

# XPath

## Concepts et Technologies XML

Année 2007-08

### Plan du cours

1. les DTD modulaires
2. les espaces de noms dans XML
3. la représentation sous forme arborescente d'un document XML
4. XPath, le langage d'expressions sur les arbres XML
5. Premiers pas en XSLT (langage de transformations de documents XML)

### Les DTD modulaires

Première possibilité : **en ajoutant les possibilités externes.**

La règle est simple pour éviter les conflits :

- si plusieurs déclarations d'un même élément coexistent, seule la première est prise en compte ;
- si plusieurs déclarations d'attributs pour un même élément co-existent, alors on les concatène ;
- si un même attribut pour un même élément est déclaré plusieurs fois, c'est la première qui compte.

## Les DTD modulaires

### Modularisation

Seconde possibilité : **utiliser des sections conditionnelles.**

Ce sont des balises spécifiques aux DTD, qui permettent de laisser le choix à l'utilisateur de la DTD de valider ou non ces sections.

Syntaxiquement, ces balises ressemblent aux balises CDATA :

```
<![test[texte de la DTD ]]>
```

*test* vaut soit *INCLUDE*, soit *IGNORE* :

- S'il vaut *INCLUDE*, le texte de la DTD sera ajouté aux déclarations ;
- s'il vaut *IGNORE*...

## Modularisation

- la valeur de *test* n'est pas remplacée manuellement !
- généralement, *test* vaut *%test-struct-cond* ;
- et l'entité *test-struct-cond* est déclarée dans le sous-module interne du document : c'est donc au plus bas niveau qu'on décide de ce qui est activé !

Les structures conditionnelles peuvent être imbriquées, mais les plus externes ont la priorité.

## Exemple - la DTD

On propose une nouvelle DTD pour le message :

```
<![%test ;[
<!ELEMENT message (destinataire,expediteur?,objet?,
                    corps,formulepolitesse,
                    sign_complete) >
]]>

<!ELEMENT message (destinataire,expediteur?,objet?,
                    corps,formulepolitesse,
                    signature) >
```

## Exemple - le document XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE mistigri SYSTEM "mistigri.dtd"
[
<!ENTITY test "IGNORE">
]>
```

## Les espaces de noms

### Les espaces de noms

Un document XML peut être composé

- de nombreuses balises différentes,
- qui ont des rôles différents
- et qui doivent être traitées par des applications XML différentes

Les noms simples des éléments ne permettent pas de gérer facilement de tels documents :

- traitement des homonymies sur les éléments
- détection rapide des éléments à traiter pour les applications

### Les espaces de noms

Les espaces de noms permettent d'étendre le nommage en qualifiant le nom de l'élément par un URI.

```

<eltRacine>
  <monElt xmlns="http://www.chezmoi.net/mesElements">
    <a> ... </a>
  </monElt>
  <monElt xmlns="http://www.chezlui.net/sesElements">
    <b> ... </b>
  </monElt>
</eltRacine>

```

Les deux élément **monElt** sont différents : ils n'appartiennent pas au même espace de noms

### Les espaces de noms

- habituellement, déclaration de tous les espaces de noms nécessaires dans l'élément racine du document
- à chaque espace de nom est associé un préfixe
- les éléments sont préfixés pour indiquer à quel espace de noms ils appartiennent.

