XPath

XML Path language

Yves Bekkers

Mise à jour : 9 octobre 2003

Déplacements dans un document XPath

- XPath :
  - Un langage fonctionnel pour adresser les sous-arbres d'un arbre XML
- Norme utilisée dans de nombreux outils XML
  - **XSLT** : langage de transformation d'arbre XML
  - **XPointer** : Extension des URLs
  - **XQuery** : langage d'interrogation de documents
  - **Java, C++, Javascript** : évaluateurs Xpath

Un langage fonctionnel

- **Programme** : une expression
  - 1 + 1 ➔ 2
  - 2.0* (4-1) ➔ 6
  - concat('bonjour', " " , "monsieur") ➔ *bonjour monsieur*
- **Exécution** : évaluation de l'expression
- **Résultat typé**
  - booléen
  - chaîne
  - nombre décimal signé
  - ensemble de nœuds
    - expression = chemin dans une arbre

Opérandes et opérateurs pour les types simples

Constantes

- Chaînes
  - `'Paris'`
  - `"That's rubbish"`
  - `'He said "Boo"'`
- Valeurs numériques
  - `12`
  - `3.05`
  - `- 5.25`
- Valeurs Booléennes
  - `true()`
  - `false()`

Opérateurs

- Expression numériques :
  - `+, -, *, div, mod`
  - `position(), last(), count(nds),`
  - `string-length(expr)`
- Expression booléenne :
  - `or, and, not(...)`
  - `boolean(...),`
  - `=, !=, <, <=, >=` (à écrire < et >)
- Expression Chaîne
  - …

Opérateurs (suite)

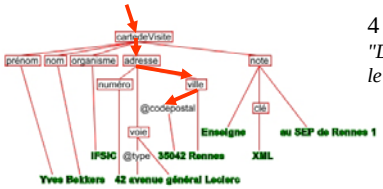
- Expressions chaîne
  - `string(exp)`
  - `concat(exp1, exp2, …)`
  - `substring(expr,start),`
  - `substring(expr,start,length)`
  - `substring-before(expr,expr)`
  - `substring-after(expr,expr)`

Conversions de type

- Explicites
  `boolean(1 + 1)` ➔ `true`
  `boolean(1 – 1)` ➔ `false`
  `string-length("Boo")` ➔ `3`
- Implicites
  `boolean("Boo")` ➔ `true`
  équivalent à `boolean("Boo")`

Type ensemble de nœuds

Chemin absolu dans un arbre

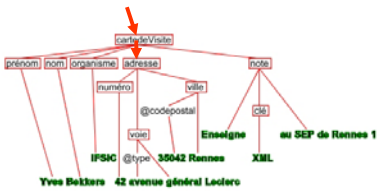


4 étapes
"Depuis la racine, le code postal"

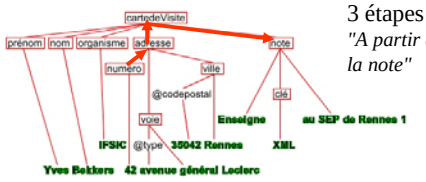
- Langage de chemin strict
`/child::cartedeVisite/child::adresse`
`/child::ville/attribute::codepostal`
- Langage de chemin étendu
`/cartedeVisite/adresse/ville/@codepostal`

Conversions de type

`boolean(/root)` ➔ `false`
`boolean(/cartedeVisite/adresse)` ➔ `true`
`count(/cartedeVisite)` ➔ `1`



Chemin relatif



3 étapes
"A partir du numéro, la note"

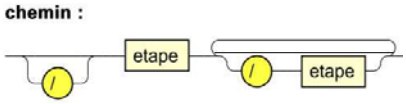
- Langage de chemin strict
`parent::*/parent::*/child::note`
- Langage de chemin étendu
`../../note`

Composition des étapes

- Un chemin est évalué comme la composition des étapes interprétées de gauche à droite avec
  - Un contexte initial fournit par l'application cliente
  - Tout nœud résultat d'une étape est utilisé comme nœud d'entrée de l'étape suivante
- Le résultat est un ensemble de nœuds

Expression de chemin

- séquence d'étapes



- Chemin absolu
  - /étape1/étape2/étape3/...
- Chemin relatif
  - étape1/étape2/étape3/...

