Plan

- XPATH
- **XSL(T)**

Qu'est ce qu'une feuille de style ?

- Ensemble d'instructions qui contrôlent une mise en page d'un document
  - passage de la partie logique à la partie physique
  - possibilité d'utiliser différentes feuilles de style pour des rendus différents à partir d'une même source
    - papier, Web, téléphone…

Spécifications de feuilles de style

- DSSSL - Document Style and Semantics Specification Language
  - Standard lié à SGML pour la présentation et la conversion de documents
- CSS - Cascading Style Sheet
  - Syntaxe simple pour assigner des styles à des éléments XML (géré par les navigateurs web)
- XSL - Extensible Stylesheet Language
  - Combinaison des possibilité de DSSSL et CSS avec une syntaxe XML
    - une feuille de style XSL est un fichier XML

XSL

- Extensible Stylesheet Language
  - Transformer du XML vers un autre format
    - XML, HTML, texte…
    - pas seulement pour la présentation, mais aussi la conversion entre structures de données
- Pendant le développement de XSL, on s'est aperçu que XSL faisait deux choses différentes
  - définir des éléments pour présenter du contenu
  - définir une syntaxe pour transformer des éléments XML et des structures de documents
- XSL a donc été divisé entre
  - XSL – XML Stylesheet Language (XSL-FO)
  - XSLT – XSL Transformations

Possibilités de XSL/XSLT

- Rajouter du texte à du contenu
- Effacer, créer, réordonner et trier des éléments
- Réutiliser des éléments ailleurs dans le document
- Transformer des données entre deux formats XML différents
- Spécifier les objets de formatage (FO) à appliquer à chaque type d'élément
- Utiliser un mécanisme récursif pour explorer le document
- …

XSLT

- Langage de programmation déclaratif
- Décrire des transformations
  - d'un fichier (arbre) d'entrée
  - vers un fichier (arbre) de sortie
  - dans un document XML (lui-même un arbre)
- Description des transformations
  - modèles ou règles (templates) de transformation qui décrivent les traitements appliqués à un nœud
  - chaque modèle correspond à un motif (pattern) qui décrit des éléments auxquels il s'applique en utilisant Xpath
- Espace de nom spécifique

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

Spécifier une feuille de style

- Utiliser une instruction de traitement dans le prologue du document XML qui doit être transformé

```
<?xml-stylesheet
 href="le-style.xsl"
 type="application/xml+xsl" ?>
```

- Possibilité de mettre plusieurs choix
  - le processeur XSL choisira la feuille de style la plus adéquate

Spécification XSLT

- Disponible sur http://www.w3.org/TR/xslt
  - définit 34 éléments et leurs attributs
  - mais on peut faire se débrouiller en utilisant juste
    - **stylesheet**
    - **template**
    - **apply-templates**
    - **output**
- A connaître pour utiliser XSL
  - les espaces de nom (*namespaces*)
  - XPath

Élément de feuille de style

- L'élément racine est **stylesheet**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version = "1.0" >
      <xsl:template ... >
         <!-- traitements à effectuer -->
      </xsl:template >
      ...
      <xsl:template ... >
         <!-- traitements à effectuer -->
      </xsl:template >
</xsl:stylesheet>
```

Espace de nom XSL

Un premier exemple

```
<?xml version="1.0"?>
<xsl:stylesheet
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      version="1.0">
<xsl:template match="doc">
<out>Résultat : <xsl:value-of select="."/></out>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<doc>Hello</doc>
```

L'application de la feuille de style XSL au document XML de départ donne le document de sortie

```
<out>Résultat : Hello</out>
```


12 éléments de premier niveau

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:import href="..."/>
   <xsl:include href="..."/>
   <xsl:strip-space elements="..."/>
   <xsl:preserve-space elements="..."/>
   <xsl:output method="..."/>
   <xsl:key name="..." match="..." use="..."/>
   <xsl:decimal-format name="..."/>
   <xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>
   <xsl:attribute-set name="..."> ... </xsl:attribute-set>
   <xsl:variable name="...">...</xsl:variable>
   <xsl:param name="...">...</xsl:param>
   <xsl:template match="..."> ... </xsl:template> ou
   <xsl:template name="..."> ... </xsl:template>
</xsl:stylesheet>
```


