



XPATH – XSLT

Yannick Prié
UFR Informatique – Université Lyon 1

UE2.2 – Master SIB M1 – 2006-2007



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)
 - Xpath2.0
 - CR : Candidate Recommendation depuis 3/11/05, dernière version au 8/06/06
- 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:///2005-2006/XML-CM3.ppt`

- Ressources distantes

`http://www.univ-lyon1.fr/`

`http://liris.cnrs.fr/yprie/ens/O5-06/SIB-RDD/CM1-2pp.pdf`



Exemples d'utilisations DTD

- 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">
```

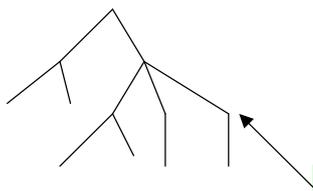
```
<!ENTITY ent SYSTEM
```

```
"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>
```



```
...  
<!ELEMENT el ...>  
<!ATTLIST el label ID #REQUIRED>  
...
```

DTD

```
...  
<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` \neq `person/title`



Xpath : principe général

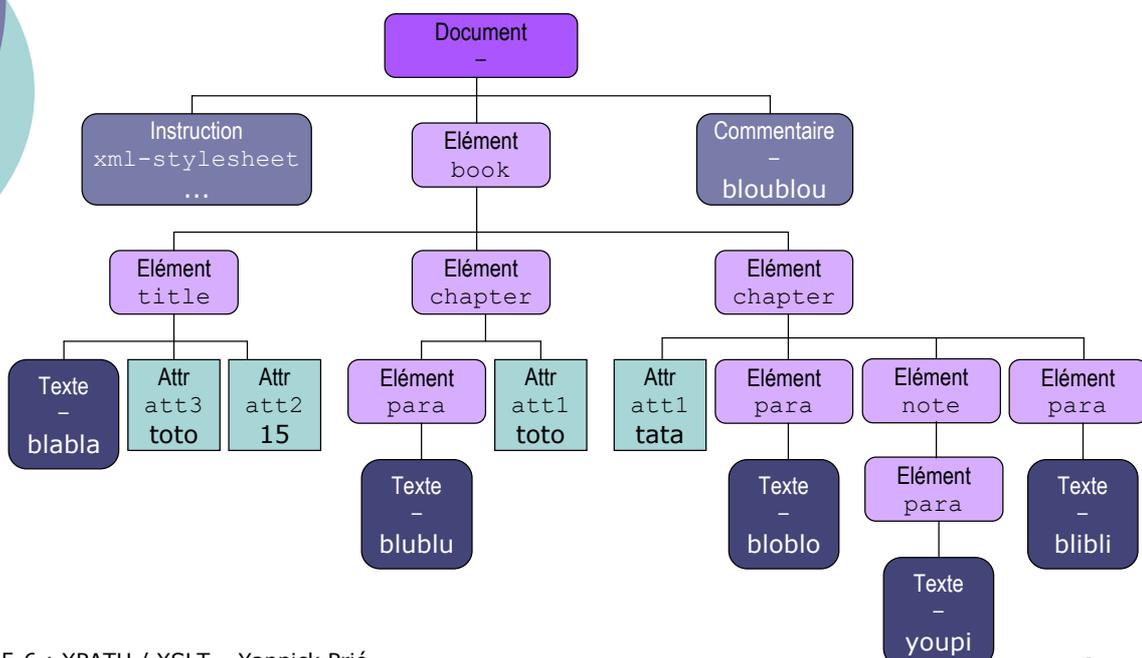
- Décrire un modèle de chemin dans un arbre XML
→ **expression**
- Récupérer les nœuds qui répondent à ce chemin en utilisant l'expression
→ résultat de l'**application de l'expression à l'arbre XML**
- Une expression sera utilisée et appliquée au sein de différentes syntaxes
