XPath

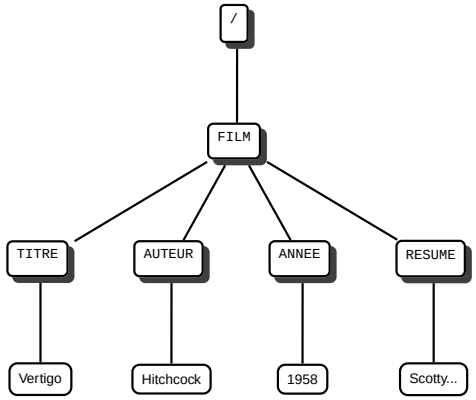
Bernd Amann et Philippe Rigaux

December 9, 2003

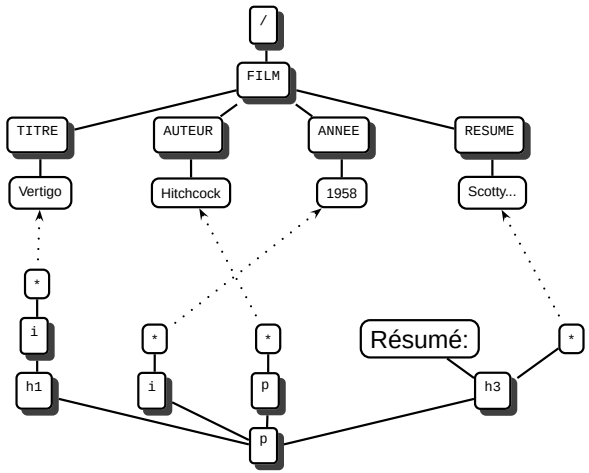
XPath: Sélectionner des Fragments XML

Exemple: Document XML

Document XML:

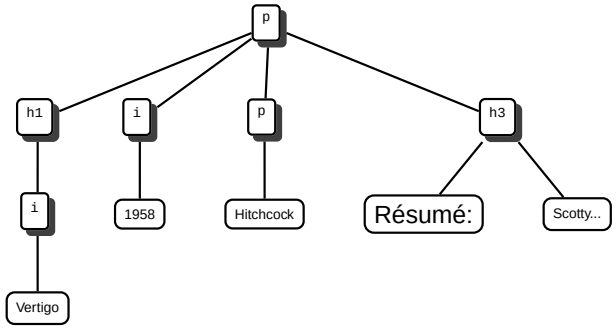


Exemple: Transformation d'un document XML



Exemple: Résultat HTML

Fragment HTML:



Référencer des fragments XML avec XPATH

Il faut pouvoir extraire des fragments (nœuds) XML d'un document ⇒ XPath.

XPath est utilisé par

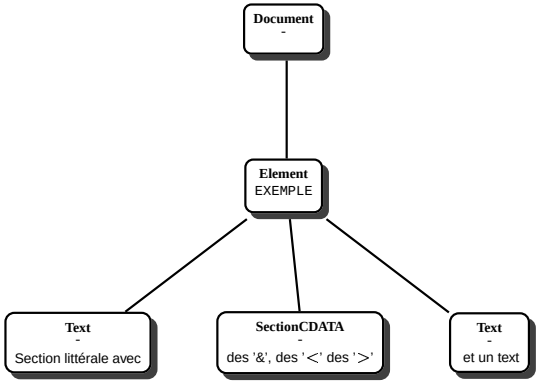
- XSLT pour sélectionner des règles de transformation
- XQuery pour l'interrogation de documents XML
- XML Schéma pour créer des clés et références
- XLink pour créer des liens entre documents/fragments XML

XPath: Le modèle

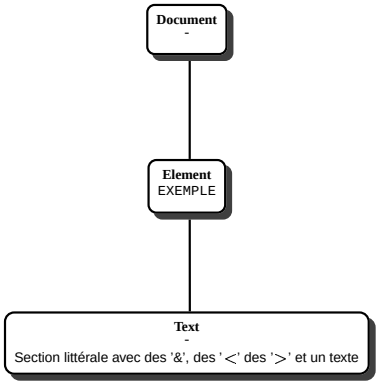
Le langage permet de désigner un ou plusieurs nœuds dans un document XML, à l'aide **d'expressions de chemin**.