Éléments import / include

- Pour composer une feuille de style à partir de plusieurs fichiers XSL

```
<xsl:stylesheet … >
  <xsl:import href="tables.xsl" />
  <xsl:import href="features.xsl" />
  <!-- ordre important, seul cas pour
       les éléments de premier niveau -->
<xsl:template … > … </xsl:template>
  …
</xsl:stylesheet>
```

- Inclure des fichiers XML : **xsl:include**
  - comportement équivalent à **xsl:import**
  - mais pas de possibilité d'écraser une définition importée par une définition de plus haut-niveau → erreur si deux définitions similaires

Élément output

- Pour spécifier le format de sortie

```
<xsl:output      method="xml"
                 indent="yes"
                 encoding="iso-8859-1" />
```

- Attributs de **output**
  - **method** : xml, html, text
  - **indent** : yes, no
  - **encoding**
  - **standalone** (si on génère du XML)
  - …

Principe du traitement XSLT

- Effectué sur une liste de noeuds
  - la liste initiale contient uniquement le nœud racine du document XML à traiter
- Pour chaque nœud de la liste
  - recherche de templates (formes, patterns) qui lui correspondent
  - exécution des templates
    - écriture sur la sortie
      - réécriture
    - mise à jour de la liste
      - appel de nouveaux templates, *etc.*
- Écrire une feuille de style = écrire des templates
  - plus ou moins complexes

Élément template

- Pour spécifier une règle de transformation

```
<xsl:template match="expression">
  …
</xsl:template>
```

- L'attribut `match` a pour valeur une expression Xpath (limitation aux axes child, attribute, descendant-or-self)
- Le résultat de cette expression est le noeud contextuel au sein du template
- On commence toujours par s'intéresser à la racine ("/" en Xpath).
- Remarque
  - si plus d'une réponse comme résultat de l'expression Xpath, il faut utiliser des règles de priorité pour déterminer quelle règle utiliser

Élément template (suite)

- Contenu de l'élément xsl:template
  - du **texte**, qui peut contenir des balises
    - ce texte est inséré dans l'arbre destination
      - ex. : "<out>Résultat : </out> »
  - des **instructions** qui décrivent des traitements à effectuer
    - le résultat de leur exécution sera inséré à leur plae dans l'arbre destination
      - ex. : <xsl:value-of select="." />
- Exemple

```
<xsl:template match="doc">
  <out>Résultat : <xsl:value-of select="."/></out>
</xsl:template>
```

Traduction : *à chaque fois que l'on trouve un élément doc, il faut écrire une chaîne, en y insérant le contenu de l'élément doc trouvé*

Quelques « éléments instructions » à mettre dans un élément template

- **xsl:apply-templates**
  - Signifie qu'on doit continuer à appeler les règles sur les éléments courants. L'attribut **select** permet de spécifier éventuellement l'élément
- **xsl:template**
  - Permet de charger/appeler un template spécifique (par son nom)
- **xsl:choose**
  - Structure conditionnelle de type "case" (utilisé en combinaison avec xsl:when et/ou xsl:otherwise)
- **xsl:if**
  - Permet d'effectuer un test conditionnel sur le modèle indiqué
- **xsl:comment**
  - Crée un commentaire dans l'arbre résultat
- **xsl:copy**
  - Copie le noeud courant dans l'arbre résultat (mais pas les attributs et enfants)
- **xsl:copy-of**
  - Copie le noeud sélectionné et ses enfants et attributs
- **xsl:element**
  - Crée un élément avec le nom spécifié
- **xsl:for-each**
  - Permet d'appliquer un canevas à chaque noeud correspondant au modèle

