XPATH – XSLT

Yannick Prié
UFR Informatique – Université Lyon 1

UE2.2 – Master SIB M1 – 2004-2005

Objectif du cours

- Xpath
  - syntaxe permettant de désigner des informations dans un arbre XML
  - sous la forme de chemins (*paths*)
- XSL – XSLT
  - expression en XML de transformations à opérer sur un arbre XML
    - transcodage d'un document XML vers un autre
    - présentation de documents XML aux utilisateurs

Plan

- **XPATH**
- XSLT

XPath

- Spécification W3C
- Version actuelle : 1.0 (16/11/1999)
- Objectif :
  - localiser des documents / identifier des sous-structures dans ceux-ci
- Utilisé par d'autres spécifications XML
  - XPointer, XQuery, XSLT, ...

Localisation de documents XML

- La localisation du document XML est prise en charge par une URL
- Uniform resource locator
  protocole://adresse/chemin
- Ressources locales
  file:///2004-2005/XML-CM3.ppt
- Ressources distantes
  http://www.univ-lyon1.fr/
  http://www710.univ-lyon1.fr/~yprie/Enseignement/SIB/SIB-UE3-bloc4/CM4.6-7.pdf

Exemples d'utilisations

- Ressources locales

```
<!ENTITY ent SYSTEM
  "file:///c:/ents/ent.xml">
<!ENTITY ent SYSTEM "../folder/ent.xml">
```

- Ressources distantes

```
<!ENTITY ent SYSTEM
  "http://w3c.org/ents/ent.xml">
<!ENTITY ent SYSTEM
  "ftp://w3c.org/gifs/pic.gif">
```

Repérer des fragments XML

- Avec les identificateurs XML
  - Utilisation des attributs identificateurs uniques (type d'attribut ID)

```
<link href="../files/detail.xml#part3">
  Link to Part 3 </link>
```

```
…                                   DTD
<!ELEMENT el …>
<!ATTLIST el label ID #REQUIRED>
…
```

```
…
<el label="part3">
…                                   xml
```


Les limites des identificateurs

- Les identificateurs ne sont pas suffisants
  - donner un identificateur unique à chaque élément peut être pénible
  - l'identité des éléments peut ne pas être connue
  - les identificateurs ne peuvent pas identifier des fragments de texte
  - il peut être peu pratique d'identifier des objets en listant tous leurs identificateurs

Contexte et éléments XML

- La signification d'un élément peut dépendre de son contexte
  - `<book><title>…</title></book>`
    `<person><title>…</title></person>`
- Supposons que l'on cherche le titre d'un livre, pas le titre d'une personne
- Idée
  - exploiter le contexte séquentiel et hiérarchique de XML pour spécifier des éléments par leur contexte (i.e. leur position dans la hiérarchie)
  - exemple : `book/title ≠ person/title`

Xpath : principe général

- Décrire un modèle de chemin dans un arbre XML
  → expression
- Recupérer les nœuds qui répondent à ce chemin en utilisant l'expression
  → résultat de l'application de l'expression à l'arbre XML
- Une expression sera utilisée et appliquée au sein de différentes syntaxes
  - URL : `http://abc.com/getQuery?/book/intro/title`
  - XSL : `<xsl:pattern match="chapter/title">…</xsl:pattern>`
  - Xpointer :

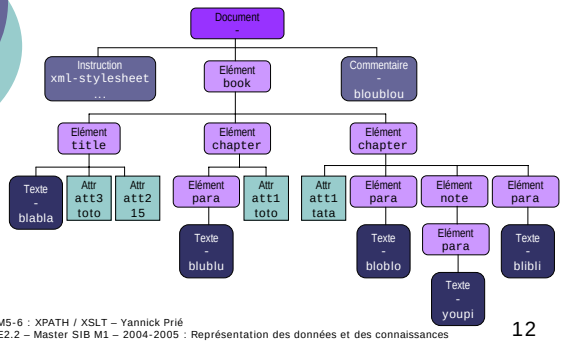
```
<link href="./doc.xml#xptr(book/intro/title)">
  Link to introductory title
</link>
<!-- désignation comme valeur de l'attribut href de
l'élément titre situé dans l'élément intro, situé dans
l'élément book, éléments qui se trouvent dans le fichier
doc.xml, lui-même situé dans le dossier où se trouve le
fichier XML en cours -->
```


