/XMLHttpRequestExample/example.html


http://www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/
~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/
Google
XMLHttpRequest Object Demo
Category:
Choose One
Items:
Choose a Category First


http://www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/
~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/
Google
XMLHttpRequest Object Demo
Category:
Top 10 Just Added
Items:
A Snow Capped Romance - 36 Crazyfists
Conspiracy - 3kStatic
50 Foot Wave - EP - 50 Foot Wave
De La Isla 'el Caiman - 90 Millas
Mer de Noms - A Perfect Circle
Jazz Workshop, Vol. 3 - Ada Moore
Dreams of Water Themes - Adventure Time
Honkin' On Bobo - Aerosmith
Road Runner - Single - Aerosmith
Music Is a Virus - Air Liquide

http://www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/

~turcotte/teaching/csi-3540/lectures/21/XMLHttpRequestExample/

Google

XMLHttpRequest Object Demo

Category:

Top 10 Just Added

Items:

- A Snow Capped Romance - 36 Crazyfists
- Conspiracy - 3kStatic
- 50 Foot Wave - EP - 50 Foot Wave
- De La Isla 'el Caiman - 90 Millas
- Mer de Noms - A Perfect Circle
- Jazz Workshop, Vol. 3 - Ada Moore
- Dreams of Water Themes - Adventure Time
- Honkin' On Bobo - Aerosmith
- Road Runner - Single - Aerosmith
- Music Is a Virus - Air Liquide

Conspiracy
3kStatic

Release Date: May 23, 2003
Total Songs: 13
Genre: Electronic
Price: $9.99
Copyright 2003 INgrooves

```
<html ...>
<head>...</head>
<body>
<h1>XMLHttpRequest Object Demo</h1>
<hr />

<form>
<p>Category:<br />
<select onchange="loadDoc(event)">
   <option value="">Choose One</option>
   <option value="songs.xml">Top 10 Songs</option>
   <option value="albums.xml">Top 10 Albums</option>
   <option value="newreleases.xml">Top 10 New Releases</option>
   <option value="justadded.xml">Top 10 Just Added</option>
</select>
</p>
<p>Items:<br />
<select size="10" id="topics" onchange="showDetail(event)">
   <option value="">Choose a Category First</option>
</select>
</p>
</form>
<div id="details"><span></span></div>
</body>
<html>
```

```
// invoked by "Category" select element change;
// loads chosen XML document, clears Topics select
// element, loads new items into Topics select element
function loadDoc( evt ) {
    // equalize W3C/IE event models to get event object
    evt = (evt) ? evt : ((window.event) ? window.event : null);
    if ( evt ) {
        // equalize W3C/IE models to get event target reference
        var elem = (evt.target) ? evt.target : ((evt.srcElement) ? evt.srcElement : null);
        if ( elem ) {
            try {
                if ( elem.selectedIndex > 0 ) {
                    loadXMLDoc( elem.options[ elem.selectedIndex ].value );
                }
            }
            catch( e ) {
                var msg = (typeof e == "string") ? e : ((e.message) ? e.message : "Unknown Error");
                alert("Unable to get XML data:\n" + msg);
                return;
            }
        }
    }
}
```