- XPath est fondé sur une représentation arborescente (DOM) du document XML
- Objectif : référencer nœuds (éléments, attributs, commentaires, ...) dans un document XML
- Un typage simplifié par rapport à celui de DOM ⇒ pas d'entité, pas de section littérale

Exemple : l'arbre DOM...



Puis l'arbre XPath...



Expressions XPath : Sémantique

Une expression XPath :

- s'évalue en fonction d'un **nœud contexte**
- désigne un ou plusieurs chemins dans l'arbre à partir du nœud contexte
- a pour résultat
  - un ensemble de nœuds
  - ou une valeur, numérique, booléenne ou alphanumérique

Expressions XPath : Syntaxe

Un chemin XPath est une suite **d'étapes** :

$$[/]étape_1/étape_2/.../étape_n$$

Deux variantes : Un chemin peut être

- **absolu** : `/FILM/RESUME`

  Le nœud contexte est alors la racine du document.
- ou **relatif** : `RESUME/text()`

  Le nœud contexte est un nœud dans le document (pas forcément la racine).

Étapes XPath

Une étape : trois composants

[axe::]filtre[prédicat1][prédicat2]...

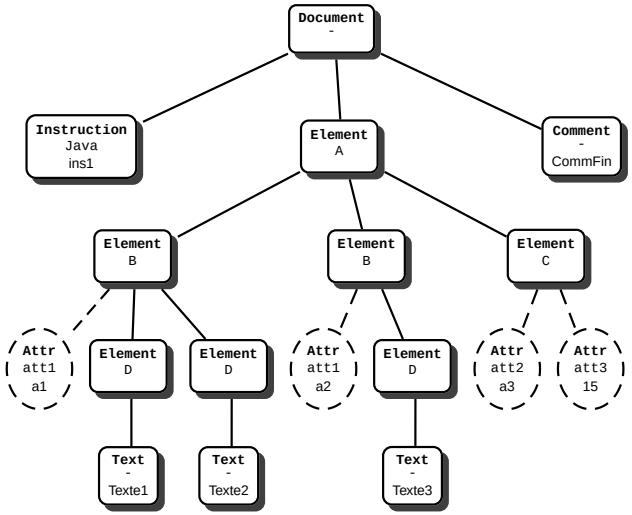
- **l'axe** : sens de parcours des nœuds (par défaut: *child*).
- **le filtre** : type des nœuds/noms des éléments qui seront retenus
- **le(s) prédicat(s)** que doivent satisfaire les nœuds retenus

Les axes XPath

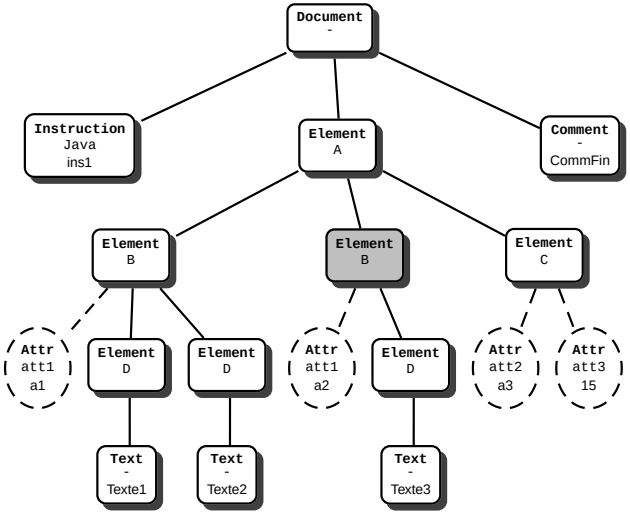
Un axe XPath recouvre les deux notions suivantes :

- un **sous-ensemble des nœuds de l'arbre** relatif au nœud contexte ;
- l'**ordre de parcours** de ces nœuds à partir du nœud contexte.