Document/arbre/nœuds Xpath

- Dans XML
  - nœud = élément = noeud de l'arbre XML
- Dans Xpath, extension :
  - l'arbre contient toutes les information repérables dans ses noeuds de différentes types
    - Document = noeud racine
    - Noeuds éléments
    - Noeuds attributs
    - Noeuds textes
    - Noeuds instructions de traitement
    - Noeuds commentaires
    - (Noeuds espaces de nom)

(adapté d'après le cours de T. Accary)

Exemple de référence



Version XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="fichier.xsl" type="text/xsl"?>
<book>
<title att3="toto" att2="15">blabla</title>
<chapter att1="toto">
<para>blublu</para>
</chapter>
<chapter att1="tata">
<para>bloblo</para>
<note>
<para>youpi</para>
</note>
<para>blibli</para>
</chapter>
</book>
<!-- bloublou -->
```


Chemin de localisation

- Les expressions identifient des noeuds par leur position dans la hiérarchie XML
- Permet de
  - monter/descendre dans la hiérarchie de l'arbre XML
  - aller voir les voisins (frères) d'un noeud
  - *en fait : suivre des axes*
- Un chemin peut être
  - relatif
  - absolu

Chemins relatifs

- On se place dans le contexte d'un noeud
- A partir de là, on explore l'arbre XML, et on garde les noeuds qui vérifient l'expression
- Exemple
  - **para** (ou **child::para**) sélectionnera les fils du noeud courant qui ont le nom '**para**'

