Module MO73
Documents électroniques

Manipulation et présentation de documents XML

Contexte

- SGML : origine, complexe, peu utilisé
- HTML cumule le pire de deux mondes
  - Peu d'information structurelle
  - Faible capacité de présentation
- Eléments de formatage mêlés au document
  - Par défaut tous les documents ont la même présentation
- XML permet une bonne structuration, mais sans mise en page
- Comment manipuler et présenter à la fois :
  - Des **données semi structurées** (Ex: des documents)
  - Des **données structurées** (Ex: des tableaux XML)
  - Voire un mélange des deux (Ex: document avec tableau ou tableau avec champs semi structurés)

Objectifs

- Représentation de données avec XML
  - Statique
  - dynamique
    - génération par une base de données ou une application
- Manipulation des données XML
  - Interrogation
  - Transformation
- Présentation des données ou des résultats
  - Par transformation vers un format propriétaire ou non XML
    - HTML, PDF, RTF, …
  - Native (en XML) avec des feuilles de styles vers des langages XML
    - FO (formating object), XHTML, Scalable Vector Graphics (SVG)

Plan du cours

- Localisation de fragments dans un doc. XML
  - *XPath*
- Transformation de documents XML
  - *The Extensible Stylesheet Language (XSLT)*
- *Présentation de document XML et HTML*
  - *Avec une feuille externe : Cascading Style Sheets (CSS)*
  - *Avec un langage XML :The Extensible Stylesheet Language (XSL)*
  - *Description d'images vectorielles dynamiques : Scalable Vector Graphics (SVG)*
- Programmation et XML
  - *Document Object Model (DOM)*
  - *Simple API for XML (SAX)*
- Intégration de XML dans un serveur Web
  - *Apache Cocoon, connexion à un SGBD relationnel*
- *Entrepôts de données XML ?*

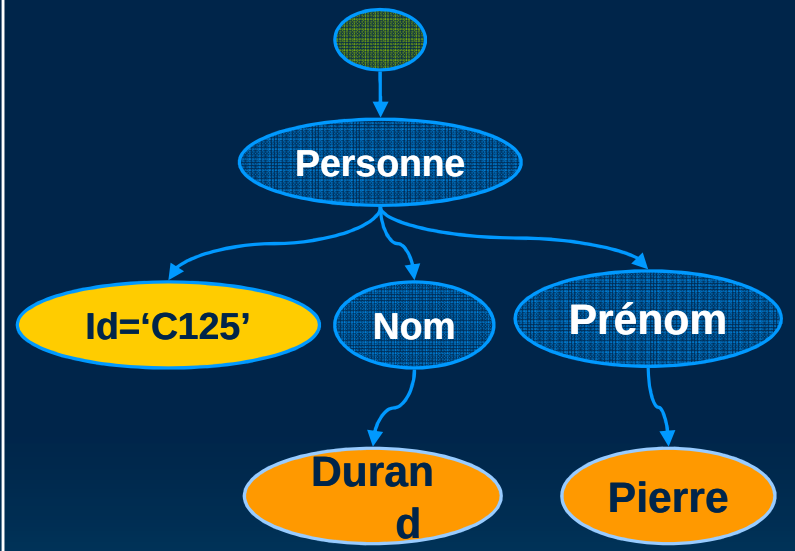
XML Path Language (XPath)

Version 1.0 - W3C Recommendation 16 November 1999

Modèle de document (1)

- Un modèle arborescent peut être associé aux documents XML
- La racine représente le document
- Les éléments, attributs et textes sont représentés par des nœuds.

Un arbre de document




```
<?xml version='1.0'?>
<Personne id='C125'>
    <nom>Durand</nom>
    <prenom>Pierre</prenom>
</Personne>
```

Modèle de document (2)

```
<guide>
...
<itinéraire numéro="6">
 <nom>Col d'Izoard</nom>
 <commentaire>Cette montée pédestre, au haut lieu du cyclotourisme, vous évite les
      lacets de la route et, du fond du vallon du torrent d'Izoard, vous permet d'admirer,
      loin des foules et de la motorisation, les rochers déchiquetés de la Casse Déserte
      (cargneules).</commentaire>
 <cotation>1</cotation>
 <départ>
  <lieu>Brunissard</lieu>
  <altitude>1760</altitude></départ>
 <arrivée>
  <lieu>Col d'Izoard</lieu>
  <altitude>2360</altitude></arrivée>
 <temps unité="heure">2</temps>
 <montée>De <lieu>Brunissard</lieu> remonter (NW) la route de l'Izoard, <lieu>D
      902</lieu>, jusqu'au premier virage de l'entrée des bois ; la quitter, à gauche, pour
      traverser (N) le groupe de <lieu>chalets de La Draye</lieu>. Continuer (N) par le
      sentier balisé <lieu>GR 58</lieu> qui remonte le ravin du torrent du Col d'Izoard.
      Juste avant d'arriver à la base des éboulis de la <lieu>Casse Déserte</lieu>, le
      <lieu>GR 58</lieu> monte à l'Est et notre itinéraire continue (N) dans le thalweg, au
      pied de la <lieu>Casse Déserte</lieu>, puis s'élève dans les pentes Sud du col,
      recoupant la route <lieu>D 902</lieu>, pour aboutir au <lieu>Col d'Izoard</lieu>
      (2360 m). Stèle et Pavillon du Cyclotourisme.</montée>
<descente>Sur <lieu>Brunissard</lieu> par l'itinéraire de montée. Sur Cervières, par le
      <lieu>refuge d'Izoard</lieu> et la route <lieu>D902</lieu>, sur le versant Nord du
      col.</descente>
</itinéraire>
...
</guide>
```

La structure d'un arbre de document

- **Le nœud document** constitue la racine de l'arbre d'un document, il a un fils unique qui est un nœud élément qui représente l'élément de niveau supérieur du document (le premier dans l'ordre de lecture) aussi appelé élément du document
- Un **nœud élément** est étiqueté par le nom de l'élément qu'il représente, il a pour fils les nœuds représentants les attributs, les éléments et les fragments de texte qui le constituent. Parmi ces fils, les nœuds éléments et les nœuds textes sont appelés ses enfants. Ces enfants sont ordonnés selon l'ordre de lecture du document ;
- Un **nœud attribut** est étiqueté par le nom et la valeur de l'attribut qu'il représente. Il a pour père un nœud élément et n'a pas de nœud fils ;
- Un **nœud texte** est étiqueté par le fragment de texte qu'il représente. Il a pour père un nœud élément et n'a pas de nœud fils.

XPath

Localisation de fragments de documents

- Documents arborescents:
  - navigation dans l'arbre du document

Une expression XPath permet de décrire un modèle de chemin dans l'arbre du document

Exemples intuitifs – chemins absolus

- La syntaxe est similaire à l'adressage dans les systèmes de fichiers
- Le / initial indique que le chemin est absolu
- Les noms d'éléments indiquent les pas à suivre
- Un chemin permet de sélectionner un ensemble de nœuds
- Plusieurs utilisations possible : sélection d'un nœud ou d'un valeur, ou d'un sous-arbre.

/livre

```
<livre>
  <chapitre>
    …
  </chapitre>
</livre>
```

/livre/chapitre

```
<livre>
  <chapitre> …</chapitre>
  <chapitre> …</chapitre>
</livre>
```

Exemples intuitifs – Chemins de profondeur arbitraire

- Si le **chemin commence** par //, il est absolu et tous les éléments qui vérifient le critère suivant sont sélectionnés.

//section

```
<livre>
  <chapitre>
    <section>…</section>
  </chapitre>
  <section>…</section>
</livre>
```

//chapitre/section

```
<livre>
  <chapitre>
    <section>…</section>
  </chapitre>
  <section>…</section>
</livre>
```

Exemples intuitifs – Eléments et Attributs

- Le symbole ***** sélectionne n'importe quel **élément**
- Les noms d'attributs sont préfixés avec @

//section/*

```
<livre>
  <chapitre>
    <section num='1'><a/><b/><c/></section>
  </chapitre>
  <section num='2'><a/></section>
</livre>
```

//chapitre/section/@num

```
<livre>
  <chapitre>
    <section num='1'>…</section>
  </chapitre>
  <section num='2'>…</section>
</livre>
```

Exemples intuitifs – Prédicats 1

- Les contraintes que doivent vérifier un nœud sont exprimées entre crochets : **Prédicat**
  - Position : Un nombre indique la position dans **l'ensemble de nœuds sélectionnés**

/livre/chapitre[position()=1] ou /livre/chapitre[1]

```
<livre>
  <chapitre>
    <section num='1'>…</section>
  </chapitre>
 <section num='2'>…</section>
</livre>
```

/livre/section[position()=last()] ou /livre/section[last()]

```
<livre>
  <chapitre>
    <section num='1'>…</section>
  </chapitre>
  <section num='1'>…</section>
  <section num='2'>…</section>
</livre>
```

Exemples intuitifs – Prédicats 2

- Une expression XPath peut être utilisée dans un prédicat pour spécifier **un modèle de contenu**.
- La fonction **count**(chemin) retourne le nombre de nœuds vérifiant ce chemin

/livre/chapitre[section/@num]

```
<livre>
  <chapitre>
    <section num='1'>…</section>
  </chapitre>
  <chapitre>
    <section>…</section>
  </chapitre>
</livre>
```

//section[count(a)=3]

```
<livre>
  <chapitre>
    <section num='1'><a/><a/><a/></section>
  </chapitre>
  <section num='2'><a/></section>
</livre>
```

Exemples intuitifs – Prédicats 3

- Il est possible d'utiliser des expressions booléennes dans les prédicats
- Et d'appliquer **SUCCESSIVEMENT** des prédicats : ATTENTION à l'ordre

/*[section[@num] or chapitre]

```
<livre>
  <chapitre num='1'>
    <section num='1'>…</section>
  </chapitre>
  <chapitre num='2'>
    <section>…</section>
  </chapitre>
  <section num='1'>…</section>
</livre>
```

//chapitre[@num][2]

//chapitre[2][@num]

```
<livre>
 <chapitre>…</chapitre>
 <chapitre num='1'>…</chapitre>
 <chapitre num='2'>…</chapitre>
 <chapitre num='3'>…</chapitre>
 <chapitre></chapitre>
</livre>
```

Exemples intuitifs – Valeurs

- Les valeurs des attributs ou celles contenues dans les éléments peuvent être utilisées.

/livre/chapitre[titre='ab']

```
<livre>
  <chapitre>
    <titre>ab</titre>
    …
  </chapitre>
  <chapitre>
    <titre>cd</titre>
    …
  </chapitre>
</livre>
```

/livre/chapitre[@titre='cd']

```
<livre>
  <chapitre titre='ab'>…
  </chapitre>
  <chapitre titre='cd'>…
  </chapitre>
</livre>
```

Exemples intuitifs – Descendants

- Un axe peut être associé au *test de nœud* pour préciser une direction différente de *fils direct* (*child*)
- L'axe ***descendant*** permet d'atteindre tous les fils des nœuds sélectionnés. Il ne filtre donc que des éléments

/descendant::*

```
<livre>
  <section>….</section>
  <chapitre>
    <section><t/>….</section>
    <section><t/>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

//chapitre/descendant::* ou
/*/descendant-or-self/chapitre/descendant::*

```
<livre>
  <section>….</section>
  <chapitre>
    <section><t/>….</section>
    <section><t/>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

Exemples intuitifs – Navigation verticale ascendante

- XPath permet aussi une navigation verticale ascendante avec les axes *parent* et *ancestor.*

//section/parent::* ou //*[section]

```
<livre>
  <section>….</section>
  <chapitre>
    <section>….</section>
    <section>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

//t/ancestor::* ou //*[]

```
<livre>
  <section>….</section>
  <chapitre>
    <section><t/>….</section>
    <section><t/>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

Exemples intuitifs – Navigation horizontale

- XPath permet aussi une navigation horizontale avec les axes *following* et *preceeding.*
- Cette navigation peut être limitée aux frères

//chapitre/section[2]/preceding::*

```
<livre>
  <section>….</section>
  <chapitre>
    <section>….</section>
    <section>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

//section/preceding-sibling::*

```
<livre>
  <section>….</section>
  <chapitre>
    <section>….</section>
    <section>….</section>
  </chapitre>
  <section>
      <section>….</section>
  </section>
</livre>
```

Définition d'un chemin XPath

- Si D est l'ensemble des nœuds d'un document, et C un ensemble de nœuds de départs (nœud contexte)

- Un pas XPath est de la forme:

  Axe:test de nœud [prédicat$_1$]…[prédicat$_n$]

  Il sélectionne des nœuds de D atteints à partir des nœuds de C

- Un chemin XPath relatif est de la forme:

  Pas$_1$ /Pas$_2$/…/Pas$_n$

  Il sélectionne des nœuds de D atteints en suivant les n pas.

  Le pas i produit le contexte du pas i+1.

- Un chemin XPath absolu est de la forme:

  /chemin_relatif

  C_0 est initialisé avec la racine du document

Tests de noeuds

- Les nœuds peuvent être filtrés selon leur type grâce à des fonctions:
  - element()
  - text()
  - comment()
  - processing-instruction()
  - node() *(ne sélectionne pas les attributs)*
- Pour atteindre les attributs, on utilise un axe : *attribute*

Axes et écriture abrégées

- *child : c'est l'axe par défaut*
- *descendant: peut être abrégé en // (/livre//section)*
- *parent*
- *ancestor*
- *following-sibling*
- *preceding-sibling*
- *following*
- *preceding*
- *attribute : abrégé en préfixant le test de nœud avec @*
- *namespace axis*
- *self : sélectionne le nœud contexte*
- *descendant-or-self*
- *ancestor-or-self*

Fonctions sur les chaînes

- *string* **string**(*object*?)
- *string* **concat**(*string*, *string*, *string**)
- *boolean* **starts-with**(*string*, *string*)
- *boolean* **contains**(*string*, *string*)
- *string* **substring-before**(*string*, *string*)
- *string* **substring-after**(*string*, *string*)
- *string* **substring**(*string*, *number*, *number*?)
- *number* **string-length**(*string*?)
- *string* **normalize-space**(*string*?)

Fonctions hypertextes

- **Changement de document:**
  - document(//a/@href)/title
- Suivi des liens internes (ID et IDREF)
  - Id(//note[1]/@iref)/titre
  - Nécessite la DTD

Exemples 1

- child::para
- child::*
- child::text()
- child::node()
- attribute::name
- attribute::*
- descendant::para
- ancestor::div
- ancestor-or-self::div
- descendant-or-self::para
- self::para
- child::chapter/descendant::para
- child::*/child::para
- /

Exemples 2

- /descendant::para
- /descendant::olist/child::item
- child::para[position()=1]
- child::para[position()=last()]
- child::para[position()=last()-1]
- child::para[position()>1]
- **following-sibling::chapter[position()=1]**
- **preceding-sibling::chapter[position()=1]**
- /descendant::figure[position()=42]
- /child::doc/child::chapter[position()=5]/child::section[position()=2]
- child::para[attribute::type="warning"]
- **child::para[attribute::type='warning'][position()=5]**
- **child::para[position()=5][attribute::type="warning"]**
- child::chapter[child::title='Introduction']
- child::chapter[child::title]
- child::*[self::chapter or self::appendix]
- **child::*[self::chapter or self::appendix][position()=last()]**

Exemple 3

- /guide/itinéraire[départ/lieu = "Brunissard"]
  - sélectionne les itinéraires dont le lieu de départ est Brunissard.
- //itinéraire[@numéro = "6"]/nom/text()
  - sélectionne le nom de l'itinéraire numéro 6.
- /child::guide/child ::*/child::lieu/child ::text()
  - a pour valeur la séquence des lieux qu'ils soient de départ ou d'arrivée.
- /descendant-or-self::node()/child::itinéraire[cotation = "1"]/attribute::numéro
  - a pour valeur la séquence des numéros d'itinéraires concernant des randonnées faciles, c'est-à-dire dont la cotation est égale à 1.
- //itinéraire[temps < 3][position() <= 10]
  - a pour valeur la séquence des 10 premiers itinéraires du guide dont le temps de montée est inférieur à 3 heures.
- //itinéraire[contains(montée, "raide") and contains(montée, "éboulis")]/commentaire
  - a pour valeur l'ensemble des commentaires des itinéraires dont une partie de la montée est raide ou se déroule sur des éboulis.
- //itinéraire[.//lieu[text() = "GR 51"]]
  - a pour valeur la séquence des itinéraires dont la description mentionne le GR 51, et donc contient un élément lieu dont la valeur textuelle est "GR 51".

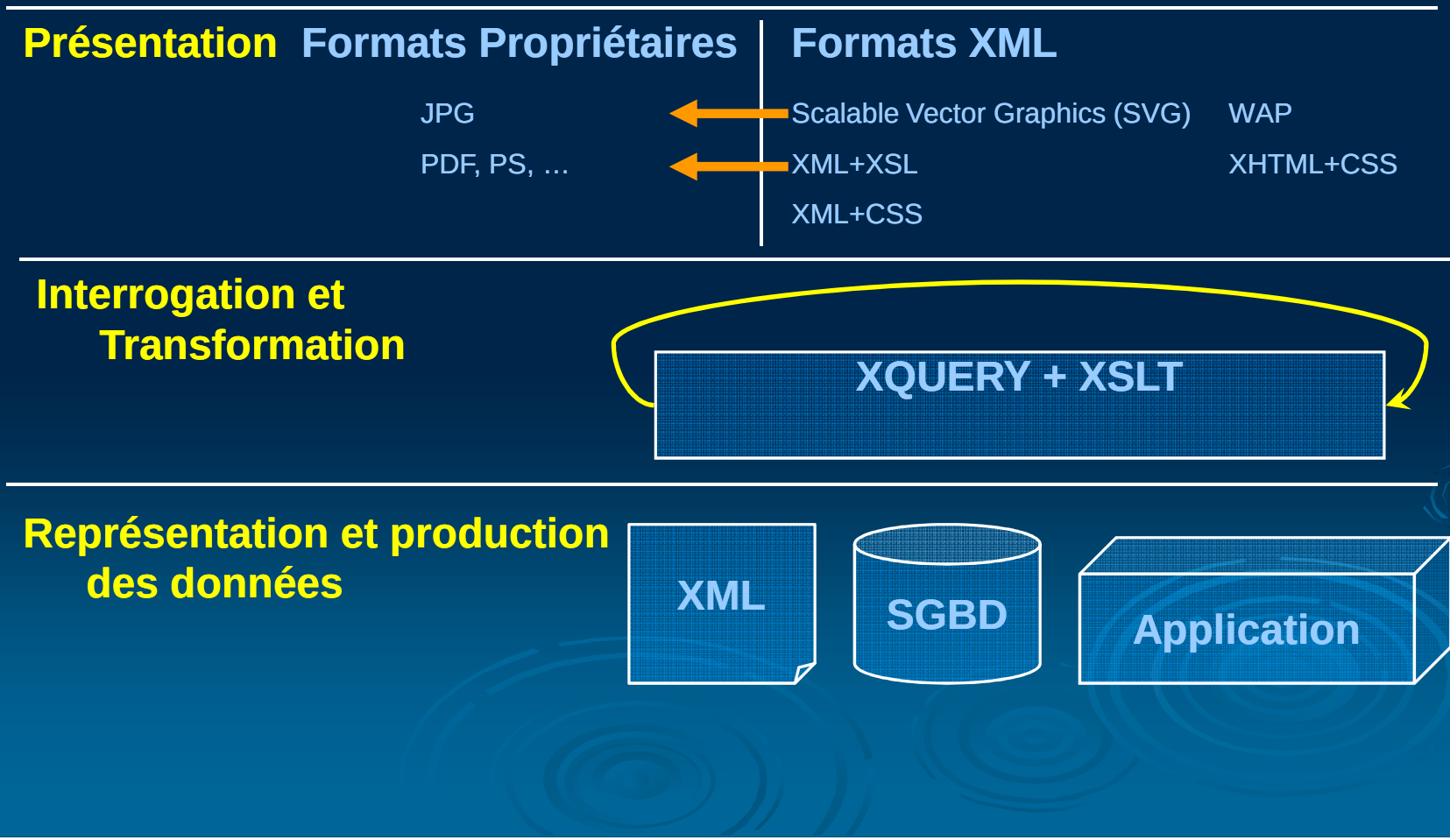
Transformation et présentation de documents XML

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

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
  - **xmlns:xsl** définit xsl comme le préfixe (**namespace**) des balises appartenant au langage XSLT

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

Template

- 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
    - Fragment de documents xml + balises prédéfinies

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

