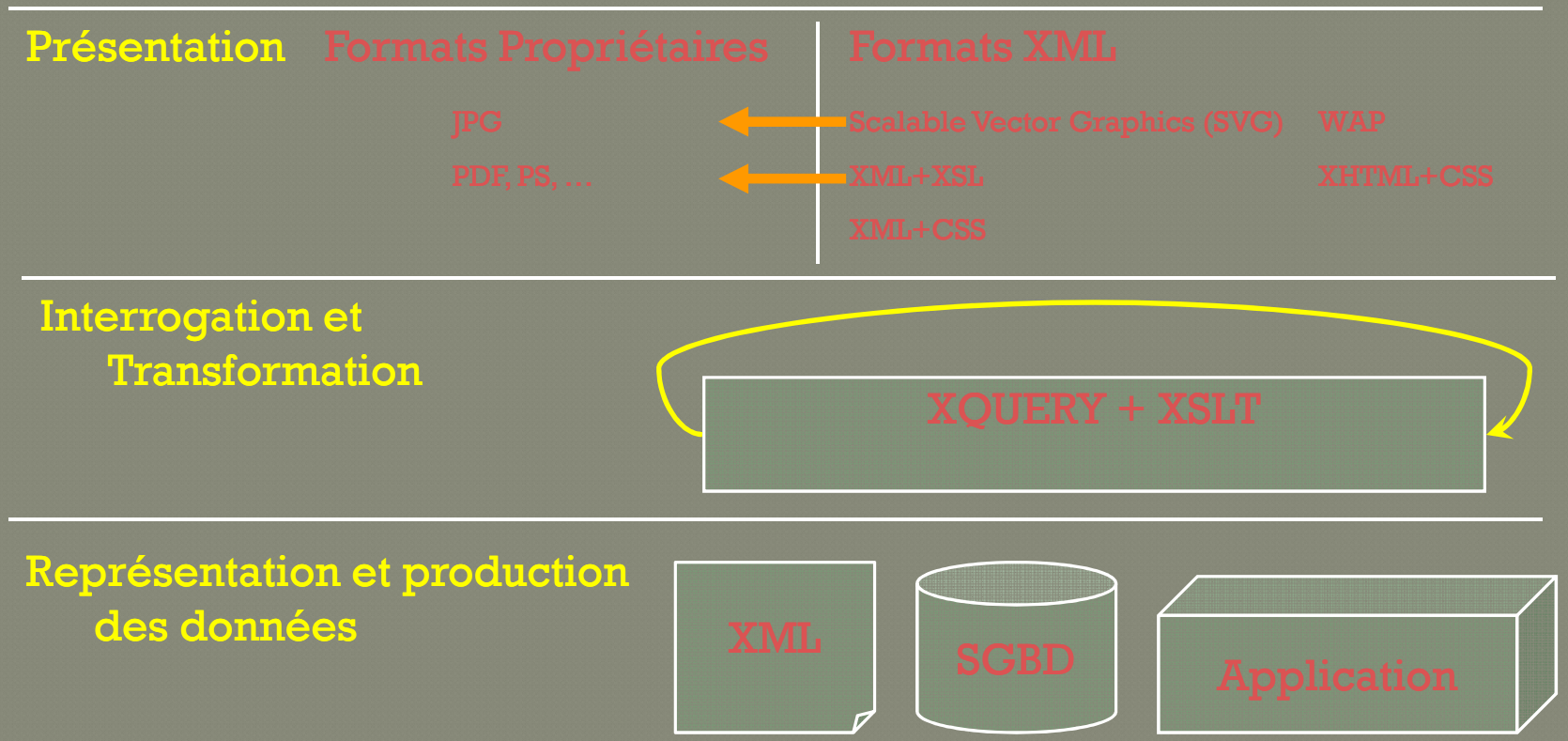
Transformation et présentation de documents XML

XSL Transformations (XSLT)
Extensible Stylesheet Language (XSL)
Cascading Style Sheets (CSS)

Principes généraux



Le langage XSLT

XSL Transformations (XSLT)
Version 1.0
W3C Recommendation 16 November 1999

Version 2.0
W3C Recommendation 23 January 2007

http://www.w3.org/tr/xslt

Introduction

- XSLT est un langage de transformation de documents XML
- Une feuille de style est un document XML
- Elle s'applique à un document XML pour produire du XML ou du texte
- Une feuille XSLT est composée de règles appelées ***templates***

Mécanisme général

- On parcourt récursivement l'arbre du document source
- Pour chaque nœud on applique la règle correspondante qui indique
  - ce qui doit être produit dans l'arbre du document résultat
  - si le parcours récursif du document source continue

Une feuille de style très simple

- Feuille de style vide
  - Attribut version obligatoire (2.0 est possible)
  - **xmlns:xsl** définit xsl comme le préfixe (**namespace**) des balises appartenant au langage XSLT
    - L'espace de nom permet au processeur XSLT de reconnaitre les « instructions »du langage XSLT