```

<moi:eltRacine
  xmlns:moi="http://www.chezmoi.net/mesElements"
  xmlns:lui="http://www.chezlui.net/sesElements">
  <moi:monElt>
    <a> ... </a>
  </moi:monElt>
  <lui:monElt>
    <b> ... </b>
  </lui:monElt>
</moi:eltRacine>

```

## La représentation arborescente d'un arbre XML

### La représentation arborescente d'un document XML

- un document XML est représenté par un arbre
- qui comprend les éléments, les attributs, les zones de texte
- mais aussi les commentaires, les instructions de traitement, ...
- bonne compréhension : permet de se déplacer dans les arbres ! sous-tend :
  - l'API DOM
  - XPath

### Les noeuds d'un arbre

Il y en a 7 types de noeuds différents.

**Les noeuds internes :**

- **racine**
  - se situe au-dessus de l'élément racine du document (*recette, message, html...*)
  - contient ce qui se trouve au-dessus de l'élément racine
    - les instructions de traitement,
    - les commentaires,
    - etc...
    - **sauf la déclaration de document**
  - sous la **racine** on peut trouver **tous** les types de noeuds
- **élément** :
  - peut contenir toutes les sortes d'autres éléments excepté le noeud racine
  - c'est une feuille s'il n'a ni attribut ni contenu

### Les noeuds d'un arbre

#### Les feuilles de l'arbre

- **attribut** :
  - c'est un noeud en-dessous de l'élément qu'il qualifie
  - **ce n'est pas un fils** de l'élément qu'il qualifie
- **texte** :
  - la donnée de base d'un document XML!
  - toujours enfant d'un noeud élément (pas forcément fils unique)
  - c'est une chaîne de caractères contigus sans noeuds intermédiaires

### Les noeuds d'un arbre

#### Les feuilles de l'arbre

- **instruction de traitement**
- **commentaire**
- **espace de noms** :
  - a une signification particulière et une influence sur le sous-arbre qu'il domine
    - n'est pas un attribut
    - n'est pas un noeud à part entière
    - mais **n'est pas fils** de l'élément qu'il qualifie

### La représentation arborescente

Cette représentation arborescente n'est pas isomorphe au document texte original :

- le prologue du document (déclaration XML + déclaration de type de document)
  - les sections CDATA
  - les appels d'entités
  - le contenu de la DTD
- ne sont pas conservés.

# Langage XPath

## XPath

[XPath](#) : langage de localisation et de sélection d'ensembles de noeuds dans un document XML.

Une [expression XPath](#) retourne

- soit un ensemble de noeuds
- soit un booléen
- soit un nombre
- soit une chaîne de caractères

La dernière recommandation de XPath date de Novembre 1999 et il s'agit de la version 1.0.

## Se déplacer dans les arbres

Tout d'abord, [localiser des noeuds](#) :

- on retrouve des éléments qu'on avait déjà vus avec les sélecteurs dans les CSS (actuellement CSS2).
- avec XPath, on peut aller plus loin (CSS3 pourrait implémenter complètement XPath).

*Mais ...*

## Contexte de recherche

Une expression XPath est calculée à partir d'un [contexte](#).

Un [contexte](#) est déterminé par :

- un noeud (le [noeud-contexte](#));
- un couple d'entiers positifs : [l'indice du noeud-contexte](#) dans l'ensemble dont il est issu, [la taille de cet ensemble](#) ;
- des [variables](#) et leur valeur ;
- une librairie de [fonctions](#) ;
- un ensemble d'[espaces de noms](#).

## Chemin de localisation

Un [chemin de localisation](#) est

- soit un [chemin absolu](#) : depuis la racine, il commence par /
- soit un [chemin relatif](#) : il commence depuis le noeud-contexte.

Un chemin est ensuite constitué d'*étapes* qui permettent d'avancer dans la localisation.

Syntaxe d'un chemin : *étape/étape/étape/...*

## Chemin de localisation

Une *étape* a trois composantes :

- un [axe](#) : il détermine la direction à prendre
- un [test de noeud](#) : il précise le type de noeud à sélectionner

- des **prédicats** optionnels qui permettent d'affiner la sélection

Syntaxe :

*axe::test-de-noeud[prédicat]\**

Chaque élément peut-être omis.

### Axes

Les axes possibles sont :

- **parent** : le parent du noeud contextuel
- **ancestor** : tous les noeuds placés au-dessus du noeud contextuel
- **ancestor-or-self** : idem avec le noeud contextuel
- **child** : les enfants ... (rappel : **sauf les attributs et les espaces de noms**)

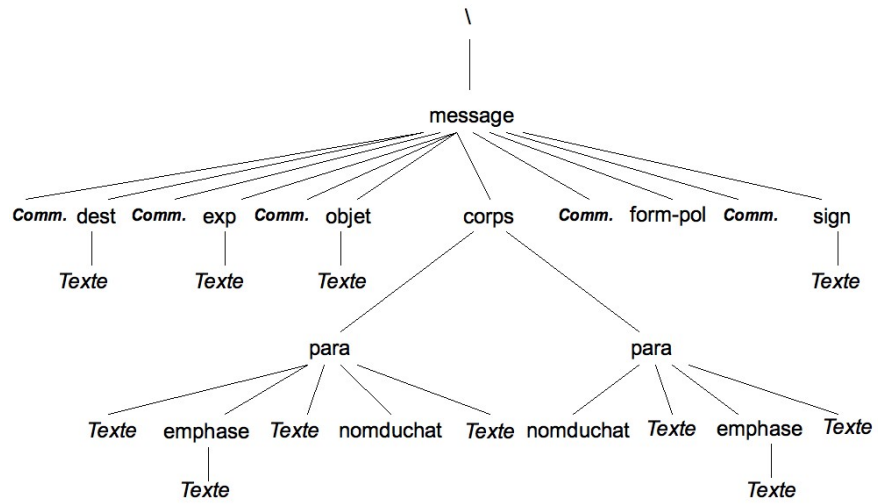
### Axes

- **descendant** : tous les noeuds en-dessous du noeud contextuel
- **descendant-or-self** : idem avec le noeud contextuel
- **self** : le noeud lui-même
- **attribute** : les attributs du noeud contextuel
- **namespace** : tous les noeuds d'espaces de noms actifs pour le noeud contextuel

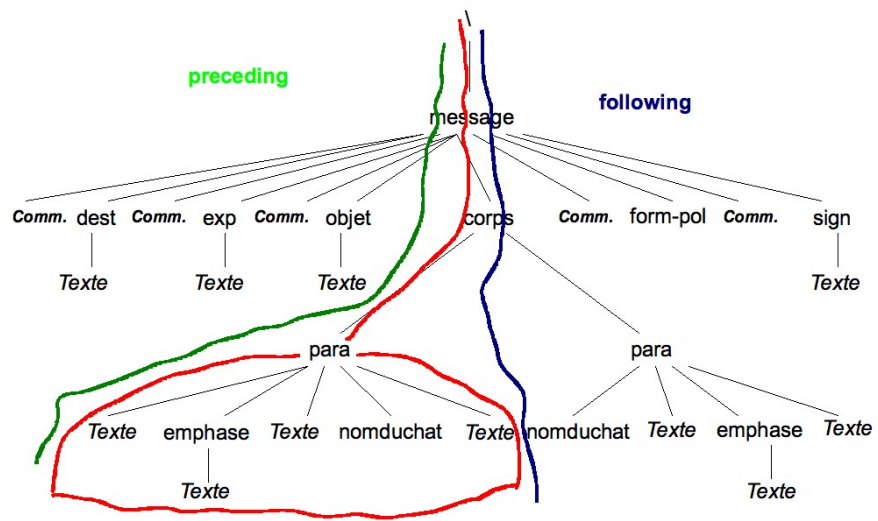
### Exemple

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<message priorité="importante">
  <!-- le destinataire --><dest>M. Dupont</dest>
  <!-- l'expéditeur --><exp>Melle. Dumoulin</exp>
  <!-- le sujet --><obj>alimentation du chat</obj>
  <corps>
    <para>Conformément à vos instructions, je donne
    <emphase>trois</emphase> rations de croquettes
    par jour à <nomduchat />.</para>
    <para><nomduchat /> a cependant
    pris <emphase>deux</emphase> kilos pendant
    les vacances.</para>
  </corps>
  <!-- pour finir --><form-pol style="simple"/>
  <!-- signature --><sign>Melle. Dumoulin</sign>
</message>
```