```
<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>
```

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

```
<?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>
```

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>
```

Les feuilles de style (CSS2)

Cascading Style Sheets, level 2
CSS2 Specification
W3C Recommendation *12-May-1998*

http://www.w3.org/tr/css2

Qu'est-ce que CSS?

- Puissant langage de formatage de document
  - Visuel (navigateur, impression)
  - mais aussi audio
- Attachement de propriétés de style aux éléments d'un document
- Utilisable dans un document ou par référence externe
- Syntaxe propriétaire

Exemple de CSS

- En HTML sans CSS
  - ```
    <EM>
      <FONT face="Arial" color="green">
        <H3>This is a green, italic, Arial H3 header.</H3>
      </FONT>
    </EM>
    ```
- En HTML avec CSS
  - ```
    h4 { font-family: Arial;
      font-style: italic;
      color: green }

    ... <h4>This is a green, italic, Arial H4 header</h4>
    ```

Que peut-on faire avec CSS ?

- Sélectionner un ou des éléments et modifier
  - les couleurs du texte et du fond
  - les polices de caractères
  - les propriétés du texte (espacement...)
- Créer des boîtes pour manipuler
  - Les marges
  - Les blocs
  - Les objets flottants

Feuille de style interne avec HTML

- La feuille de style peut être incluse dans le document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>

  <HEAD>
    <TITLE>Une page HTML</TITLE>
    <STYLE type="text/css">
      H1 { color: blue;font-size: 120% }
       P { font-family : fantasy;}
    </STYLE>
  </HEAD>

  <BODY>
   <H1>Titre</H1>
   <P>Et voilà du texte.
  </BODY>

</HTML>
```

Feuille de style externe avec HTML

- La feuille de style peut être référencée par le document (HTML)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
 <HEAD>
  <TITLE>Une page HTML</TITLE>
  <LINK rel="stylesheet"
        href="monstyle.css" type="text/css">
 </HEAD>

 <BODY>
  <H1>Titre</H1>
  <P>Et voilà du texte.</P>
 </BODY>

 </HTML>
```

L'élément LINK contient :

Le type de lien : vers une feuille de style (REL)

L'adresse de la feuille de style (HREF)

Le type de feuille de style (type)

CSS et XML

- CSS peut aussi être utilisé avec XML
- Pour définir les blocs

```
Personne { display: inline }
Doc, Titre, Para { display: block }
```

Mon document

Bonjour Mr Dupont, ...

```
<?xml version='1.0'?>
<?xml-stylesheet href="monstyle.css" type="text/css"?>
 <Doc>
  <Titre>Mon document</Titre>
  <Para>Bonjour </Personne>Mr Dupont</Personne>,…</Para>
</Doc>
```

Associer une présentation aux éléments

- Une feuille de style est composée de règles
- Chaque règle est composée d' :
  - Un sélecteur
    - Le plus simple : un nom d'élément HTML (Ex: H1)
  - Une déclaration : un ou plusieurs couples (<propriété : valeur>)
    - Ex: **color : blue**
    - Quelques propriétés seront présentées plus tard

```
H1 { color : blue }
```

Héritage de propriétés

- Beaucoup de propriétés associées à un élément sont héritées par les éléments contenus dans celui-ci

```
H1 {color : red }
EM {text-decoration : underline}
```

```
<H1 STYLE="color:red">
Voila mon <EM STYLE="text-decoration :underline">Titre</EM>
</H1>
```

Eléments et contexte

- Distinguer les éléments selon le contexte
- Mécanisme simple
- Filtrer selon l'imbrication des éléments
- Si le sélecteur est composé d'une liste d'élément, la règle ne s'applique qu'aux éléments qui possèdent tous les parents listés

```
DIV BLOCKQUOTE EM {color : blue}
<DIV>
  <P> Et maintenant une citation :
    <BLOCKQUOTE> <EM>Bonjour </EM> à tous !</BLOCKQUOTE>
  </P>
</DIV>

<EM>Et pas là</EM>
```

- On peut imposer une filiation directe avec le symbole '>' (Ex: DIV > BLOCKQUOTE)

Sous-classer les éléments

- Distinguer des sous-classes d'éléments
- Utiliser l'attribut HTML CLASS pour assigner un rôle à un élément

```
P {color : blue}
P.Attention {background : red}

<PCLASS="Attention">Avertissement</P>
<P>Paragraphe quelquonque</P>
```

Sélection par IDentificateur

- Affecter un formatage à UN élément précis
- Utiliser l'attribut ID

```
#special {color: red}

<P ID="special">Un cas à part</ID>
<P>Un paragraphe quelquonque</P>
```

Les pseudo-classes

- Aspects de la présentation indépendants de la structure du document
- Première ligne d'un document
- Liens visités
- ...

```
A:visited {bgcolor : yellow}
P.initial:first-letter {font-size:200%}
```

Résoudre les conflits : les cascades

- A la fois l'auteur du document et le lecteur peuvent spécifier un style
- Possibilité de conflits
- Il est nécessaire de savoir comment ces conflits sont résolus
- CSS fournit un mécanisme de priorité : le style qui a la plus grande priorité gagne :
  - Un style peut être "! important" ou "! normal"
  - En cas d'égalité le style de l'auteur gagne

```
BODY { color: black !important;
background: white !important; }
```

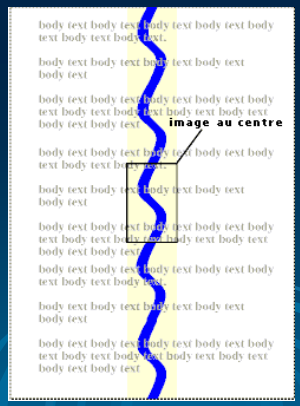
Couleurs et Fontes

- Deux propriétés pour spécifier les couleurs : **color et background**
- Cinq propriétés pour contrôler les fontes :
  - font-family : {serif, sans-serif,monospace,cursive,fantasy}
  - font-style : {normal, italic, oblique}
  - font-variant : {normal, small-caps}
  - font-size : taille de la fonte spécifiée de manière absolue ou relative à la taille courante
    - font-size : 12pt;
    - font-size: 110%;
    - font-size: .3 em;
  - font-weight : Le poids de la fonte
    - bold ou bolder
    - 100,200,....,900

Background

- ```
  BODY { background-image: url("logo.png");
   background-attachment: fixed;
   background-position: 100% 100%;
   background-repeat: no-repeat; }
  ```

- ```
  BODY { background: white url("pendant.gif");
   background-repeat: repeat-y;
   background-position: center; }
  ```



Formater du texte

- word-spacing : espacement entre les mots
- letter-spacing :espacement entre les lettres
- text-decoration : {none, underline, overline, blink}
- vertical-align : ajuster l'alignement vertical d'un élément
- text-transform : {uppercase, lowercase}
- text-align : {left, right, center, justified}
- text-indent :fixe l'indentation de la première ligne d'un bloc de texte
- line-height : distance ente les bases des lignes de textes consécutives

Un exemple complet

```
<HTML>

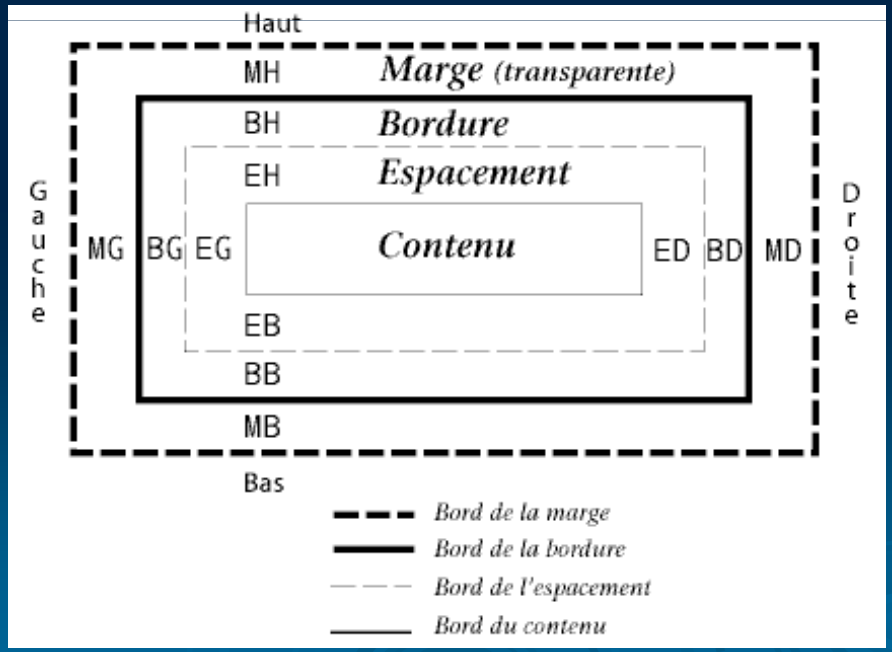
  <HEAD>
    <STYLE type="text/css">
      BODY { background : blue;
             color : white;
             font-family : times, serif;
             font-size : 10pt }
      A:link { color : red }
      A:visited { color : yellow }
      H1 {font-family : arial,helvetica, sans-serif;
          font-size : 20pt;
          font-style : bold }
      DIV.warning { margin-left : 0.5in;
                    margin-right : 0.5in;
                    color : yellow;
                    background : red }
      BLOCKQUOTE { background : white;
                   color : black;
                   font-size : 8pt;
                   font-style : italic }
      UL {list-style :url(bullet.gif) disc }
    </STYLE>
  </HEAD>

  <BODY>
    <H1>Le titre du document</H1>
    Du texte dans le corps du document <BLOCKQUOTE>Une citation </BLOCKQUOTE>
    <DIV CLASS="warning" >Attention : une liste va suivre</DIV>
    <UL>
      <LI>Item 1</LI>
      <LI>Item1</LI>
    </UL>
  </BODY>

</HTML>
```

Des boîtes imbriquées

- La présentation CSS est basée sur des rectangles concentriques correspondant aux éléments
- Chaque élément a une marge, une bordure et une espacement intérieur (padding)



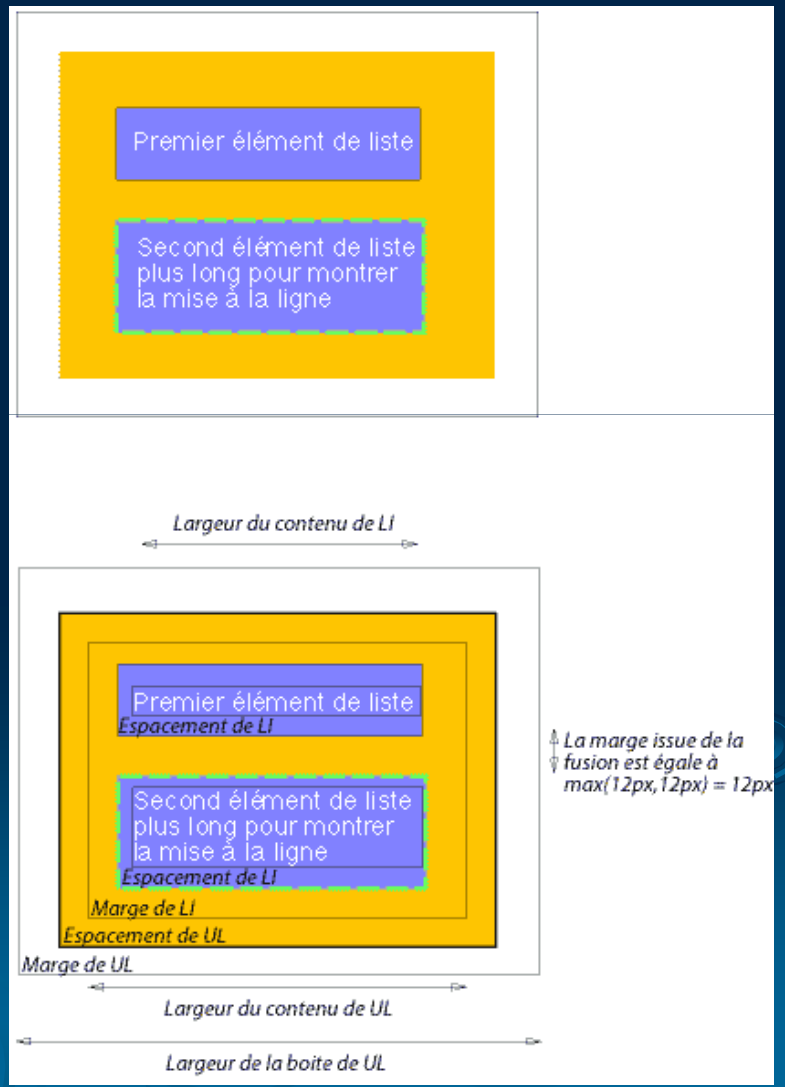
Exemple de boîtes

```
<HTML>
 <HEAD>
  <TITLE>Exemples de boîtes</TITLE></HEAD>

  <STYLE type="text/css">
  UL { background: #ff9933; /* orange */
   margin: 12px 12px 12px 12px;
   padding: 3px 3px 3px 3px;}

  LI { color: white;
   background: #3366cc; /* bleu */
   margin: 12px 12px 12px 12px;
   padding: 12px 0px 12px 12px;
   list-style: none }

  LI.withborder { border-style: dashed;
   border-width: medium;
   border-color: green; }
  </STYLE>
 </HEAD>
 <BODY>
  <UL>
   <LI>Premier élément de liste
   <LI class="withborder">
   Second élément de liste plus long
   pour montrer la mise à la ligne. </UL>
 </BODY>
</HTML>
```



Objets flottants

```
<HTML>
 <HEAD>
 <TITLE>Un document avec des cadres en CSS2</TITLE>

 <STYLE type="text/css">
 BODY { height: 8.5in } /* Nécessaire pour les hauteurs en pourcentage plus bas */

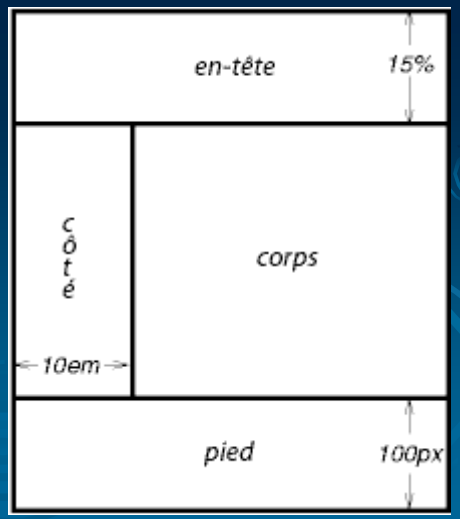
 #entete { position: fixed; width: 100%; height: 15%; top: 0; right: 0; bottom:
auto; left: 0; }

 #cote { position: fixed; width: 10em; height: auto; top: 15%; right: auto; bottom:
100px; left: 0; }

 #corps { position: fixed; width: auto; height: auto; top: 15%; right: 0; bottom:
100px; left: 10em; }

 #pied { position: fixed; width: 100%; height: 100px; top: auto; right: 0; bottom:
0; left: 0; }
 </STYLE>

 </HEAD>
 <BODY>
  <DIV id="entete"> ... </DIV>
  <DIV id="cote"> ... </DIV>
  <DIV id="corps"> ... </DIV>
  <DIV id="pied"> ... </DIV>
 </BODY>
</HTML>
```



Autres média ?