 - URL : `http://abc.com/getQuery?/book/intro/title`
 - XSL : `<xsl:pattern match="chapter/title">...</xsl:pattern>`
 - Xpointer :
`<link href="./doc.xml#xptr(book/intro/title)">`
`Link to introductory title`
`</link>`
`<!-- la valeur de l'attribut href est celle de l'élément`
`title 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
 - arbre XML = élément XML
- Dans Xpath
 - arbre XPATH = arbre avec toutes les informations repérables dans un document XML:
 - noeuds éléments (= nœud XML)
 - noeud racine (représente tout le doc XML)
 - 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>bibli</para>
</chapter>
</book>
<!-- bloublou -->
```



Chemins de localisation

- Les expressions identifient des noeuds par leur position dans la hiérarchie
- 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
 - à partir de l'endroit où l'on est
 - absolu
 - à partir de la racine



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`



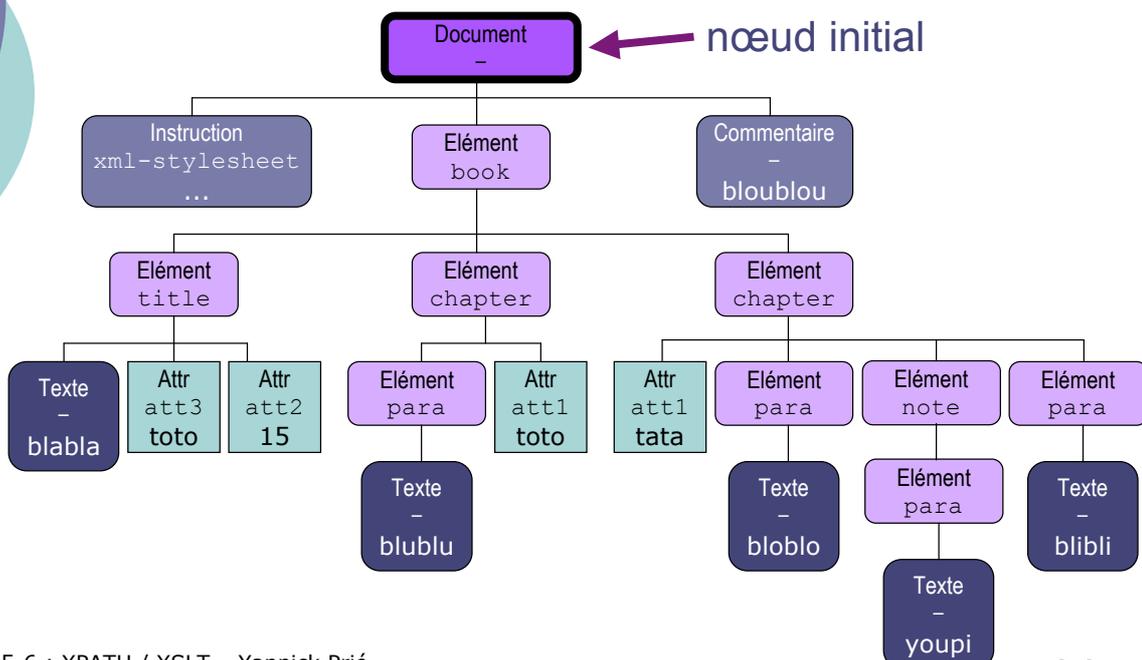
Notion d'étape Xpath

- Une étape contient 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 :: chapter [@att1 = "toto"]`
- Remarques