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

   <!– ICI mettre un ensemble de règles (template) -->

</xsl:stylesheet>
```

Template (1/2)

- Un *template* est composé de deux parties
  - La première sélectionne des nœuds avec le langage XPath (*attr.* ***match***)
  - La seconde indique ce qui doit être produit
    - Litéraux : Fragment de documents XML
    - Instructions XSLT (éléments prédéfinis)
  - Une template est évalué dans le contexte d'un nœud donné

```
<document>
    <titre>Un titre</titre>
    <auteur>L'auteur</auteur
    >
</document>
```

```
<xsl:template match="titre">
        <h1>
            <xsl:value-of select="."/>
        </h1>
</xsl:template>
```

Template (2/2)

- Un *template* est composé de deux parties
  - [1] Pattern ::= LocationPathPattern | Pattern '|' LocationPathPattern [2] LocationPathPattern ::= '/' RelativePathPattern? | IdKeyPattern (('/' | '//') RelativePathPattern)? | '//'? RelativePathPattern [3] IdKeyPattern ::= 'id' '(' Literal ')' | 'key' '(' Literal ',' Literal ')' [4] RelativePathPattern ::= StepPattern | RelativePathPattern '/' StepPattern | RelativePathPattern '//' StepPattern [5] StepPattern ::= ChildOrAttributeAxisSpecifier NodeTest Predicate* [6] ChildOrAttributeAxisSpecifier ::= AbbreviatedAxisSpecifier | ('child' | 'attribute') '::'

```
<xsl:template match="titre">
        <h1>
            <xsl:value-of select="."/>
        </h1>

</xsl:template>
```

Feuille de style

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

<xsl:template match="/">
<html>
 <head> <title> Mon livre </title> </head>
  <body> <xsl:apply-templates/> </body>
</html>
</xsl:template>

<xsl:template match='contenu'>
        <p> <xsl:value-of select="."/> </p>
</xsl:template>

<xsl:template match='titre'>
        <h1> <xsl:value-of select="."/> </h1>
</xsl:template>
</xsl:stylesheet>
```

```
<document>
  <titre>Un titre</titre>
  <contenu>blabla</contenu>
</document>
```

```
<html>
 <head>
      <title> Mon livre </titre>
 </head>
      <h1>Un titre</h1>>
      <p>blabla</p>
</html>
```

Algorithme général

- Règles par défaut:
  - L'arbre du document source est parcouru à partir de la racine
  - Si aucune règle ne correspond à un élément le parcours continu avec ses fils
  - Les nœuds textes sont recopiés dans l'arbre résultat
- Si une règle vérifie le nœud courant le *template* est instancié dans l'arbre résultat
  - ***L'application aux fils doit être explicite***
    - <xsl:apply-templates/>
    - <xsl:apply-templates select =« XPath relatif ou absolu»/>

Accès aux données sources

- En allant « chercher » la valeur
  - <xsl:value select='***XPATH***'/>
    - Copie la valeur du premier nœud texte indiqué par le chemin XPath (relatif au nœud contexte)
- En provocant l'application des règles à un ensemble de nœuds
  - <xsl:apply-templates select='*XPATH*'/>
    - Applique l'ensemble des règles aux nœuds atteints par le chemin XPath

Priorité

- Il peut arriver que plusieurs règles puissent être appliquées
- La priorité peut être spécifiée explicitement par l'attribut *priority*
- Sinon c'est
  - la règle la plus spécifique qui est appliquée
  - La dernière règle qui correspond

Priorité - Exemple

```
<xsl:template select="Para">
  <P><xsl:apply-templates</P>
</xsl:template>

<xsl:template
     select='Para[@type="citation"]'>
    <BLOCKQUOTE>
      <xsl:apply-templates/>
    </BLOCKQUOTE>
</xsl:template>
```

Pour <Para>...</Para> La première règle s'applique.

Pour <Para type="citation">...</Para> Seule la seconde s'applique.

Construction d'éléments et d'attributs

- Il est possible de créer des éléments ou des attributs à l'aide de valeurs extraites du document source
- Deux modes de construction:
  - Instanciation explicite
    <title num='1'><xsl:value-of select='@title'></title>
  - Appel de fonction

```
<xsl:element name='{@title}'>
  <xsl:attribute name='size'>
                <xsl:value-of select='{count(Section)}'/>
  </xsl:attribute>
</xsl:element>
```

  - Méthode mixte
    <title name='{@title}'/>
  - Les { } indique qu'il s'agit d'un chemin XPath qui doit être remplacé par sa valeur.