```
<chapter>            //noeud courant
  <para>…</para>     //Sélectionné
  <note>
    <para>…</para>   //Non sélectionné
  <note>
  <para>…</para>     //Sélectionné
</chapter>
```


Chemins absolus

- Expression identique aux chemins relatifs, mais
  - tout chemin absolu commence par '**/**'
  - signifie qu'on part de l'élément racine
- Exemple
  - Trouver tous les éléments '**para**' dans un document
    - **//para**
    - **/descendant-or-self::node()/para**

Chemins à plusieurs étapes

- Séparer les étapes par des '**/**'
- Exemple
  - **book/title** (version courte)
  - **child::book/child::title** (version longue)
  - Depuis le noeud courant, on sélectionne d'abord book, qui devient le contexte courant, puis on sélectionne `title`

Notion d'étape Xpath

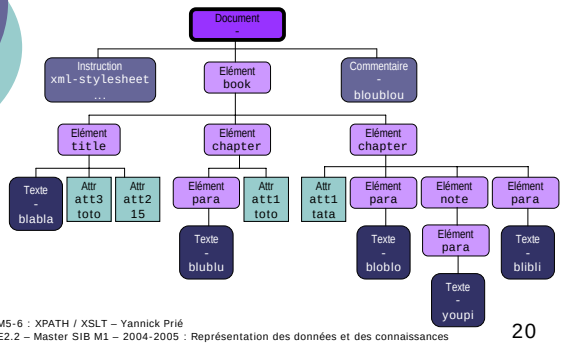
- Trois composants :
  Axe :: Filtre [Prédicat]
  - Axe
    - sens de parcours des nœuds
  - Filtre
    - type des nœuds retenus
  - Prédicats (on peut les enchaîner)
    - propriétés satisfaites par les nœuds retenus
- Exemple
  - **child::A[@att1 = "val1"]**
- Remarques
  - Il existe une syntaxe bavarde (verbose) et une syntaxe raccourcie, plus pratique
  - Possibilité de multiples expressions séparées par '|'

Evaluation d'une expression Xpath

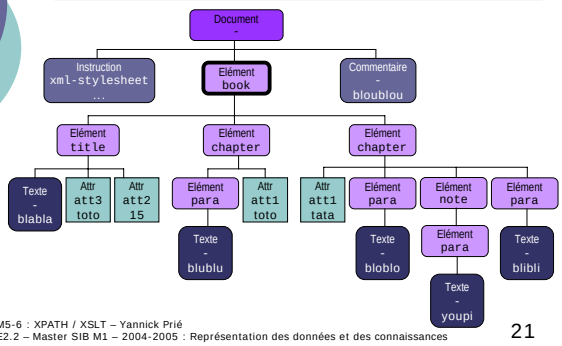
- On part du nœud contexte (ou de la racine), on évalue l'étape 1 : récupération d'un ensemble de nœuds
- Pour chacun de ces nœuds
  - il devient le nœud contexte
  - on évalue l'étape 2 : récupération d'un ensemble de nœuds
    - Pour chacun de ces nœuds
      - …

`/child::book/child::chapter/attribut::att1`

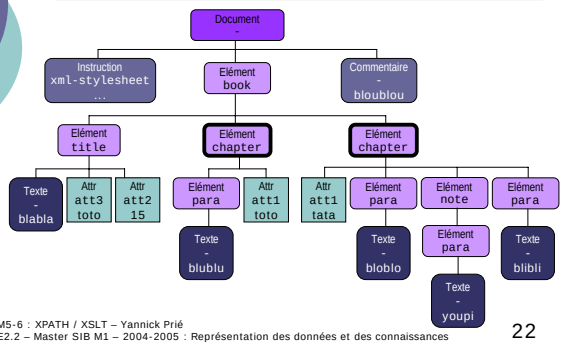
`/book/chapter/@att1` : nœud initial



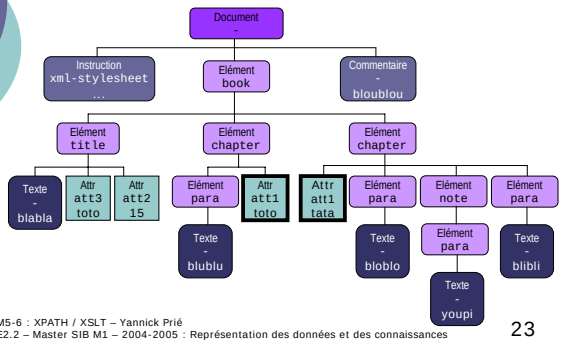
`/book/chapter/@att1` : Etape 1



`/book/chapter/@att1` : Etape 2



`/book/chapter/@att1` : Etape 3

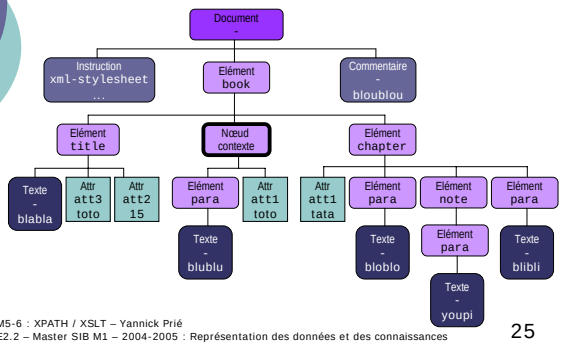


Axe :: Filtre [Prédicat]

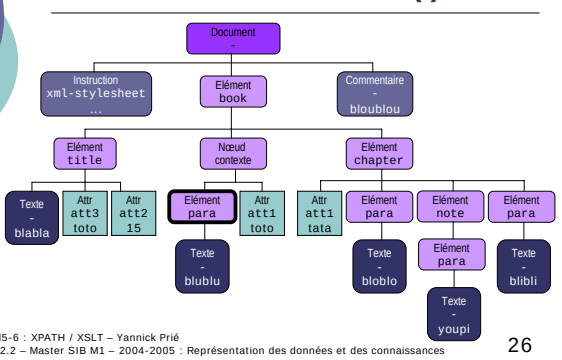
Axes : directions à suivre