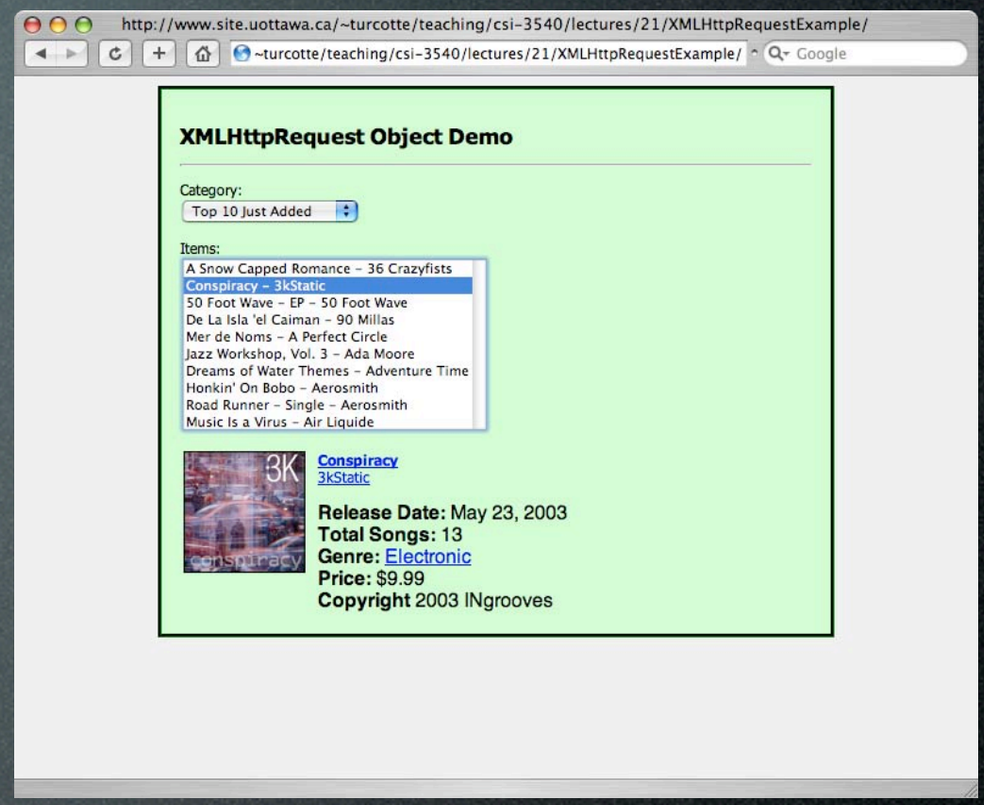
```
// global request and XML document objects
var req;

// retrieve XML document (reusable generic function);
// parameter is URL string (relative or complete) to
// an .xml file whose Content-Type is a valid XML
// type, such as text/xml; XML source must be from
// same domain as HTML file

function loadXMLDoc( url ) {
    // branch for native XMLHttpRequest object
    if ( window.XMLHttpRequest ) {
        req = new XMLHttpRequest();
        req.onreadystatechange = processReqChange;
        req.open( "GET", url, true );
        req.send( null );
    // branch for IE/Windows ActiveX version
    } else if ( window.ActiveXObject ) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
        if (req) {
            req.onreadystatechange = processReqChange;
            req.open("GET", url, true);
            req.send();
        }
    }
}
```

```
// handle onreadystatechange event of req object

function processReqChange() {
    // only if req shows "loaded"
    if ( req.readyState == 4 ) {
        // only if "OK"
        if ( req.status == 200 ) {

            clearTopicList();
            buildTopicList();

        } else {

            alert("There was a problem retrieving the XML data:\n" +
                req.statusText);

        }
    }
}
```

```
// empty Topics select list content

function clearTopicList() {
    var select = document.getElementById( "topics" );
    while ( select.length > 0 ) {
        select.remove( 0 );
    }
}


 // add item to select element the less
// elegant, but compatible way.

function appendToSelect( select, value, content ) {
    var opt;
    opt = document.createElement( "option" );
    opt.value = value;
    opt.appendChild( content );
    select.appendChild( opt );
}
```

```
// fill Topics select list with items from
// the current XML document

function buildTopicList() {

    var select = document.getElementById( "topics" );
    var items = req.responseXML.getElementsByTagName( "item" );

    // loop through <item> elements, and add each nested
    // <title> element to Topics select element
    for ( var i = 0; i < items.length; i++ ) {
        appendToSelect( select, i,
            document.createTextNode( getElementTextNS( "", "title", items[ i ], 0 ) ) );
    }

    // clear detail display
    document.getElementById( "details" ).innerHTML = "";
}
```

```
<html ...>
<head>...</head>
<body>
<h1>XMLHttpRequest Object Demo</h1>
<hr />

<form>
<p>Category:<br />
<select onchange="loadDoc(event)">
   <option value="">Choose One</option>
   <option value="songs.xml">Top 10 Songs</option>
   <option value="albums.xml">Top 10 Albums</option>
   <option value="newreleases.xml">Top 10 New Releases</option>
   <option value="justadded.xml">Top 10 Just Added</option>
</select>
</p>
<p>Items:<br />
<select size="10" id="topics" onchange="showDetail( event )">
   <option value="">Choose a Category First</option>
</select>
</p>
</form>
<div id="details"><span></span></div>
</body>
<html>
```



```
// display details retrieved from XML document

function showDetail( evt ) {
    evt = (evt) ? evt : ((window.event) ? window.event : null);
    var item, content, div;
    if ( evt ) {
        var select = (evt.target) ? evt.target : ((evt.srcElement) ? evt.srcElement : null);
        if ( select && select.options.length > 1 ) {
            // copy <content:encoded> element text for
            // the selected item
            item = req.responseXML.getElementsByTagName( "item" )[ select.value ];
            content = getElementTextNS( "content", "encoded", item, 0 );
            div = document.getElementById( "details" );
            div.innerHTML = "";
            // blast new HTML content into "details" <div>
            div.innerHTML = content;
        }
    }
}
```

Ressources

- Langage de balisage extensible (XML) 1.0 [http://pages.videotron.com/fyergeau/w3c/xml10/REC-xml-19980210.fr.html] 2007
- Les espaces de nommage dans XML 1.1 [http://www.yoyodesign.org/doc/w3c/xml-names11] 2007

Ressources (suite)

- Java API for XML Processing (JAXP) Specification 1.3 [http://java.sun.com/xml/downloads/jaxp.html] 2007
- Java API for XML Code Samples [http://java.sun.com/developer/codesamples/xml.html] 2010-03-12
- SAX [http://www.saxproject.org] 2007
- <oXygen/> XML Editor & XSLT Debugger [http://www.oxygenxml.com] 2007

Ressources (suite)

- The GNU JAXP Project [http://www.gnu.org/software/classpathx/jaxp/jaxp.html] 2007
- Langage XML Path (XPath) Version 1.0 [http://xmlfr.org/w3c/TR/xpath] 2007
- Transformations XSL (XSLT) [http://xmlfr.org/w3c/TR/xslt/] 2007
- XML: Looking at the Forest Instead of the Trees par Guy Lapalme [http://www.iro.umontreal.ca/~lapalme/ForestInsteadOfTheTrees/] 2007