Élément apply-templates

- Indique au processeur XSL de traiter les éléments enfants directs des éléments courants en leur appliquant les règles définies dans la feuille XSL
  - « continuer le traitement sur les enfants »
- Traitement récursif

```
<p>C'est <b>très</b> important</p>
---------------------------
<xsl:template match="p">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="b">
  <xsl:apply-templates/>
</xsl:template>
```

Élément apply-Templates (2)

- Autre exemple

```
<xsl:template match="livre">
  <html:p>
    Un livre : <xsl:apply-templates/>
  </html:p>
</xsl:template>
```

- Remarque
  - On ne peut pas ré-arranger la structure hiérachique d'un document XML source (le document XSL serait mal formé)

```
<xsl:template match="firstname">
  <html:p><xsl:apply-templates/>
</xsl:template>
<xsl:template match="lastname">
  <xsl:apply-templates/></html:p>
</xsl:template>
```


apply-Templates : attribut **select**

- L'attribut select permet de spécifier certains éléments enfants auxquels la transformation doit être appliquée
  - plus spécifique que <xsl:apply-templates />
- Utilisation de patterns Xpath pour sélectionner les enfants

```
<xsl:template match="elt-pere">
<xsl:apply-templates
        select="elt-fils[@type='title']"/>
</xsl:template>
```

- Remarque
  - plusieurs éléments possèdent cet attribut
    - **apply-templates, value-of, copy-of, param, sort, variable, with-param**

Élément xsl:value-of

- Pour convertir l'objet spécifié par un attribut '**select**' en une chaîne de caractères
- Non récursif

```
<p>A <b>hidden</b> word</p>
----------------------------
<xsl:template match="p">
  <e><xsl:value-of select="."/></e>
</xsl:template>
<xsl:template match="b">
  <xsl:value-of select="."/>
</xsl:template>
```

donnera

```
<e>A hidden word.<e>
```


Valeurs d'attributs

- Utiliser l'élément **xsl:value-of** avec un attribut **select**

```
<full-name first="John" second="Smith"/>
----------------------------------------
<xsl:template match="full-name">
  <person>
    <xsl:value-of select="@first"> +
    <xsl:value-of select="@second">
  </person>
  <person name="{@first} {@second}" />
</xsl:template>
----------------------------------------
<person>John + Smith</person>
<person name="John Smith"/>
```


Règles par défaut : racine/éléments

- Quand aucune règle n'est sélectionnée, XSLT applique des règles par défaut
- Première règle par défaut
  - pour les éléments et la racine du document.

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

  - on demande l'application de règles pour les fils du noeud courant
  - conséquence
    - Pas obligatoire de faire une règle pour la racine du document à transformer

Règles par défaut : texte et attributs

- Par défaut, on insère dans le document résultat la valeur du noeud Text, ou de l'attribut.
- Deuxième règle par défaut

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

- Conséquence
  - Si on se contente des règles par défaut, on obtient la concaténation de noeuds de type text()
    - par défaut, les nœuds attributs ne sont pas considérés comme enfants
- Programme minimal :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform" />
```

Règles par défaut : autres nœuds

- Pour les instructions de traitement et les commentaires : on ne fait rien.
- Troisième règle par défaut

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

- Si on ne les sélectionne pas explicitement, en définissant une règle pour les traiter, il ne se passe rien.

Élément sort

- Permet de spécifier que les éléments sont triés suivant une certaine propriété

```
<list>
  <item>ZZZ</item>
  <item>AAA</item>
  <item>MMM</item>
</list>
```

```
<xsl:template match="list">
  <xsl:apply-templates>
    <xsl:sort/>
  </xsl:apply-templates>
</xsl:template>
```

AAABBBZZZ

bbbcccaaa

```
<list>
  <item code="Z">aaa</item>
  <item code="A">bbb</item>
  <item code="M">ccc</item>
</list>
```

