Le langage de requêtes XPATH

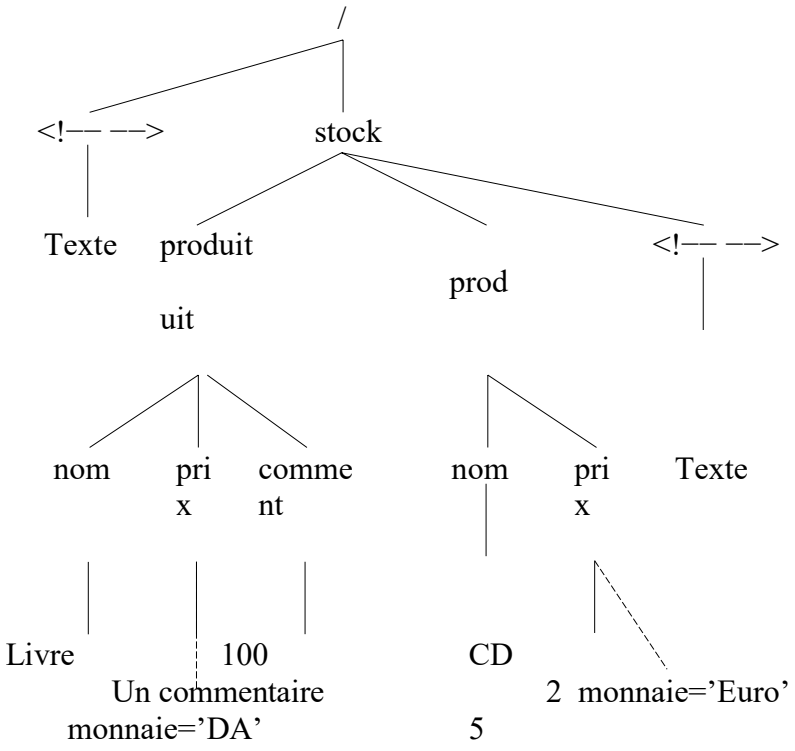
Table des matières

1 Introduction

1.1 Présentation

- Le langage XPATH offre un moyen d'identifier un ensemble de nœuds dans un document XML.
- Toutes les applications ayant besoin de repérer un fragment de document XML peuvent utiliser ce langage.
- Les feuilles de style XSL, les pointers XPOINTER et les liens XLINK utilisent de manière intensive les expressions XPATH.
- XPATH est un premier pas vers un langage d'interrogation d'une base de données XML (XQuery).

1.2 Structure d'arbre d'un document XML




```
<!-- Texte -->
<stock>
  <produit>
    <nom> Livre </nom><prix  monnaie="DA">  100  </prix>
    <comment> Un commentaire  </comment>
  </produit>
  <produit>
    <nom> CD </nom><prix monnaie="Euro"> 25 </prix>
  </produit> <!-- Texte -->
</stock>
```

1.3 Les expressions XPATH

La forme générale d'une expression XPATH est :

- `10` ou `10.12` un nombre réel (codé en virgule flottant)
- `'Hello'` ou `"Hello"` une chaîne de caractères
- `nom de fonction()` l'appel à une fonction
- `expr1 + expr2` une opération (`+` , `-` , `mod` , `div`) (nombre)
- `expr1 = expr2` un test d'égalité (`=` , `!=`) (boolén)
- `expr1 > expr2` une comparaison (`<` , `<=` , `>` , `>=`) (boolén)
- `expr1 or expr2` ou `expr1 and expr2` (booléen)
- `(expression)` une parenthèse
- `chemin` un chemin (ensemble de nœuds)

1.4 Les chemins

Un **chemin** a la forme suivante :

- `/sélecteur1/sélecteur2/...` un chemin absolu
- `sélecteur1/sélecteur2/...` un chemin relatif

Chaque **sélecteur** sélectionne un ensemble de nœuds en fonction du résultat du sélecteur précédent.
L'ensemble initial est soit le nœud courant (forme relative) soit la racine (forme absolue).
Exemple : /stock/produit/comment

```
{ /                                              } --> stock -->
{ /stock                                         } --> produit -->
{ /stock/produit[1], /stock/produit[2] } --> comment -->
{ /stock/produit[1]/comment }
```

1.5 Les sélecteurs de nœuds

Les sélecteurs de nœuds sont de la forme :

axe`::`*filtre*`[` *condition1* `][` *condition2* `]...`

- l'**axe** indique un sens de recherche,
- le **filtre** sélectionne un type de nœud,
- les **conditions** sélectionnent sur le contenu.

Les parties *axe*`::` et `[`*condition*`]` sont optionnelles.