- self:: (abrégé : .)
- child:: (abrégé : rien)
- attribut:: (abrégé :@)
- parent:: (abrégé : ..)
- descendant:: (abrégé : //)
- ancestor::
- ancestor-or-self::
- following::
- following-sibling::
- preceding::
- preceding-sibling::

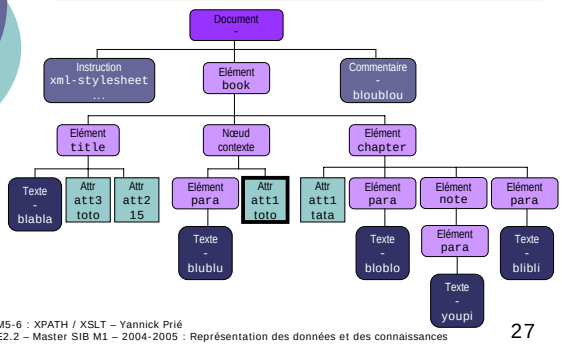
Axe self : `self::node()`



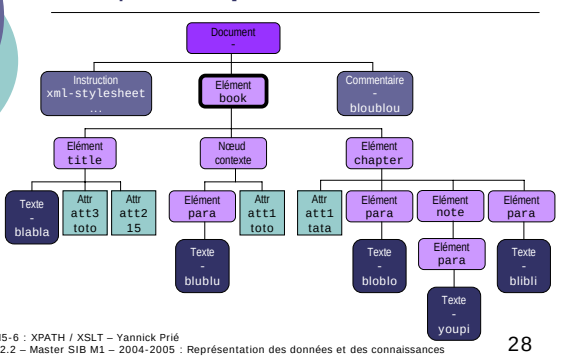
Axe child : `child::node()`



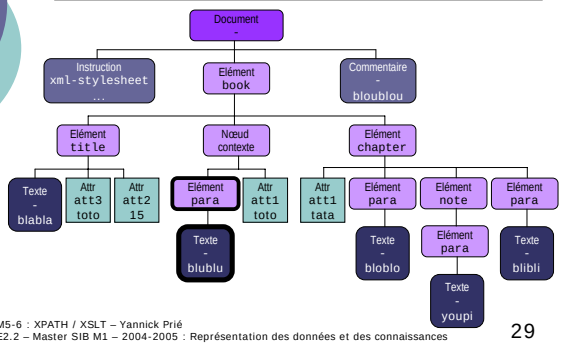
Axe attribute : `attribute::att1`



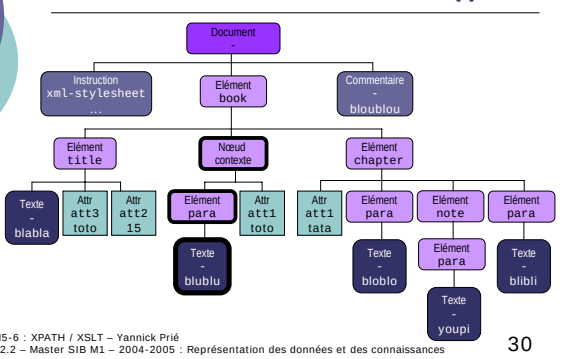
Axe parent : `parent::book`



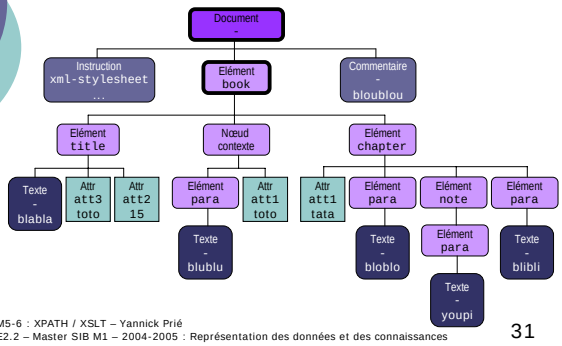
Axe descendant : `descendant::node()`



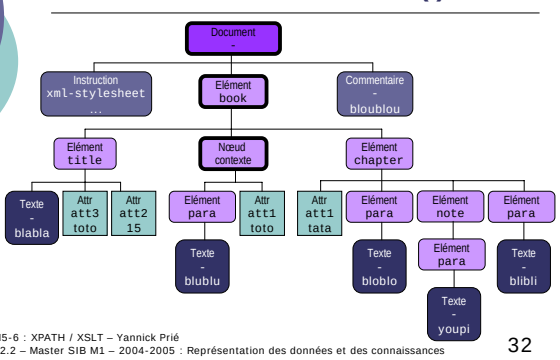
Axe descendant-or-self : `descendant-or-self::node()`



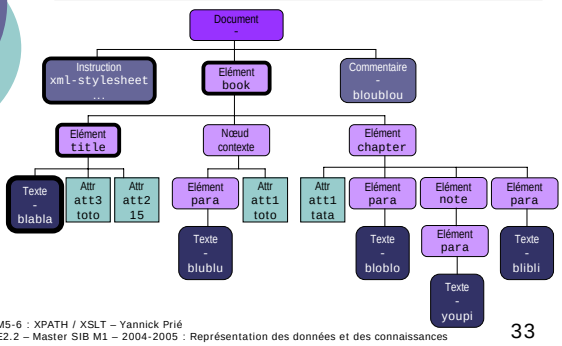
Axe ancestor : `ancestor::node()`



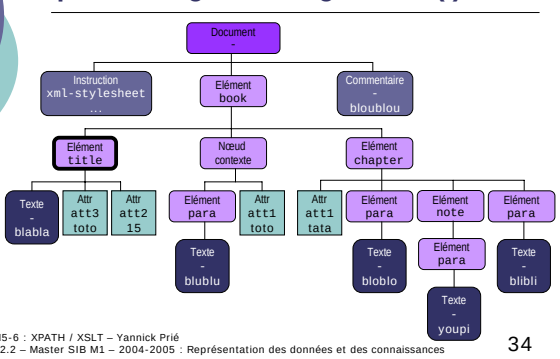
Axe ancestor-or-self : `ancestor-or-self::node()`



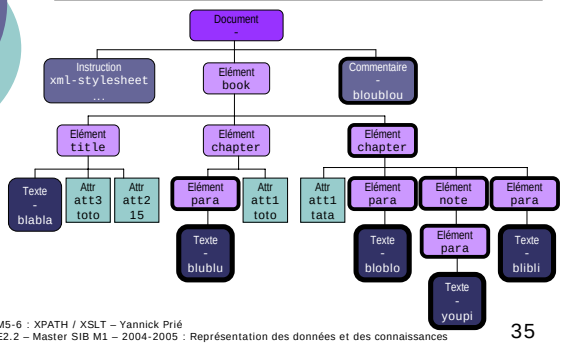
Axe preceding : `preceding::node()`



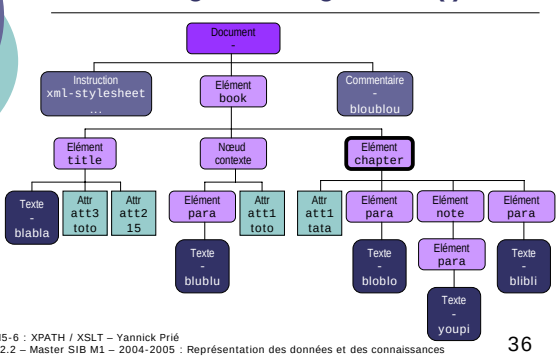
Axe preceding-sibling : `preceding-sibling::node()`



Axe following : `following::node()`



Axe following-sibling : `following-sibling::node()`



Filtres

- Filtrage par le nom
  - Éléments
  - Attributs
  - Instructions de traitement
- Filtrage par le type

Filtrage par le nom

- Nom connu
  - /book/chapter/note
- Nom inconnu
  - Utiliser le joker '*****' pour tout élément simple
    - **A/*/B** permet de trouver **A/C/B** et **A/D/B**
    - Version longue : **child::***
  - Utilisation de plusieurs astérisques, plusieurs niveaux de correspondance
    - attention à contrôler ce qu'il se passe
      - nombre de niveaux
      - éléments trouvés

Filtrage par le type

- text()
  - Contenu textuel
- comment()
  - Commentaire
- processing-instruction()
  - Instruction de traitement
- node()
  - Tout type de noeud

Quelques exemples

- **chapter//para** (noeud contexte = book)
  child::chapter/descendant-or-self::node()/child::para
- **.//para** (noeud contexte = book)