 - il existe une syntaxe bavarde (verbose) et une syntaxe raccourcie, plus pratique
 - possibilité de multiples expressions séparées par '|'
 - équivalent d'un OU

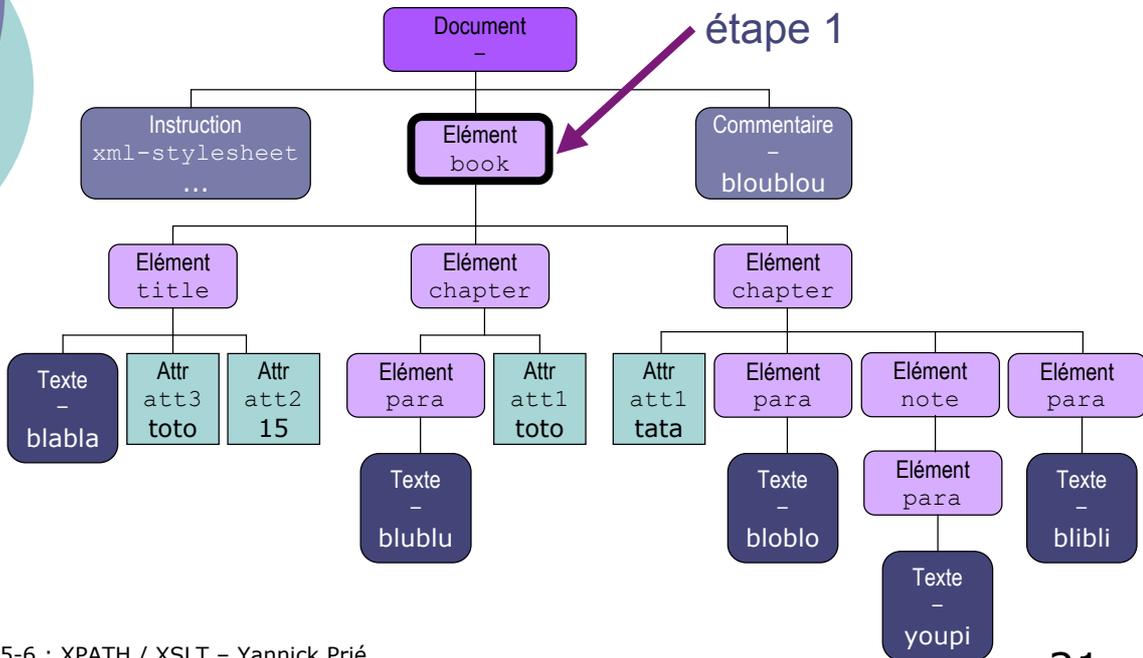
Evaluation d'une expression Xpath

- Expression = séquence d'étapes
- 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
 - ...

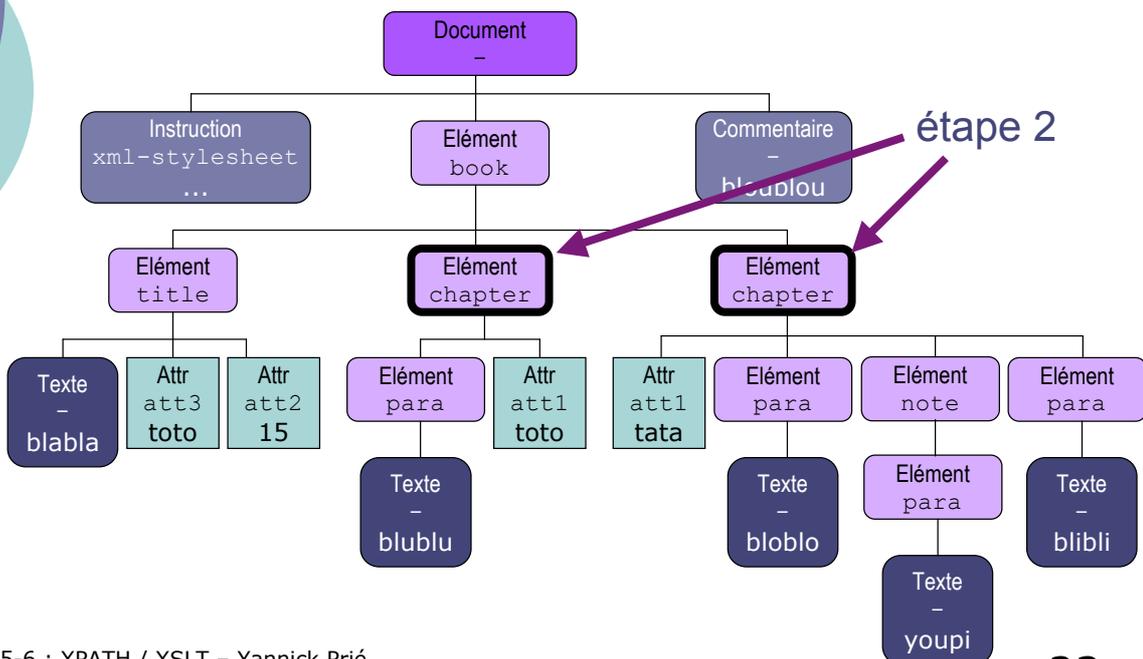
`/child::book/child::chapter/attribute::att1`
`/book/chapter/@att1` (expression raccourcie)



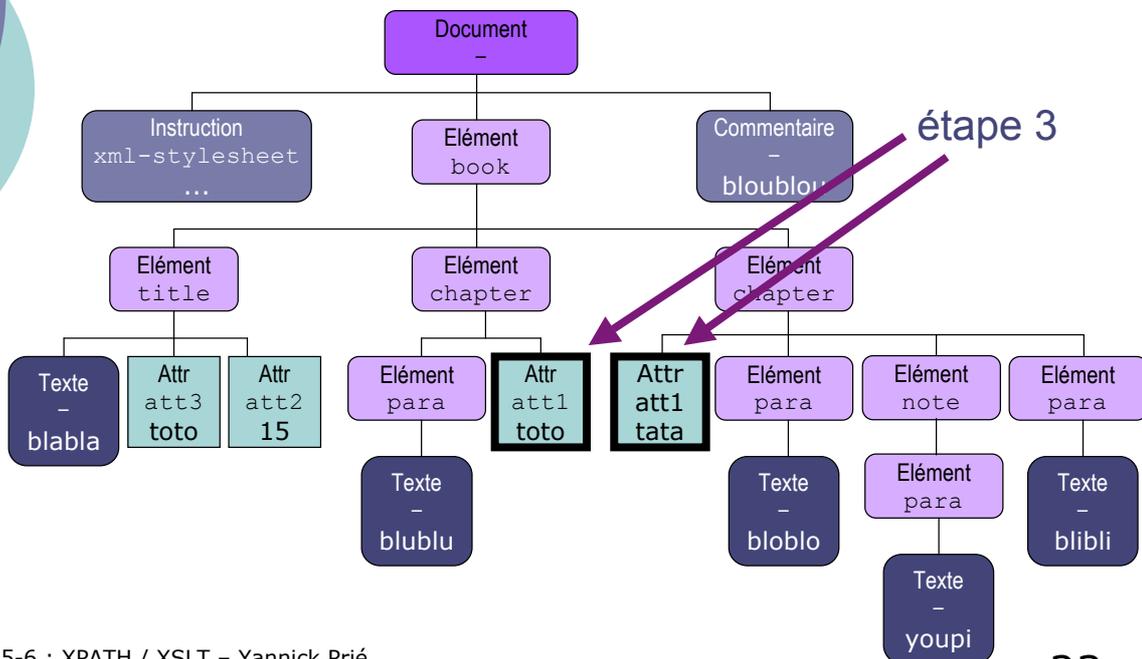
`/child::book/child::chapter/attribute::att1`
`/book/chapter/@att1`



`/child::book/child::chapter/attribute::att1`
`/book/chapter/@att1`



/child::book/child::chapter/attribute::att1
/book/chapter/@att1



CM5-6 : XPATH / XSLT – Yannick Prié
UE2.2 – Master SIB M1 – 2006-2007 : Représentation des données et des connaissances

23

Axe :: Filtre [Prédicat]

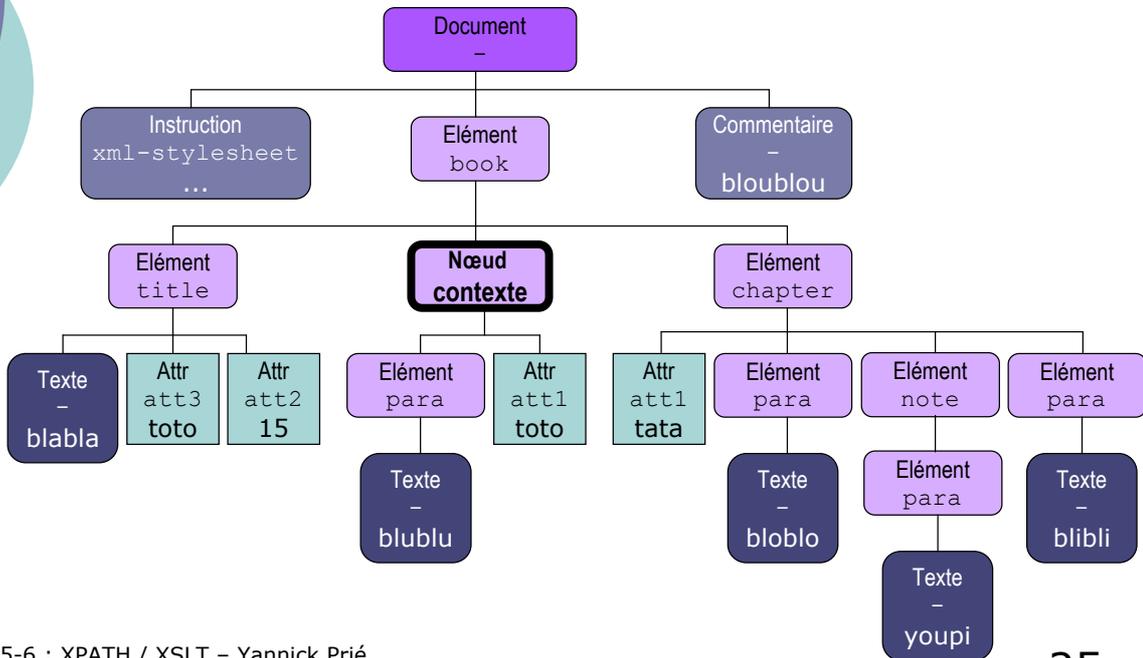
Axes : directions à suivre

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

CM5-6 : XPATH / XSLT – Yannick Prié
UE2.2 – Master SIB M1 – 2006-2007 : Représentation des données et des connaissances

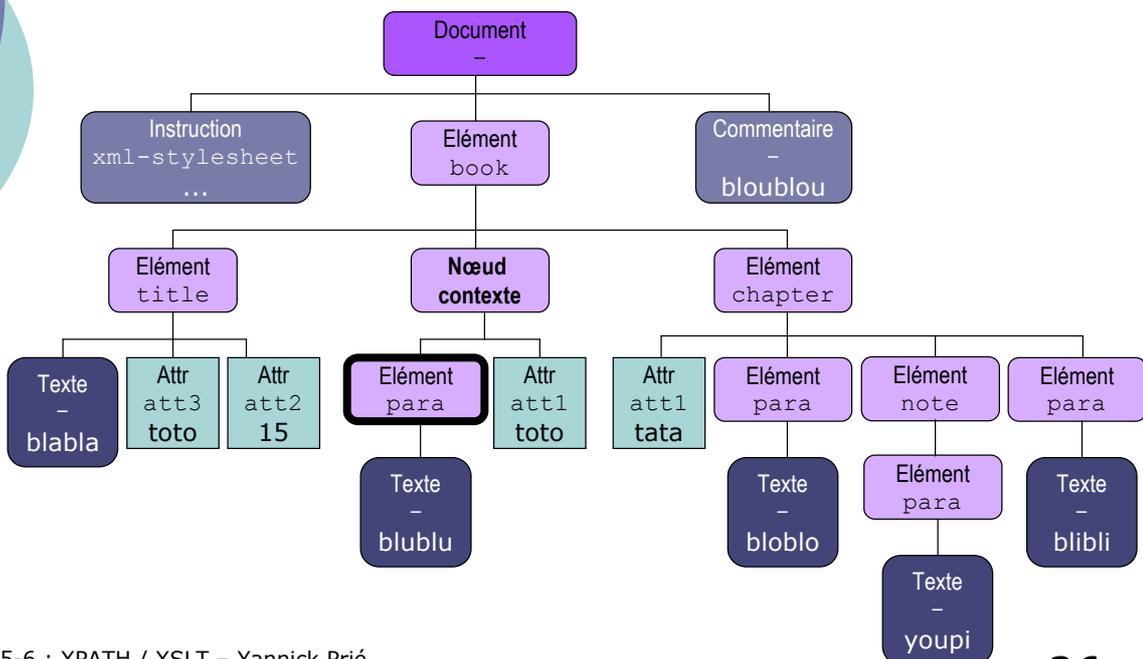
24

Axe self : self::node ()