2 Les axes de recherche

2.1 Les axes en avant

Les axes qui permettent de descendre dans l'arbre :

child les fils du nœud courant (c'est l'axe par défaut). Ces deux expressions sont identiques :

```
produit/child::nom
```

```
produit/nom
```

self le nœud courant

```
/stock/produit[ prix > 10 ]/self::produit[ prix < 99 ]
```

```
/stock/produit/self::prix                    résultat vide
```

descendant les descendants du nœud courant

```
/stock/descendant::prix
```

descendant-or-self les descendants du nœud courant plus lui-même.

2.2 Les axes en arrière

Les axes qui permettent de remonter dans l'arbre :

parent le nœud parent du nœud courant (si il existe),

```
/stock/produit/comment/parent::produit/prix
```

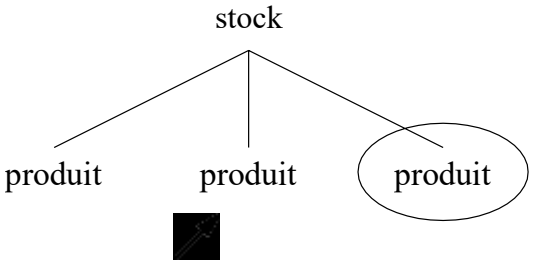
ancestor les ascendants du nœud courant

```
/stock/produit/comment/ancestor::stock/produit
```

ancestor-or-self les ascendants du nœud courant ou le nœud courant

2.3 Les axes à droite et à gauche

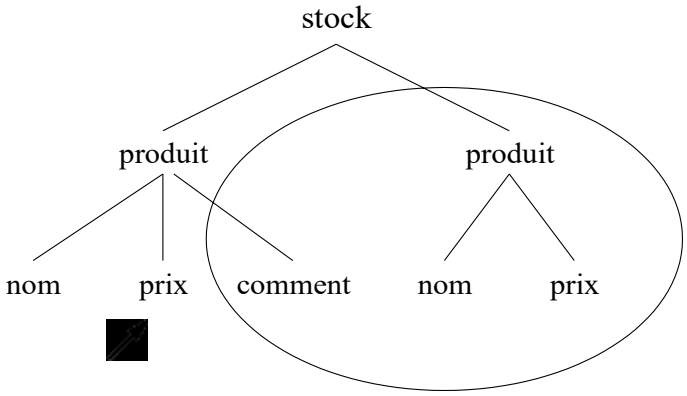
following-sibling les nœuds frères placés après le nœud courant,