  self::node()/descendant-or-self::node()/child::para
- **../title** (noeud contexte = chapter)
  parent::node()/child::title
- **note|/book/title** (noeud contexte = 2ème chapter)
- **./*** (noeud contexte = book)
- /comment()
- **./para/text()** (noeud contexte = book)
- **/descendant::node()/@att2**

Filtres avec prédicats

- Les chemins de localisation ne sont pas forcément assez discriminants
  - peuvent fournir une liste de noeuds
- On peut filtrer ces listes avec des prédicat
  - le prédicat est indiqué entre crochets '**[ ]**'
- Le prédicat le plus simple utilise la fonction **position()**
  - **para[position() = 1]** `//1er para`
  - **item[2]** `//2eme item`
- Possibilité de combiner les tests avec '**and**' et '**or**'
  - ***[self::para and self::item]**

Tests sur les positions / texte

- **last()**
  - Récupère le dernier frère dans la liste
- **count()**
  - Evalue le nombre d'items dans la liste
    child::chapter[count(child::para) = 2]
- **id(…)**
  - Récupère l'identificateur de l'élément
    child::chapter[id("chap1")]
- **string(…)**
  - Récupère le texte d'un élément en enlevant toutes balises

Exemples

- **/book/chapter[@att1]**
  - les nœuds chapter qui ont un attribut att1
- **/book/chapter[@att1='tata']**
  - les nœuds chapter qui ont un attribut att1 valant 'tata'
- **/book/chapter/descendant::text()[position()=1]**
  - Le premier nœud de type Text descendant d'un /book/chapter
  - S'abrège en /book/chapter/descendant::text()[1]
- **/book/chapter[count(para)=2]**
  - Les nœuds chapter qui ont deux enfants de type para
- **child::chapter[note]**
  - Les nœuds chapter qui ont de enfants note

Prédicat : divers

- Pour les booléens
  - not(), and, or
- Pour les numériques
  - <, >, !=
  - +, -, *, div, mod
  - number() pour essayer de convertir
  - autres opérateurs : **round(), floor(), ceiling()**
- Exemples
  - **para[not(position() = 1)]**
  - **para[position() = 1 or last()]**
  - **//node()[number(@att2) mod 2 = 1]**