```
<xsl:template match="list">
  <xsl:apply-templates>
    <xsl:sort select="@code"/>
  </xsl:apply-templates>
</xsl:template>
```


Attributs de l'élément sort

- Attribut '**order**'
  - pour classer croissant ou décroissant
    - '**ascending**' ou '**descending**'
- Attribut '**data-type**'
  - pour indiquer si les données à prendre en compte sont une simple chaîne ou doivent être interprétées comme des nombres
    - '**text**' (par défaut) ou '**number**'
- Attribut '**case-order**'
  - ordre majuscules / minuscules
    - '**lower-first**' ou '**upper-first**'

Élément number

- Pour la numérotation automatique

```
<xsl:template match="item">
  <xsl:number/><xsl:apply-templates/>
</xsl:template>
```

- Attributs
  - **level** = '**single**' ou '**any**' ou '**multiple**'
  - **count** = "**list-1|list-2**"
  - **format** = "**1.A**" (également "**I**" et "**i**")
  - **from** = "**3**"
  - **grouping-separator** = "**,**"
  - **grouping-size** = "**3**"
  - **value** = "**position()**"

Attribut mode

- Attribut de l'élément template
- Permet de spécifier quelle règle utiliser en fonction de l'élément retrouvé

```
<xsl:template match="chapter/title">
  <html:h1><xsl:apply-templates/></html:h1>
</xsl:template>

<xsl:template match="chapter/title" mode="h3">
  <html:h3><xsl:apply-templates/></html:h3>
</xsl:template>

<xsl:template match="intro">
  <xsl:apply-templates
      select="//chapter/title" mode="h3"/>
</xsl:template>
```

Spécifie le mode à utiliser

Élément variable

- On peut déclarer et utiliser des variables en XSLT
  - `<xsl:variable name="colour">red</xsl:variable>`
  - définition de la variable colour avec valeur red
- Une variable est référencée avec la notation $
  - `<xsl:value-of select="$colour"/>`
- On peut aussi l'utiliser dans les éléments de sortie
  - `<ajr:glyph colour="{$colour}"/>`

Appel explicite de templates

- Si on a besoin plusieurs fois du même formatage
  - on nomme le template pour pouvoir l'appeler

```
<xsl:template name="CreateHeader">
  <html:hr/>
  <html:h2>***<xsl:apply-templates/>***</html:h2>
  <html:hr/>
</xsl:template>
...

<xsl:template match="title">
  <xsl:call-template name="CreateHeader" />
</xsl:template>

<xsl:template match="head">
  <xsl:call-template name="CreateHeader" />
</xsl:template>
```


Passer des paramètres à un template

- L'élément **param**, une variable spéciale
  - `<xsl:param name="nom">valeur par defaut</xsl:param>`
  - `<xsl:with-param name="nom">nouvelle valeur</xsl:with-param>`
- L'élément **call-template** peut passer une nouvelles valeur de **param** à un **template**

```
<xsl:template match="name">
  <xsl:call-template name="salutation">
    <xsl:with-param name="greet">Hello </xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="salutation">
  <xsl:param name="greet">Dear </xsl:param>
  <xsl:value-of select="$greet"/>
  <xsl:apply-templates/>
</xsl:template>
```

remplacera la valeur par défaut

valeur par défaut

Créer des éléments

- Utiliser l'élément '**Element**'
- Très puissant si on l'utilise avec des variables

```
<xsl:template name="CreateHeader">
  <xsl:param name="level">3</xsl:param>
  <xsl:element namespace="html" name="h{$level}">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<xsl:template match="title">
  <xsl:call-template name="CreateHeader">
    <xsl:with-param name="level">1</xsl:with-param>
  </xsl:call-template>
</xsl:template>
```


Copier des éléments

- Élément '**copy**'
  - copie le nœud courant (mais pas les fils et les attributs)

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
  <xsl:copy>
    Header: <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

- Pour créer de nouveaux attributs : xsl:attribute

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
  <xsl:copy>
    <xsl:attribute name="style">purple</xsl:attribute>
    Header: </xsl:apply-templates />
  </xsl:copy>
</xsl:template>
```

Crée des éléments copiés avec un attribut style qui vaut purple
Ex. <h3 style="purple">