preceding-sibling les nœuds frères placés avant le nœud courant,

2.4 Les axes avant et après

following les nœuds placés après dans le document,



preceding les nœuds placés avant dans le document,

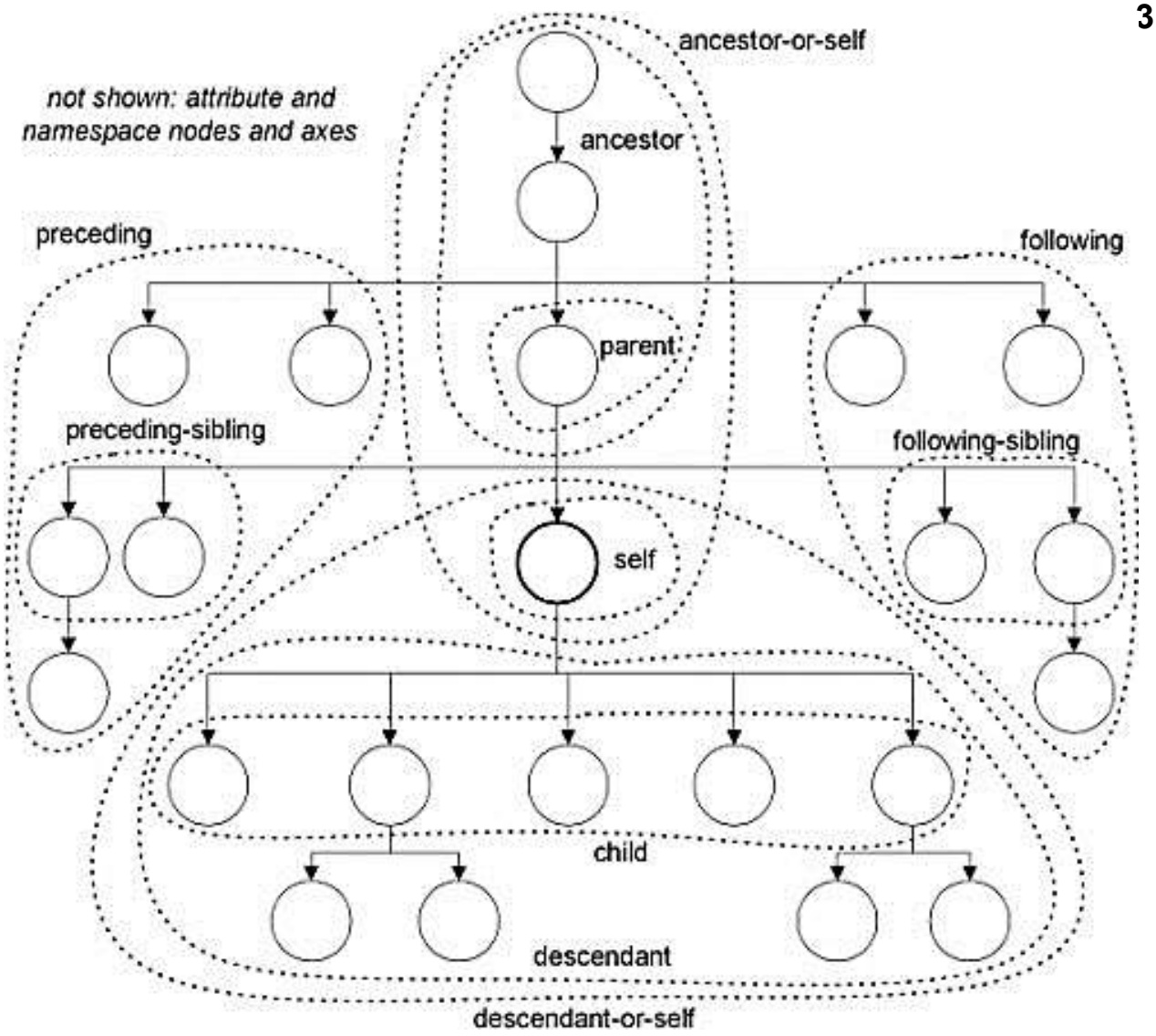
2.5 Les axes pour les attributs

attribute les nœuds de type attribut du nœud courant,

```
/stock/produit/prix/attribute::monnaie
```

namespace les nœuds de type espace de nom du nœud courant,

3 Les



filtres

3.1 Filtrer les nœuds nommé

identificateur *les nœuds de l'axe qui portent ce nom.*

```
/stock/produit/prix/attribute::monnaie
```

* les nœuds de l'axe qui ont un nom (attribut ou élément)

```
/stock/*/prix/attribute::*
```

3.2 Filtrer les nœuds textuels

text() tous les nœuds de type texte de l'axe.

```
/stock/produit/*/text()
```

3.3 Filtrer les commentaires

comment() tous les nœuds de type commentaire de l'axe.

3.4 Filtrer les instructions de traitement

processing-instruction() tous les nœuds de type instruction de traitement de l'axe,

3.5 Filtrer les nœuds

node() tous les nœuds de l'axe sauf la racine,

```
/stock/produit/prix/node()
```

4 Simplifications

Afin d'éviter une trop grande lourdeur, les simplifications suivantes sont autorisés :

originale	simplifiée	exemple
child::		/stock/produit
attribute::	@	/prix/@monnaie
/descendant-or-self::node()/	//	//prix
self::node()	.	prix[. = 10]
parent::node()	..	prix/../nom
[position() = x]	[x]	produits[4]

5 Les types de base de XPATH

Il existe quatre types de base : booléen, chaine de caractères, nombre réel et ensemble de nœuds.

Chaîne de caractères :

- 'chaine de caractères' ou "chaine de caractères"
- string(expression)

```
string(10)            = '10'
string(20.300)        = '20.3'
string(/stock/*/nom) = ' Livre '          (le premier noeud)
string(/stock/produit) = ' Livre 50 ... ' (concaténation des nœuds text())
string(/stockx)       = ''
```

Booléen :

- true() ou false()
- boolean(expression) renvoie vrai ssi l'argument est
  - ▷ le booléen vrai
  - ▷ un nombre différent de zéro
  - ▷ la chaine de longueur supérieure à zéro
  - ▷ un ensemble de nœuds non vide

```
boolean(10)              = true
boolean(0)               = false

boolean('hello')         = true
boolean('')              = false

boolean(true())          = true
boolean(false())         = false

boolean(/stock/produit) = true
boolean(/stock/comment) = false
```

Nombre réel :

- 100 ou 234.56
- number(expression)

```
number(10)                  = 10

number('10.12')             = 10.12
number(' 10.120 ')          = 10.12
number(' 10x ')             = NaN (not a number)

number(true())              = 1
number(false())             = 0

number(/stock/*/prix)  = 50 (le premier)
number(/stock/prix)         = NaN
```

6 Les conditions

Rappel : les sélecteurs de nœuds sont de la forme :

axe::*filtre*[*condition1*][*condition2*]...

Définition : Un sélecteur calcule

- les nœuds de l'**axe**
- qui respectent le **filtre** et
- pour lesquels les **conditions** (traduites en booléen) sont vraies.