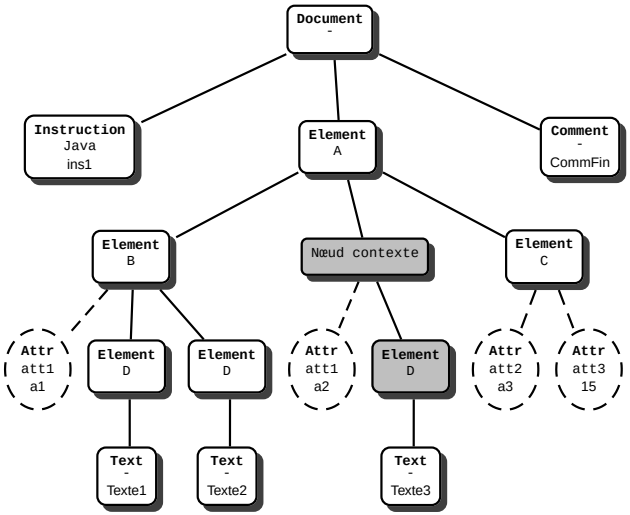
Exemple de référence



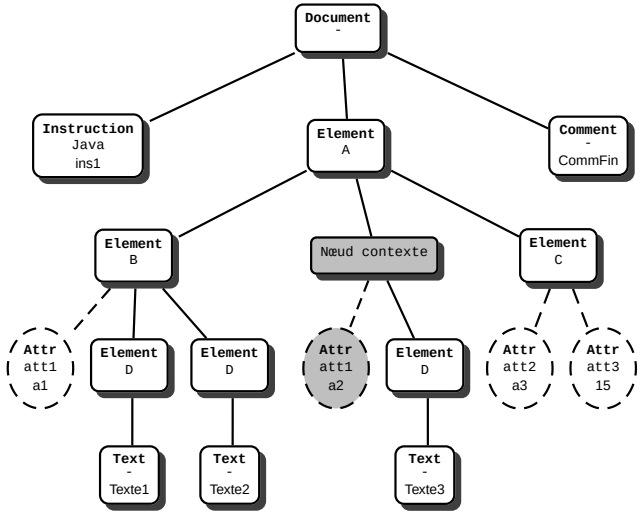
Exemple : le nœud contexte



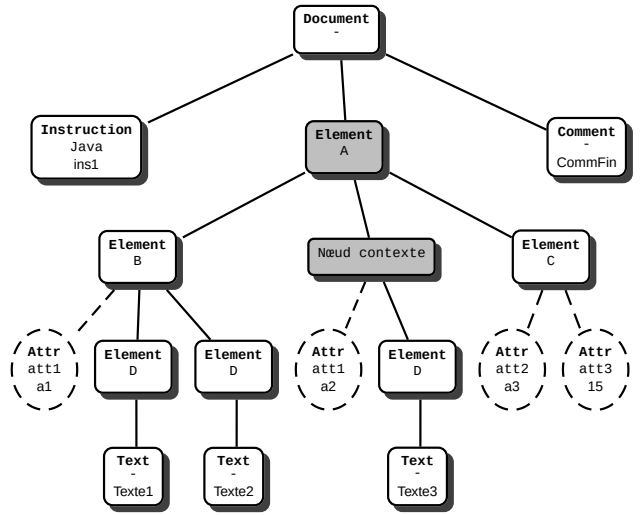
`child::D` ou `D`



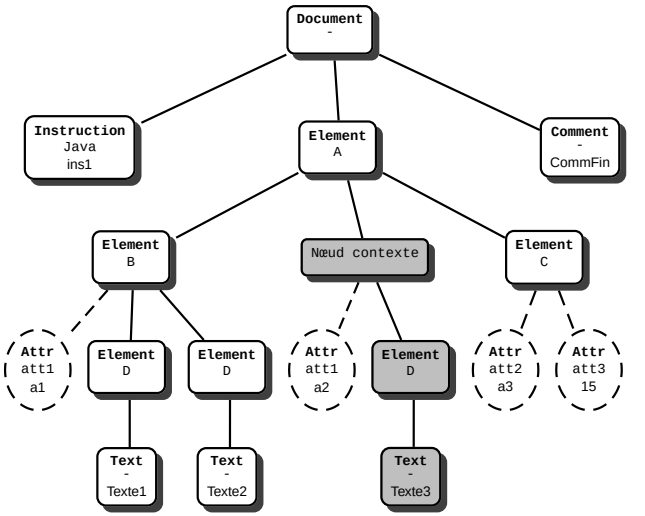