- **all**
  - convient pour tous les appareils ;
- **aural**
  - destiné aux synthétiseurs de parole.;
- **braille**
  - destiné aux appareils braille à retour tactile ;
- **embossed**
  - destiné aux appareils à impression braille ;
- **handheld**
  - destiné aux appareils portatifs (typiquement ceux avec petits écrans, monochromes et à bande passante limitée) ;
- **print**
  - destiné à un support paginé opaque et aux documents vus sur écran en mode aperçu avant impression.;
- **projection**
  - destiné aux présentations en projection, par exemple avec des projecteurs ou des impressions pour des transparents ;
- **screen**
  - destiné principalement aux moniteurs couleurs ;
- **tty**
  - destiné aux médias utilisant une grille de caractères fixe, tels les télétypes, les terminaux ou les appareils portatifs aux capacités d'affichage réduites. Les auteurs ne devraient pas utiliser de valeurs exprimées en pixel avec ce type de média ;
- **tv**
  - destiné aux appareils du type télévision (avec ces caractéristiques : basse résolution, couleur, défilement des pages limité, sonorisé).

```
@media screen { BODY { font-size: 12pt } }
@media print  { BODY { font-size: 10pt } }
```

Importer d’autres feuilles CSS

- `@import "mystyle.css";`
- `@import url("mystyle.css");`
- `@import url("fineprint.css") print;`
-
```
@media screen { BODY { font-size: 12pt } }
@media print {
    @import "print-main.css";
    BODY { font-size: 10pt }
}
```

Conclusion sur CSS

- CSS est un bon langage de mise en page
- Il commence à être bien supporté par les navigateurs
- CSS2 offre des fonctionnalités de manipulations "simples"
- La gestion des tableaux est normalisée

CSS pour HTML 4.0

```
ADDRESS, BLOCKQUOTE, BODY, DD, DIV, DL, DT, FIELDSET, FORM,
FRAME, FRAMESET, H1, H2, H3, H4, H5, H6, IFRAME, NOFRAMES, OL,
P, UL, CENTER, DIR, HR, MENU, PRE { display: block }

OBJECT, APPLET { display: inline }

 LI { display: list-item }

HEAD { display: none }

TABLE { display: table }
TR { display: table-row }
THEAD { display: table-header-group }
TBODY { display: table-row-group }
TFOOT { display: table-footer-group }
COL { display: table-column }
COLGROUP { display: table-column-group }
TD, TH { display: table-cell }
CAPTION { display: table-caption }
TH { font-weight: bolder; text-align: center }
CAPTION { text-align: center }

BODY { padding: 8px; line-height: 1.12em }

H1 { font-size: 2em; margin: .67em 0 }
H2 { font-size: 1.5em; margin: .83em 0 }
H3 { font-size: 1.17em; margin: 1em 0 }
H4, P, BLOCKQUOTE, UL, FIELDSET, FORM, OL, DL, DIR, MENU { margin:
1.33em 0 }
H5 { font-size: .83em; line-height: 1.17em; margin: 1.67em 0 }
H6 { font-size: .67em; margin: 2.33em 0 }
H1, H2, H3, H4, H5, H6, B, STRONG { font-weight: bolder }

BLOCKQUOTE { margin-left: 40px; margin-right: 40px }
I, CITE, EM, VAR, ADDRESS { font-style: italic }
PRE, TT, CODE, KBD, SAMP { font-family: monospace }
PRE { white-space: pre }
BIG { font-size: 1.17em }
SMALL, SUB, SUP { font-size: .83em }
SUB { vertical-align: sub }
SUP { vertical-align: super }
S, STRIKE, DEL { text-decoration: line-through }

HR { border: 1px inset }


OL, UL, DIR, MENU, DD { margin-left: 40px }
OL { list-style-type: decimal }
OL UL, UL OL, UL UL, OL OL { margin-top: 0; margin-bottom: 0 }
U, INS { text-decoration: underline }

CENTER { text-align: center }
BR:before { content: "\A" }

ABBR, ACRONYM { font-variant: small-caps; letter-spacing: 0.1em }

A[href] { text-decoration: underline }
:focus { outline: thin dotted invert }

BDO[DIR="ltr"] { direction: ltr; unicode-bidi: bidi-override }
BDO[DIR="rtl"] { direction: rtl; unicode-bidi: bidi-override }
*[DIR="ltr"] { direction: ltr; unicode-bidi: embed }
*[DIR="rtl"] { direction: rtl; unicode-bidi: embed }

ADDRESS, BLOCKQUOTE, BODY, DD, DIV, DL, DT, FIELDSET, FORM,
FRAME, FRAMESET, H1, H2, H3, H4, H5, H6, IFRAME, NOSCRIPT,
NOFRAMES, OBJECT, OL, P, UL, APPLET, CENTER, DIR, HR, MENU,
PRE, LI, TABLE, TR, THEAD, TBODY, TFOOT, COL, COLGROUP, TD, TH,
CAPTION { unicode-bidi: embed }
/* Fin des réglages bidi */

@page { margin: 10% }

@media print {
    H1, H2, H3, H4, H5, H6 { page-break-after: avoid; page-break-inside: avoid
}
    BLOCKQUOTE, PRE { page-break-inside: avoid }
    UL, OL, DL { page-break-before: avoid } }

@media aural {
    H1, H2, H3, H4, H5, H6 { voice-family: paul, male; stress: 20; richness: 90 }
    H1 { pitch: x-low; pitch-range: 90 }
    H2 { pitch: x-low; pitch-range: 80 }
    H3 { pitch: low; pitch-range: 70 }
    H4 { pitch: medium; pitch-range: 60 }
    H5 { pitch: medium; pitch-range: 50 }
    H6 { pitch: medium; pitch-range: 40 }
    LI, DT, DD { pitch: medium; richness: 60 }
    DT { stress: 80 }
    PRE, CODE, TT { pitch: medium; pitch-range: 0; stress: 0; richness: 80 }
    EM { pitch: medium; pitch-range: 60; stress: 60; richness: 50 }
    STRONG { pitch: medium; pitch-range: 60; stress: 90; richness: 90 }
    DFN  { pitch: high; pitch-range: 60; stress: 60 } S, STRIKE { richness: 0 }
    I { pitch: medium; pitch-range: 60; stress: 60; richness: 50 }
    B { pitch: medium; pitch-range: 60; stress: 90; richness: 90 }
    U { richness: 0 }
    A:link { voice-family: harry, male }
    A:visited { voice-family: betty, female }
    A:active { voice-family: betty, female; pitch-range: 80; pitch: x-high } }
```

Les feuilles de style XSL

Extensible Stylesheet Language (XSL)
Version 1.0
W3C Recommendation 15 October 2001

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

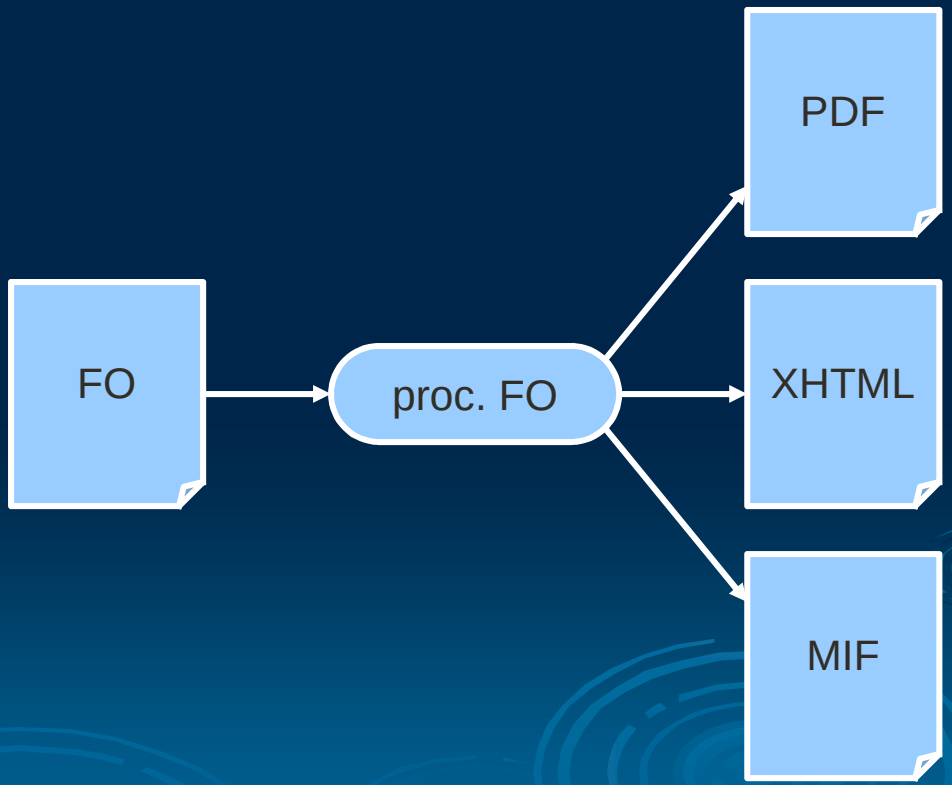
Introduction

- XSL-FO (Formatting Object) est un langage de description de page utilisant la syntaxe XML
- objets de formatage dont l'espace de noms est `xmlns:fo="http://www.w3.org/1999/XSL/Format"`
- Pour restituer le document on utilise un processeur (FOP)

```
...

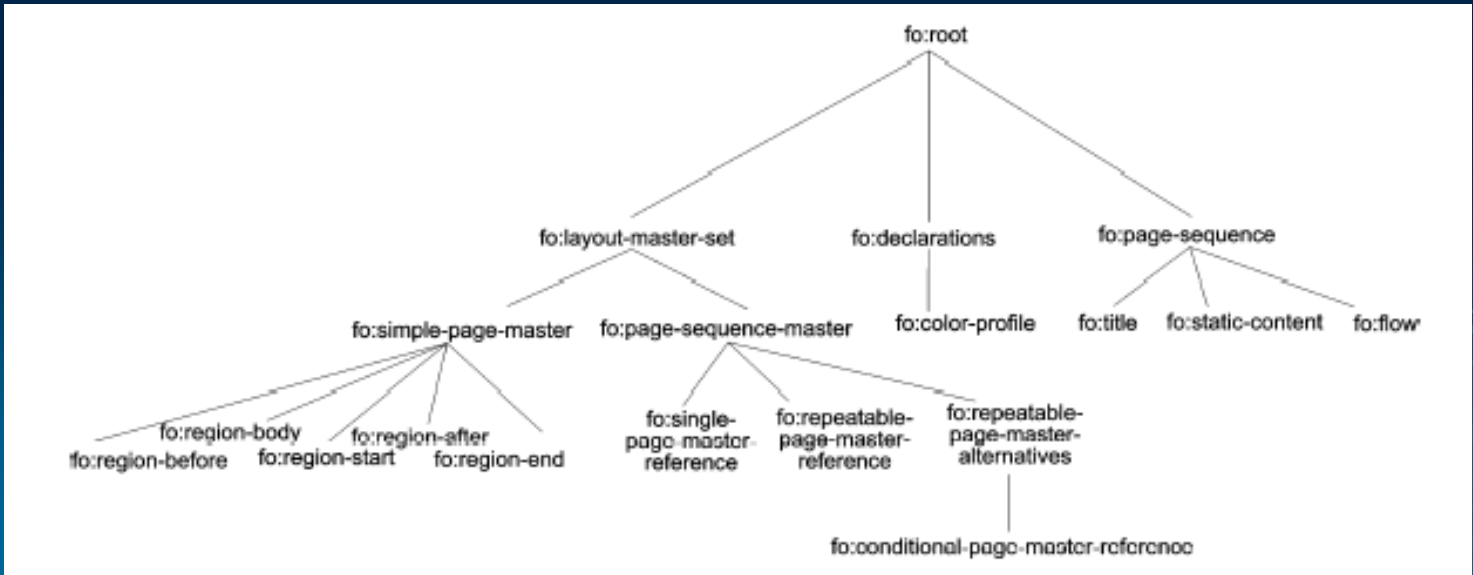
<fo:block
  font-size="18pt"
  font-family="sans-serif"
  line-height="24pt"
  space-after.optimum="15pt"
  background-color="blue"
  color="white"
  text-align="center"
  padding-top="3pt">
Extensible Markup Language (XML)1.0
</fo:block>


...
```



Structure globale d'un document FO

- Un document FO est formé d'un élément `fo:root` qui comprend 2 parties bien distinctes
  - une description des modèles de pages (`fo:layout-master-set`)
  - une description du contenu (`fo:page-sequence`)



Pour plus de détails

Tutoriel (Render X):

http://www.renderx.net/Content/support/xep/tutorial.html

Le langage SVG
Scalable Vector Graphics

http://www.w3.org/TR/SVG/

Le langage SVG
Scalable Vector Graphics

- **Contexte:**
  - Sites de plus en plus complexes donc temps de chargement plus long.
  - Il faut réduire la taille des fichiers.
- Solution:
  - Utilisation de l'image vectorielle (si l'on veut créer une image bitmap de n'importe quelle taille et de bonne qualité le fichier sera lourd)
  - Ajout de caractéristiques dynamiques
- **Proposition dans l'univers XML : SVG**
  - SVG est un langage de description de graphiques 2D en XML
  - Norme graphique vectorielle basée sur XML définie par le consortium W3C.
  - **Vectoriel :** Différence image raster / image vectorielles (**Scalable = zoomable**)
    - Possibilité de zoomer avec la même qualité d'image exemple carte, .
  - Permet un grand nombre d'utilisateurs et un grand nombre d'utilisations.
- **Environnement XML**
  - Il est possible d'utiliser tous les outils XML (parser, outils de transformation, bases de données)
  - La conformité aux espaces de nommage permet de mélanger des grammaires XML entre elles (un document HTML peut contenir des graphiques SVG, des expressions mathématiques en MathML, des présentations en SMIL,...)
- Développé par le W3C : Ce groupe rassemble des représentants de Microsoft, Autodesk, Adobe, IBM, Sun, Netscape, Xerox, Apple, Corel, HP,ILOG entre autres

Idée générale

- Manipuler trois types d'objets graphiques:
  - ***formes*** vectorielles
  - images
  - texte
- C’est-à-dire pouvoir les
  - grouper
  - transformer
  - composer dans d'autres objets
- Leur ajouter des attributs de style
- Les rendre interactifs et dynamiques
- SVG est relié aux standards suivants du W3C : XML, XML-NS, XLINK, Xpointer, XSL-T, CSS, DOM, SMIL, HTML , XHTML

Domaine d'application

- Création de sites internet
- Illustrations techniques générées dynamiquement
- Visualisation de données scientifiques
- Multimédia et divertissement
- Cartographie et SIG en ligne ainsi que leurs services
- Les services localisés (LBS), SVG Mobile
- Catalogue en ligne et E-commerce

Les outils

- Validation du code http://validator.w3.org/
- Affichage (plugin adobe)
  - IE 5/6, NS 4.x et Mozilla **<0.9.0** (> branche spéciale)
- Pensez à utiliser le menu à disposition (clic droit de la souris pour voir le source, zoomer, …)
- Editeurs:
  - XML Spy, Emacs+PSGML+TDTD, Jasc WebDraw, XML Writer, Mayura Draw

Objets de base

- Types des données de base
  - Angle : un entier suivi de *deg* (degrés), *grad* (grades), *rad* (radians)
  - Couleur : spécifiée dans le modéle RGB
    comme en HTML : *#rrggbb* ou un mot-clé dans la liste :
    aqua,black,fuchsia,gray,green,lime,maroon,navy,olive,purple,red ,silver,teal,white,yellow
  - Entier : entre - 2147483648 et 2147483647
  - Réel : soit en notation décimale soit en notation scientifique
    - (x.yyye(ou E)±**nn) entre** - 3.4e+38 et 3.4e+38
  - Longueur : nombre suivi d'un identifiant CSS (*mm,cm,m*)
  - Temps : nombre suivi de *ms* (millisecondes) ou de *s* (secondes)
  - URI : Uniform Ressource Identifier

Structure d'un fichier SVG

- **Type MIME : image/svg**
- **Déclaration XML**
  <?xml version="1.0" standalone="no"?>
  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
- **L'élément <svg>**
  - Un document SVG se compose d'un (ou plusieurs) fragment(s) délimités par la balise **<svg>**
  - Il peut y avoir plusieurs structures <svg> emboîtées dans le même document ou dans des documents composites résultants de plusieurs espaces de noms
  - La balise <svg> redéfinit l'espace utilisateur
  - *Attributs principaux de la balise <svg>*
    - x = "x0" ; position en x du coin supérieur gauche (pour les structures internes)
    - y = "y0" ; position en y du coin supérieur gauche (pour les structures internes)
    - width = "w0" ; largeur en pixels de l'espace (pour les structures externes)
    - height = "h0" ; hauteur en pixels de l'espace (pour les structures externes)
    - viewbox
    - style

Définir un style

- Peut s'appliquer :
  - soit par CSS2 (Cascadind Style Sheet Level 2)
  - soit par XSL (XML Style Language)
  - soit par l'attribut *style* commun à beaucoup de balises SVG
    - Exemple d'emploi avec une balise *text*
      - <text style="font-size: 12pt> Texte en 12 pt</text>
- Il est recommandé de ne pas mélanger les styles CSS et XSL dans le même document SVG

Grouper des éléments

- **La balise <g>**
- Regroupe et nomme des éléments qui partagent des attributs communs : couleur, style, ...
- *Exemple*

```
<g style="fill:red" id="Grands rectangles rouges">
  <rect x="100" y="100" width="200" height="200” />
  <rect x="300" y="400" width="100" height="100” />
</g>
  <g style="fill:blue" id="Petits rectangles bleus">
  <rect x="10" y="10" width="20" height="20” />
  <rect x="30" y="40" width="10" height="10” />
</g>
```

- **On peut imbriquer autant de structures <g> que l'on veut :**
  - **Les objets isolés (ne figurant pas dans une structure <g>) sont considérés comme figurant dans leur propre groupe (leurs attributs ne sont pas propagés)**

Référencement

- On peut utiliser plusieurs types de référencement
- **Référencement relatif**
  - La référence est locale
  - *Exemple*

  <linearGradient id="myGradient"> .... </linearGradient>

  ...