Associativité

- Xpath
  `(carnetDAdresse/carteDeVisite)/nom`
- Est équivalent à
  `carnetDAdresse/(carteDeVisite/nom)`

Racine d'un document

- La racine d'un document est au dessus de *l'élément racine du document*, elle contient
  - des commentaires éventuels
  - des instructions de traitements éventuelles
  - un et un seul élément (l'élément racine)

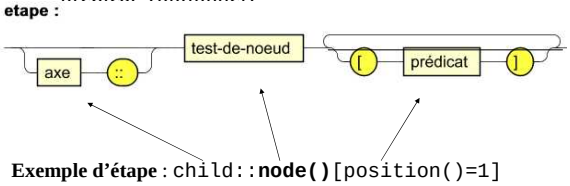
`/`	Racine du document
`/child::html`	Elément racine du document si c'est html
`/child::*`	Elément racine du document

Contexte

- Un pas de localisation est évalué par rapport à un contexte courant
- Le contexte d'interprétation d'une étape XPath est composé de
  - Un nœud contextuel
  - Une position et une taille contextuelle (2 entiers)
  - Des liaisons de variables
  - Une librairie de fonctions
  - Des déclarations d'espaces de noms
- Le contexte est tenu à jour par les applications externes qui utilisent XPath (XPointer, XSLT, XQuery)

Étape = sélectionner des nœuds

- Une étape est composée de trois parties
  - axe de déplacement (optionnel)
  - **Sélection de nœuds par leur type** (obligatoire)
  - prédicat (optionnel)



Exemple d'étape : `child::node()[position()=1]`

Prédicat - exemples

```
[@codepostal='35700']
[nom[text()='Bekkers']]
[nom='Bekkers']
[position()=last()-1]
[not(position()=1)]
[boolean(clé)]
```

- Attention

  L'opérateur "=" accepte les ensembles en tant qu'opérande avec la sémantique suivante :

  "{a,b} = {c,d,e}" est *vrai* ssi il existe au moins couple d'éléments <x,y>, x dans {a,b} et y dans {c,d,e} tel que x=y

Séquence de plusieurs prédicats

- le dernier fils du nœud courant pourvu que ce soit un élément « ville » ou un élément « rue »

```
[last()][ville or rue]
```

- le dernier des fils « ville » ou « rue » du nœud courant

```
[ville or rue][last()]
```


SELECT DISTINCT de SQL

- tous les premiers éléments "ville" du document différents les uns des autres

```
//ville[not(text()=preceding::ville/text())]
```

- équivalent de *not in*

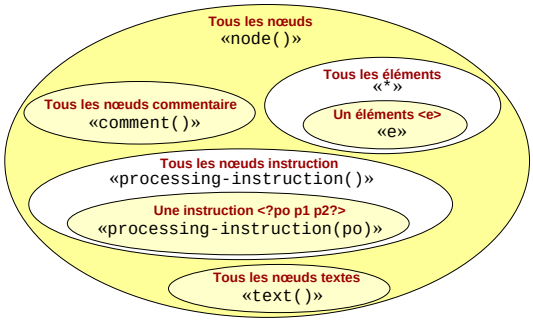
Test de type de nœud

Langage strict	Langage étendu
1. Élément `note` `Child::note`	`note`
2. Élément quelconque `Child::*`	`*`
3. Attribut `codepostal` `attribute::codepostal`	`@codepostal`
4. Attribut quelconque `attribute::*`	`@*`
5. Texte `Child::text()`	`text()`

Test de type de nœud (suite)

- Commentaire
  - `<!-- ceci est un commentaire -->`
  - `child::comment()`
- Instruction de traitement dont le nom est `monNom`
  - `<?monNom arg1 arg2 arg3?>`
  - `child::processing-instruction(monNom)`
- Toute instruction de traitement
  - `child::processing-instruction()`

Test de type de nœud

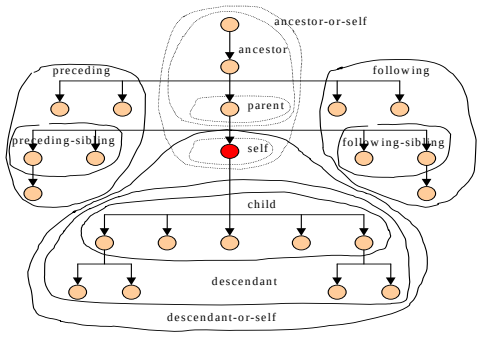


Exemple d'étape : `child::node()`

Résumé

Test	Nœuds sélectionnés
`*`	Tout élément
`ville`	Elément de nom `ville`
`text()`	Tout nœud de type texte
`processing-instruction()`	Toute instruction de traitement
`processing-instruction('proc')`	Processing instruction dont le nom est `proc`
`comment()`	Tout nœud commentaire
`node()`	Tout nœud

Axes de déplacement



+ attribute

Axe `following-sibling`

```
<a>
  <b1/>
  <b2>
     <c21/>
     <c22/>
  </b2>
  <b3>
     <c31/>
     <c32/>
  </b3>
  <b4/>
</a>
```

