XPath

Ingénierie documentaire et management des contenus

Bruno Bachimont

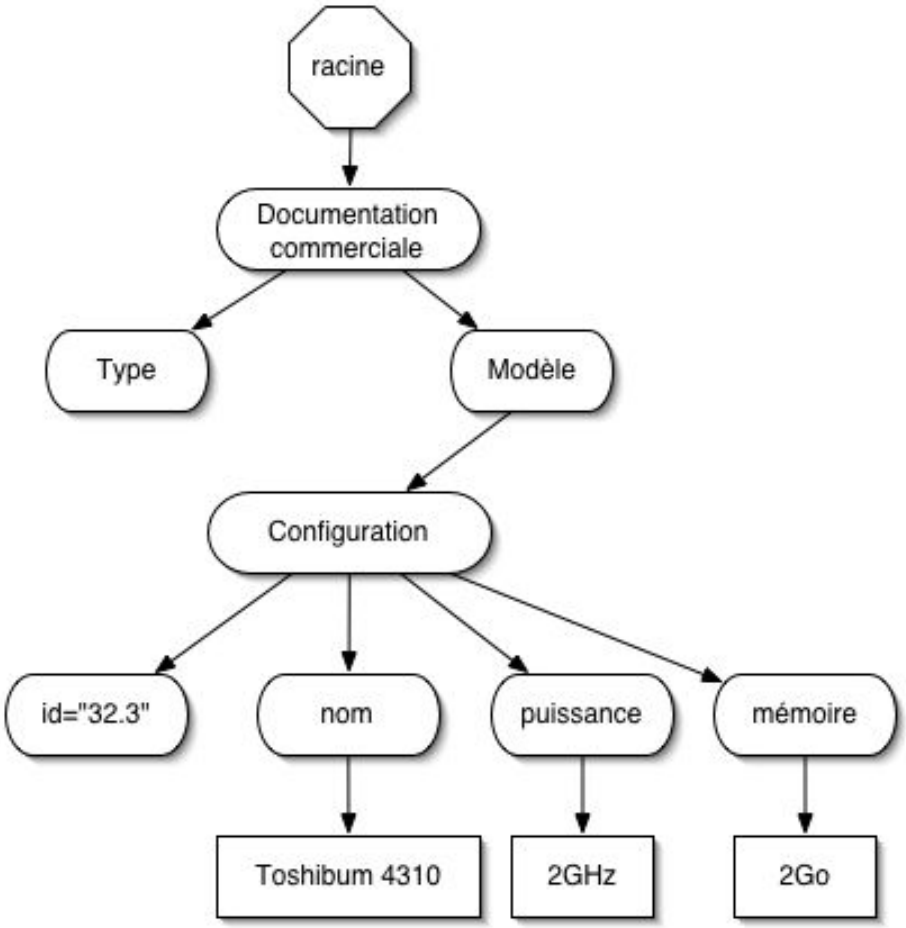
Université de Technologie de Compiègne

Filière « Ingénierie des connaissances et des supports d'information »

Présentation

- XPath est le langage proposé pour déterminer une manière de parcourir un chemin dans l'arbre d'un contenu XML pour trouver un ou plusieurs nœuds.
- Le langage XPath permet de sélectionner l'ensemble des nœuds que l'on peut atteindre en suivant, à partir d'un nœud donné de l'arbre d'un document, tous les chemins conformes à un modèle appelé chemin de localisation.

Arbre de document



Composition d'un arbre XML

L'arbre d'un document comporte 7 types de nœuds associés à chaque constituant d'un document :

- document ou racine,
- élément,
- texte,
- attribut,
- espace de noms,
- commentaire,
- instruction de traitement.

Enfants et descendants

- Les **enfants** du nœud **racine** sont ses nœuds fils de type commentaire ou instruction de traitement, et au moins un nœud élément.
- Les **enfants** d'un nœud **élément** sont ses nœuds fils de type élément, texte, commentaire ou instruction de traitement.
- Les **descendants** du nœud **racine** ou d'un nœud **élément** sont ses enfants ou les enfants de ses enfants.

  Les enfants et les descendants d'un nœud ne sont donc pas des nœuds de type attribut ou espace de noms.