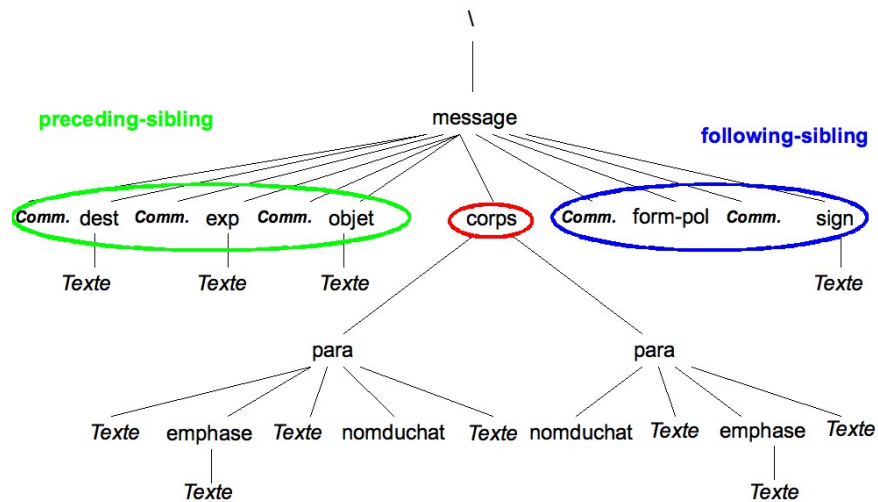
### Axes



Axes



Axes



#### Axes

- **following** :
  - dans l'arbre, les noeuds qui se trouvent à *droite du chemin* qui va de la racine au noeud-contexte ;
  - dans le document, les éléments dont *la balise ouvrante* se situe *au-dessous de la balise fermante* du noeud-contexte
- **following-sibling** : les noeuds frères qui suivent le noeud contextuel
- **preceding** :
  - dans l'arbre, les noeuds qui se trouvent à *gauche du chemin* qui va de la racine au noeud-contexte ;
  - dans le document, les éléments dont *la balise fermante* se situe *au-dessus de la balise ouvrante* du noeud-contexte
- **preceding-sibling** : frères qui précèdent