6.1 Condition d'existence

Sélectionner un nœud en fonction de son contenu :

```
/stock/produit[ prix/attribute::monnaie ]
```

```
/stock/produit[ prix[ attribute::monnaie ] ]
```

Ces expressions sélectionnent les nœuds « produit », à condition qu'un prix existe avec l'attribut monnaie précisé.
Remarque : les sous-expressions sont souvent relatives.

6.2 Condition de position

numéro ***vraie ssi le nœud courant a cette position dans le contexte courant.***

```
produit[ 2 ][ comment ]
```

```
produit[ comment ][ 2 ]
```

La première expression sélectionne les nœuds « produit » en deuxième position si il possède un élément fils «comment», la seconde sélectionne le deuxième nœud « produit » qui possède un élément fil «comment » .

6.3 Les conditions logiques

Les relations portent sur deux sous-expressions XPATH :
exp1 relation exp2 vraie ssi il existe n1 (dans le résultat de *exp1*) et n2 (dans le résultat de *exp2*) qui respectent la relation.
Les relations possibles sont : = != < <= > >=

produit[prix = 100]
les produits qui ont un prix à 100.

`produit[ prix != 100 ]`
les produits qui ont un prix différent de 100.

`produit[ prix < //prix ]`
tous les produits sauf les plus onéreux.

Conversion : Pour l'égalité, la conversation est automatique pour les nombres et les booléens. Pour les relations d'ordre, la conversion en nombre est automatique.

Quelques expressions vraies :

- `(10 = '10.0')`
- `(10 < '12.0')`
- `('10.3' < '12.0')`
- `(true() = "faux")`

6.4 « and » / « or »

condition1 and ***condition2***
vraie ssi les deux conditions le sont également.

```
produit[prix > 10 and comment]
```

condition1 or ***condition2***
vraie ssi au moins une des deux conditions est vraie.

7 Fonctions & opérations

Les fonctions et opérations sont utilisables dans :

- les expressions logiques,
- les clauses `<xsl:value-of .../>` des feuilles de style XSL.

7.1 Opérations et fonctions sur les nombres

`+` , `-` , `mod` , `div` ,
Opérations utilisables sur les nombres (conversion automatique).

```
produit[(prix div 10) = (prix mod 10)]
```

`sum( nœuds )` renvoie la somme des nœuds après les avoir transformés en nombre.

`count( nœuds )` renvoie le nombre de nœuds.

`floor( nombre )` arrondi par le bas.

`ceiling( nombre )` arrondi par le haut.

`round( nombre )` arrondi par le plus proche.

7.2 Fonctions sur les booléens

`true()` **et** `false()`
constantes

`not( boolean )`
vraie ssi le paramètre est faux.

```
produit[not(prix != 100)]
```

tous les prix sont égaux à 100.

7.3 Fonctions sur les nœuds

last() renvoie le nombre de nœuds dans le contexte du nœud courant.

//produit[last()]	le dernier produit
//produit[last()-1]	l'avant dernier produit

position() renvoie la position du nœud courant dans son contexte.

//produit[position() = 1]	le premier produit
//produit[position()<last()]	les produits sauf le dernier

local-name(*nœud*) renvoie la partie locale de l'étiquette d'un nœud.

namespace-uri(*nœud*) renvoie la partie espace de nom de l'étiquette d'un nœud.

name(*nœud*) renvoie l'étiquette d'un nœud.
id(*chaines*) renvoie les nœuds identifiés par au moins une des étiquettes,

id('CD')/prix
id('CD Voiture SX2500 CD')/prix
id(/reference/attribute::ref)/prix

7.4 Fonctions sur les chaînes

string(*objet*) renvoie une version chaîne du paramètre.

concat(*chaîne1* , ..., *chaîneN*) Concaténation de chaînes.

string-length(*chaîne*) renvoie la longueur d'une chaîne.

normalize-space(*chaîne*) renvoie une version normalisée (suppression des blancs au début et à la fin et remplacement de toute suite de blancs par un seul).

```
normalize-space(' AB  CD  E ')  =  'AB CD E'
```

les **blancs** comprennent **l'espace**, le **retour à la ligne** (codes 10 et 13) et la **tabulation** (code 9).

translate(*ch1* , *ch2* , *ch3*) renvoie une copie de *ch1* dans laquelle les caractères présents dans *ch2* sont remplacés par les caractères de même position dans *ch3*.

```
translate('ABCD', 'AC', 'ac') = 'aBcD'
```

substring-before(*ch1* , *ch2*) renvoie la chaîne res définie par *ch1 = res + ch2 + reste*.
substring-after(*ch1* , *ch2*) renvoie la chaîne res définie par *ch1 = reste + ch2 + res*.
substring(*chaîne, début*) et **substring(*chaîne, début, len*)** extraction de sous-chaîne.

starts-with(*ch1* , *ch2*) vraie ssi *ch1* débute par *ch2*.

contains(*ch1* , *ch2*) vraie ssi *ch1* contient *ch2*.