Ordre du document (ordre des nœuds)

L'ensemble des nœuds de l'arbre d'un document est muni d'un ordre : l'ordre du document qui est l'ordre de lecture, dans le document XML, des constituants représentés par chaque nœud.

- l'ordre des nœuds du document est celui de leur balise de début correspondante en XML (après expansion des entités).
- Les nœuds d'attribut et les espaces de noms d'un élément sont classés avant les enfants de l'élément.
- Les nœuds d'espaces de noms sont rangés avant les nœuds d'attributs.
- Chaque nœud autre que le nœud racine a exactement un parent, qui est soit un nœud d'élément soit le nœud racine.

Valeur textuelle des nœuds

Chaque nœud a une valeur textuelle :

- la valeur textuelle du nœud racine est la concaténation des valeurs textuelles de ses descendants de type texte, dans l'ordre du document,
- la valeur textuelle d'un nœud élément est la concaténation des valeurs textuelles de ses descendants de type texte, dans l'ordre du document,
- la valeur textuelle d'un nœud texte est la chaîne de caractères constituant ce texte,
- la valeur textuelle d'un nœud attribut est la chaîne de caractères constituant la valeur de cet attribut.

Expression XPath

Une expression XPath a une valeur qui a l'un des 4 types suivants :

- ensemble de nœuds,
- chaîne de caractères UCS,
- nombre (nombre flottant conforme à la norme IEEE 754),
- booléen. Une expression XPath est évaluée dans un contexte constitué par :
- un nœud : le nœud contexte,
- deux entiers : la position contextuelle et la dimension ou taille contextuelle,
- un environnement (ensemble de liaisons variable-valeur),
- une bibliothèque de fonctions prédéfinies,
- les déclarations d'espaces de noms visibles dans l'expression.

Construction d'une expression XPath

Une expression XPath est construite à partir :

- de constantes littérales de type chaîne de caractères ou nombre,
- de noms de variable,
- de chemins de localisation,
- d'opérateurs sur les ensembles de nœuds : parcours de chemin, filtrage, union, . . .
- d'opérateurs arithmétiques : `+ - * mod div` . . .
- d'opérateurs de comparaison : `= != > >= < <=` . . .
- de connecteurs logiques : `and, or, not` . . .
- d'opérateurs sur les chaînes de caractères : `contains`, . . .
- d'appels de fonctions prédéfinies.

Chemins et pas de localisation

Un chemin de localisation est un modèle de chemin dans l'arbre d'un document.

Un pas de localisation est une étape élémentaire composant un chemin de localisation.

Un chemin de localisation peut être absolu ou relatif :

Un chemin de localisation relatif commence au nœud contexte et est constitué d'une suite de pas de localisation. la séquence initiale sélectionne un certain nombre de nœuds qui sont utilisés un par un comme nœud contextuel de la prochaine étape. Les ensembles des nœuds identifiés par cette étape sont ensuite réunis.

Syntaxe : *pas de localisation*$_1$*/. . ./pas de localisation*$_n$

Un chemin de localisation absolu est un chemin de localisation relatif qui commence à la racine du document.

Un chemin absolu consiste en un / optionnellement suivi d'un chemin relatif. Un / par lui-même sélectionne le nœud racine du document contenant le nœud contextuel. S'il est suivi d'un chemin relatif, alors le chemin de localisation sélectionne l'ensemble des nœuds qui seraient sélectionnés par le chemin relatif s'appliquant au nœud racine du document contenant le nœud contextuel.

Syntaxe : */chemin de localisation relatif*

La valeur d'un chemin de localisation est l'ensemble des nœuds extrémité des chemins conforme à ce modèle.

Pas de localisation

- Un pas de localisation est caractérisé par :
  - un axe,
  - un test de nœud,
  - une suite éventuellement vide de prédicats.
- Syntaxe : *axe : :test de nœud* $prédicat_1$ *. . . prédicat*$_n$

Exemple : `child : :para[position()=1]`

- `child` est le nom de l'axe,
- `para` le nœud de test ;
- `[position()=1]` est un prédicat.

il faut choisir le premier nœud de type para fils du nœud courant.