#### Se déplacer dans les arbres

- En l'absence de précision de l'axe, c'est **child** qui est choisi  
*fiche-technique/ingredient*  
 est équivalent à  
*child::fiche-technique/child::ingredient*

#### Tests de noeuds

- Les tests de noeuds possibles :
- / : le noeud racine (pas l'élément document)
  - **node()** : tout noeud sauf le noeud racine et les attributs

- \* : dans l'axe des attributs, tout attribut, dans l'axe des espaces de noms, tout espace de noms, dans les autres axes, tout élément
- un\_nom : un\_nom d'un attribut, d'un préfixe d'espace de noms ou d'un élément, suivant l'axe

### Tests de noeuds

- text() : tout noeud de texte
- processing-instruction() : toute instruction de traitement
- processing-instruction('une\_instruct') : l'instruction de traitement une\_instruct
- comment() : tout noeud commentaire

### Raccourcis

Les raccourcis :

- \*
- @ : désigne les attributs en l'absence de qualificateur d'axe.
  - @\* est équivalent à attribute : :\*
  - @priorité est un attribut de nom priorité
- . : le noeud lui-même (équivalent à self : :\*)
- .. : le noeud parent
- // : équivalent à /descendant-or-self::node()/

### Se déplacer dans les arbres

#### Exemples

- //para est équivalent à /descendant-or-self::node()/child::para
- //para[1] n'est pas équivalent à /descendant::para[1]
- ../para est équivalent à self::node()/descendant-or-self::node()/child::para

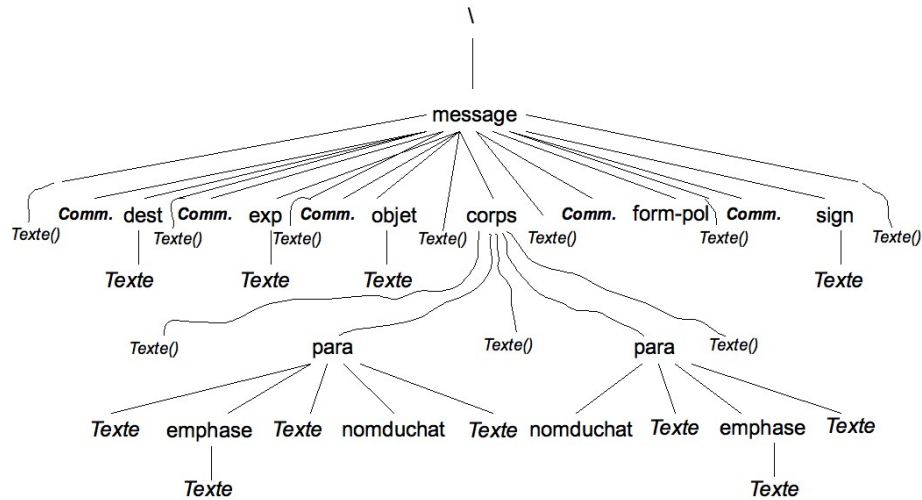
### Exemple

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<message priorité="importante">
  <!-- le destinataire --><dest>M. Dupont</dest>
  <!-- l'expéditeur --><exp>Melle. Dumoulin</exp>
  <!-- le sujet --><obj>alimentation du chat</obj>
  <corps>
    <para>Conformément à vos instructions, je donne
    <emphase>trois</emphase> rations de croquettes
    par jour à <nomduchat />.</para>
    <para><nomduchat /> a cependant
    pris <emphase>deux</emphase> kilos pendant
    les vacances.</para>
  </corps>
  <!-- pour finir --><form-pol style="simple"/>

```