attribute::att1 ou @att1



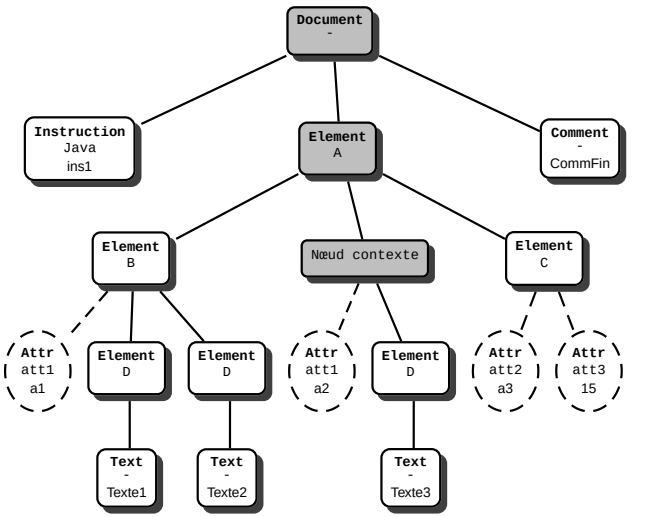
parent::A



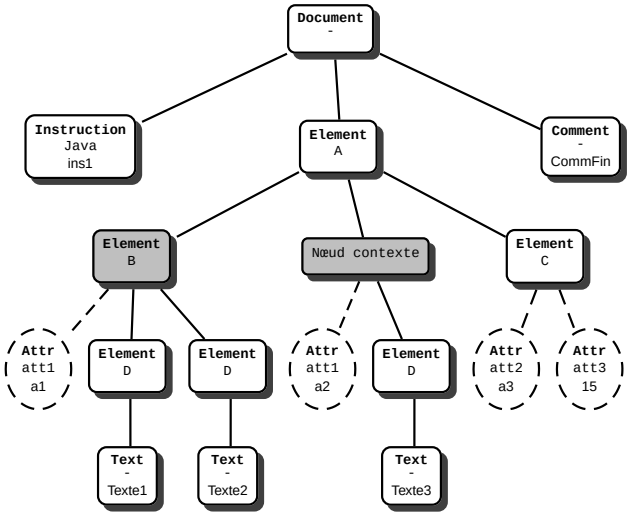
descendant::node()



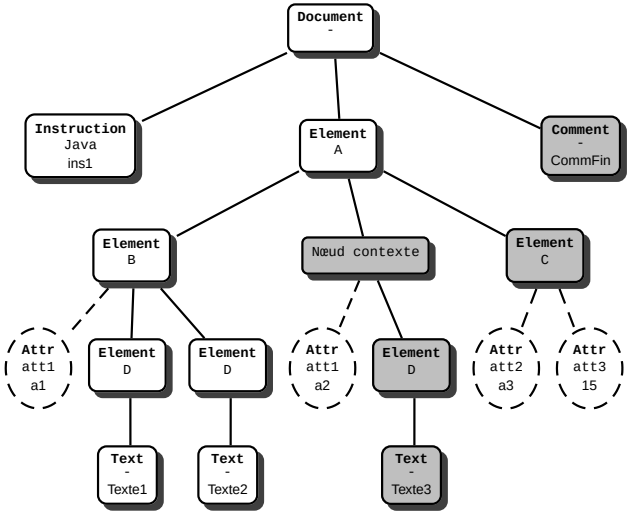
ancestor::node()