Axes

Un axe sélectionne, dans l'arbre du document et à partir du nœud contexte, l'ensemble des nœuds qui peuvent être atteints en suivant une certaine direction :

l'axe « enfant » `child` contient les enfants directs du nœud contextuel

l'axe « descendant » `descendant` contient les descendants du nœud contextuel ; un descendant est un enfant ou un petit enfant etc... Aussi, l'axe descendant ne contient jamais de nœud de type attribut ou des noms d'espace.

l'axe « parent » `parent` contient le parent du nœud contextuel, s'il y en a un

l'axe « ancêtre » `ancestor` contient les ancêtres du nœud contextuel ; cela comprend son parent et les parents des parents etc... Aussi, cet axe contient toujours le nœud racine, sauf si le nœud contextuel est lui-même la racine.

l'axe « cible suivante » `following-sibling` contient tous les nœuds de même niveau suivant le nœud contextuel ; si celui-là est un attribut ou un espace de noms, le suivant ciblé est vide.

l'axe « cible précédente » `preceding-sibling` contient tous les prédécesseurs de même niveau du nœud contextuel ; si le nœud contextuel est un attribut ou un espace de noms, la cible précédente est vide.

l'axe « suivant » `following` tous les nœuds du même document que le nœud contextuel qui sont après le nœud contextuel dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.

l'axe « précédent » `preceding` contient tous les nœuds du même document que le nœud contextuel, qui sont avant lui dans l'ordre du document, à l'exclusion de tout ancêtre, des attributs et des espaces de noms.

l'axe « attribut » `attribute` contient les attributs du nœud

contextuel ; l'axe est vide quand le nœud n'est pas un élément

l'axe « espaces de noms » `namespace` contient les nœuds des espaces de noms du nœud contextuel ; l'axe est vide quand le nœud contextuel n'est pas un élément.

l'axe « réflexif » `self` contient seulement le nœud contextuel

l'axe « descendant-ou-réflexif » `descendant-or-self` contient le nœud contextuel et ses descendants.

l'axe « ancêtres-ou-réflexif » `ancestor-or-self` contient le nœud contextuel et ses ancêtres ; aussi, cet axe contiendra toujours le nœud racine.

Sens d'un axe

Un axe a un sens : avant ou arrière.

- Un axe dont les nœuds sont soit le nœud contexte, soit des nœuds qui suivent le nœud contexte dans l'ordre du document est un axe avant.
- Un axe dont les nœuds sont soit le nœud contexte, soit des nœuds qui précèdent le nœud contexte dans l'ordre du document est un axe arrière.
- l'axe self est à la fois un axe avant et arrière, les axes ancestor, ancestor-or-self, preceding et preceding-sibling sont des axes arrières, les autres axes sont des axes avants.

Type de nœud principal

Un axe a un type de **nœud principal** : Si un axe peut contenir des éléments, alors le type principal des nœuds est un élément ; sinon, c'est le type des nœuds que l'axe peut contenir,

- le type de nœud principal de l'axe `attribute` est « attribut »,
- le type de nœud principal de l'axe `namespace` est « espace de noms »,
- le type de nœud principal des autres axes est « élément ».

Test de nœud

Un test de nœud sélectionne parmi les nœuds de l'axe, ceux qui sont d'un certain type. Un test de nœud a l'une des formes suivantes :

`nom-qualifié` où nom-qualifié est un nom : sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe et dont le nom étendu est égal au nom étendu de nom-qualifié ;

`*` : sélectionne les nœuds de l'axe ayant le même type que le type principal de l'axe ;

`node()` : sélectionne tout nœud de l'axe ;

`text()` : sélectionne tout nœud de l'axe de type texte ;

`comment()` : sélectionne tout nœud de l'axe de type commentaire ;

`processing-instruction(n)` : sélectionne tout nœud de l'axe représentant une instruction de traitement de nom `n`.

Prédicats

Un prédicat filtre un ensemble de nœuds, relativement à un axe de parcours, de manière à produire un nouvel ensemble de nœuds. La position de proximité d'un nœud, dans un ensemble de nœuds, relativement à un axe de parcours, est définie comme étant la position du nœud dans l'ensemble ordonné de nœuds vers l'avant du document si le sens est vers l'avant (forward axis) et l'inverse (reverse axis) si le sens est vers l'arrière. La première position est 1.