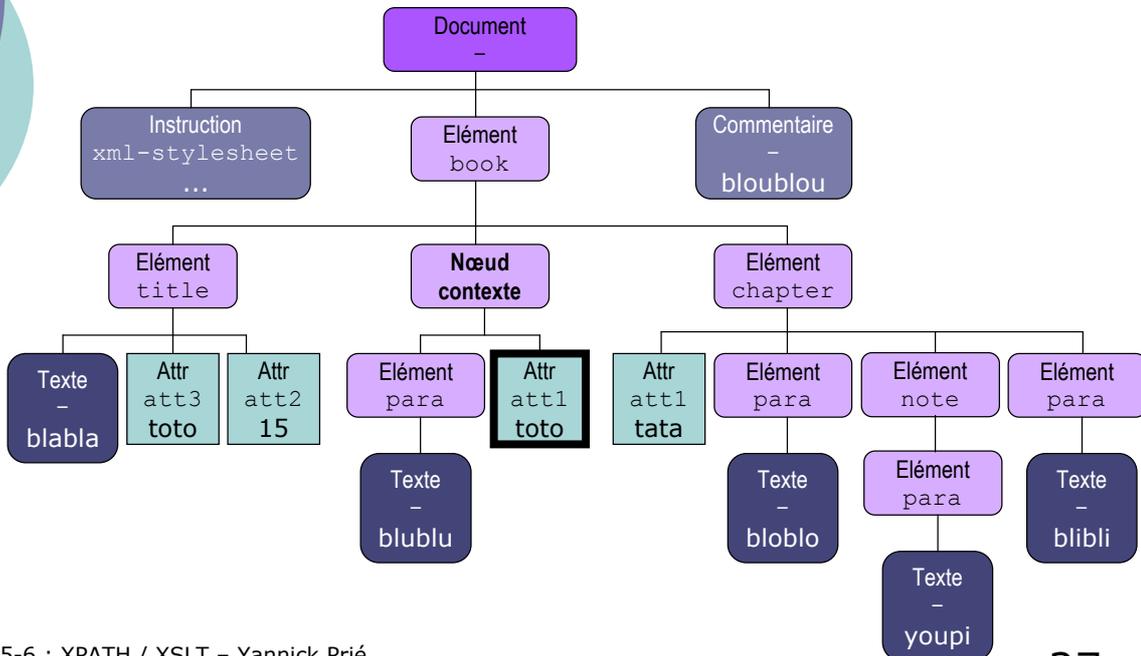
CM5-6 : XPATH / XSLT – Yannick Prié
 UE2.2 – Master SIB M1 – 2006-2007 : Représentation des données et des connaissances

Axe child : child::node ()

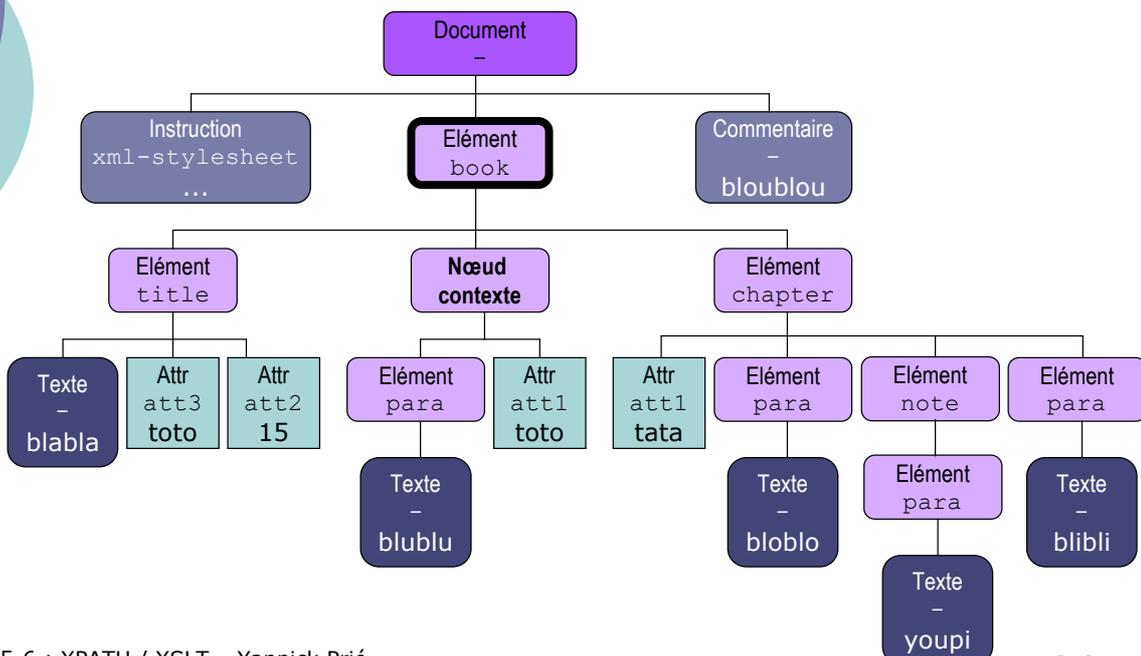


CM5-6 : XPATH / XSLT – Yannick Prié
 UE2.2 – Master SIB M1 – 2006-2007 : Représentation des données et des connaissances

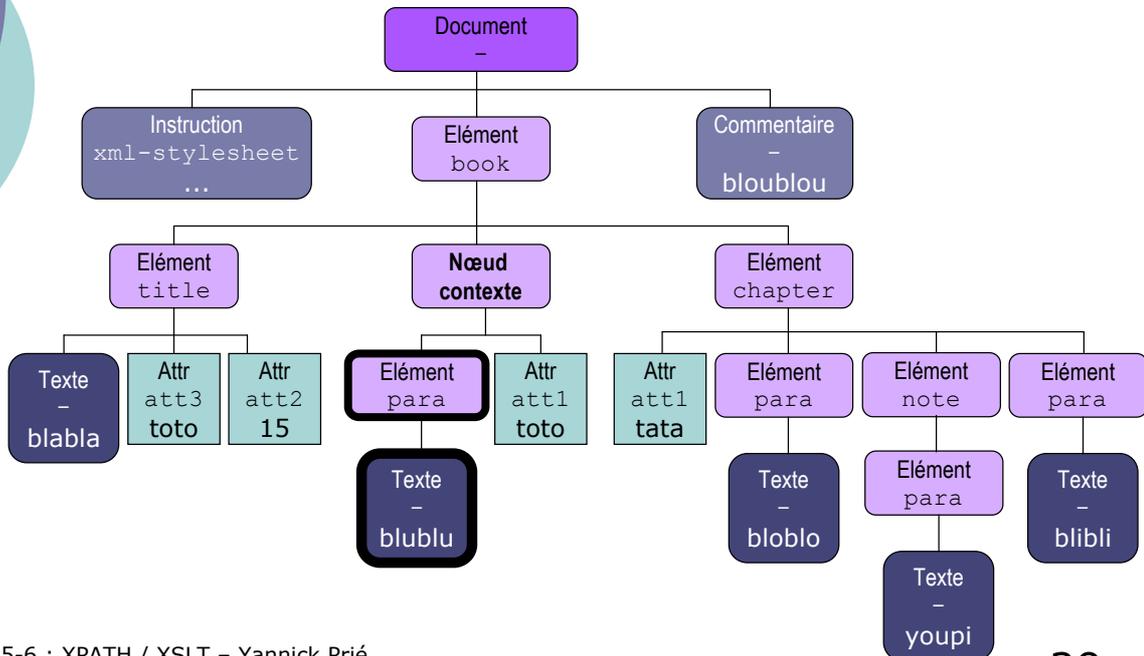
Axe attribute : attribute :: att1



Axe parent : parent :: book



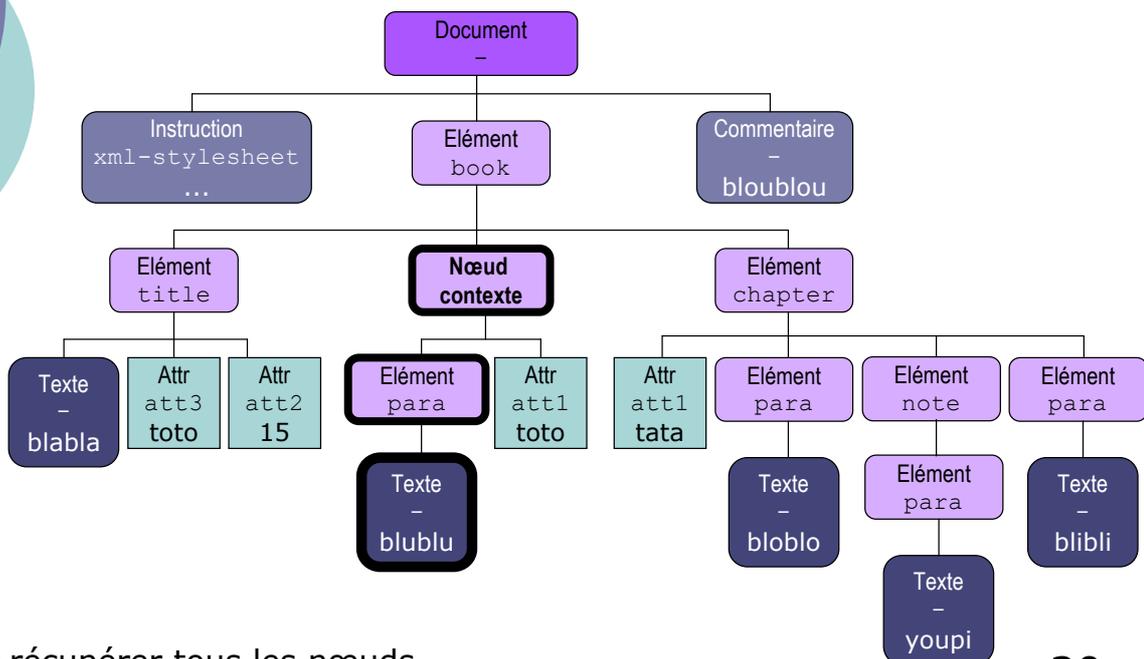
Axe descendant : descendant::node()



CM5-6 : XPATH / XSLT – Yannick Prié
UE2.2 – Master SIB M1 – 2006-2007 : Représentation des données et des connaissances

29

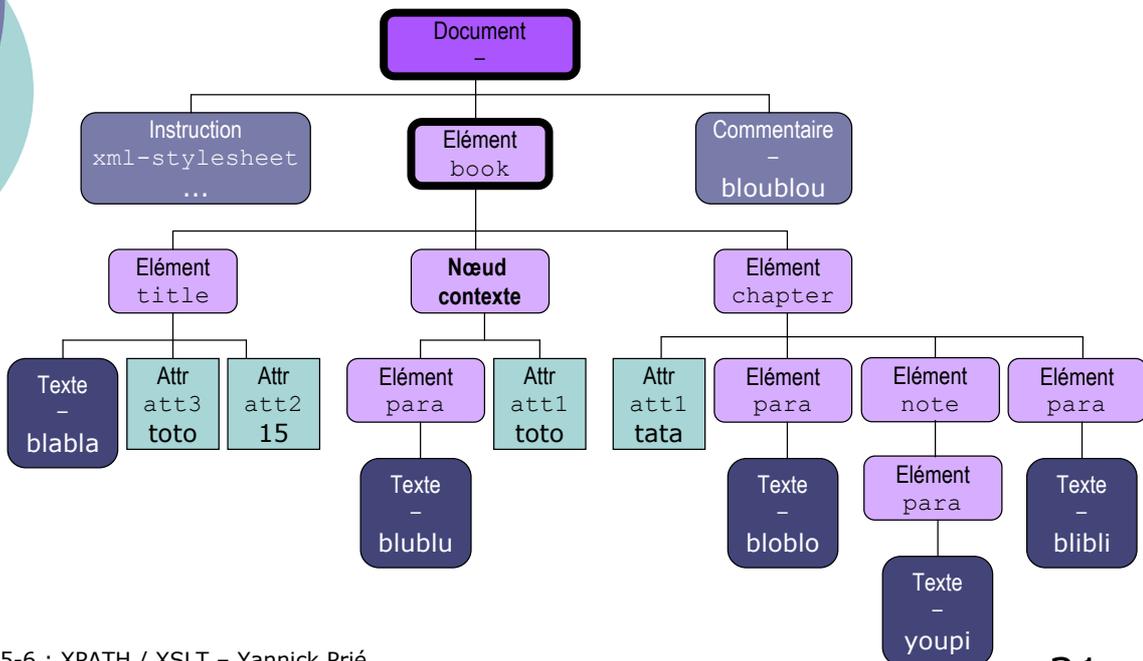
Axe descendant-or-self : descendant-or-self::node()



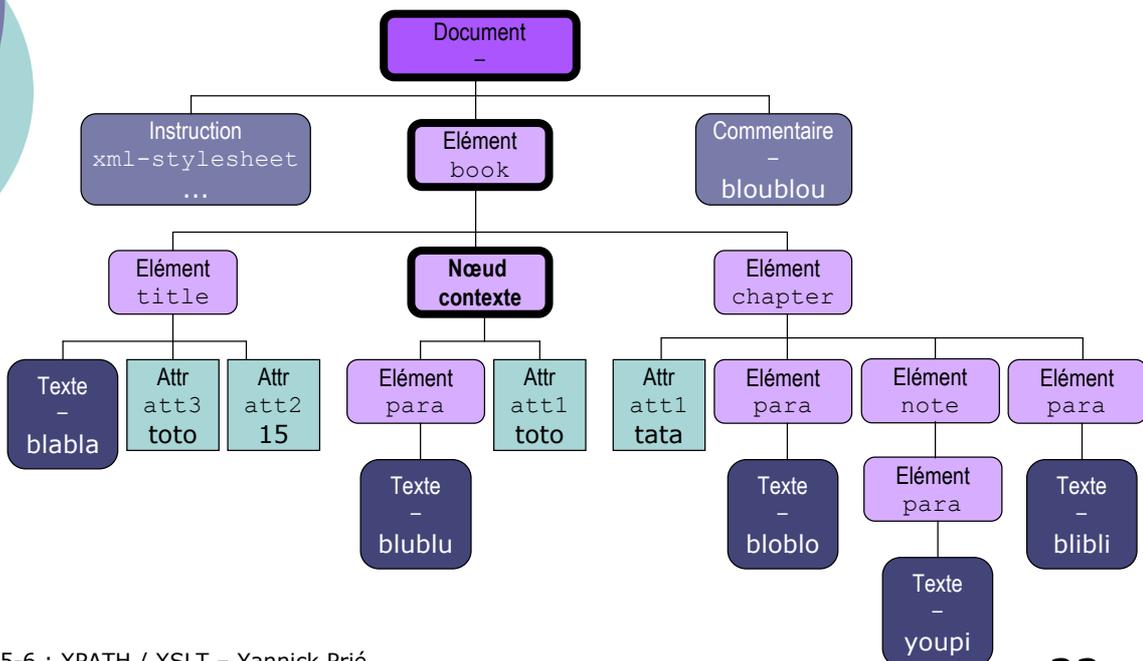
Pour récupérer tous les nœuds
/descendant-or-self::node()

30

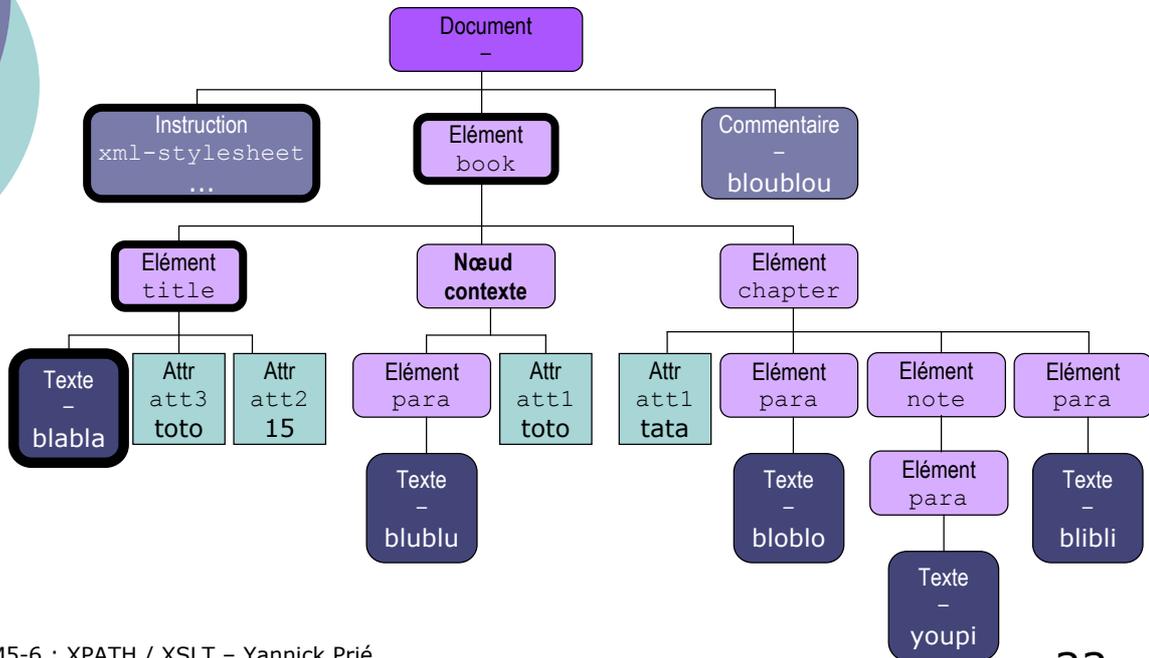
Axe ancestor : ancestor :: node ()



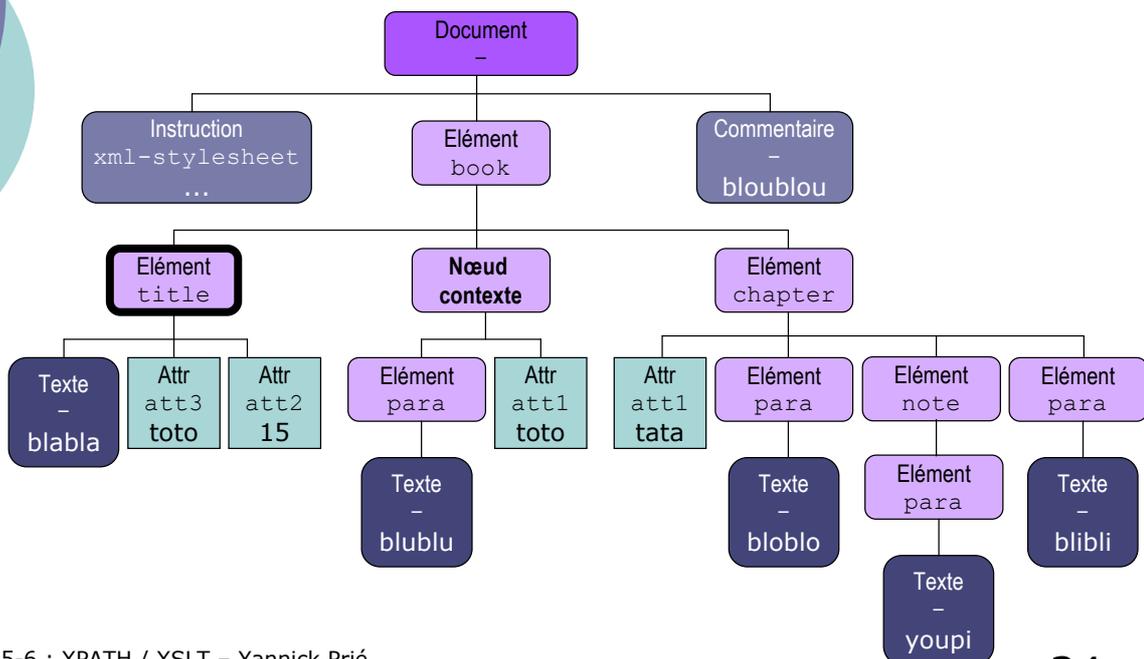
Axe ancestor-or-self : ancestor-or-self :: node ()



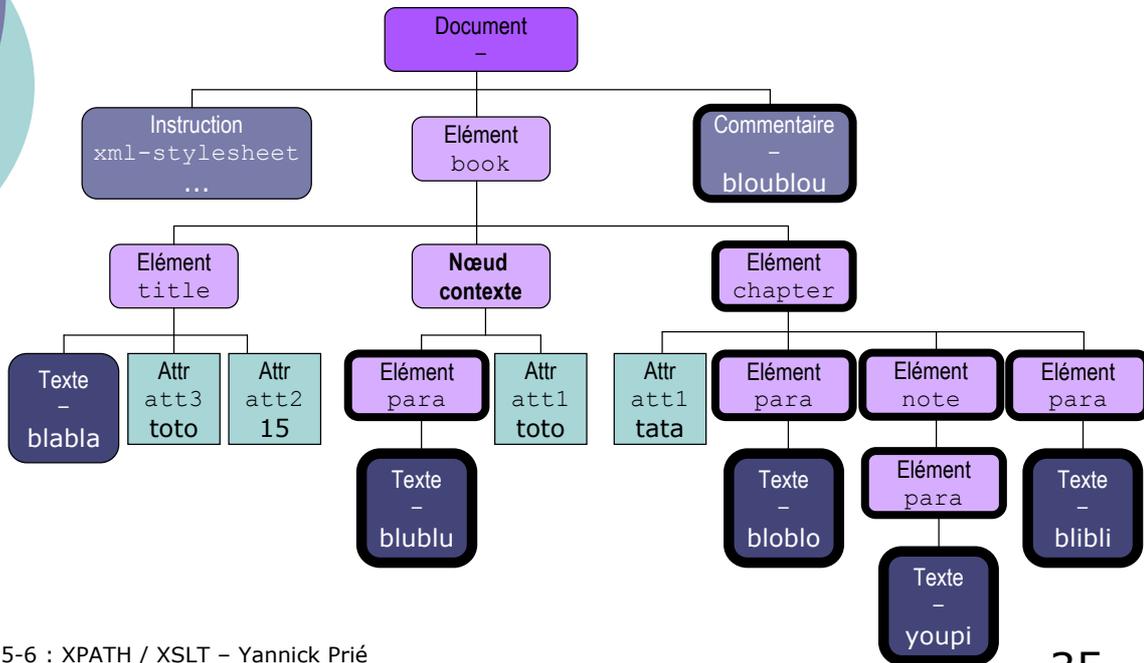
Axe preceding : preceding::node()



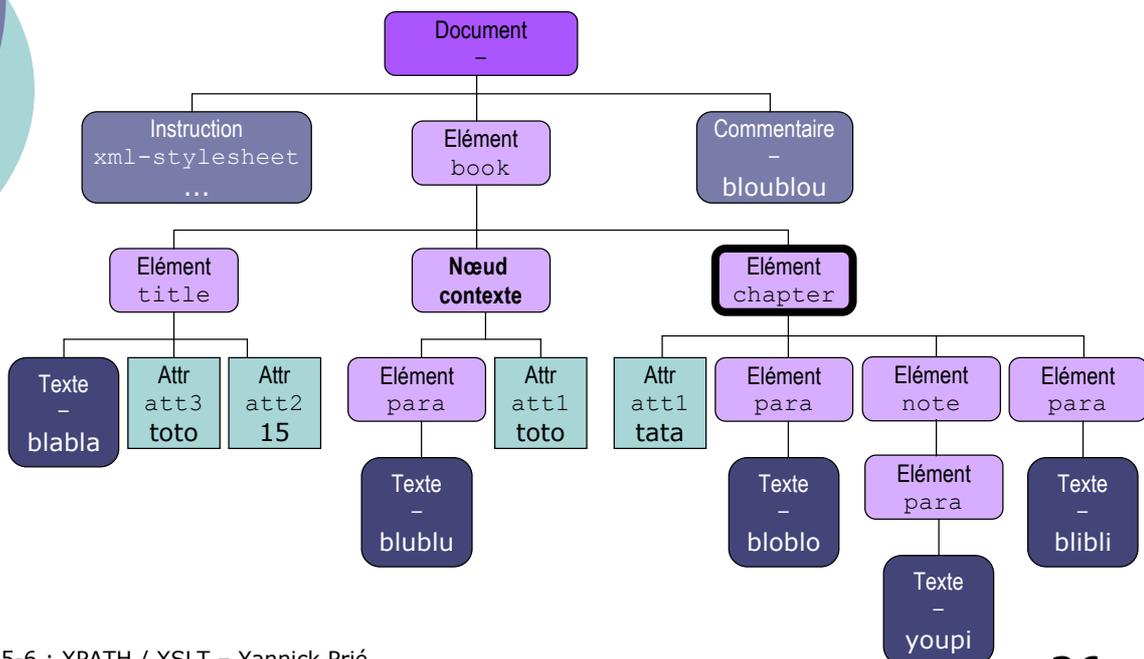