Élément attribute-set

- Utilisé pour stocker des groupes d'attributs

```
<xsl:attribute-set name="class-and-color">
 <xsl:attribute name="class">standard</xsl:attribute>
 <xsl:attribute name="color">red</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy>
 <xsl:use-attribute-sets name="class-and-color" />
  Header: <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
```


Élément copy-of

- Peut copier des fragments du fichier d'entrée sans perdre les attributs

```
<xsl:template match="body">
  <body>
    <xsl:copy-of select="//h1 | //h2" />
    <xsl:apply-templates/>
  </body>
</xsl:template>
```

Élément for-each

- Pour répéter une opération sur des éléments

```
<xsl:template match="liste">
  <xsl:for-each select="./item">
  <!-- traitement pour chaque item -->
</xsl:template>
```


Conditions

- On peut faire un test '**if**' pendant le traitement

```
<xsl:template match="para">
 <html:p>
  <xsl:if test="position() = 1">
   <xsl:attribute name="style">color: red</xsl:attribute>
  </xsl:if>
  <xsl:if test="position() > 1">
   <xsl:attribute name="style">color: blue</xsl:attribute>
  </xsl:if>
  <xsl:apply-templates/>
 </html:p>
</xsl:template>
```


Conditions (2)

- Les éléments '**choose**', '**when**', '**otherwise**'

```
<xsl:template match="para">
 <html:p>
  <xsl:choose>
   <xsl:when test="position() = 1">
    <xsl:attribute name="style">color: red</xsl:attribute>
   </xsl:when>
   <xsl:otherwise test="position() > 1">
    <xsl:attribute name="style">color: blue</xsl:attribute>
   </xsl:otherwise>
   <xsl:apply-templates/>
  </xsl:choose>
 </html:p>
</xsl:template>
```

optionnel

Un exemple XSL-FO (Formatting Objects)

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.w3.org/1999/XSL/Format"
font-size="16pt">
<layout-master-set>
<simple-page-master
   margin-right="15mm" margin-left="15mm"
   margin-bottom="15mm" margin-top="15mm"
   page-width="210mm" page-height="297mm"
   master-name="bookpage">
<region-body region-name="bookpage-body"
   margin-bottom="5mm" margin-top="5mm" />
</simple-page-master>
</layout-master-set>
<page-sequence master-reference="bookpage">
<title>Hello world example</title>
<flow flow-name="bookpage-body">
<block>Hello XSLFO!</block>
</flow>
</page-sequence>
</root>
```


Conclusion

- XSLT permet de transformer des arbres en d'autres arbres
  - changement de modèle de données
    - d'un fichier XML valide suivant une DTD à un autre, valide suivant une autre DTD
  - présentation
    - surtout en XHTML pour visualisation dans un navigateur

Exercice (suite en TP)

Ecrire une feuille de style XSLT permettant de passer du document carte1.xml à card1.xml

carte1.xml

```
<carte>
  <titre>Dr.</titre>
  <nom>Paul Durand</nom>
  <telephone inter="33">4 78 34 25
   12</telephone>
  <telephone inter="33">6 12 45 25
   12</telephone>
  <adresse>
   <rue>Impasse des Fleurs</rue>
   <code>69001</code>
   <ville>Lyon</ville>
   <pays>France</pays>
  </adresse>
  <courriel>
paul.durand@provider.com</courriel>
</carte>
```

card1.xml

```
<card>
  <name title="Dr.">
    Paul Durand</name>
  <address>
    <street>Impasse des
   Fleurs</street>
    <zipcode>69001 Lyon</zipcode>
    <country>France</country>
  </address>
  <phones>
   <phone>(33)4 78 34 25 12</phone>
   <phone>(33)6 12 45 25 12</phone>
   </phones>
</card>
```

Remerciements

- Ce cours s'appuie largement sur celui d'Alan Robinson
  http://industry.ebi.ac.uk/~alan/XMLWorkshop/
- Cours Bernd Ammann programmation XSLT