- Pour chaque nœud de l'ensemble à filtrer, l'expression du prédicat (PredicateExpr) est évaluée en utilisant le nœud comme nœud contextuel ; le nombre de nœud de l'ensemble comme dimension contextuelle et la position de proximité (proximity position) du nœud dans l'ensemble comme position contextuelle ;
- si l'expression du prédicat (PredicateExpr) est vraie pour un nœud, il est inclu dans le nouvel ensemble, sinon il est rejeté.

Expression de prédicat

Un prédicat est composé d'une expression booléenne appelée expression du prédicat.

Syntaxe : `[expression]`

Filtrage

On part donc d'un ensemble E_1 de nœuds, donné par l'axe et le test de nœud appliqué au nœud courant.

On veut obtenir un ensemble E_2 qui resultera de l'application du prédicat sur E_1.

L'algorithme indicatif est le suivant :

1. taille contexte := cardinalité de E_1
2. $E_2 := \{\}$
3. Pour chaque nœud n de E_1 :
   (a) nœud contexte := n,
   (b) position contexte := position de proximité de n dans E_1,
   (c) `Si` $[e](n) = T$ `Alors` $E_2 := E_2 \cup \{n\}$

Valeur d'un chemin de localisation

Un chemin de localisation relatif $p_1/.../p_n$ a pour valeur l'ensemble de nœuds E_2 construit de la façon suivante :

1. E_1 := nœud contexte
2. Pour i de 1 à n :
   (a) $E_2 := \{\}$
   (b) Pour chaque nœud n de E_1 :
   - $nœudcontexte := n$
   - $E_2 := E_2 \cup valeur(p_i)$
   - $E_1 := E_2$

La valeur d'un chemin de localisation absolu /c est valeur(c) dans un contexte où le nœud contexte est le nœud racine.

Valeur d'un pas de localisation

La valeur d'un pas de localisation $a :: tp_1...p_n$ à partir d'un nœud contexte n est l'ensemble de nœuds E_{n+2} construit de le façon suivante :

1. E_1 := ensemble des nœuds sélectionnés par l'axe a à partir de n
2. E_2 := sous-ensemble des nœuds de E_1 qui vérifient le test de nœud t
3. Pour i de 1 à n :
   - E_{i+2} := filtrage de E_{i+1} par le prédicat p_i

Fonctions sur les ensembles de nœuds

Position et taille contexte Les expressions `position()` et `last()` ont pour valeurs respectives la position contexte et la taille contexte.

Union L'expression `e1|e2` a pour valeur l'union des ensembles de nœuds $valeur(e_1)$ et $valeur(e_2)$.

Parcours de chemin L'expression `e/p` a pour valeur l'ensemble des nœuds extrémité des chemins de localisation conformes au modèle `p` commençant à chaque nœud de l'ensemble de nœuds $valeur(e)$. L'expression `e//p` est équivalente à `e/descendant-or-self : :node() /p`

Fonctions sur les ensembles de nœuds

Filtrage : L'expression$e[p]$ produit l'ensemble de nœuds obtenu par filtrage de l'ensemble $valeur(e)$ par le prédicat p. L'axe de filtrage est l'axe `child`.

Accès par identifiant : L'expression $id(s)$ où s est une chaîne de caractères composée d'une suite d'identificateurs $n_1, ..., n_k$ est l'ensemble des nœuds élément identifiés par ces identificateurs.

Fonctions de conversion

	type de x							
	chaîne		nombre		booléen		ensemble de nœuds	
	vide	non vide	nul	non nul	faux	vrai	vide	non vide
string(x)	x		représentation x		”false”	”true”	valeur textuel de x qui est le 1er dans l’ordre du document	
number(x)	nb plus proche de x si x est un nombre, ”NaN” sinon		x		0	1	number(string(x))	
boolean(x)	false	true	false	true	x		false	true

Fonctions de comparaison

L'expression $v_1 \theta v_2$ (où θ est `=`, `!=`, `<`, `<=`, `>` ou `>=`) est vraie dans les cas suivants et fausse dans les autres :