  <rect style="fill:url(#myGradient) />
- **Référencement absolu**
  - La référence est une URI générale
  - La référence ne se trouve pas dans le fichier courant

La balise <defs>

- Elle autorise la définition d'objets référencés **plus tard** dans le même fichier

- *Exemple* :

```
<defs>
 <linearGradient id="Gradient01"> .... </linearGradient >
</defs>
...
<rect style="fill:url(#Gradient01) ..../>
```

- Il est requis que toutes les définitions d'objets devant être référencés plus tard soient faites dans la même structure <defs>
- Il n'y a donc **qu'une structure <defs> par fichier SVG**

La balise <symbol>

- Elle permet de définir des objets graphiques réutilisables dans les cas suivants
  - Objet à instancier de multiples fois
  - Objet classique référencé par de nombreux éléments
  - Définition d'un élément de police textuel
  - ...

La balise <use>

- Référencement d'éléments définis dans une structure <defs > (forme d'inclusion)

- *Exemple*

```
<defs>
  <symbol id="s1" >
  ......
  </symbol>
</defs>
<g >
 <use xlink:href="#s1" />
</g>
```

- L'élément use peut référencer :
  - soit un élément du même fichier dont l'ancêtre est un élément <defs>
  - soit un élément d'un autre fichier dont l'ancêtre est un élément <defs>

Image

- **La balise <image>** autorise le référencement d'images entières dans une zone rectangulaire définie dans les coordonnées utilisateur
- Les formats d'images acceptés doivent être :
  - PNG
  - JPEG
  - SVG
- *Exemple :*

```
<image x="200" y="300" width="100px"
     height="100px" xlink:href="monimage.png"
/>
```

Structures des métadonnées

- Ces balises sont importantes car utilisées par les moteurs de recherche

- **La structure <desc>** autorise l'insertion de commentaires non rendus
- **La structure <title>** autorise un titre pouvant être rendu par les viewers, dans le bandeau par exemple
- *Exemple* :

```
<g>
 <title> Mon image </title>
 <desc> Cette image ne contient qu'un rectangle </desc>
 <rect ..... />
</g>
```

Traitement conditionnel

- Il est assuré comme dans SMIL par la balise switch (même principe que dans SMIL)
- Attributs :
  - xml:lang
  - system-required = liste de fonctions
    - Définit les fonctionnalités minimales devant être implémentées par exemple
      - SVGLang : toutes les fonctionnalités de la spécification
        SVGStatic : un sous-ensemble
        SVGDOMStatic : + les interfaces DOM de SVGStatic
        SVGDOMDynamic , ...
  - system-language = liste de langues
    - idem SMIL

Le système de coordonnées

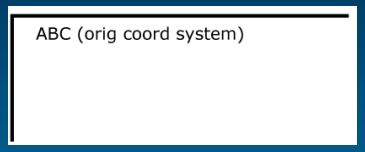
- A l'origine l'élément <svg> le plus externe établit un espace utilisateur et un point de vue confondus comme suit :

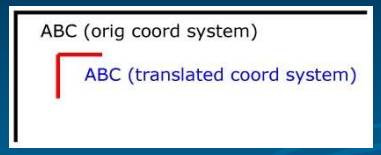
```
<?xml version="1.0" standalone="no"?>
    <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December
    1999//EN"
      "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
    <svg width="300px" height="100px">
      <desc>Example InitialCoords - SVG's initial coordinate
    system</desc>   <g style="fill:none; stroke:black; stroke-
    width:3">
        <line x1="0" y1="1.5" x2="300" y2="1.5" />
        <line x1="1.5" y1="0" x2="1.5" y2="100" />
      </g>
      <g style="fill:red; stroke:none">
        <rect x="0" y="0" width="3" height="3" />
        <rect x="297" y="0" width="3" height="3" />
        <rect x="0" y="97" width="3" height="3" />
      </g>
      <g style="font-size:14 font-family:Verdana">
        <text x="10" y="20">(0,0)</text>
        <text x="240" y="20">(300,0)</text>
        <text x="10" y="90">(0,100)</text>
      </g>
    </svg>
```

- Chaque élément <svg> interne redéfinit un nouvel espace utilisateur et un nouveau point de vue associé

Transformation et coordonnées

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="150px">
  <desc>Example xform1Orig - Simple transformations: original picture</desc>
  <g style="fill:none; stroke:black; stroke-width:3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" style="font-size:20 font-family:Verdana">
      ABC (orig coord system)
    </text>
  </g>
</svg>
```

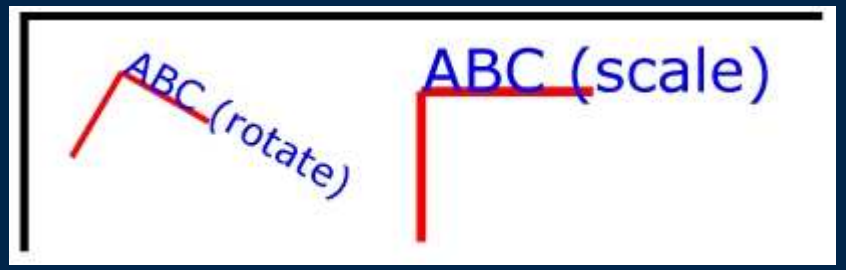





```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"  "http://www.w3.org/Graphics/SVG/SVG 19991203.dtd">
<svg width="400px" height="150px">
  <desc>Example xform1NewCoordSys - New user coordinate system</desc>
  <g style="fill:none; stroke:black; stroke-width:3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" style="font-size:20 font-family:Verdana">
      ABC (orig coord system)
    </text>
  </g> <!-- Establish a new coordinate system, which is shifted (i.e., translated) from the initial coordinate system by 50 user units along each axis. -->
  <g transform="translate(50,50)">
    <g style="fill:none; stroke:red; stroke-width:3"> <!-- Draw lines of length 50 user units along  the axes of the new coordinate system -->
      <line x1="0" y1="0" x2="50" y2="0" style="stroke:red"/>
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="30" y="30" style="font-size:20 font-family:Verdana">
      ABC (translated coord system)
    </text>
  </g>
</svg>
```

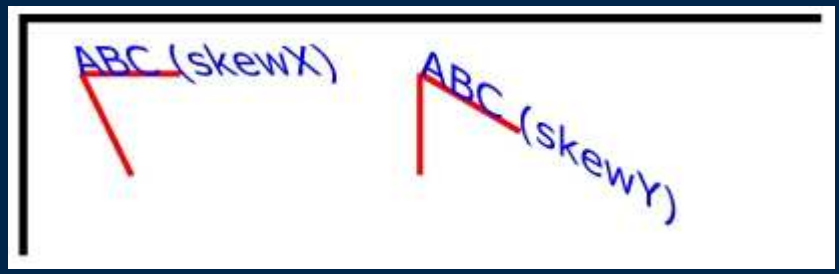
Rotation et mise à l'échelle

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="120px">
  <desc>Example RotateScale - Rotate and scale transforms</desc>
  <g style="fill:none; stroke:black; stroke-width:3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Establish a new coordinate system whose origin is at (50,30)
       in the initial coord. system and which is rotated by 30 degrees. -->
  <g transform="translate(50,30)">
    <g transform="rotate(30)">
      <g style="fill:none; stroke:red; stroke-width:3">
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" style="font-size:20; font-family:Verdana; fill:blue">
        ABC (rotate)
      </text>
    </g>
  </g>
  <!-- Establish a new coordinate system whose origin is at (200,40)
       in the initial coord. system and which is scaled by 1.5. -->
  <g transform="translate(200,40)">
    <g transform="scale(1.5)">
      <g style="fill:none; stroke:red; stroke-width:3">
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" style="font-size:20; font-family:Verdana; fill:blue">
        ABC (scale)
      </text>
    </g>
  </g>
</svg>
```



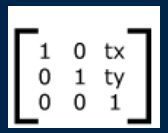
Inclinaison

```
<?xml version="1.0" standalone="no"?>
        <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
          "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
        <svg width="400px" height="120px">
          <desc>Example Skew - Show effects of skewX and skewY</desc>
          <g style="fill:none; stroke:black; stroke-width:3">
            <!-- Draw the axes of the original coordinate system -->
            <line x1="0" y1="1.5" x2="400" y2="1.5" />
            <line x1="1.5" y1="0" x2="1.5" y2="120" />
          </g>
          <!-- Establish a new coordinate system whose origin is at (30,30)
               in the initial coord. system and which is skewed in X by 30 degrees.
        -->
          <g transform="translate(30,30)">
            <g transform="skewX(30)">
              <g style="fill:none; stroke:red; stroke-width:3">
                <line x1="0" y1="0" x2="50" y2="0" />
                <line x1="0" y1="0" x2="0" y2="50" />
              </g>
              <text x="0" y="0" style="font-size:20; font-family:Verdana; fill:blue">
                ABC (skewX)
              </text>
            </g>
          </g>
          <!-- Establish a new coordinate system whose origin is at (200,30)
               in the initial coord. system and which is skewed in Y by 30 degrees.
        -->
          <g transform="translate(200,30)">
            <g transform="skewY(30)">
              <g style="fill:none; stroke:red; stroke-width:3">
                <line x1="0" y1="0" x2="50" y2="0" />
                <line x1="0" y1="0" x2="0" y2="50" />
              </g>
              <text x="0" y="0" style="font-size:20; font-family:Verdana; fill:blue">
                ABC (skewY)
              </text>
            </g>
          </g>
        </svg>
```

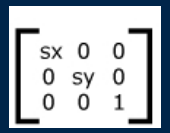


Les attributs de transformations

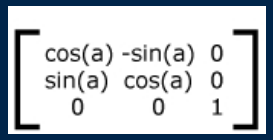
- L'attribut transform a comme valeur possible une liste de transformations séparées par des blancs et/ou une virgule. Ces transformations individuelles sont appliquées l'une après l'autre dans l'ordre où elles apparaissent. Ce peut être :
  - matrix (a,b,c,d,e,f) ; les valeurs e et f peuvent recevoir des unités CSS



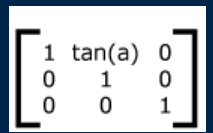
Translation



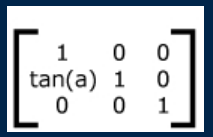
Mise à l'échelle



Rotation



Inclinaison X



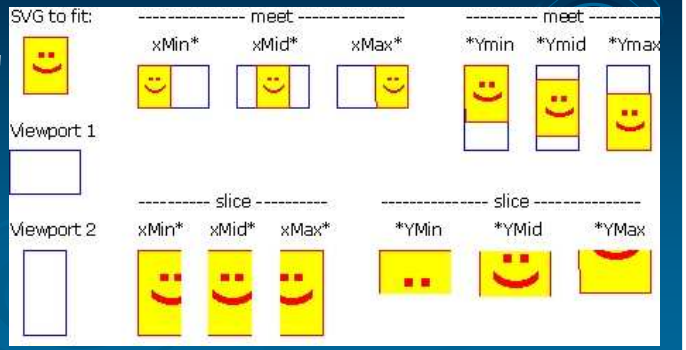
Inclinaison Y

  - translate (tx, [ty]) ; tx et ty peuvent recevoir des unités CSS
  - scale (sx, [sy]) ; si sy n'est pas fourni, on suppose sy = sx
  - rotate (alpha) spécifie une rotation d'angle alpha autour de l'origine de l'espace de coordonnées utilisateur courantes
  - skewX (beta) spécifie une inclinaison d'angle beta le long de l'axe des x
  - skewY (beta) spécifie une inclinaison d'angle beta le long de l'axe des y
- Toutes les valeurs numériques sont réelles ; les valeurs d'angle sont exprimées dans les unités précisées pour les angles

- L'attribut **transform** est appliqué à l'élément AVANT tout calcul de coordonnées ou de longueurs pour cet élément

Les points de vues

- Pour contraindre un ensemble d'objets graphiques à rester confiné dans un espace donné, on peut établir un nouveau point de vue
- Tous les éléments qui peuvent établir un nouveau point de vue ont l'attribut viewBox
  - viewBox (minx , miny, largeur, hauteur)
- On peut contrôler la mise à l'échelle lors du changement de point de vue par l'attribut preserveAspectRatio = *align* [meet ou slice]
  - rien (défaut) ; la mise à l'échelle s'arrange pour que les valeurs extrêmes en x et en y touchent les bords du rectangle de point de vue
  - xMinYmin ; Mise à l'échelle uniforme ; min des x = x min du point de vue min des y = y min du point de vue
  - XMidYMin ; Mise à l'échelle uniforme ; val moyenne des x = x moyen du point de vue min des y = y min du point de vue
  - <svg preserveAspectRatio="xMaxYMax slice"



Formes basiques,path,traitement de l'image,texte,liens

La balise <path>

- L'objet graphique de base est un chemin (path)
  - Contour d'une forme dont on peut spécifier l'épaisseur, la couleur, le remplissage

- **Syntaxe** : <path d="path data" nominalLenght="number" />
  - *path data* est un ensemble de commandes élémentaires (détaillées plus loin)
  - nominalLenght est la longueur totale du chemin en coordonnées utilisateur

- **Commandes de base**
  - M ou m : moveto : x,y démarre un nouveau sous-chemin
  - Z ou z : closepath : ferme un sous-chemin en traçant une ligne droite entre le point courant et le dernier moveto/lineto
  - L ou l : lineto : x , y trace une ligne droite entre le point courant et le point (x,y)
  - H ou h : horizontal lineto : x trace une ligne horizontale entre le point courant et le point (x,y0)
  - V ou v : vertical lineto : y trace une ligne verticale entre le point courant et le point (x0,y)