Axe preceding-sibling : preceding-sibling::node()



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
- Pour un attribut
 - @nom-attribut

Filtrage par le type

- `text()`
 - Contenu textuel
- `comment()`
 - Commentaire
- `processing-instruction()`
 - Instruction de traitement
- `node()`
 - Tout type de nœud
- `@*`
 - Toutes les valeurs de tous les attributs

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 = 2^{ème} 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
- Qu'on peut filtrer à nouveau avec des prédicats
 - prédicat indiqué entre crochets '[']'
 - si ce qui est dans le prédicat est Vrai : on garde
- Le prédicat le plus simple utilise la fonction `position()`
 - `para[position() = 1]` //1er para
 - `chapter[2]` //2eme chapter
- Possibilité de combiner les tests avec 'and' et 'or'
 - `//*[self::chapter and @att1="tata"]`

Premier test
Deuxième test

Tests sur les positions / texte

- `last()`
 - Récupère le dernier noeud dans la liste
- `count()`
 - Evalue le nombre d'items dans la liste
 - `child::chapter [count(child::para) = 2]`
- `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
- `//chapter[child::note]`
 - Les nœuds chapter qui ont des enfants note

Prédicat : divers

- Pour les booléens
 - `not()`, `and`, `or`
- Pour les numériques
 - `<`, `>`, `!=` (différent)
 - `+`, `-`, `*`, `div` (division entière), `mod` (reste div entière)
 - `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]`
 - les nœuds avec un attribut att2 impair

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>`
 - l'expression précédente ne fonctionne pas (`note/text()` donne "there")
 - utiliser plutôt `note[contains(., "hello")]`
 - '.' est le noeud courant, et on parcourera tous les enfants

Tests sur les chaînes (2)

- `starts-with(chaine, motif)`
 - `note[starts-with(., "hello")]`
- `string(chaine)`
 - `note[contains(., string("pi"))]`
- `string-after(chaine, terminateur)`
- `string-before(chaine, terminateur)`
- `substring(chaine, offset, longueur)`

Tests sur les chaînes (3)

- **normalize(*chaîne*)**
 - enlève les espaces en trop
- **translate(*chaîne*, *source*, *replace*)**
 - **translate(., "+", "plus")**
- **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::para]** (dans le contexte de chapter)
 - Tout élément note ou para fils du nœud contexte



Quelques fonctions

- S'appliquent sur un ensemble de nœuds
