XPath

Dan VODISLAV

CY Cergy Paris Université

Licence Informatique L3

Plan

- Principes généraux
- Syntaxe XPath
  - Expressions de chemin
  - Étapes: Axes, Filtres, Prédicats
- Types et opérateurs XPath
- Fonctions XPath prédéfinies
- XPath 2.0

XPath

- Langage de sélection d'un ensemble de nœuds dans un document XML
  - Basé sur le modèle DOM de représentation arborescente, mais *simplifié*:
    - Pas d'entités (entités remplacées par leur contenu)
    - Pas de différence entre texte et CDATA
  - Utilise des *expressions de chemin* pour désigner des nœuds dans l'arbre
- Les expressions XPath sont utilisées dans d'autres langages: XSLT, XQuery, XLink, …
- Versions
  - Version 1.0 (1999): la plus courante, présentée ici
  - Version 2.0 (2007): seules les principales différences seront présentées
  - Versions 3.0 (2014) et 3.1 (2017)

Expressions XPath

- Une expression de chemin XPath
  - S'évalue en fonction d'un ***nœud contexte***
  - Désigne *un ou plusieurs chemins* dans l'arbre du document à partir du nœud contexte
  - A pour résultat *un ensemble de nœuds* de l'arbre (qui se situent au bout des chemins)
- XPath offre aussi des *expressions littérales* qui produisent une valeur (texte, numérique, booléenne)
    - Ex: 1 + 2 est une expression XPath aussi
- Types de nœuds
  - *Document*, *Element*, *Attribute*, *Text*, *Comment*, *ProcessingInstruction*

  Remarque : les nœuds *Attribute* sont adressés d'une façon particulière

Syntaxe XPath

- Expression de chemin: suite d'**étapes**

  [/]étape$_1$/étape$_2$/…/étape$_n$

  – "/" au début indique un *chemin absolu* (partant de la racine du document)

  – Absence "/" au début: *chemin relatif* (partant du nœud contexte)

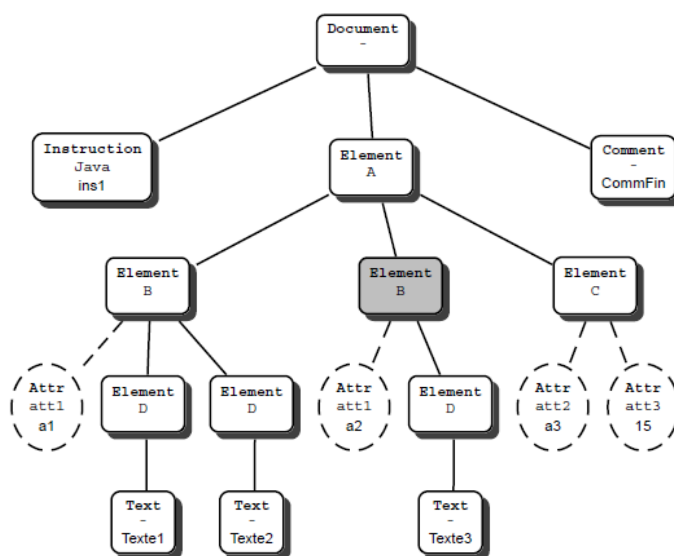
- Idée: chaque étape successive détermine un ensemble de nœuds

  – La première étape part du nœud contexte

  – Le reste du chemin utilise chacun des nœuds désignés par l'étape précédente comme nœud contexte pour la suite de l'évaluation

Exemple

```
<?xml version="1.0" ?>
<?java ... ?>
<A>
  <B att1="a1">
    <D>Texte 1</D>
    <D>Texte 2</D>
  </B>
  <B att1="a2">
    <D>Texte 3</D>
  </B>
  <C att2="a3" att3="15"/>
</A>
<!- CommFin -->
```



- L'expression /A/B/D/text()

  – Chemin absolu, première étape (A) produit un nœud

  – Seconde étape (B) produit deux nœuds à partir du nœud produit par la première

  – Troisième étape (D) est évaluée à partir de chacun des nœuds B → trois nœuds D

  – Dernière étape (text()) → résultat: l'ensemble des trois nœuds *Text*

Étapes XPath

- Étape = axe + filtre + prédicats

  `axe::filtre[prédicat`$_1$`]…[prédicat`$_n$`]`

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

- Axe
  - Optionnel (par défaut `child`)
  - Spécifie "la direction" que prend le chemin par rapport au nœud contexte
  - Définit *un ensemble des nœuds* et *l'ordre* dans lequel on les considère
- Filtre
  - Obligatoire, décrit le sous-ensembles de nœuds de l'axe retenu
- Prédicats
  - Optionnels, décrivent un filtrage supplémentaire
  - Conditions à satisfaire par les nœuds, combinées par l'opérateur "ET" logique