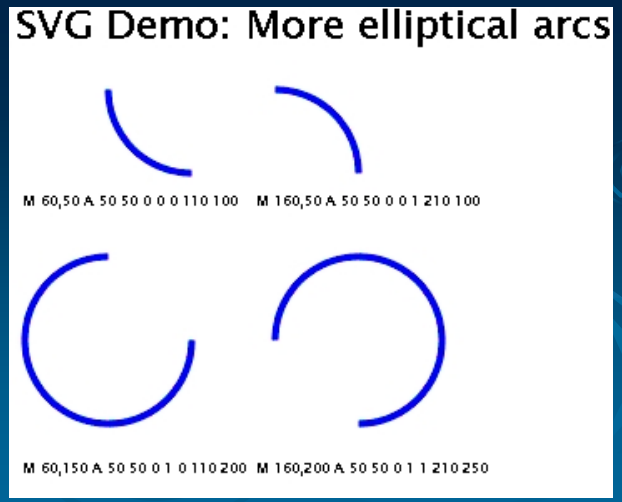
- ```
  <?xml version="1.0" standalone="no"?>
  <!DOCTYPE svg SYSTEM "svg-19991203.dtd" > <svg width="600px" height="200px">
    <text x="5" y="20" style="font-size:24">SVG Test: Path using horizontal & vertical lines</text>
    <g style="fill:none; stroke:blue">
      <path d="M 50,50  h 50 v 50 h -50 z"/>
      <path d="M 110,50 h 50 v 50 h -50 v -50"/>
      <path d="m 50,110 h 50 v 50 h -50 z" />
    </g>
  </svg>
  ```


  ```

Dessiner des courbes

- Il existe de plus trois groupes de commandes qui dessinent **des courbes** :
  - C ou c, S ou s : Courbes de Bézier cubiques . On spécifie un point de départ, un point d'arrivée et 2 points de contrôle
  - Q ou q, T ou t : Courbes de Bézier quadratiques . On spécifie un point de départ, un point d'arrivée et 1 point de contrôle
  - A ou a : Arc elliptique . On spécifie un morceau d'ellipse par 2 rayons et un sens de parcours

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<svg width="500" height="700" >
 <g style="text-rendering:optimizeLegibility;shape-rendering:default">
  <text x="5" y="20" style="font-size:24">SVG Demo: More elliptical arcs</text>
  <g style="stroke-width:4 ; fill:none; stroke:blue" >
   <path d="M 60,50   A 50 50 0 0 0 110 100" />
   <path d="M 160,50  A 50 50 0 0 1 210 100" />
   <path d="M 60,150  A 50 50 0 1 0 110 200" />
   <path d="M 160,200 A 50 50 0 1 1 210 250" />
  </g>
  <g style="font-size:8" >
   <text x="10"  y="120">M 60,50  A 50 50 0 0 0 110 100</text>
   <text x="150" y="120">M 160,50 A 50 50 0 0 1 210 100</text>
   <text x="10"  y="280">M 60,150 A 50 50 0 1 0 110 200</text>
   <text x="150" y="280">M 160,200 A 50 50 0 1 1 210 250</text>
  </g>
 </g>
</svg>
```



Formes élémentaires (1)
Rectangles, cercles, ellipses

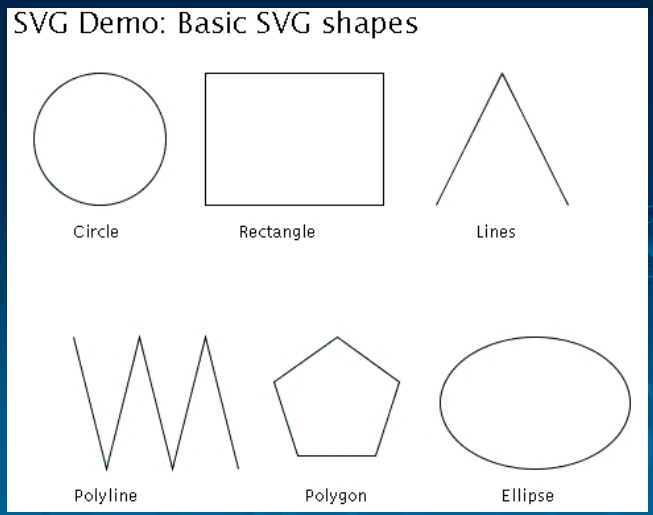
- Il existe un certain nombre de formes prédéfinies (chemins particuliers) qui permettent de s'affranchir de la description complète d'un chemin
- **les rectangles**
  - <rect x="coord" y="coord" width="longueur" height="longueur" rx="longueur" ry="longueur" />
    - x : coordonnée x du coté du rectangle de plus bas x
    - y : coordonnée y du coté du rectangle de plus bas y
    - width : largeur du rectangle
    - height : hauteur du rectangle
    - rx : pour des rectangles à coins arrondis rayon en x de l'ellipse assurant le raccord
    - ry : pour des rectangles à coins arrondis rayon en y de l'ellipse assurant le raccord
- **les cercles**
  - <circle cx="coord" cy="coord" r="longueur" />
    - cx : coordonnée x du centre du cercle, 0 par défaut
    - cy : coordonnée y du centre du cercle, 0 par défaut
    - r : rayon du cercle
- **les ellipses**
  - <ellipse cx="coord" cy="coord" rx="longueur" ry="longueur" />
    - cx : coordonnée x du centre de l'ellipse, 0 par défaut
    - cy : coordonnée y du centre de l'elipse, 0 par défaut
    - rx : rayon de l'ellipse suivant l'axe des x
    - ry : rayon de l'ellipse suivant l'axe des y

Formes élémentaires (2) lignes, polylignes et polygones

- **les lignes**
  - <line x1="coord" x2="coord" y1="coord" y2="coord" />
    - x1 : coordonnée x du point de départ
    - y1 : coordonnée y du point de départ
    - x2 : coordonnée x du point d'arrivée
    - y2 : coordonnée y du point d'arrivée
- **les polylignes**
  - Une polyligne est un ensemble de lignes droites connectées entre elles. Elle définit une forme ouverte.
  - <polyline points="liste de points" />
    - *liste de points* est une liste de coordonnées x , y séparées par des virgules. Ces coordonnées sont exprimées dans l'espace utilisateur
- **Les polygones**
  - Un polygone est un ensemble de lignes droites connectées entre elles définissant une forme fermée
  - <polygon points="liste de points" />
    - *liste de points* est une liste de coordonnées x , y séparées par des virgules. Ces coordonnées sont exprimées dans l'espace utilisateur

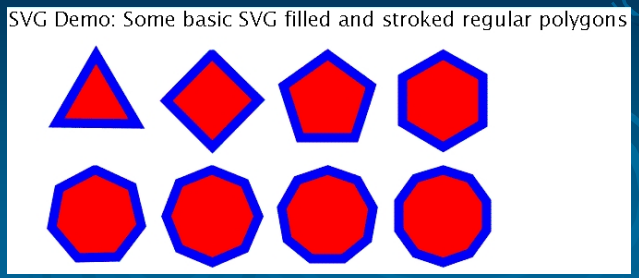
Formes élémentaires
Exemple (1)

```
<?xml version="1.0" standalone="no"?>
      <!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
      <svg width="500" height="500" >
        <g style="text-rendering:optimizeLegibility;shape-rendering:default">
          <text x="5" y="20" style="font-size:22">SVG Demo: Basic SVG shapes</text>
          <g style="stroke:black; fill:none; shape-rendering:default" >
            <circle cx="70"  cy="100" r="50"  />
            <rect x="150" y="50" width="135" height="100" />
            <line x1="325" y1="150" x2="375" y2="50" />
            <line x1="375" y1="50"  x2="425" y2="150" />
            <polyline points="50,250,75,350,100,250,125,350,150,250,175,350" />
            <polygon points="250,250,297,284,279,340,220,340,202,284,250,250" />
            <ellipse cx="400" cy="300" rx="72" ry="50" />
          </g>
          <g style="text-rendering:optimizeSpeed">
            <text x="50"  y="175">Circle</text>
            <text x="175" y="175">Rectangle</text>
            <text x="355" y="175">Lines</text>
            <text x="50"  y="375">Polyline</text>
            <text x="225" y="375">Polygon</text>
            <text x="375" y="375">Ellipse</text>
          </g>
        </g>
      </svg>
```



Formes élémentaires
Exemple (2)

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<svg width="680" height="500" >
        <g style="text-rendering:optimizeLegibility">
          <text x="5" y="20" style="font-size:22">SVG Demo: Some basic SVG filled and stroked regular
            polygons</text>
         <g style="stroke:blue; fill:red; shape-rendering:default; stroke-width:10" >
            <polygon points="99,50,143,125,56,124,99,50" />
            <polygon points="225,50,275,99,225,150,175,100,224,50" />
            <polygon points="350,50,397,84,379,140,320,140,302,84,350,50" />
            <polygon points="475,50,518,74,518,125,475,150,431,124,431,75,475,50" />
            <polygon points="99,175,138,193,148,235,122,269,79,270,51,237,59,195,97,175" />
            <polygon
               points="225,175,260,189,275,225,260,260,225,275,189,260,175,225,189,189,224,175" />
            <polygon
               points="350,175,382,186,399,216,393,250,367,271,332,271,306,250,300,216,317,186,350,
                       175" />
            <polygon
               points="475,175,504,184,522,209,522,240,504,265,475,275,445,265,427,240,427,209,445,
                       184,475,175" />
         </g>
       </g>
</svg>
```



La balise <text>

- Le texte suit les recommandations générales des caractères XML . Il n'effectue ni retour à la ligne ni césure automatique
- La balise <text> est traitée comme un objet graphique
- En tant que telle, elle subit l'influence :
  - des changements de coordonnées
  - du mode de rendu
  - du clipping

<text x="coord" y="coord" />

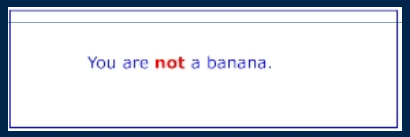
- x représente l'abscisse de départ du texte
  - s'il n'est pas suivi d'unité, la valeur est calculée dans l'espace utilisateur
  - s'il est suivi d'une unité CSS ou de %, la valeur est calculée par rapport au point de vue
- y représente l'ordonnée de départ du texte
  - si elle n'est pas suivie d'unité, la valeur est calculée dans l'espace utilisateur
  - si elle est suivie d'une unité CSS ou de %, la valeur est calculée par rapport au point de vue

- A l'intérieur d'un élément <text>, on peut ajuster la position du texte, la valeur du texte ou la police du texte grâce à l'élément <tspan>
  - <tspan x="coord+" y="coord+" dx="coord+" dy="coord+" rotate="auto|nombre" />

Exemple de textes

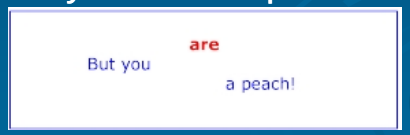
- **Exemple 1**

```
<?xml version="1.0" standalone="no"?>
    <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
      "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
    <svg width="10cm" height="3cm">
      <desc>Example tspan01 - using tspan to change visual attributes</desc>
        <g style="font-family:Verdana; font-size:12pt">
        <text x="2cm" y="1.5cm" style="fill:blue">
          You are <tspan style="font-weight:bold; fill:red">not</tspan> a banana.
        </text>
      </g>
    </svg>
```



- **Exemple 2**

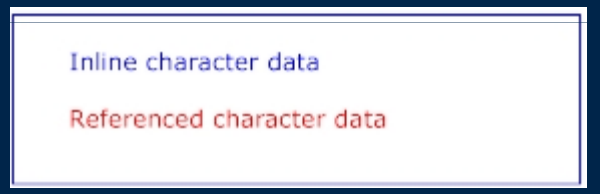
```
<?xml version="1.0" standalone="no"?>
    <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
      "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
    <svg width="10cm" height="3cm">
      <desc>Example tspan02 - using tspan's dx and dy attributes
          for incremental positioning adjustments</desc>
       <g style="font-family:Verdana; font-size:12pt">
        <text x="2cm" y="1.5cm" style="fill:blue">
          But you <tspan dx="2em" dy="-.5cm" style="font-weight:bold; fill:red"> are </tspan>
            <tspan dy="1cm"> a peach! </tspan>
        </text>
      </g>
    </svg>
```



Les textes

- A l'intérieur de la balise <text>, on peut :
  - soit spécifier le texte directement
  - soit référencer le texte d'un autre élément par la balise <tref>

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
      "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
    <svg width="10cm" height="3cm">
      <defs>
        <text id="ReferencedText">
          Referenced character data
        </text>
      </defs>
      <desc>Example tspan04 - inline vs reference text content</desc>   <text x="1cm"
        y="1cm" style="font-size:12pt; fill:blue">
        Inline character data
      </text>
      <text x="1cm" y="2cm" style="font-size:12pt; fill:red">
        <tref xlink:href="#ReferencedText"/>
      </text>
    </svg>
```



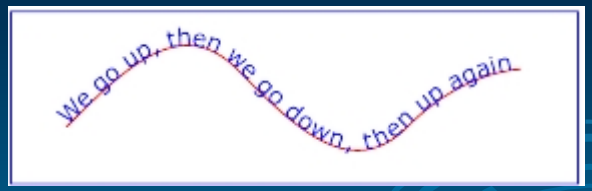
Lier le texte avec un chemin

- On peut imposer au texte de suivre un chemin prédéfini par la balise <path>

<textPath starOffset="longueur|pourcentage" xlink:href="uri" />

- starOffset est le décalage par rapport au début du texte
- une longueur représente une distance le long du chemin mesurée selon la métrique de l'espace utilisateur
- un pourcentage représente un pourcentage par rapport au chemin entier selon la métrique de l'espace utilisateur

```
<?xml version="1.0" standalone="no"?>
      <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
        "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
      <svg width="10cm" height="3cm" viewBox="0 0 1000 300">
        <defs>
          <path id="MyPath"
              d="M 100 200
                 C 200 100 300   0 400 100
                 C 500 200 600 300 700 200
                 C 800 100 900 100 900 100" />
        </defs>
        <desc>Example toap01 - simple text on a path</desc>   <use
      xlink:href="#MyPath" style="stroke:red" />
        <text style="font-family:Verdana; font-size:42.3333; fill:blue">
          <textPath xlink:href="#MyPath">
            We go up, then we go down, then up again
          </textPath>
        </text>
      </svg>
```



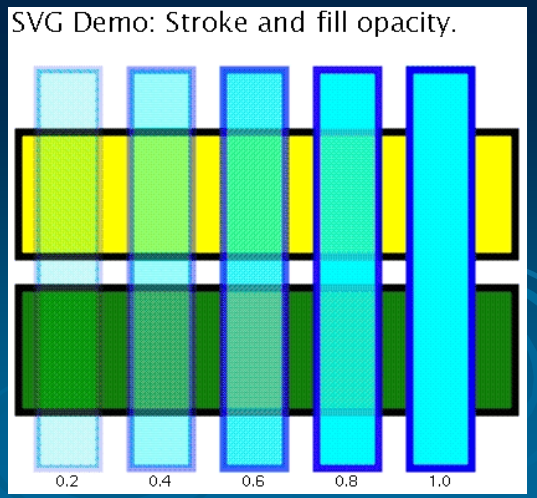
Traitement de l'image : Le rendu

- Les éléments <path>,<text> et les formes de base peuvent être remplis et coloriés (c'est à dire peints sur les bords). On appelera cette opération le rendu
- En SVG, on peut rendre avec :
  - une couleur simple
  - un gradient (linéaire ou radial)
  - un motif (vecteur ou image)
  - des motifs personnalisés disponibles par extension

- Deux propriétés fill et stroke se partagent les attributs suivants :
  - couleur
  - uri de la couleur ou du gradient
- Propriétés de fill (remplissage)
  - opacité
- Propriétés de stroke (dessin)
  - épaisseur
  - jonction de lignes
  - arrondi des angles
  - pointillés
- La propriété de couleur s'applique aux propriétés fill et stroke

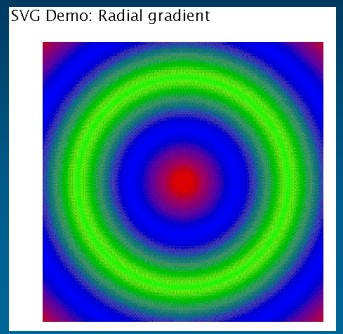
Exemple de remplissage

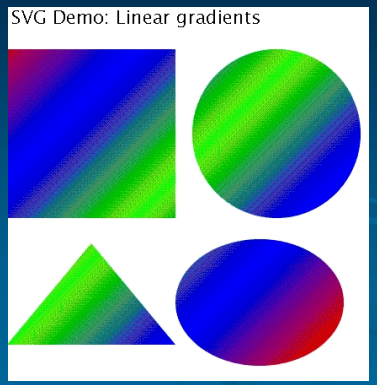
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<svg width="500" height="500" >
<g style="stroke:black; stroke-width:6; text-rendering:optimizeLegibility; shape-rendering:default" >
   <text x="5" y="20" style="font-size:22">SVG Demo: Stroke and fill opacity.</text>
   <rect x="10"  y="100" width="400" height="100" style="fill:yellow"  />
   <rect x="10"  y="225" width="400" height="100" style="fill:green"  />
   <g style="stroke:blue;fill:cyan">
          <rect x="25"  y="50"  width="50"  height="320" style="stroke-opacity:0.2;fill-opacity:0.2"  />
          <rect x="100" y="50"  width="50"  height="320" style="stroke-opacity:0.4;fill-opacity:0.4"  />
          <rect x="175" y="50"  width="50"  height="320" style="stroke-opacity:0.6;fill-opacity:0.6"  />
          <rect x="250" y="50"  width="50"  height="320" style="stroke-opacity:0.8;fill-opacity:0.8"  />
          <rect x="325" y="50"  width="50"  height="320"   />
    </g>
   <text x="40"  y="385">0.2</text>
   <text x="115" y="385">0.4</text>
   <text x="190" y="385">0.6</text>
   <text x="265" y="385">0.8</text>
   <text x="340" y="385">1.0</text>
 </g>
</svg>
```



Gradients et motifs (ne marche plus)

- En SVG, on peut remplir un objet ou en souligner le contour par :
  - une couleur
  - un gradient (de couleurs)
  - un motif
- Un gradient de couleurs consiste en une transition douce entre deux couleurs selon
  - un vecteur
  - Les gradients de couleur peuvent être :
    - linéaires
    - radiaux






```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<svg width="500px" height="600px" >
 <g style="text-rendering:optimizeLegibility;shape-rendering:default">
  <defs>
   <linearGradient id="grad1" x1="0" y1="0" x2="400" y2="400">
    <stop offset="0%"   style="stop-color:#FF0000"/>
    <stop offset="25%"  style="stop-color:#0000FF"/>
    <stop offset="50%"  style="stop-color:#00FF00"/>
    <stop offset="75%"  style="stop-color:#0000FF"/>
    <stop offset="100%" style="stop-color:#FF0000"/>
   </linearGradient>
  </defs>
<text x="5" y="20" style="font-size:22">SVG Demo: Linear
      gradients</text>
  <path style="fill:url(#grad1)" d="M0 50 h200 v200 h-200 z"/>
  <circle  style="fill:url(#grad1)" cx="320" cy="150" r="100" />
  <polygon style="fill:url(#grad1)" points="0,400,200,400,100,280" />
  <ellipse style="fill:url(#grad1)" cx="300" cy="350" rx="100" ry="75"/>
  <rect style="fill:url(#grad1)" x="0" y="450" width="400" height="100" />
 </g>
</svg>
```

Les Liens

- On distingue les liens extra et intra document SVG
- **Liens extra document :**
  - Les liens en dehors du document courant sont pris en charge par l'élément <a>, analogue à l'élément correspondant de HTML ou SMIL
  - A noter que l'élément <a> utilise la syntaxe de XLink (en cours de spécification)
- *Exemple :*

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
    <desc>This valid svg document draws a triangle which is a hyperlink
    </desc>
    <a xlink:href="http://www.w3.org">
        <path d="M 0 0 L 200 0 L 100 200 z"/>
    </a>
</svg>
```

- **Les Liens : liens internes**
  - Nécessite de spécifier un fragment SVG
  - Analogue à HTML : MyDrawing.svg#MyView
  - Référence compatible avec XPointer : MyDrawing.svg#xptr(id('MyView'))
  - Spécification d'une vue SVG : MyDrawing.svg#svgView(viewBox(0,200,1000,1000))

Animations

Introduction culture générale animation...