XSLT fonctions étendues

Itération et tri
Structure conditionnelle
Variables
Modes

Répétition

- Parfois la simple récursivité ne suffit pas
- Une structure de répétition existe **xsl:for-each**
  - Elle permet de parcourir un ensemble de noeuds
- Les instructions sont appliquées successivement à chaque noeud sélectionné

```
<xsl:template match="/">
  <R>
   <xsl:for-each select=".//Titre">
    <H1><xsl:apply-templates/></H1>
   </xsl:for-each>
  </R>
</xsl:template>
```

```
<?xml version="1.0" encoding="utf-8"?>
<R>
  <H1>titre1</H1>
  <H1>titresect1</H1>
  <H1>titre2</H1>
  <H1>titresect2</H1>
</R>
```

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

       <Chapitre>
        <Titre>titre1</Titre>
        <Section>
          <Titre>titresect1</Titre>
        </Section>
       </Chapitre>

       <Chapitre>
        <Titre>titre2 </Titre>
        <Section>
          <Titre>titresect2 </Titre>
        </Section>
       </Chapitre>

     </Texte>
```

- Les nœuds résultats peuvent être triés par **xsl:sort**

Tri

- Les nœuds résultats peuvent être triés par xsl:sort

```
<xsl:template match="/">
  <R>
   <xsl:for-each select=".//Titre">
    <xsl:sort select='.'/>
    <H1><xsl:apply-templates/></H1>
   </xsl:for-each>
  </R>
</xsl:template>
```

```
<?xml version="1.0" encoding="utf-8"?>
<R>
 <H1>titre1</H1>
 <H1>titre2</H1>
 <H1>titresect1</H1>
 <H1>titresect2</H1>
</R>
```

Traitement conditionnel

- xsl:if permet le traitement conditionnel d'un noeud

```
<xsl:template match="list">
  <xsl:for-each select="entry">
    <xsl:value-of select="."/>
    <xsl:if test="not(position()=last())">
      <xsl:text>,   </xsl:text>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

```
<list>
    <entry
    type=«c»>item1</entry>
    <entry
    type=«b»>item2</entry>
    <entry
    type=«a»>item3</entry>
</list>
```

item1,item2,item3

Copie de nœud

- Copie du nœud courant
  - <xsl:copy select='.'/>
- Copie de l'arbre
  - <xsl:copy-of select='.'/>

Variables

- XSLT permet de définir des variables

```
<xsl:variable name='X'>valeur</xsl:variable>
```

- Les variables sont visibles dans toute la sous arborescence
- Un type particulier de variables: les paramètres

```
<xsl:param name='X'/>Default</param>
```

Inclusion de feuilles de style

- Dans un document XML
  - Exprimée dans le document source
  - Avec une instruction de traitement
- Dans un feuille de style
  - <!-- Category: top-level-element -->
    <xsl:include
    href = *uri-reference* />
  - <xsl:import
    href = *uri-reference* />

Utilisation de plusieurs feuilles de style

- Utilisation de deux éléments de haut niveau
  - *<xsl:include href = uri-reference />*
  - *<xsl:import href = uri-reference />*

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

Les modes

- Produire plusieurs résultats à partir d'un nœud

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

    <xsl:template match="/">
        <xsl:apply-templates select="//titre" mode="red"/>
        <xsl:apply-templates select="//titre" mode="blue"/>
        <xsl:apply-templates select="//titre"/>
    </xsl:template>

    <xsl:template match="titre" mode="red">
        <div style="color:red">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>

    <xsl:template match="titre" mode="blue">
        <div style="color:blue">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>

    <xsl:template match="titre">
        <div style="color:purple">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>
</xsl:stylesheet>
```

Templates nommés et Fonctions

- XSL permet de nommer un *template* et de l'appeler explicitement à n'importe quel endroit

```
<xsl:template name = qname>
   …
      <xsl:template

      <xsl:call-template name = qname>
   …
</xsl:call-template>
```

- Cela permet de définir des fonctions

Créer dynamiquement une feuille de style

- **Pb l'espace de nom est déjà utilisé !**
- **Utilisation d'un ALIAS**

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

   <xsl:namespace-alias stylesheet-prefix="axsl" result-prefix="xsl"/>

   <xsl:template match="/">
     <axsl:stylesheet>
       <xsl:apply-templates/>
     </axsl:stylesheet>
   </xsl:template>

   <xsl:template match="block">
     <axsl:template match="{.}">
        <fo:block><axsl:apply-templates/></fo:block>
     </axsl:template>
   </xsl:template>
</xsl:stylesheet>
```

- génèrera une feuille de style XSLT à partir d'un document de la forme :

```
<elements>
    <block>p</block>
    <block>h1</block>
    <block>h2</block>
    <block>h3</block>
    <block>h4</block>
</elements>
```

Passage de paramètres

```
<xsl:template name="ItemApuce">
  <xsl:param name= "puce">- </xsl:param>
  <fo:block>
    <xsl:value-of select="$puce"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="li">
  <xsl:call-template name="ItemApuce">
    <xsl:with-param name="format">* </xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template match="dd">
  <xsl:call-template name="ItemApuce">
    <xsl:with-param name="format"># </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Créer des ensembles d'attributs

- Il est possible de créer des ensembles d'attributs, de les nommer et de les réutiliser

```
<xsl:template match="chapter/heading">
  <fo:block quadding="start"
            xsl:use-attribute-sets="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>
```