Expression Xpath

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

`/c21/following-sibling::*`

Réponse : 1 élément

`<c22/>`

Raccourci

/ descendant-or-self::node() = //

Axe `following`

```
<a>
  <b1/>
  <b2>
     <c21/>
     <c22>
        <d/>
     </c22>
  </b2>
  <b3>
     <c31/>
     <c32/>
  </b3>
  <b4/>
</a>
```

Expression Xpath

`//c21/following::*`

Réponse : 6 éléments

`<c22/>`

`<d/>`

`<b3>`

`<c31/>`

`<c32/>`

`<b4/>`

Axe `ancestor-or-self`

```
<a>
  <b1/>
  <b2>
     <c21/>
     <c22/>
  </b2>
  <b3>
     <c31/>
     <c32/>
  </b3>
  <b4/>
</a>
```

Expression Xpath

`//c22/ancestor-or-self::*`

Réponse : 3 éléments

`<c22/>`

`<b2>`

`<a>`

Axe `attribute`

```
<a>
  <b1/>
  <b2>
     <c21/>
     <c22/>
  </b2>
  <b3 id='i1'>
     <c31/>
     <c32/>
  </b3>
  <b4/>
</a>
```

Expression Xpath

`/*/*[attribute::id='i1']`

`/*/*[@id='i1']`

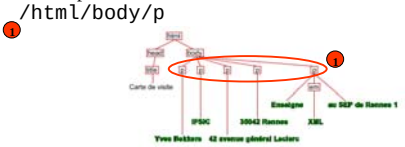
Réponse : 1 élément

`<b3 id='i1'>`

Filtrage à l'aide du prédicat

Une expression XPath s'évalue en un ensemble de nœuds

```
/html/body/p
```



Filtrage par la position

```
/html/body/p[position()=1]
```

Filtrage par le contenu

```
/html/body/p[em]
/html/body/p[position()=last()]
                        /em[text()='XML']
```


Les raccourcis importants

Langage strict	Langage étendu
`child::nom`	`nom`
`child::*`	`*`
`attribute::id`	`@id`
`attribute::*`	`@*`
`descendant-or-self::node()`	`//`
`self::node()`	`.`
`parent::node()`	`..`
`child::personne [string(nom)='Bekkers']`	`personne[nom='Bekkers']`

Conversions de type implicites

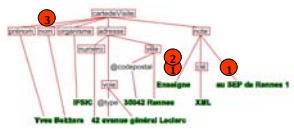
- Un prédicat en langage étendu
  `[.='Bekkers']`
- En langage strict
  `[self::node()='Bekkers']`
- En langage strict avec conversion explicite
  `[string(self::node())='Bekkers']`

Conversions de type implicites (2)

- Un prédicat en langage étendu
  `[clé]`
- En langage strict
  `[child::clé]`
- En langage strict avec conversion explicite
  `[boolean(child::clé)]`

Quelques exemples

Quelques exemples



1. `/descendant-or-self::note/child::text()` *langage stricte*
   `//note/text()` *langage étendu*

2. `/descendant-or-self::note/child::text()[position()=1]`
   `//note/text()[position()=1]`
   `//note/text()[1]`

3. `/descendant-or-self::nom[string(self::nom)='Bekkers']`
   `//nom[.='Bekkers']`

Quelques exemples

`//figure`	Tous les éléments "figure" du document
`figure`	Tous les éléments "figure" fils du nd courant
`./figure`	Tous les éléments "figure" fils du nd courant
`*/figure`	Tous les éléments "figure" petit fils du nd courant
`adresse/@rue`	Les attributs "rue" des éléments "adresse" fils du nd courant
`adresse/text()`	Tous les textes situés directement sous l'élément "adresse"

Quelques exemples - suite

`*[last()]`	Le dernier fils du nœud courant
`figure[lg]`	Tous les éléments "figure" fils du nd courant, pourvu qu'ils aient un fils "lg"
`ancestor::tst`	L'élément "tst" englobant le plus intérieur
`./@*`	Tous les attributs de l'élément courant
`XXX[@WIDTH and not(@WIDTH="20")]`	Les éléments XXX fils du nd courant pourvu qu'ils aient un attribut "WIDTH" avec une valeur différente de 20

Opérateur union

- Tous les éléments pere ou mere du document

```
//pere | //mere
```


Quelques questions

Un fichier source

```
<carnetDAdresse>
  <carteDeVisite id="i1">
    <prénom>Yves</prénom>
    <nom>Bekkers</nom>
    <organisme>IFSIC</organisme>
    <adresse>
      <numéro>42</numéro>
      <voie type="avenue">général Leclerc</voie>
      <ville codepostal="35042">Rennes</ville>
    </adresse>
    <note>Enseigne <clé>XML</clé>
       au SEP de Rennes 1</note>
  </carteDeVisite>
…
```