`preceding-sibling::node()`



`following::node()`



Axes: Résumé

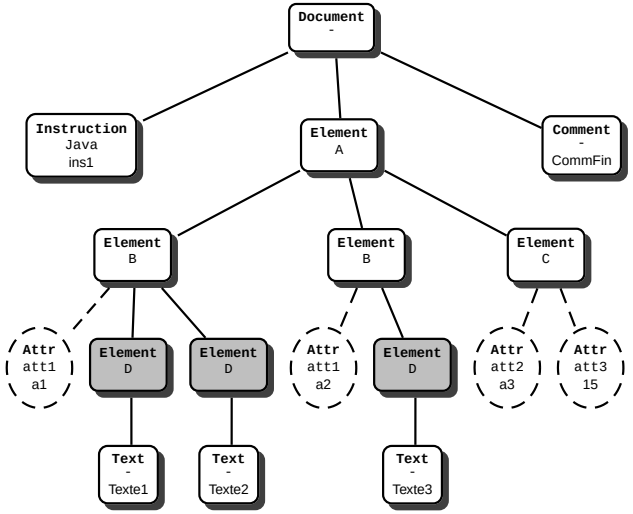
- nœuds enfants et descendants: `child, descendant, descendant-or-self`
- nœud parent et ancêtres: `parent, ancestor`
- frères: `preceding-sibling, following-sibling`
- nœuds précédents et suivants: `preceding, following`
- nœud même: `self`

Filtres

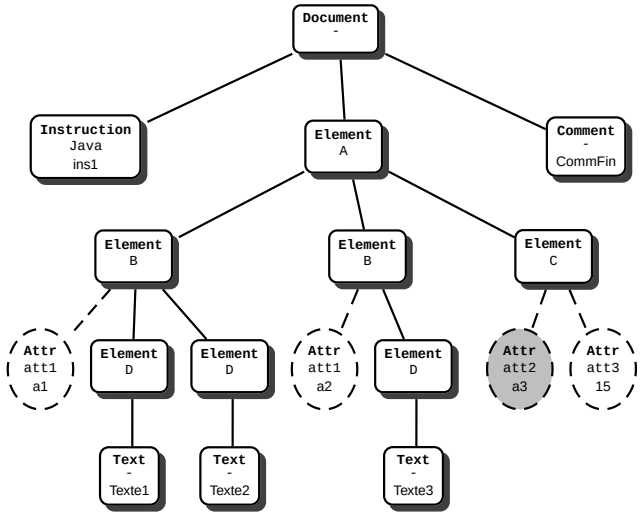
Deux manières de filtrer les nœuds :

- par leur nom :
  - possible pour les types de nœuds qui ont un nom : **Element**, **ProcessingInstruction** et **Attr**
- par leur type DOM.