- Les éléments d'animation SVG ont en été développés en collaboration avec le Groupe de Travail Multimédias Synchronisés du W3C (SYMM), les développeurs de la spécification du Langage d'Intégration Multimédia Synchronisé (SMIL) version 1.0 [SMIL1].
  Un fichier SMIL (The Synchronous Multimedia Integration Language) peut contenir du texte, des images fixes et en mouvement, des animations, ainsi que du son. Le code décrit quels éléments multimédia sont présents dans le document, où ils sont situés, à quel moment ils apparaissent et combien de temps ils durent.
  (Les formats de fichiers acceptés dans SMIL sont les suivants : .swf, .rm, .wav, .aif, .mov, .mp3, .gif, .jpg, .rt, .rm, .avi, .mov, .asf, .viv, .mpeg.)
- Le Groupe de Travail SYMM, en collaboration avec le Groupe de Travail SVG, a édité la spécification SMIL Animation [SMILANIM] http://www.w3.org/TR/smil20/animation.html#animationNS-Introduction, qui représente un jeu de fonctions d'animation XML d'usage général.
  SVG incorpore les fonctions d'animation définies dans la spécification SMIL Animation et fournit des développements propres à SVG.

Une petite animation

- Un language d'animation doit permette de spécifier quant commence l'animation d'un élément, quand cette animation s'arrête, si cette animation doit se répéter et combien de fois. Et biensûr, ll doit permettre de décrire ce qui se passe durant l'animation.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="250" height="100" viewBox="0 0 250 100">
<title>Simple Animation</title>
<rect x="10" y="10" width="200" height="20" style="stroke: black; fill: none;">
<animate
attributeName="width"
attributeType="XML"
from="200" to="20"
begin="0s" dur="5s"
repeatCount="indefinite"
fill="freeze" />
</rect>
</svg>
```

..\..\..\..\..\..\..\Bureau\TP SVG\rectangle diminue.svg

Svg et animations

- 'animate'permet aux attributs et aux propriétés **scalaires** de recevoir différentes valeurs au cours du temps.
- 'set'un raccourci pratique pour l'élément 'animate', utile lors de l'assignation de valeurs d'animation à des attributs et propriétés **non-numériques**, telle que la propriété 'visibility'
- 'animateMotion' déplace un élément le long d'un tracé de mouvement
- 'animateColor'modifie la valeur de couleur des attributs et des propriétés particuliers au cours du temps

- En supplément, SVG comprend les extensions à SMIL Animation compatibles suivantes :
  - 'animateTransform' modifie l'un des attributs de transformation de SVG au cours du temps, tel que l'attribut transform
  - l'attribut path SVG autorise la spécification de toute fonction, issue de la syntaxe des données de tracé de SVG, dans l'attribut path d'un élément 'animateMotion' (SMIL Animation ne permet qu'un sous-ensemble de la syntaxe des données de tracé de SVG dans un attribut path)
  - 'mpath' elementSVG ajoute un attribut keyPoints à l'élément 'animateMotion' pour offrir un contrôle précis des animations d'un tracé de mouvement
  - l'attribut keyPoints SVG ajoute un attribut keyPoints à l'élément 'animateMotion' pour offrir un contrôle précis des animations d'un tracé de mouvement
  - l'attribut rotate SVG ajoute un attribut rotate à l'élément 'animateMotion' pour le contrôle automatique de la rotation d'un objet, de façon à ce que son axe-x se dirige dans la même direction (ou dans la direction opposée) que le vecteur tangent directionnel du tracé de mouvement

Une autre animation

```
<svg width="300" height="250" viewBox="0 0 300 250">
    <title>Multiple Animations on a Single Object</title>
    <rect x="10" y="10" width="20" height="20"
      style="stroke: black; fill: green; style: fill-opacity: 0.25;">
    <animate attributeName="width" attributeType="XML"
      from="20" to="250" begin="0s" dur="8s" fill="freeze"
      repeatCount="indefinite" />
    <animate attributeName="height" attributeType="XML"
      from="20" to="200" begin="0s" dur="8s" fill="freeze"
      repeatCount="indefinite"/>
    <animate attributeName="fill-opacity" attributeType="CSS"
      from="0.25" to="1" begin="0s" dur="3s" fill="freeze"
      repeatCount="indefinite"/>
    <animate attributeName="fill-opacity" attributeType="CSS"
      from="1" to="0.25" begin="3s" dur="3s" fill="freeze"
      repeatCount="indefinite"/>
    </rect>
</svg>
```

..\..\..\..\..\..\Bureau\TP SVG\multipleanim.svg

Animation le long d’un chemin

```
<svg width="300" height="200" viewBox="0 0 300 200">
    <title>Animation Along a Complex Path</title>
    <!-- show the path along which the triangle will move -->
    <path d="M50,125 C 100,25 150,225, 200, 125"
          style="fill: none; stroke: blue;"/>
    <!-- Triangle to be moved along the motion path.
          It is defined with an upright orientation with the base of the
          triangle centered horizontally just above the origin. -->
    <path d="M-10,-3 L10,-3 L0,-25z" style="fill: yellow; stroke: red;">
      <animateMotion
          path="M50,125 C 100,25 150,225, 200, 125"
          dur="6s" fill="freeze" repeatCount="indefinite"/>
    </path>
</svg>..\..\..\..\..\..\Bureau\TP SVG\animatemotion.svg
```

Animation le long d’un chemin

```
<svg width="300" height="200" viewBox="0 0 300 200">
    <title>Motion Along a Complex Path Using mpath</title>
    <path id="cubicCurve"
      d="M50,125 C 100,25 150,225, 200, 125"
      style="fill: none; stroke: blue;"/>
    <path d="M-10,-3 L10,-3 L0,-25z" style="fill: yellow;
      stroke: red;" >
      <animateMotion dur="6s" rotate="auto" fill="freeze"
        repeatCount="indefinite">
        <mpath xlink:href="#cubicCurve"/>
      </animateMotion>
    </path>
</svg>..\..\..\..\..\..\..\Bureau\TP SVG\animatemotion2.svg
```

Découpage et masquage

```
<?xml version="1.0" standalone="no"?>
    <!DOCTYPE svg SYSTEM "svg-19991203.dtd" >
<svg width="500" height="500" >
  <defs>
        <clipPath>
          <ellipse cx="200" cy="220" rx="80" ry="120" id="myClip"  />
        </clipPath>
      </defs>
  <g style="text-rendering:optimizeLegibility;shape-rendering:default" >
        <text  x="5" y="25" style="font-size:24">SVG Demo: An image
    clipped by an ellipse.</text>
    <ellipse cx="200" cy="220" rx="90" ry="130" style="fill:#CE8A77" />
        <image x="40" y="100" width="320" height="240"
    xlink:href="krl1.jpg" style="clip-path:URL(#myClip)"/>
      </g>
    </svg>
```

Effets de filtre

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG
December 1999//EN"
 "http://www.w3.org/Graphics/SVG/SVG-
19991203.dtd">
<svg width="278px" height="118px">
 <defs>
<filter id="Shadow">
<feGaussianBlur in="SourceAlpha"
  stdDeviation="3"
  result="blurredAlpha" />
 <feOffset in="blurredAlpha"
   dx="2" dy="1"
   result="offsetBlurredAlpha" />
 <feDiffuseLighting in="blurredAlpha"
   diffuseConstant=".5"
    surfaceScale="5"
    resultScale="5"
    LightColor="white"
    result="bumpMapDiffuse" >
 <feDistantLight azimuth="135" elevation="60"/>
</feDiffuseLighting>
<feComposite in="bumpMapDiffuse"
in2="SourceGraphic"
  operator="arithmetic" k1="1"
  result="litPaint" />
<feSpecularLighting in="blurredAlpha"
      surfaceScale="5"
      specularConstant=".5"
      specularExponent="10"
      lightColor="white"
      result="bumpMapSpecular" >
     <feDistantLight azimuth="135" elevation="60"/>
    </feSpecularLighting>
    <feComposite in="litPaint"
   in2="bumpMapSpecular"
     operator="arithmetic" k2="1" k3="1"
     result="litPaint" />
    <feComposite in="litPaint"
      in2="SourceAlpha"
      operator="in"
      result="litPaint" />
    <feMerge>
      <feMergeNode in="offsetBlurredAlpha" />
       <feMergeNode in="litPaint" />
    </feMerge>
   </filter>
   </defs>
    <desc>Example filters02 - text with shadowing
  effect</desc>
 <text style="font-size:36px; fill:red; filter:url(#Shadow)"
       x="10px" y="70px">Shadowed Text</text>
  </svg>
```

Insertion de scripts

- Il est possible d'insérer des « javascripts »
- <?xml version="1.0" standalone="no"?>

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
      "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
    <svg width="4in" height="3in">
      <defs>
        <script><![CDATA[
          /* Beep on mouseclick */
          MouseClickHandler() { beep(); }
         ]]>
        </script>
      </defs>
      <circle onclick="MouseClickHandler()" r="85"/>
    </svg>
```

Programmation XML

SAX un parser pour XML

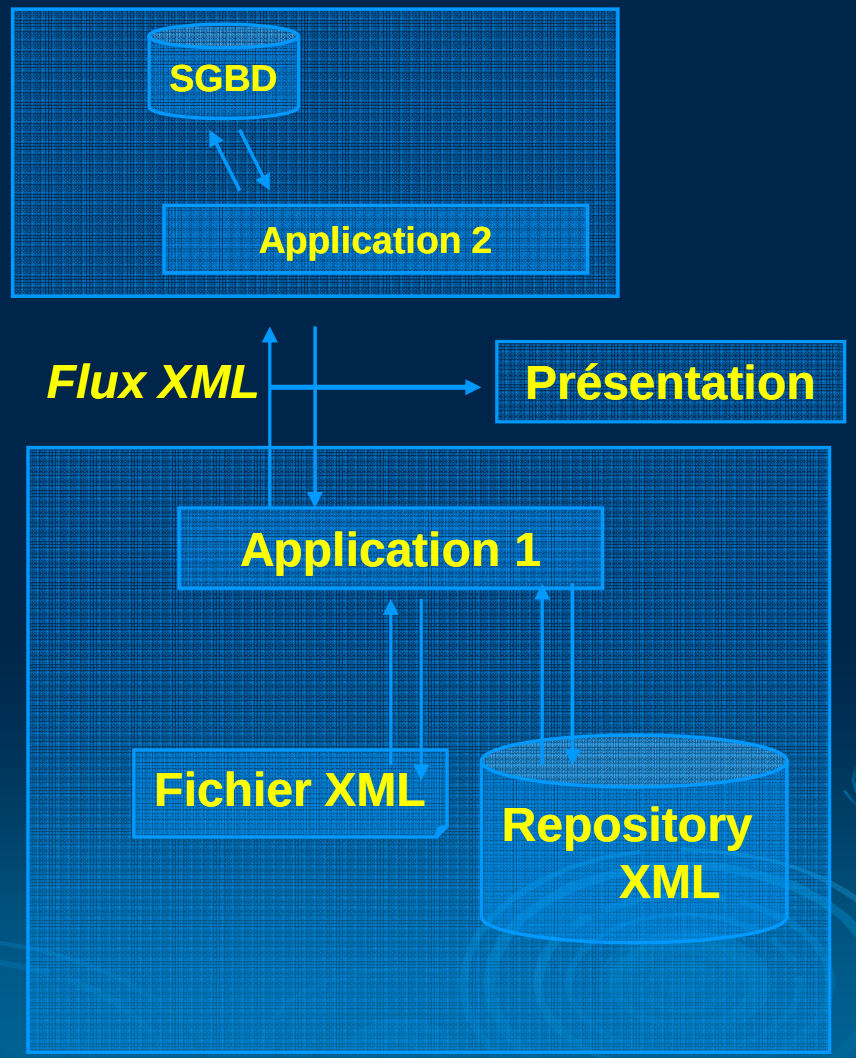
DOM: un modèle de document et une API

Programmation XML

- Pour quoi faire ?
  - Format de stockage
    - Plus puissant que les fichiers fixes
    - Plus standard que les fichiers formatés
    - Plus souple et plus « simple » qu'une base de données
  - Format d'échange
    - Entre applications
    - Vers un format standard

Un programme utilisant XML

- XML comme format d'échange
- Stockage de documents XML
  - Fichiers textes
  - Repository natif
- Manipulation de données XML



Objectifs

- Analyser un document XML
- Manipuler un ou plusieurs documents XML
  - Représenter un document
  - Localiser et des fragments
  - Créer des fragments
- Créer un document XML

The Simple API for XML

http://www.saxproject.org/

Présentation de SAX

- API évènementielle de parsing de documents XML
- Développée en collaboration par les membres de la liste XML-DEV
- A l'origine uniquement disponible pour Java
  - Disponible aussi avec Perl, C, ...
- SAX2: version plus récente supportant les namespaces
- Nombreux parseurs publics
  - XP de James Clark, Aelfred, Saxon
  - MSXML4 de Microsoft
  - Xerces et Crimson de Apache

API évènementielle

- Rapporte les évènements liés à l'analyse syntaxique:
  - ouverture d'élément
  - Fin d'élément
  - ...
- L'arbre du document n'est pas construit
  - Exemple : Recherche les éléments `Personne` contenant le texte « `durand` »

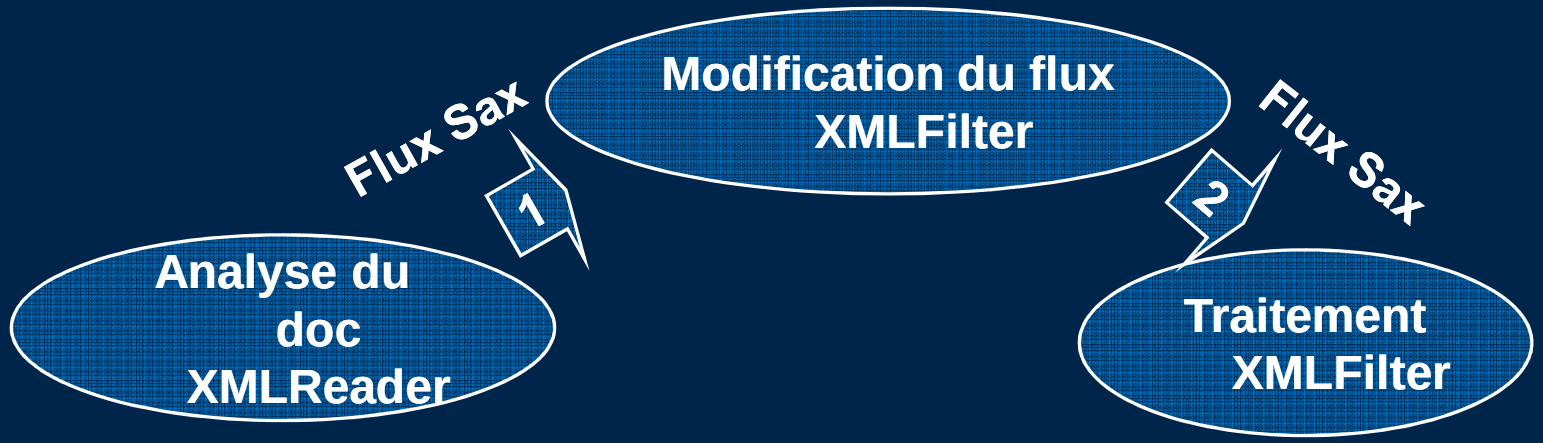
Les interfaces essentielles (SAX2)

- XMLReader
  - setContentHandler
  - setErrrorHandler
  - parse
- ContentHandler
  - startDocument
  - endDocument
  - startElement
  - endElement
  - characters
- InputSource
- ErrorHandler
  - fatalError
  - error
  - warning

Utilisation de SAX

- Définir une classe qui indique comment réagir aux évènements de l'analyse du document
  - Sous-classe de **ContentHandler**
- Utiliser cette classe de la manière suivante:
  - Instancier la classe **XMLReader**
    - (choix d'une implantation de sax)
  - Instancier la **InputSource** pour accéder au document XML source
  - Associer une instance de la sous-classe de **ContentHandler** au **XMLReader**
  - L'analyse est lancée avec la méthode ***parse(inputSource)***
- Les méthodes de la classe sont appelées par le XMLReader au cours du traitement

Les filtres SAX



- Modification à la volée du flot d'événements
- Fonctionnement en chaîne

Exemple d’analyse avec SAX