Tests sur les chaînes

- Possibilité de tester si les chaînes contiennent des sous-chaînes
  - **<note>hello there</note>**
    - **note[contains(text(), "hello")]**
  - **<note>hello there</note>**
    - **note[contains(., "hello")]**
  - '.' est le noeud courant, et parcourera tous les enfants

Tests sur les chaînes (2)

- **starts-with(*string, pattern*)**
  - **note[starts-with(., "hello")]**
- **string(*exp*)**
  - **note[contains(.,string("pi"))]**
- **string-after(*string, terminator*)**
- **string-before(*string, terminator*)**
- **substring(*string, offset, length*)**

Tests sur les chaînes (3)

- **normalize(*string*)**
  - enlève les espaces en trop
- **translate(*string, source, replace*)**
  - **translate(., ";+", ",")**
- **concat(*strings*)**
- **string-length(*string*)**

Encore des exemples

- **/book/chapter/child::para[child::note or text()]**
  - Tout élément para fils de chapter ayant au moins un fils note ou un fils text
- **/descendant::chapter[attribute::att1 or @att2]**
  - Tout élément chapter ayant un attribut att1 ou att2
- **//*[note]**
  - Tout élément ayant un fils note
- ***[self::note or self::p]** (dans le contexte de chapter)
  - Tout élément note ou p fils du nœud contexte

Conclusion

- Xpath permet de retrouver toutes sortes d'information dans les documents XML
  - requêtes
  - transformation : lire une information sous une forme, l'écrire sous une autre forme → XSL/XSLT

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 (XML-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 formattage (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 (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="text/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**
  - Nécessaires pour utiliser XSL
    - les espaces de nom (*namespaces*)
    - XPath

Elément 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

- **foo.xml:**

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

- **foo.xsl:**