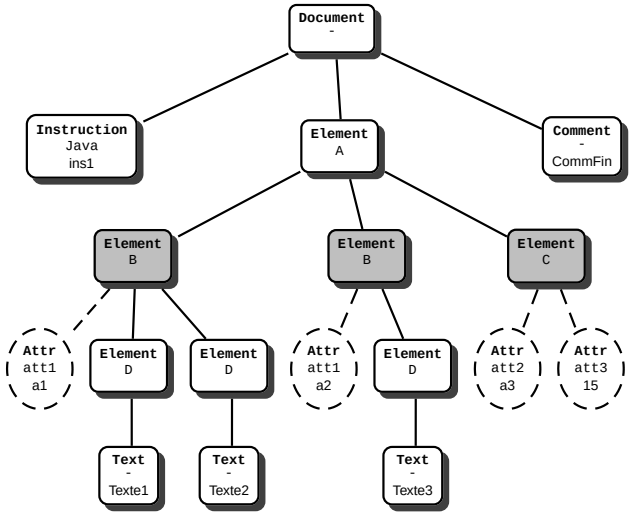
Chemin absolu : `/A/B/D`



`/descendant::node()/@att2`



Nom générique : `/A/*`



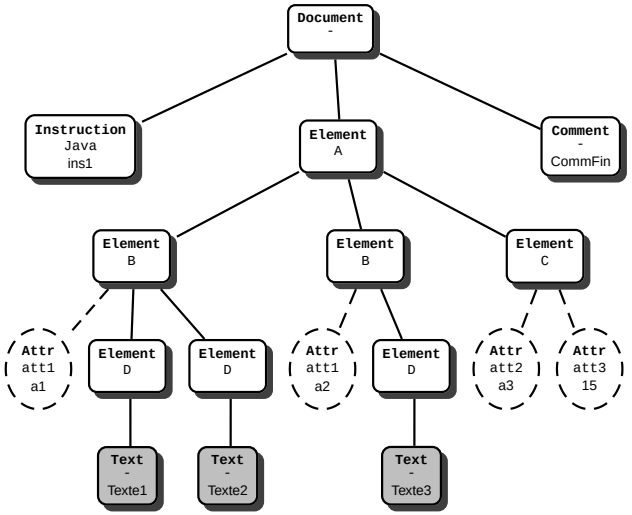
Filtrage sur le type de nœud

- `text()` : Nœuds de type **Text**
- `comment()` : Nœuds de type **Comment**

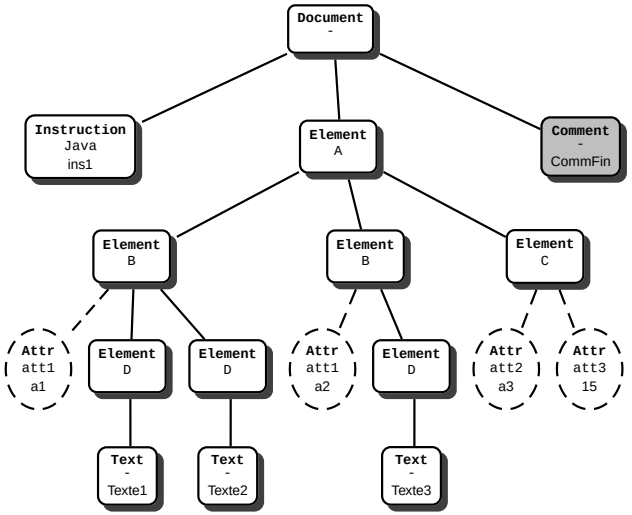
  Exemple : `/comment()`
- `processing-instruction()` : Nœuds de type **ProcessingInstruction**

  Exemple : `/processing-instruction()`, ou `/processing-instruction('java')`
- `node()` : Tous les types de nœud

/A/B//text()



/comment()



Notation abrégée de `parent::A`

- La notation abrégée "`..`" désigne le père du nœud contexte, **quel que soit son type**.
- L'expression "`..`" est équivalent à `parent::node()` (le filtre `node()` désigne tous les types de nœuds sauf les attributs).
- Pourquoi "`..`" est différent de `parent::*` (le filtre `*` désigne tous les *éléments*).

`self::node()` ou "."

- L'expression "." désigne le nœud contexte lui-même.
- Généralement elle est complétée par un filtre.
- Dans notre document exemple :
  - `self::node()` et `self::B` retournent le nœud contexte.
  - `self::A` renvoie un ensemble vide. Pourquoi?

Prédicats

- **Prédicat** : expression booléenne constituée d'un ou plusieurs tests, composés avec les connecteurs logiques habituels `and` et `or`
- **Test** :
  - toute expression XPath, dont le résultat est convertie en booléen;
  - une comparaison, un appel de fonction.

$\Rightarrow$ il faut connaître les règles de conversion

Quelques exemples

`/A/B[@att1]:` Les nœuds `/A/B` qui ont un attribut `@att1`

`/A/B[@att1='a1']:` Les nœuds `/A/B` qui ont un attribut `@att1` valant 'a1'

`/A/B/descendant::text()[position()=1]:` Le premier nœud de type **Text** descendant d'un `/A/B`.

`/A/B/descendant::text()[1]:` idem

Pour bien comprendre

Dans l'expression `/A/B[@att1]` :

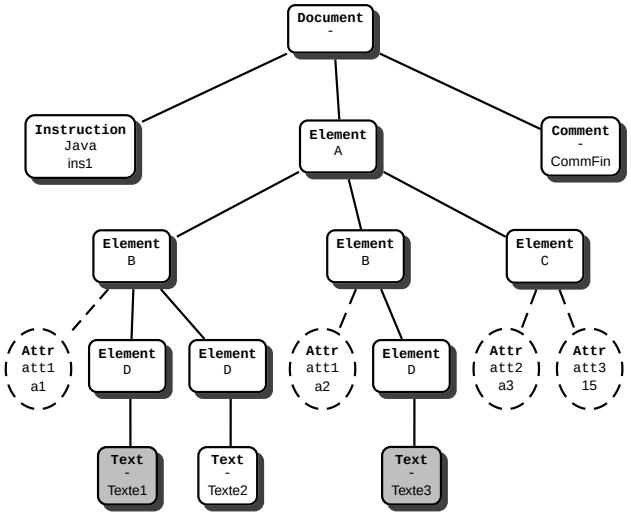
- On s'intéresse aux nœuds de type `B` fils de l'élément racine `A`.
- Parmi ces nœuds on ne prend que ceux pour lesquels le prédicat `[@att1]` s'évalue à `true`
- Cette expression s'évalue avec pour nœud contexte un élément `B`
- `[@att1]` vaut `true` ssi `@att1` renvoie un ensemble de nœuds **non vide**