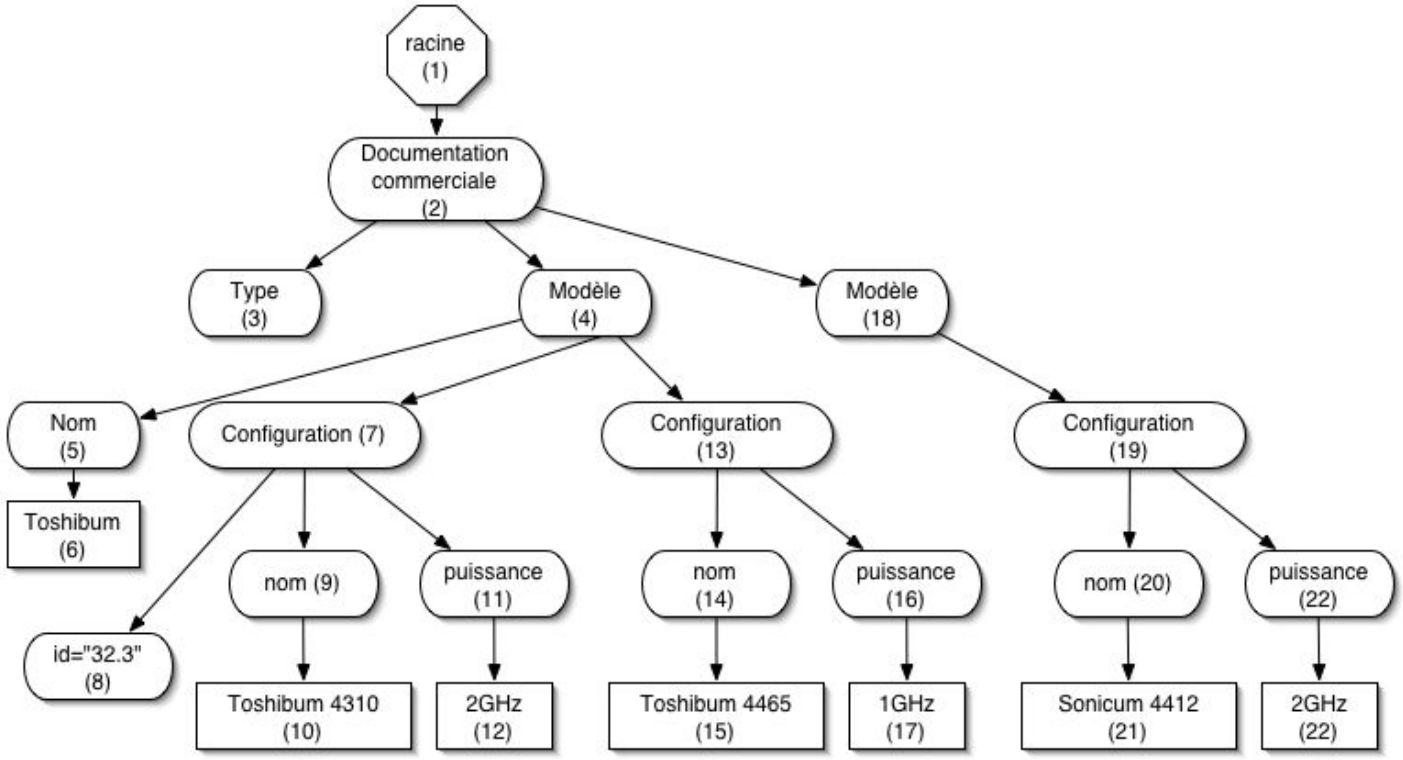
- v_1 et v_2 sont des ensembles de nœuds et il existe un nœud n_1 de v_1 et un nœud n_2 de v_2 tel que $valeur - textuelle(n_1) \theta valeur - textuelle(n_2)$ est vraie ;
- v_1 (resp. v_2) est un ensemble de nœuds et
  - v_2 (resp. v_1) est un booléen et `boolean`$(v_1)\theta v_2$ (resp. $v_1 \theta$ `boolean`(v_2)) est vraie ;
  - v_2 (resp. v_1) est une chaîne ou un nombre et il existe un nœud n de v_1 (resp. v_2) tel que $valeur - textuelle(n) \theta v_2$ (resp. $v_1 \theta valeur - textuelle(n)$) est vraie ;
- aucun des opérandes n'est un ensemble de nœuds et
  - θ est `=` ou `!=` et
    - l'un des opérandes est un booléen et `boolean`(v_1) θ `boolean`(v_2) est vraie ;

  - l'un des opérandes est un nombre et `number`(v_1) θ `number`(v_2) est vraie ;
  - l'un des opérandes est une chaîne et `string`(v_1) θ `string`(v_2) est vraie ;
- θ est `<`, `<=`, `>` ou `>=` et `number`(v_1) θ `number`(v_2) est vraie.