```
<?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>
```

- **foo.out:**

```
<out>Résultat : Hello</out>
```


Eléments 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>
```

Eléments import / include

- Importer plusieurs fichiers XSL

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

- Inclure des fichiers XML
  - 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

Elément output

- Spécifie le format de sortie

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

- Attributs de **stylesheet**
  - **method** : xml, html, text
  - **indent** : yes, no
  - **encoding**
  - **standalone** (si on a choisit xml)
  - ...

Elément template

- Spécifie une règle de transformation

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

- L'attribut match a pour valeur une expression Xpath (limitation aux axes child, attribute, //)
- 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

Elément template (suite)

- Contient
  - du texte, des balises
    - Exemple : <out>Résultat : </out>
  - des instructions
    - la description des traitements à effectuer
    - Exemple : <xsl:value-of select="."/>
- Le contenu est inséré dans l'arbre destination
- Le résultat des instructions est inséré à la place de celles-ci dans l'arbre destination

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


Quelques instructions dans template

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

Elément apply-templates

- Indique au processeur XSL de traiter les éléments enfants directs en leur appliquant les règles définies dans la feuille XSL.
- 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>
```

Elément apply-Templates (2)

- Autre exemple

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

- Remarque
  - On ne peut pas ré-arranger la structure hiérachique d'un document XML source (le document 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
- Utiliser les patterns Xpath permet de sélectionner des enfants spécifiques

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

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

Elément 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">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="b">
  <xsl:value-of select="."/>
</xsl:template>
```


Valeurs d'attributs

- Utiliser l'élément '**value-of**'
- Les attributs sont identifiés par le préfixe '@'
  -

```
<full-name first="John" second="Smith"/>
----------------------------------------

<xsl:template match="full-name">
  <ajr:person>
    <xsl:value-of select="@first">
    <xsl:value-of select="@second">
  </ajr:person>
  <ajr: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 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 /

Règles par défaut : texte et attributs

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

```
<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.
- Programme minimal :

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

Règles par défaut : autres nœuds

- Pour les instructions de traitement et les commentaires : on ne fait rien.

```
<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.

Elément 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>
```

```
<list>
  <item code="Z">…</item>
  <item code="A">…</item>
  <item code="M">…</item>
</list>
```

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


Elément sort (2)

- Attribut '**order**'
  - '**ascending**' ou '**descending**'
- Attribut '**data-type**'
  - '**text**' (par défaut) ou '**number**'
- Attribut '**case-order**'
  - '**lower-first**' ou '**upper-first**'

Elément 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

- Fait partie de l'élément template
- Permet de spécifier quelle règle utiliser

```
<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>
```


Elément variable

- On peut déclarer et utiliser des variables en XSLT
  - `<xsl:variable name="colour">red</xsl:variable>`
  - On peut utiliser aussi `<value-of select="attribute">`
- 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}"/>`

Réutiliser des templates

- Si on a besoin plusieurs fois du même formattage

```
//A named template called by call-template
<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="prefix">default</xsl:param>`
  - `<xsl:with-param name="prefix">new</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>
```


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

- Elément '**copy**'

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

- Les attributs ne sont pas préservés
  - Il faut créer de nouveaux attributs

```
<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>
```

`<value-of select="@style"/>`

Elément 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 use-attribute-sets name="class-and-color">
  Header: <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
```


Elément 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>
```

Element 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>
```

optional

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
  - Présentation → XHTML

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