 - id(liste identificateurs) : récupère les éléments ayant ces identificateurs
 - nécessité d'avoir DTD / schéma
 - Ex. id('id54' '678')
 - count()
 - compte le nombre de nœuds. Ex. count(//para)
 - max()
 - rend la valeur maximale
 - sum()
 - rend la somme (les nœuds doivent correspondre à des valeurs numériques, traductibles par number())
 - distinct-values() (Xpath 2.0)
 - élimine les doublons



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
- Nous avons vu les grands principes
 - pour la description systématique de la syntaxe
 - sites de références
 - pour plus d'exemples
 - sites avec tutoriaux
- Ce cours : présentation de XPATH 1.0
 - des améliorations dans XPATH 2.0



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 possibilités 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...
 - Pour présenter les informations
 - Pour transformer les informations
 - d'un format à un autre
- 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

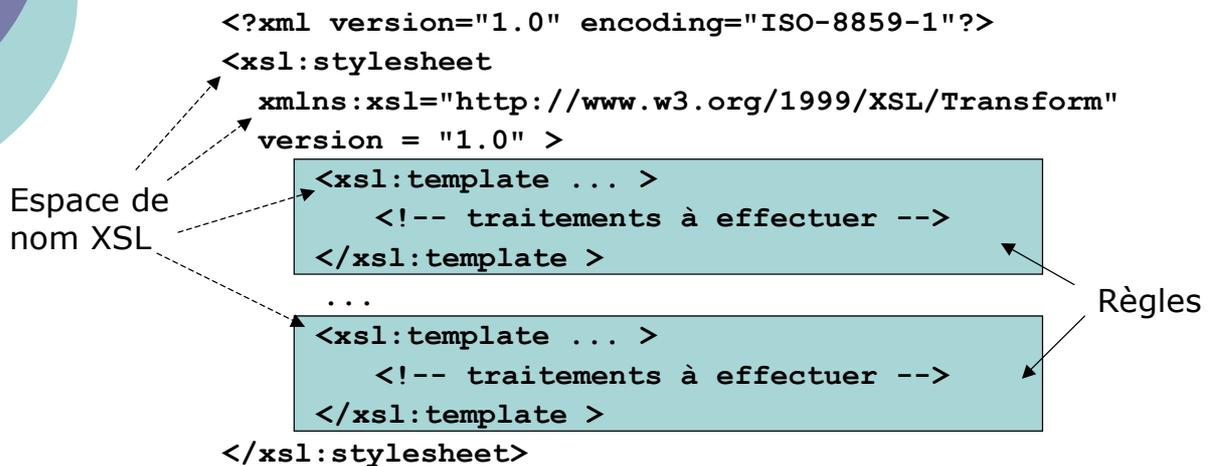


Spécification XSLT

- 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**



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>
```