- ```
  <?xml version="1.0"?>
   <doc>
     <para>Hello, world!</para>
   </doc>
  ```

- ```
  start document
  start element: doc
  start element: para
  characters: Hello, world!
  end element: para
  end element: doc
  end document
  ```

Implantation en Java

```
import java.io.FileReader; import org.xml.sax.XMLReader; import org.xml.sax.Attributes;
import org.xml.sax.InputSource; import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;

public MySAXApp () { super(); }

public class MySAXApp extends DefaultHandler {
  public static void main (String args[]) throws Exception {
    XMLReader xr = XMLReaderFactory.createXMLReader();
    MySAXApp handler = new MySAXApp();
    xr.setContentHandler(handler);
    xr.setErrorHandler(handler);
    FileReader r = new FileReader(args[0]);
    xr.parse(new InputSource(r)); } }

public void startDocument () { System.out.println("Start document"); }

public void endDocument () { System.out.println("End document"); }

public void startElement (String uri, String name, String qName, Attributes atts) {
        if ("".equals (uri)) System.out.println("Start element: " + qName);
        else System.out.println("Start element: {" + uri + "}" + name); }

public void endElement (String uri, String name, String qName) {
        if ("".equals (uri)) System.out.println("End element: " + qName);
        else System.out.println("End element: {" + uri + "}" + name); }

public void characters (char ch[], int start, int length) {
        System.out.print("Characters: \"");
        System.out.print(ch[i]);
        System.out.print("\"\n"); }
}
```

Conclusion sur SAX

- Interfaces événementielles simples
  - Fonctionnement en pipe-line
  - Faible coût en mémoire
  - Bonnes performances
- Difficile à utiliser si
  - Traitement multiple sur un document
  - Utilisation de la structure arborescente
    - Analyse avec un automate (Changement d'états)

Document Object Model (DOM)

Document Object Model Level 2
(W3C Recommendation)
13 November 2000

Document Object Model Level 2 Core
Document Object Model Level 2 Views
Document Object Model Level 2 Events
Document Object Model Level 2 Style
Document Object Model Level 2 Traversal and Range

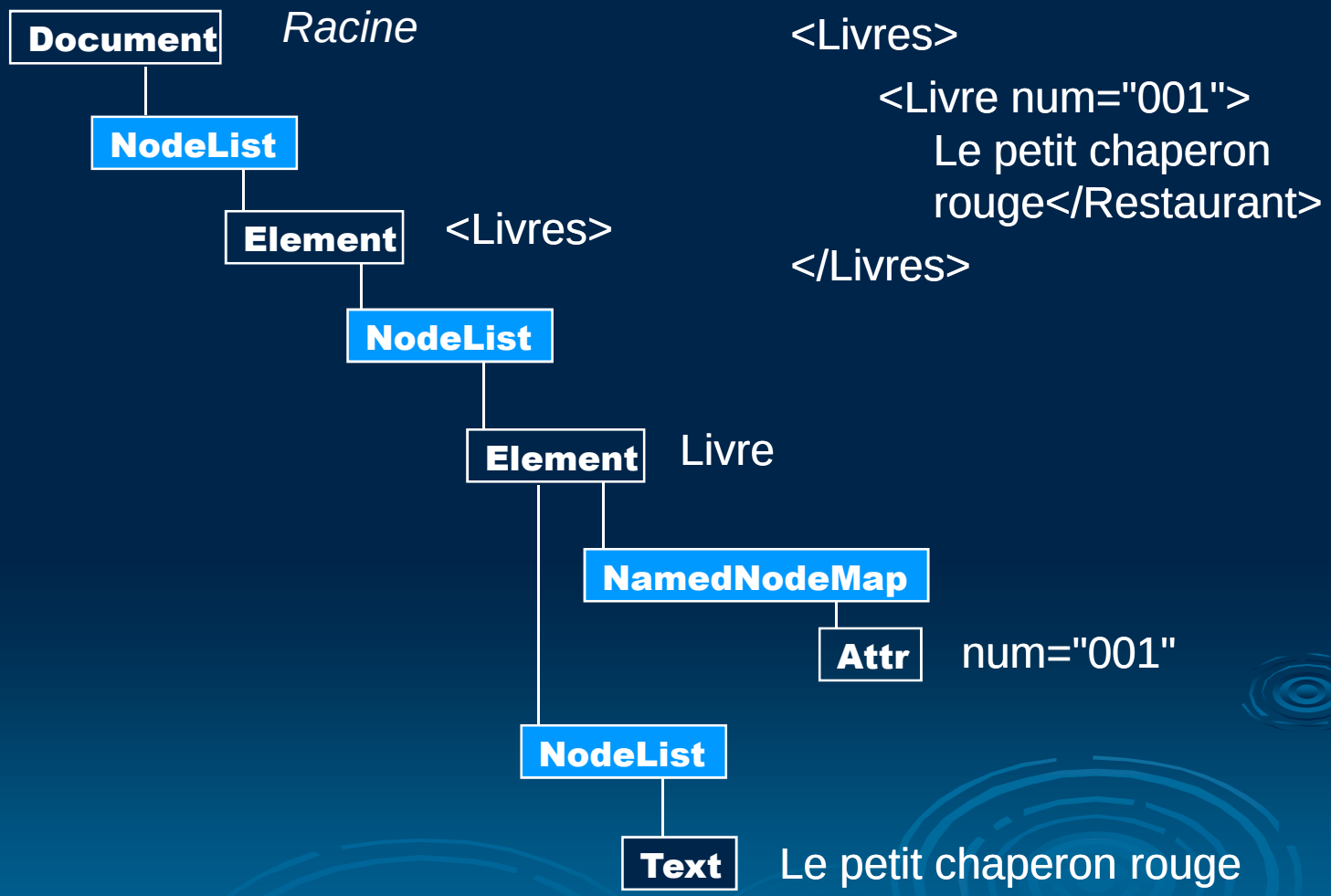
Introduction au DOM

- Interface standard d'accès à un document XML
- Indépendant
  - Du langage de programmation
  - De la plateforme
- Différents bindings: Java, C++, …
- Définit une interface:
  - Un modèle arborescent (ensemble de nœuds)
  - Une API

DOM une API pour documents

- Standard W3C
- Fonctionne avec HTML et XML
- Propose un modèle pour représenter un document
  - Produit par un "parser"
- Interface de navigation
  - Définie en IDL CORBA
  - Peut être utilisée en:
    - Java, C++
    - C#, VB
    - Python

Un d'arbre DOM



DOM, une forêt de nœuds

- Navigation via un arbre générique de nœuds
  - **Node**
  - **NodeList** (Parent/Child)
  - **NamedNodeMap** (attributs)
- Tout nœud hérite de **Node**

Nœuds DOM et leurs fils

- **Document** → Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- **DocumentFragment** → Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **DocumentType** → no children
- **EntityReference** → Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **Element** → Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- **Attr** → Text, EntityReference
- **ProcessingInstruction** → no children
- **Comment** → no children
- **Text** → no children
- **CDATASection** → no children
- **Entity** → Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **Notation** → no children

Le Dom définit aussi une structure de liste de nœuds : NodeList.

Les interfaces DOM

- Interfaces fondamentales
  - DOMImplementation
  - Document
  - Comment
  - DocumentFragment
  - Element
  - Attr(ibute)
  - NamedNodeMap
  - CharacterData
    - Comment
    - Text
- Interfaces étendues XML
  - ProcessingInstruction
  - DocumentType
  - CDATASection
  - Notation
  - Entity
  - EntityReference

Mise en oeuvre - Navigation

- A partir de l'instance de Document
  - getDocumentElement() *obtenir l'élément racine*
- Pour un élément
  - getName(), getLocalName(), getNamespaceURI() : *obtenir le nom et l'espace de nom de l'élément*
  - getChildNodes() : *obtenir la liste des noeuds fils*
  - getFirstNode(), getLastNode(), getNextSibling() : *navigation*
  - getAttributes() : *obtenir la table de hachage des attributs*
- Pour les attributs
  - getName() : *obtenir le nom de l'attribut*
  - getValue() : *obtenir la valeur de l'attribut*
- Pour les noeuds texte
  - getNodeValue() : obtenir *valeur du noeud texte*

Méthodes de parcours dans DOM

- NodeIterator

```
NodeIterator iter=
((DocumentTraversal)document).createNodeIterator(
     root,
     NodeFilter.SHOW_ELEMENT, null);
while (Node n = iter.nextNode()) printMe(n);
```

- TreeWalker

```
processMe(TreeWalker tw) {
     Node n = tw.getCurrentNode();
     nodeStartActions(tw);
     for (Node child=tw.firstChild(); child!=null;
child=tw.nextSibling())
               { processMe(tw); }
     tw.setCurrentNode(n);
     nodeEndActions(tw);
}
```

- NodeFilters

```
NamedAnchorFilter myFilter = new NamedAnchorFilter();
NodeIterator iter=
     ((DocumentTraversal)document).createNodeIterator( node,
               NodeFilter.SHOW_ELEMENT, myFilter);
```

Mise en œuvre - Construction

- Créer une instance de **Document**
  - En général, dépend de l'implémentation
- Construire les nœuds de l'arbre
  - L'instance de **Document** sert d'usine à objets (*factory*)
  - *createElementNS(namespaceURI, qName), createAttributeNS(…)…*
- Ajouter des nœuds en fixant des liens avec les nœuds existants.
  - *appendChild(node), replaceChild(node1, node2), insertBefore(node)*
  - *setAttributeNS(…)*

Bilan DOM

- Une interface objet standard
  - Navigation
  - Construction
- Des concepts simple
  - Interface vaste mais intuitive
- Performances actuellement limitées
  - Coût mémoire important

Interface IDL pour le nœud *document*