Contexte d'évaluation

Une étape s'évalue en tenant compte d'un **contexte** constitué de

- un nœud contexte, position initiale du chemin ;
- ce nœud fait lui-même partie d'un ensemble obtenu par évaluation de l'étape précédente
  - on connaît la **taille** de cet ensemble (fonction *last*())
  - on connaît la **position** du nœud contexte dans cet ensemble (fonction *position*())

/A/B/descendant::text()[1]



Typage avec XPath

On peut effectuer des comparaisons, des opérations. Cela implique un **typage** et des conversions de type.

Types XPath :

- *les numériques*
- *les chaînes de caractères*
- *les booléens* (`true` et `false`)
- enfin *les ensembles de nœuds*

Numériques

Notation décimale habituelle

- Comparaisons habituelles (<, >, !=)
- Opérations : +, -, *, div, mod
- La fonction *number*() permet de tenter une conversion
- Si la conversion échoue on obtient `NaN` (*Not a Number*). **À éviter...**

Ex : `//node()[number(@att1) mod 2=1]`

Conversions

Deux conversions sont toujours possibles.

- **Vers une chaîne de caractères**.
  - utile pour la production de texte en XSLT (balise `xsl:value-of`)
- **Vers un booléen**
  - utile pour les tests effectués dans XSLT (`xsl:if`, `xsl:when`)

Conversions booléennes

- **Pour les numériques** : 0 ou `NaN` sont `false`, tout le reste est `true`
- **Pour les chaînes** : une chaîne vide est `false`, tout le reste est `true`
- **Pour les ensembles de nœuds** : un ensemble vide est `false`, tout le reste est `true`

Comparaisons : des règles de conversion étranges...

Fonctions XPath

Quelques fonctions utiles dans les prédicats :

- *concat(chaîne1, chaîne2, ...)* pour concaténer des chaînes
- *contains(chaîne1, chaîne2)* teste si *chaîne1* contient *chaîne2*
- *count (expression)* renvoie le nombre de nœuds désignés par *expression*
- *name()* renvoie le nom du nœud contexte
- *not(expression)* : négation

Prédicats: Exemples

Axes « d'avancement » :

- `child::*[3]`: le 3e enfant
- `child::*[position()=3]`: idem
- `child::*[last()]`: le dernier enfant
- `descendant::*[last()]`: le dernier descendant

La position d'un nœud dépend de l'axe choisi.

Prédicats: Axes inverses

Axes « inverses » :

- `ancestor::*[1]`: le premier ancêtre du nœud contexte (dernier dans l'ordre du document).
- `preceding-sibling::*[last()]`: le dernier frère qui précède le nœud contexte (premier dans l'ordre du document).

Prédicats: Exemples

Test du nombre d'occurrences :

- `CINEMA[count(SEANCE) > 1]`: cinémas avec au moins 2 séances
- `FILM[count(ACTEUR) = 0]`: films sans acteur
- `FILM[not(ACTEUR)]`: films sans acteur

Prédicats: Sélection par valeur

Sélection par valeur:

- `FILM[not(ACTEUR[NOM='Willis'])]`: films sans Bruce Willis
- `FILM[ACTEUR/NOM='Willis']`: films avec Bruce Willis
- `FILM[ACTEUR[NOM='Willis']]`: idem

Prédicats: Exemples

Test sur la structure : chemins imbriqués avec connecteurs logiques (qualifiers)

- `ACTEUR[NOM and DATENAISSANCE]` ou `ACTEUR[NOM][DATENAISSANCE]`: les acteurs avec un nom et une date de naissance
- `FILM[@TITRE = 'Brazil' and ACTEUR/NOM = 'De Niro']`: le film Brazil avec l'acteur De Niro
- `FILM[@TITRE = 'Brazil'][ACTEUR/NOM = 'De Niro']`: idem

Remarque sur les prédicats

Attention: Les deux premières expressions sont equivalentes, mais pas la troisième! Pourquoi?