Élément output

- Pour spécifier le format de sortie

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

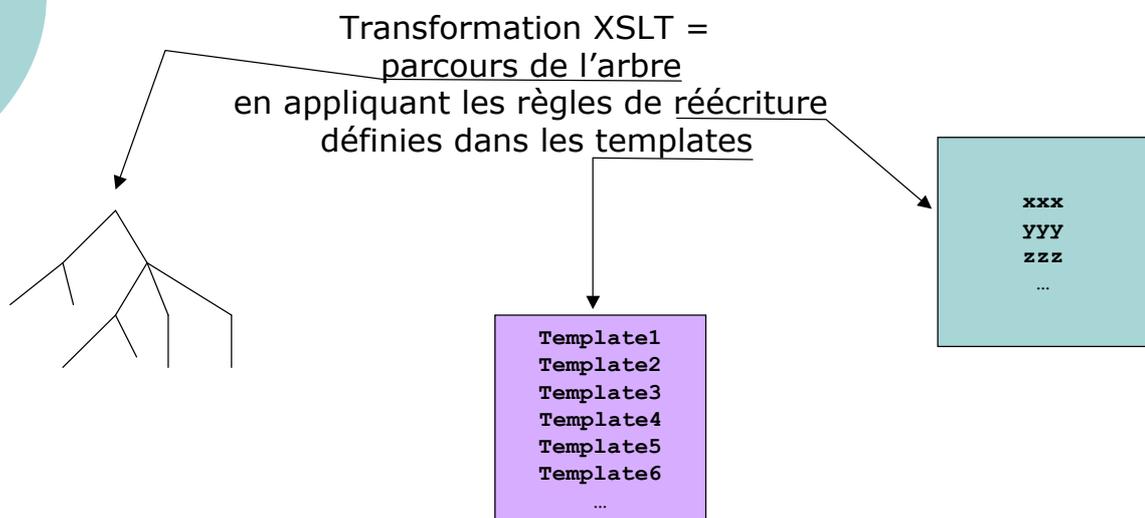
- Attributs de **output**

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

Principe du traitement XSLT

- Effectué récursivement 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

Feuilles de style XSL





Élément template

- Pour spécifier une règle de transformation

```
<xsl:template match="expression">
...
</xsl:template>
```
- L'attribut `match` a pour valeur une expression Xpath
 - limitée aux axes `child`, `attribute`, `descendant-or-self`
- Le résultat de cette expression devient noeud contextuel au sein du template
- On commence toujours par s'intéresser à la racine 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 place 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

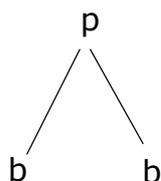
Quelques « éléments instructions » à mettre dans un élément template

- **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

67

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 cette  
<b>chose</b>.</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 hiérarchique 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 select
 - **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}" />
  XXXXX SYNTAXE SPECIALE XXXXXX
</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 noeuds attributs ne sont pas atteints
- 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 défaut</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 '**E**lement'
- 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 '**c**opy'

- 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>
 <xsl:attribute name="style">color: blue</xsl:attribute>
 </xsl:otherwise>
 <xsl:apply-templates/>
 </xsl:choose>
</html:p>
</xsl:template>
```



## 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



## 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	card1.xml
<pre>&lt;carte&gt;   &lt;titre&gt;Dr.&lt;/titre&gt;   &lt;nom&gt;Paul Durand&lt;/nom&gt;   &lt;telephone inter="33"&gt;4 78 34 25   12&lt;/telephone&gt;   &lt;telephone inter="33"&gt;6 12 45 25   12&lt;/telephone&gt;   &lt;adresse&gt;     &lt;rue&gt;Impasse des Fleurs&lt;/rue&gt;     &lt;code&gt;69001&lt;/code&gt;     &lt;ville&gt;Lyon&lt;/ville&gt;     &lt;pays&gt;France&lt;/pays&gt;   &lt;/adresse&gt;   &lt;courriel&gt;     paul.durand@provider.com&lt;/courriel&gt; &lt;/carte&gt;</pre>	<pre>&lt;card&gt;   &lt;name title="Dr."&gt;     Paul Durand&lt;/name&gt;   &lt;address&gt;     &lt;street&gt;Impasse des     Fleurs&lt;/street&gt;     &lt;zipcode&gt;69001 Lyon&lt;/zipcode&gt;     &lt;country&gt;France&lt;/country&gt;   &lt;/address&gt;   &lt;phones&gt;     &lt;phone&gt;(33) 4 78 34 25 12&lt;/phone&gt;     &lt;phone&gt;(33) 6 12 45 25 12&lt;/phone&gt;   &lt;/phones&gt; &lt;/card&gt;</pre>

## Remerciements

---

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