Questions

1. Tous les éléments `<note>`
2. Contenus et attributs des éléments `<note>`
3. Le troisième élément `<carteDeVisite>`
4. Le contenu de l'élément `<adresse>` de la troisième carte de visite

XPath 2

La séquence

- Introduction de la *séquence*
  - Une séquence est composée de 0 *items* ou plus
  - Un item est soit un *nœud* soit une *valeur simple*
  - Une séquence peut être non homogène
- Notation
  - Séquence vide : `()`
  - Séquence singleton : `(1)`
  - Séquences générales : `(1,2,3) (1,personne)`

Exemple 2

Expression XPath	*Résultat*
`10,12,20` ou `(10,12,20)`	`10 12 20`
`("a")`	`a`
`()`	
`10,'bonjour',true()`	`10 bonjour true`

Exemple 3

- Document

```
<order>
   <item name="choux" quantité="5" prix="9"/>
   <item name="carotte" quantité="3" prix="15"/>
   <item name="pomme_de_terre" quantité="10" prix="7"/>
   <item name="tomate" quantité="4" prix="17"/>
</order>
```

Expression	*Résultat*
`item/@name`	`choux carotte pomme_de_terre tomate`
`for $i in item return concat($i/@name,',', $i/@quantité,',',$i/@prix)`	`choux,5,9 carotte,3,15 pomme_de_terre,10,7 tomate,4,17`
`sum(for $i in item return $i/@quantité*$i/@prix)`	`228`

Expression `for`

Expression	*Résultat*
`for $i in (10,20) return ($i+1)`	`11 21`
`for $i in (10,20), $j in (1,2) return ($i+$j)`	`11 12 21 22`
`for $i in (10 to 14), $j in ($i*$i) return $j`	`100 121 144 169 196`
`sum(for $i in (10 to 14), $j in ($i*$i) return $j)`	`730`
`for $i in item return concat($i/@name,',', $i/@quantité,',',$i/@prix)`	`choux,5,9 carotte,3,15 pomme_de_terre,10,7 tomate,4,17`
`sum(for $i in item return $i/@quantité*$i/@prix)`	`228`

Fonction `index-of`

- Déclaration

```
index-of($seqParam as xs:anyAtomicType*,
         $srchParam as xs:anyAtomicType)
         as xs:integer*
```

- Exemples
  - `index-of((10, 20, 30, 40), 35)`
    rend `()`
  - `index-of((10, 20, 30, 30, 20, 10), 20)`
    rend `(2, 5)`
  - `index-of(("a", "sport", "a", "not"), "a")`
    rend `(1, 3)`

Fonction `remove`

- Déclaration
  - **remove**($target **as** item()*, $position **as** xs:integer) **as** item()*
- Exemples si `$x = ("a", "b", "c")`
  - remove($x, 0) rend ("a", "b", "c")
  - remove($x, 1) rend ("b", "c")
  - remove($x, 6) rend ("a", "b", "c")
  - remove((), 3) rend ()

Fonction `empty`

- Déclaration
  empty($arg **as** item()*) **as** xs:boolean
- Exemples
  - empty(remove(("hello", "world"), 1)) rend false.

Expressions booléennes et quantifications

Expression	*Résultat*
1>2	false
count(item)=4	true
some $q **in** item/@quantité **satisfies** $q=5	true
some $q **in** item/@quantité **satisfies** $q=50	false
every $q **in** item/@quantité **satisfies** $q>2	true
every $q **in** item/@quantité **satisfies** $q>6	false

Expressions `if`

Expression	*Résultat*
if (1>2) **then** 'vrai' **else** 'faux'	faux

Les types - expression `instance of`

Expression	*Résultat*
Valeurs atomiques et éléments	
1 **instance of** xs:integer	true
1 **instance of** xs:decimal	true
'bonjour' **instance of** xs:decimal	false
'bonjour' **instance of** xs:string	true
. **instance of** element()	true
. **instance of** element(order)	true
. **instance of** element(nom)	false

Les types - expression `instance of` (2)

Séquences	
1,2 **instance of** xs:decimal+	1 true
(1,2) **instance of** xs:decimal+	true
. **instance of** element()+	true
. **instance of** element()*	true
item **instance of** element()	false
item **instance of** element()+	true
item **instance of** element()*	true

Convertir une chaîne en une date

Expression	*Résultat*
`xs:date('2002-04-30')`	`30-04-2002`
`xs:dateTime('1966-07-31T15:00:00Z')`	`1966-07-31T15:00:00Z`

`Cast as, Treat as castable as`

- L'expression `E cast as T` convertit la valeur de l'expression `E` dans le type `T`. `T` doit être un type atomique (QName).
- L'expression `E treat as T` est conçue pour effectuer des tests de type statique
- L'expression `E castable as T` test si la valeur de `E` est convertible en une valeur de type `T`. `T` doit être un type atomique.