```
interface Document : Node {
  readonly attribute DocumentType     doctype;
  readonly attribute DOMImplementation  implementation;
  readonly attribute Element          documentElement;
  Element             createElement(in DOMString tagName)
                                  raises(DOMException);
  DocumentFragment   createDocumentFragment();
  Text                createTextNode(in DOMString data);
  Comment             createComment(in DOMString data);
  CDATASection        createCDATASection(in DOMString data)
                                  raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                      in DOMString data)
                                  raises(DOMException);
  Attr                createAttribute(in DOMString name)
                                  raises(DOMException);
  EntityReference     createEntityReference(in DOMString name)
                                  raises(DOMException);
  NodeList            getElementsByTagName(in DOMString tagname);
  // Introduced in DOM Level 2:
  Node                importNode(in Node importedNode,
                                 in boolean deep)
                                      raises(DOMException);
  // Introduced in DOM Level 2:
  Element             createElementNS(in DOMString namespaceURI,
                                  in DOMString qualifiedName)
                                  raises(DOMException);
  // Introduced in DOM Level 2:
  Attr                createAttributeNS(in DOMString namespaceURI,
                                  in DOMString qualifiedName)
                                  raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList            getElementsByTagNameNS(in DOMString
        namespaceURI,
                                      in DOMString localName);
  // Introduced in DOM Level 2:
  Element             getElementById(in DOMString elementId);
};
```

Interface IDL pour les nœuds *element*

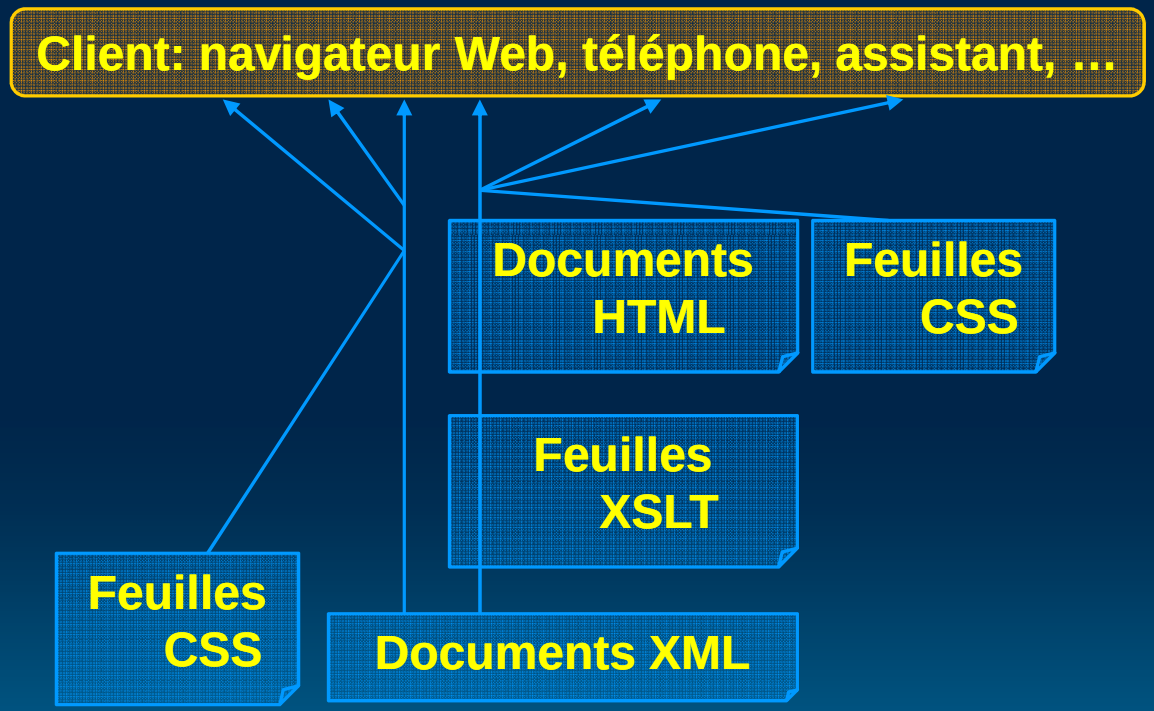
```
interface Element : Node {
  readonly attribute DOMString        tagName;
  DOMString           getAttribute(in DOMString name);
  void                setAttribute(in DOMString name,
                                   in DOMString value)
                                       raises(DOMException);
  void                removeAttribute(in DOMString name)
                                       raises(DOMException);
  Attr                getAttributeNode(in DOMString name);
  Attr                setAttributeNode(in Attr newAttr)
                                       raises(DOMException);
  Attr                removeAttributeNode(in Attr oldAttr)
                                       raises(DOMException);
  NodeList            getElementsByTagName(in DOMString name);
  // Introduced in DOM Level 2:
  DOMString           getAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName);
  // Introduced in DOM Level 2:
  void                setAttributeNS(in DOMString namespaceURI,
                                     in DOMString qualifiedName,
                                     in DOMString value)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  void                removeAttributeNS(in DOMString namespaceURI,
                                        in DOMString localName)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  Attr                getAttributeNodeNS(in DOMString namespaceURI,
                                         in DOMString localName);
  // Introduced in DOM Level 2:
  Attr                setAttributeNodeNS(in Attr newAttr)
                                         raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList            getElementsByTagNameNS(in DOMString
        namespaceURI,
                                           in DOMString localName);
  // Introduced in DOM Level 2:
  boolean             hasAttribute(in DOMString name);
  // Introduced in DOM Level 2:
  boolean             hasAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName);
```

Intégration de XML dans un serveur XML

Utilisation de cocoon:
Serveur Web Apache
Tomcat et les servlets Java
Source : cocoon.apache.org

XML et le Web

Qui réalise les transformations ?



XML et le Web

- Exécution par le client
  - Demande un client « dernier cri »
  - Nécessite le téléchargement de l'intégralité des documents
    - Problèmes de débit et de sécurité
- Exécution par le serveur
  - Serveur Web classique construit dynamiquement

Intégration des outils XML dans un serveur Web

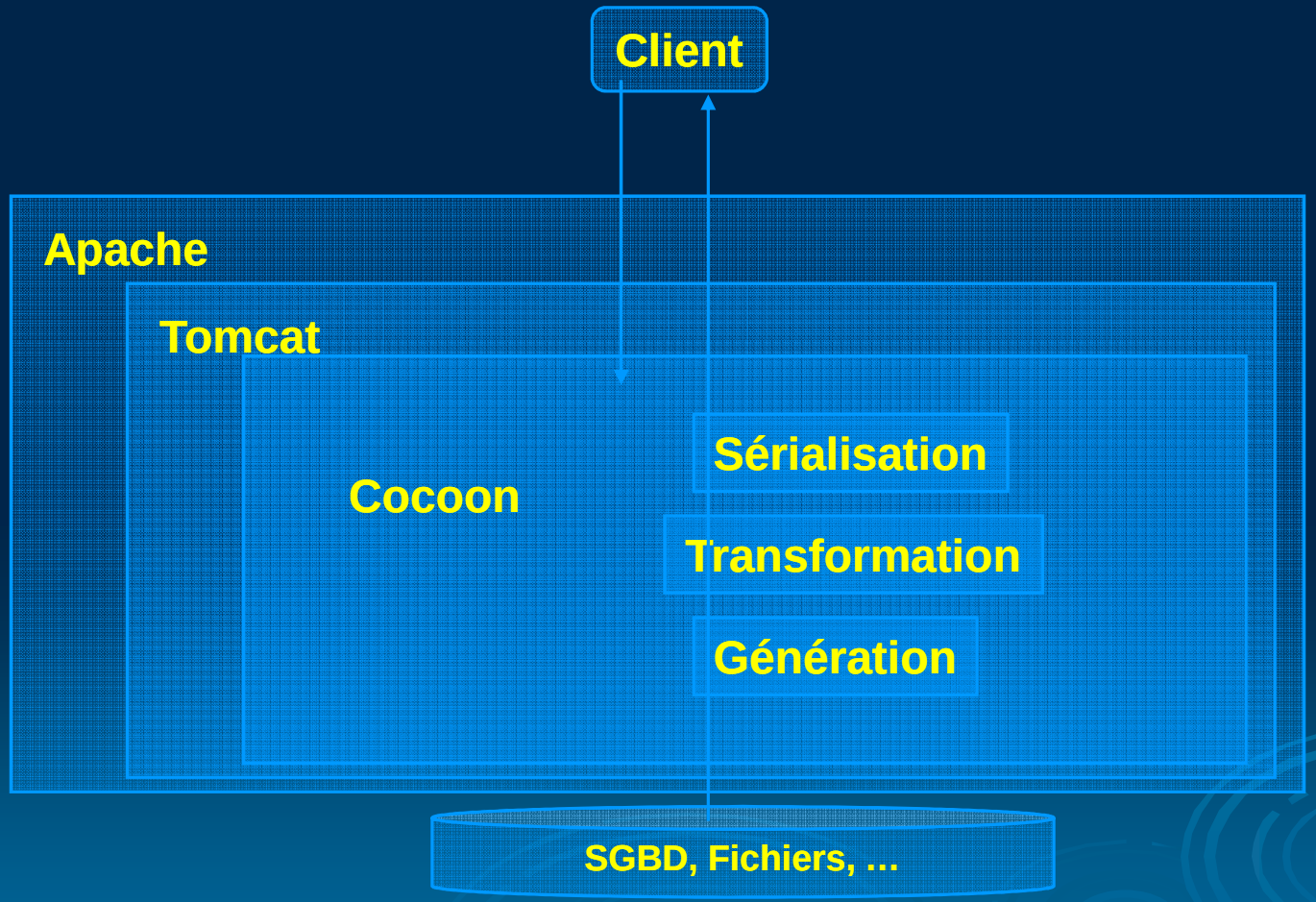
- Parseur : Xerces
- Transformation XSLT : Xalan
- Transformation de XML-FO en pdf : FOP
- …

FAIRE REALISER LES TRANSFORMATIONS PAR LE SERVEUR WEB

Les servlets Java

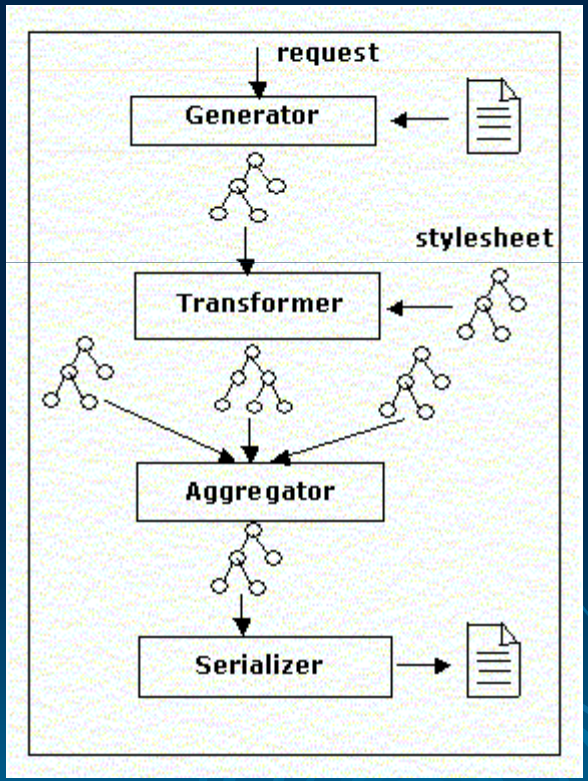
- http://java.sun.com/products/servlet/
- Programmes Java
  - Compilés à la volée
  - Exécutés par le serveur
- Partage d’une machine virtuelle
- Gestion d’un cache des applications
- Nécessite un serveur :
  - Apache Tomcat, JBOSS, …

Cocoon : une servlet Java



Cocoon fonctionnement

- Un utilisateur demande un document (virtuel ?) : Requête.
- Le document est produit :
  - A partir d'une ou plusieurs sources
  - En appliquant des transformations
  - Et une serialisation

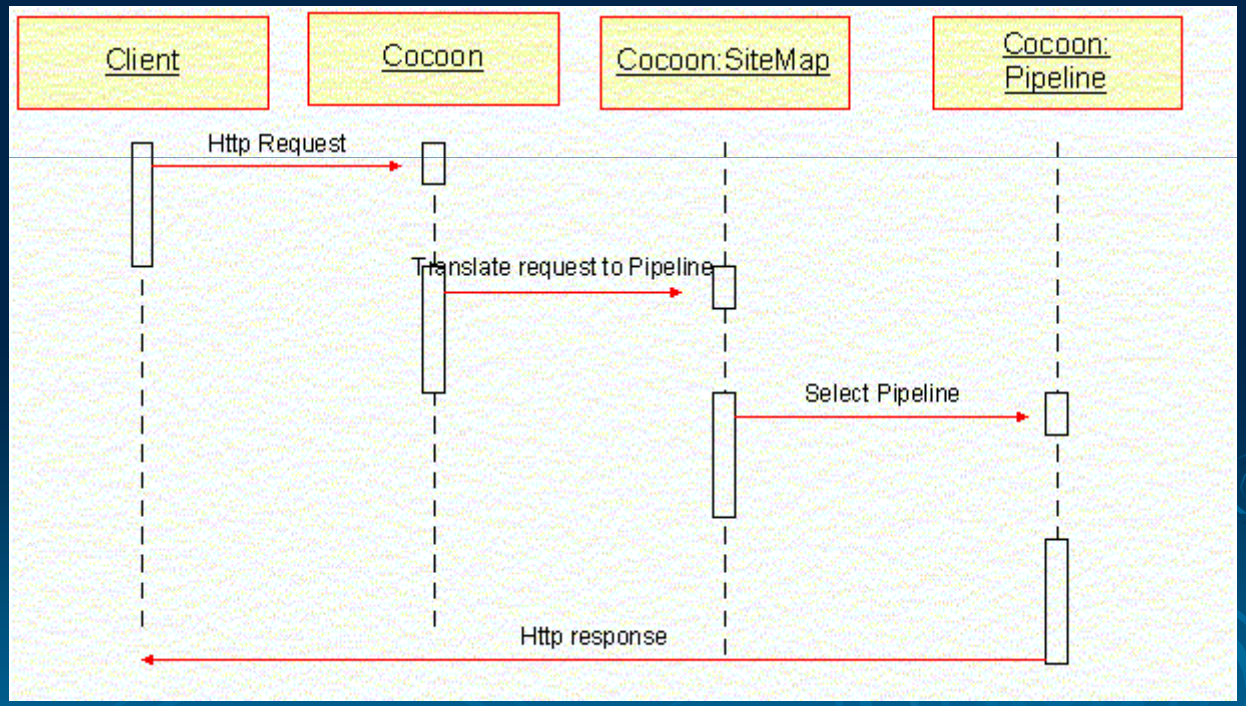


Composants de cocoon

- **Generators**
  - Production du **document initial** du pipe-line
  - file generator, directory generator, XSP generator, JSP generator,Request generator
- **Transformers**
  - Transformation **dans le pipe-line** d'un document en un autre
  - **XSLT Transformer, Log Transformer, SQL Transformer, I18N Transformer, ...**
- **Serializers**
  - Création du **rendu final** du document XML (pas forcément en XML)
  - HTML Serializer, FOP Serializer, Text Serializer, XML Serializer, ...

Le sitemap

- La description de la production du document est faite dans le **sitemap**



Sitemap

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>
...
</map:components>

<map:views>
...
</map:views>
<map:pipelines>
<map:pipeline>
<map:match>
...
</map:match>
...
</map:pipeline>
...
</map:pipelines>
...
</map:sitemap>
```

Une règle du sitemap

```
<map:match pattern="hello.html">

    <map:generate src="docs/samples/hello-page.xml"/>

    <map:transform src="stylesheets/page/simple-
    page2html.xsl"/>

    <map:serialize type="html"/>

</map:match>
```

Cocoon XSP

- Insertion de « code » dans les documents XML
- Utilisation de code existant les tag-lib
  - Balises XML traduites en code java
- La page entière est traduite en un programme java
  - Compilé à la volé
  - Exécuté pour produire un document

eXtensible Server Pages (XSPs) une page

```
<?xml version="1.0"?>

<xsp:page language="java" xmlns:xsp="http://apache.org/xsp">

  <xsp:logic>
  static private int counter = 0;
  private synchronized int count()
  {
    return counter++;
  }
  </xsp:logic>

  <page>
  <p>I have been requested <xsp:expr>count()</xsp:expr> times.</p>
  </page>

</xsp:page>
```

eXtensible Server Pages (XSPs) le code Java

```
package org.apache.cocoon.www.docs.samples.xsp;

import java.io.File;
// A bunch of other imports

public class counter_xsp extends XSPGenerator {
   // .. Bookkeeping stuff commented out.
  /* User Class Declarations */
  static private int counter = 0;
  private synchronized int count() {
    return counter++;
  }
  /* Generate XML data. */
  public void generate() throws SAXException {
    this.contentHandler.startDocument();
    AttributesImpl xspAttr = new AttributesImpl();
    this.contentHandler.startPrefixMapping("xsp", "http://apache.org/xsp");
    this.contentHandler.startElement("", "page", "page", xspAttr);
    // Statements to build the XML document (Omitted)
    this.contentHandler.endElement("", "page", "page");
    this.contentHandler.endPrefixMapping("xsp");
    this.contentHandler.endDocument();
  }
```

Exemples de tag-lib

- ESQL
  - Construction de documents XML à partir d'une requête SQL

```
<esql:connection>
  <esql:pool>connectionName</esql:pool>
  <esql:execute-query>
    <esql:query>SELECT mycolumn1,mycolumn2 FROM
    table</esql:query>
    <esql:results>
      <table>
        <esql:row-results>
          <tr>
            <td><esql:get-string column="mycolumn1"/></td>
            <td><esql:get-string column="mycolumn2"/></td>
          ...
```

- Forms
- Sendmail

Un exemple de XSP+XSLT

- Source + XSLT : ***sample.xsl***
  - ```
    <?xml version="1.0"?>
    <xsp:page language="java"
     xmlns:xsp="http://www.apache.org/1999/XSP/Core">
     <page title="Time of Day">
      <xsp:logic> Date now = new Date(); </xsp:logic>
       To the best of my knowledge, it's now
      <xsp:expr>now</xsp:expr> 
     </page>
    </xsp:page>
    ```
- Résultat
  - ```
    <html><head><title>Time of Day</title></head>
    <body><h3 style="color: navy; text-align: center">Time of
    Day</h3> <p>It's now Thu Dec 23 20:11:18 PST 1999</p>
    </body> </html>
    ```

Applications XML

Exemples de langages décrits avec XML

Mathematical Markup Language

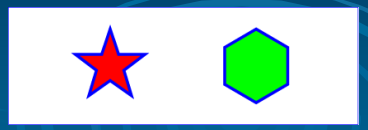
- Description d’expressions mathématiques:
  - Echange entre applications
  - Présentation

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
 <head>
  <title>MathML's Hello Square</title>
 </head>
 <body>
  <p> This is a perfect square:</p>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <mrow>
    <msup>
     <mfenced>
       <mrow> <mi>a</mi> <mo>+</mo> <mi>b</mi> </mrow>
     </mfenced>
     <mn>2</mn>
    </msup>
   </mrow>
  </math>
 </body>
</html>
```

Scalable Vector Graphics (SVG)

- Description d'images vectorielles en XML
- Interprétable (IE6, adobe plugin, mozilla)
- Rasterisable → PNG, JPEG

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN" "http://www.w3.org/TR/2001/REC-
SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example polygon01 - star and hexagon</desc>
  <rect x="1" y="1" width="1198" height="398" fill="none" stroke="blue" stroke-
width="2" />
  <polygon fill="red" stroke="blue" stroke-width="10" points="350,75 379,161
469,161 397,215 423,301 350,250 277,301 303,215 231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10" points="850,75 958,137.5
958,262.5 850,325 742,262.6 742,137.5" />
</svg>
```



Synchronized Multimedia Integration Language (SMIL)

- Description de présentations multimédia
- Synchronisation d’évènements

```
<par>
 <seq>
  <par>
   <img src="image1.jpg" region="foo1" fill="freeze" erase="never" .../>
   <audio src="audio1.au"/>
  </par>
  <par>
   <img src="image2.jpg" region="foo2" fill="freeze" erase="never" .../>
   <audio src="audio2.au"/>
  </par>
  ...
  <par>
   <img src="imageN.jpg" region="fooN" fill="freeze" erase="never" .../>
   <audio src="audioN.au"/>
  </par>
 </seq>
</par>
```

Exercices SVG

- Créer un fichier SVG qui représente un histogramme à 5 colonnes
- Créer un document XML et sa DTD qui représente les notes d'élèves dans cinq matières pour un nombre de contrôles variables (par élève et par matière)
  - Les descriptions des élèves et de leur notes par matière seront séparées
- Écrire les expressions XPath qui permettent d'extraire les informations importantes du documents XML
- Ecrire les feuilles de style XSLT qui permettent de transformer ce document XML en SVG pour le visualiser : courbes des notes, histogramme par matière, …
- Rendre les documents SVG dynamiques et animés, réaction à la souris pour afficher des informations supplémentaires, lien entre les vues, …

Projet 2004-2005

- Le sujet du projet est très libre, il s'agit de mettre en place un outils de gestion d'un agenda. Un agenda est principalement constitué d'un carnet de contacts, d'un calendrier et d'une liste de tâches.

- L'objectif pratique du projet est de mettre en application les concepts suivants
  - Représentation de données avec XML
  - Utilisation de schémas et DTD standards (XHTML, SVG,)
  - Production de documents XML à partir d'une base de données relationnelle
  - Transformation de documents XML avec XSLT
  - Définition de feuilles de style avec CSS
  - Construction dynamique d'images vectorielles interactives avec SVG
  - Intégration avec cocoon (Pages XSP, Gestion des formulaires, Gestion des session)
  - (Facultatif) Programmation en Java

Projet 2004-2005

- Vous devrez réaliser un portail web qui permettra à un ensemble d'utilisateurs de gérer leur agenda. Votre portail devra proposer deux accès
  - administrateur et utilisateur.
- Pour un administrateur : gestion (ajout, suppression, mot de passe) d'un ensemble d'utilisateurs via une base de données (l'utilisation des bases de données avec cocoon sera étudiée en TP).
- Pour un utilisateur :
  - Login
    - Une possibilité de gestion de sessions avec cocoon sera présentée en TP.
  - Accès à un environnement personnalisé pour :
    - Consulter, Ajouter, supprimer ou modifier des contacts.
    - Consulter, Ajouter, supprimer ou modifier des rendez-vous.
    - Consulter, Ajouter, supprimer ou modifier des tâches.
    - Visualiser une synthèse : prochain rendez-vous de la journée, tâches restantes, ...
  - La présentation sera réalisée en XHTML+CSS, le calendrier pourra aussi être présentée en SVG. Un possibilité d'imprimer le calendrier via pdf sera proposée.

Projet 2004-2005

- Dans un premier temps, vous pouvez :
  - Prévoir la structure du portail en utilisant UML.
  - Prévoir les formats XML internes pour représenter les contacts, les rendez-vous, les tâches. Il est conseillé de regarder les références suivantes :
    - Vcard et Vcal http://www.imc.org/pdi/,
    - une proposition de format XML http://www.w3.org/TR/vcard-rdf,
    - le format utilisé par jabber http://www.jabber.org/jeps/jep-0054.html.
  - Prévoir les transformations du format interne pour produire les parties du site.
  - Mettre en place les parties suivantes en fonction de l'avancement en TP
    - Représentation de données dans un SGBDR
    - Production dynamique des documents XML internes
    - Mises à jour des données
      - Utilisation de formulaires avec cocoon
      - Mise à jour de documents XML
    - Gestion des sessions utilisateur
- Pour l'exemple, il est conseillé de regarder le site http://www.plaxo.com.