- `FILM[ACTEUR and position()=1]`
- `FILM[position()=1][ACTEUR]`
- `FILM[ACTEUR][position()=1]`

Un cas épineux : les espaces

Un document XML avec espaces

```
<?xml version="1.0"
      encoding="ISO-8859-1"?>
<?xml-stylesheet href="ex.xsl"
                 type="text/xsl"?>
<!-- commentaire -->
<!DOCTYPE A SYSTEM "ex.dtd" >
<A>
 <B at1=' val1 '
    at2=' avec blanc '
 />
 <C> </C>
 <D> texte </D>
</A>
```


Où sont les espaces ?

Un peu partout !

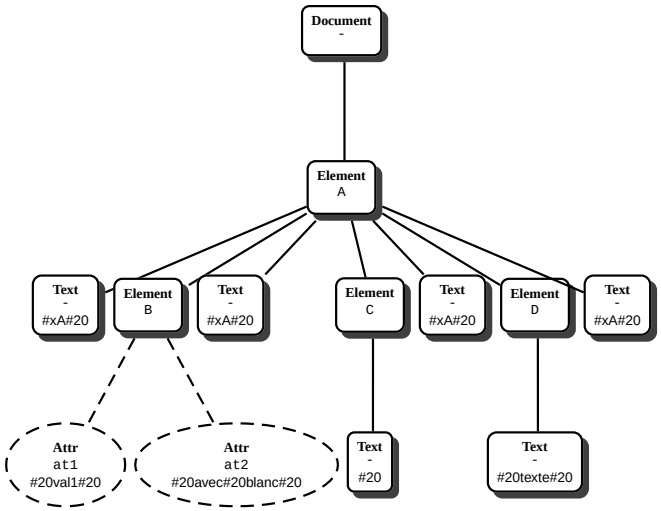
- avant la balise ouvrante de l'élément racine ;
- à l'intérieur d'une balise ;
- à l'intérieur d'un élément ;
- à l'intérieur de la valeur d'un attribut ;
- entre deux balises ;
- entre deux attributs.

Tous n'ont pas la même importance

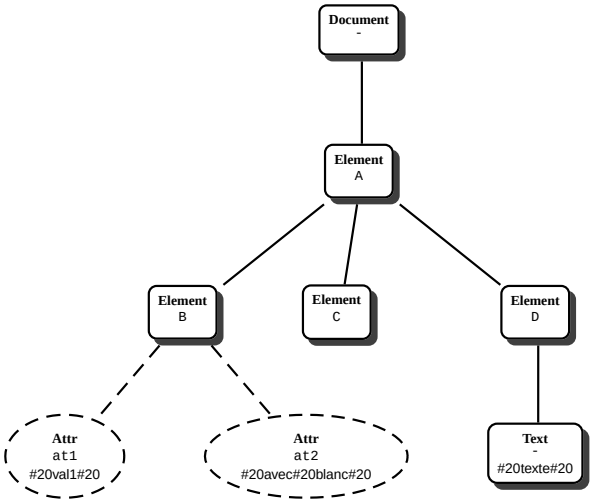
Les espaces dans XPath

- Les espaces du prologue ou de l'épilogue sont supprimés.
- Les espaces consécutifs sont réduits à un seul.
- Les fins de lignes sont représentées par le caractère #xA.
- En général, les espaces en début et fin d'attributs sont supprimées.

L'arbre XPath du document



Instruction `xsl:strip-space`



XPath : Résumé

XPath est un langage pour extraire des noeuds dans un arbre XML :

- On navigue dans l'arbre grâce à des axes de navigation.
- Un chemin de navigation est une séquence d'étapes.
- Dans chaque étape on choisi un axe, un filtre et éventuellement des prédicats.
- Le résultat d'une étape (d'une séquence d'étapes) est un séquence de noeuds.