Exemple

- Traiter une valeur en fonction de son type

```
if ($x castable as voiture)
   then $s cast as voiture
   else if ($x castable as personne)
   then $s cast as personne
   else $x cast as xs:string
```


Différence d'ensemble et intersection

- Différence d'ensemble
  - L'expression `E1 except E2` sélectionne tous les nœuds qui se trouvent dans `E1` sauf s'ils sont dans `E2`
  - Les deux expressions doivent retourner une séquence de noeuds.
  - Les résultats sont rendus dans l'ordre du document d'entrée.
- Intersection
  - L'expression `E1 intersect E2` sélectionne tous nœuds qui se trouvent dans `E1` et dans `E2`.
  - Les deux expressions doivent retourner une séquence de noeuds.
  - Les résultats sont rendus dans l'ordre du document d'entrée.

Union

- L' opérator | est présent XPath 1.0
- Le mot clé `union` est un synonyme de la barre dans XPath2.
- L'expression `E1 | E2` sélectionne tous nœuds qui se trouvent dans `E1` ou dans `E2` ou dans les deux.
  - Les deux expressions doivent retourner une séquence de noeuds.
  - Les résultats sont rendus dans l'ordre du document d'entrée

Conclusions XPath

- XPath est puissant et indispensable
  - permet d'adresser n'importe quel nœud d'un document par sa position ou par contexte
  - Il est utilisé dans de nombreuses autres normes
    - XPointer (XLink, XInclude)
    - XSLT
    - XQuery
- Mais XPath n'est pas fait pour remplacer SQL et les bases de données
  - efficacité ?
- Dans une feuille de style XSLT, XPath est rarement employé dans toute sa généralité

Visualiser des résultats d'évaluation XPath

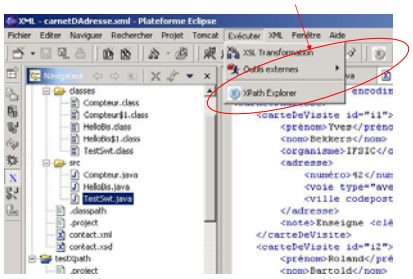
Visualiser des résultats d'évaluations XPath

- Voici trois moyens pour tester un chemin XPath
  1. Avec MSExplorer 6 et son XPath Visualizer
     Se présente comme une page HTML
     Utilise MSXML 3 (qu'il faut installer)
     `http://www.vbxml.com/xpathvisualizer/`
  2. XMLSpy et sa commande `Evaluate XPath`
     `http://www.altova.com`
  3. Eclipse et son plugin XPathExplorer
     http://sourceforge.net/project/showfiles.php?group_id=54719
  4. Eclipse et Xpath Developper
     http://www.bastian-bergerhoff.com/eclipse/features

Eclipse et le plugin XPathExplorer

1. Lancer Eclipse
2. dans une perspective quelconque
3. Choisir le menu **`Exécuter> XPath Explorer`**

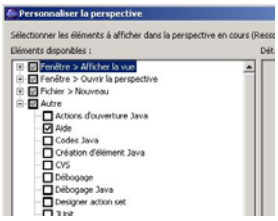
Icônes XPath Explorer



XPathExplorer (lancé par Eclipse)

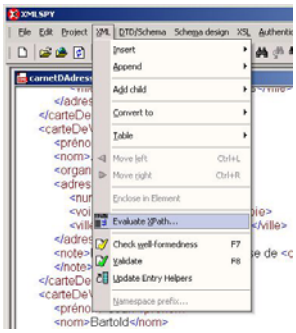
Faire venir les icônes XPath Explorer dans la perspective courante

- Si les icônes d'XPath Explorer ne sont pas visibles dans la perspective courante, les rendre visibles comme suit :
  - Menu **`Fenêtre>Personnaliser la perspective …`**
  - Ouvrir **`Autre`**
  - sélectionner **`XPath Explorer`**



XMLSpy

1. Lancer XMLSpy
2. Ouvrer le document XML à explorer
3. Choisir le menu **`XML>Evaluate XPath …`**



Fenêtre d'évaluation XPath d'xmlSpy