```
<!-- signature --><sign>Melle. Dumoulin</sign>
</message>
```

## Axes



*Texte()* : noeud blanc – le texte ne contient que des séparateurs (espaces, retours à la ligne, tabulations, ...)

## Exemples !

Le [noeud-contexte](#) est *corps*.

Que retourne ...

<code>child::node()</code>	<code>child::*</code>
<code>parent::*</code>	<code>descendant-or-self::text()</code>
<code>parent::message</code>	<code>ancestor-or-self::corps</code>
<code>parent::para</code>	<code>ancestor-or-self::/</code>
<code>self::para</code>	<code>descendant::comment()</code>
<code>preceding-sibling::*</code>	<code>following-sibling::node()</code>
<code>descendant::para</code>	<code>descendant::para[child::nomduchat]</code>
<code>self::corps</code>	<code>descendant::para[child::nomduchat]</code>

## Exemples !

Le [noeud-contexte](#) est *corps*.

- `child::node()` : tous les noeuds fils de *corps*, i.e. ici les deux noeuds *para* et trois noeuds blancs
- `child::*` : \* est plus restrictif que `node()` puisqu'en ne conservant que les éléments, il oublie les commentaires et les instructions de traitements, et les zones de texte. Ici, il n'y a plus que les deux noeuds *para*.

- `parent::*` : le noeud père, c'est-à-dire le noeud `message`
- `descendant-or-self::text()` : toutes les zones de texte en-dessous de corps (y compris les noeuds blancs)

### Exemples !

- `parent::message` : même chose, retourne le noeud `message`
- `ancestor-or-self::corps` : le noeud `corps` lui-même
- `self::corps` : idem
- `parent::para` : l'ensemble vide !
- `ancestor-or-self::/` : le noeud racine
- `self::para` : l'ensemble vide !
- `descendant::comment()` : les commentaires plus bas (s'il y en avait)

### Exemples !

- `preceding-sibling::*` : l'objet, le destinataire, et l'expéditeur
- `following-sibling::node()` : la signature, la formule de politesse et les deux commentaires
- `descendant::para` : les deux noeuds `para`
- `descendant::para[child::nomduchat]` : les `para` qui ont le nom du chat dans leur texte, à ne pas confondre avec ...
- `descendant::para[child::nomduchat]` : qui retourne les noeuds `nomduchat`

### Prédicats

Le dernier filtre est l'évaluation d'une expression dont le résultat est `converti en un booléen`.

Principales règles :

- si c'est `un nombre` : il est comparé à la position du noeud dans le contexte ; `para[1]` est équivalent à `child::para[position()=1]` i.e. le premier fils du noeud contexte, s'il s'agit d'un noeud `para`
- si c'est `une chaîne de caractères` : faux si c'est la chaîne vide, vrai sinon ;
- si c'est `un ensemble de noeuds` : faux si c'est un ensemble vide, vrai sinon.

### Expressions

- les opérateurs logiques `or`, `and` (il existe une fonction `not(Expr)`)
- les opérateurs de comparaison `=`, `!=`, `<`, `<=`, `>`, `>=`
- les opérateurs arithmétiques `+`, `-`, `mod`, `div`, `*`
- l'opérateur union pour des ensembles de noeuds `||`

### Fonctions

- pour convertir en booléen `boolean(expr)`
- fonctions sur des ensembles de noeuds :
  - `number last()` : taille du contexte
  - `number position()` : indice du noeud sélectionné dans le contexte

- `number count(node-set)` : taille de l'ensemble de noeuds donné en paramètre
- `node id(string)` : le noeud dont l'identifiant est donné en paramètre

### Fonctions

- pour convertir en chaîne de caractères : `string(node ou node-set)`
- fonctions sur des nombres :
  - `number sum(node-set)`
  - `number floor(number)`
  - `number ceiling(number)`
  - `number round(number)`

### Fonctions

- fonctions sur des chaînes de caractères :
  - `concat(chaine, chaine, ...)`
  - `normalize-space(chaine)`
  - `substring(chaine, debut, taille)`
  - `substring-after(chaine, sous-chaine)`
  - `substring-before(chaine, sous-chaine)`
  - `contains(chaine, sous-chaine)`
  - `starts-with(chaine, prefixe)`
  - `string-length(chaine)`

### Exercices

Pour le document `GrandPrix.xml`, donnez les expressions XPath qui retournent (à partir de la racine) :

1. tous les noeuds éléments de nom `attr`
2. tous les noeuds éléments qui ont un attribut `order`
3. le fils de nom `attr` de `cast`
4. le premier fils de nom `name` d'un `Composer`
5. le noeud attribut de nom `Crew` (attribut de `filmography`)
6. le quatrième fils de `filmography`
7. les noeuds attributs de nom `order`
8. le noeud contextuel étant maintenant `remainder`, retournez le sous-arbre de racine `Editor`