Conversion automatique

- Si la valeur v d'un prédicat n'est pas un booléen, elle est convertie en booléen de la façon suivante :
  - si v est un nombre égal à la position contexte, v est convertie en `true` sinon v est convertie en `false` ;
  - si v n'est pas un nombre, elle est convertie en `boolean`(v). On peut donc écrire `[i]` au lieu de `[position() = i]`.
- Si la valeur v de l'argument d'une fonction prédéfinie n'a pas le type attendu (c.-à-d. celui de l'argument formel correspondant), v est automatiquement convertie, si cela est possible, en utilisant les fonctions de conversion `string`, `number` et `boolean`.

Arbre de document



Exemple

```
/descendant : :modèle[child : :nom = "Toshibum"]
/child : :Configuration[position() = 2]
/child : :nom
/child : :text()
```

Exemple-1

- 1er pas :
  - nœud contexte 0 :
  - descendant → 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22
  - modèle → $\{4, 18\}$
  - `child : :nom = "Toshibum"` :
  - nœud contexte = 4
    - `child : :nom` → $\{5\}$
    - $\{5\}$ = `"Toshibum"` → `true`, car valeur-textuelle(5) = `"Toshibum"`
    - → le nœud 4 est sélectionné
  - nœud contexte = 18 : ne vérifie le prédicat
- → 4

2e pas

- nœud contexte : 4
- `child : :Configuration`! 7, 13
- `[position() = 2]`

nœud contexte 7 :

  - `position()` → 1 ;
  - `position() = 2`! false ;
  - → le nœud 7 est éliminé ;

nœud contexte 13 :

  - `position()` → 2 ;
  - `position() = 2` → `true` ;
  - → le nœud 13 est sélectionné ;

→ {13}

Et pour finir

`child : :nom` nom contexte = 13 ;

- `child` → {14, 16}
- `nom` → {14}

`child : :text()` nom contexte = 14

- `child` → {15}
- `text()` → {16}

Le résultat du chemin de localisation est `Toshibum 4465`

Abréviations

Afin de faciliter la lecture des chemins de localisation, les abréviations suivantes sont autorisées : l'axe child est l'axe par défaut :

- `t` est l'écriture abrégée de `child : :t`
- `@` est l'écriture abrégée de attribute `: :`
- `.` est l'écriture abrégée du pas de localisation `self : :node()`
- `..` est l'écriture abrégée du pas de localisation `parent : :node()`
- `//` est l'écriture abrégée de `/descendant-or-self : :node()/`

Exemples

– nom des machines « Sonicum » possédant une puissance supérieure à 1GHz :

```
/Documentation/Modèle[nom="Sonicum"]
/configuration[number(puissance) > 2])
/nom/text()
```

– nom de la deuxième machine Toshibum possédant une cotation « ** » :

```
/Documentation/Modèle[nom="Toshibum"]
/configuration[cotation = "**"] [2]) /nom/text()
```

N.B : on applique en cascade les prédicats : `[2]` s'applique au contexte rapporté par `[cotation = "**"]`, qui signifie : `[position() = 2]`.

Encore des exemples

- Nom de toutes les configurations de type « SAV compris » :
  `//configuration[@type = "SAV compris"] /nom/text()`
- Descriptions mentionnant les comparaisons de processeurs :
  `//description[ contains(text(), "benchmarking")]`
- nom de la configuration qui précède et de la configuration qui suit « Toshibum 4465 » :

```
((//configuration[nom="Toshibum
4465"]/preceding-sibling : :configuration[1] )
|
(//configuration[nom="Toshibum
4465"]/following-sibling : :configuration[1]))
/nom/text()
```

N.B. : cette expression n'est pas un chemin de localisation car elle ne commence pas par le symbole / ni par un pas/étape de localisation.