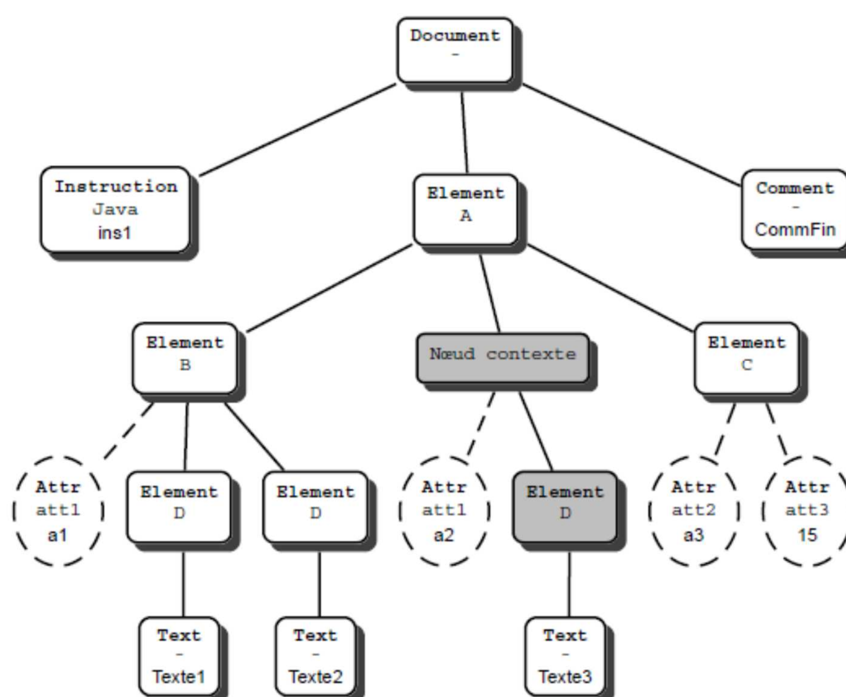
Axes XPath

- Douze axes
  - `child` (axe par défaut): enfants directs du nœud contexte
  - `parent`: nœud parent
  - `attribute`: nœuds attribut du nœud contexte
  - `descendant`: nœuds descendants du nœud contexte
  - `descendant-or-self`: descendants, y compris le nœud contexte
  - `ancestor`: nœuds ancêtres du nœud contexte
  - `ancestor-or-self`: ancêtres, y compris le nœud contexte
  - `following`: nœuds suivants dans l'ordre du document
  - `following-sibling`: frères suivants dans l'ordre du document
  - `preceding`: nœuds précédents dans l'ordre du document
  - `preceding-sibling`: frères précédents dans l'ordre du document
  - `self`: le nœud contexte lui-même
- Attributs: seul l'axe `attribute` désigne des nœuds attribut !

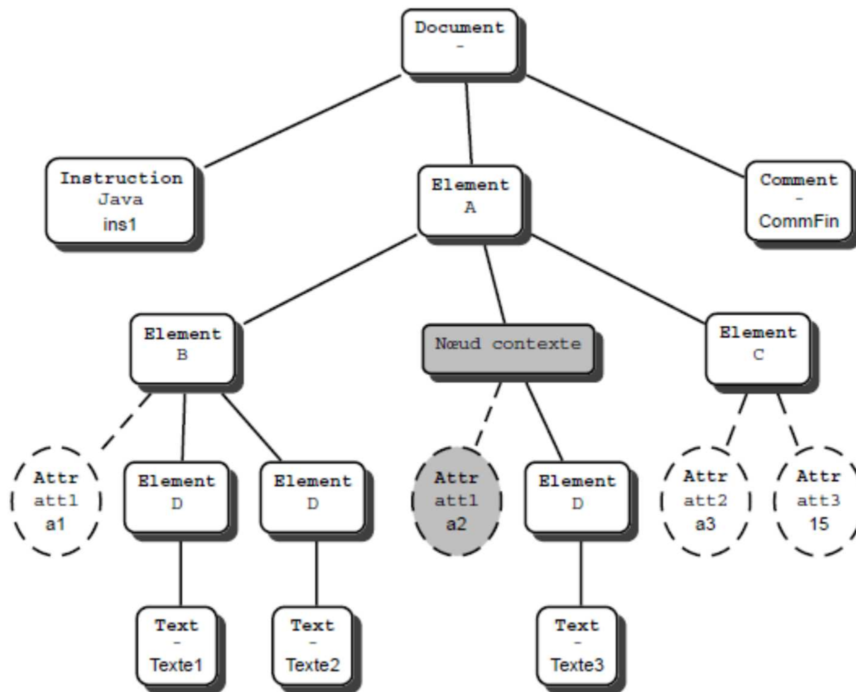
Filtres

- Deux façons de filtrer les nœuds d'un axe:
  - Par leur ***nom***
    - Pour les nœuds qui ont un nom (*Element*, *Attribute*, *ProcessingInstruction*)
    - * : n'importe quel nom
  - Par leur ***type***
    - *text()* : nœuds de type texte
    - *comment()* : nœuds de type commentaire
    - *processing-instruction()* : nœuds de type instruction de traitement
      - On peut spécifier un nom en paramètre (ex. processing-instruction('java'))
    - *node()* : tous les types de nœud
- Remarque: le filtre est appliqué à un *axe* donné
  - `child::*`, `descendant::*` ou `attribute::*` utilisent le même filtre mais désignent des nœuds différents

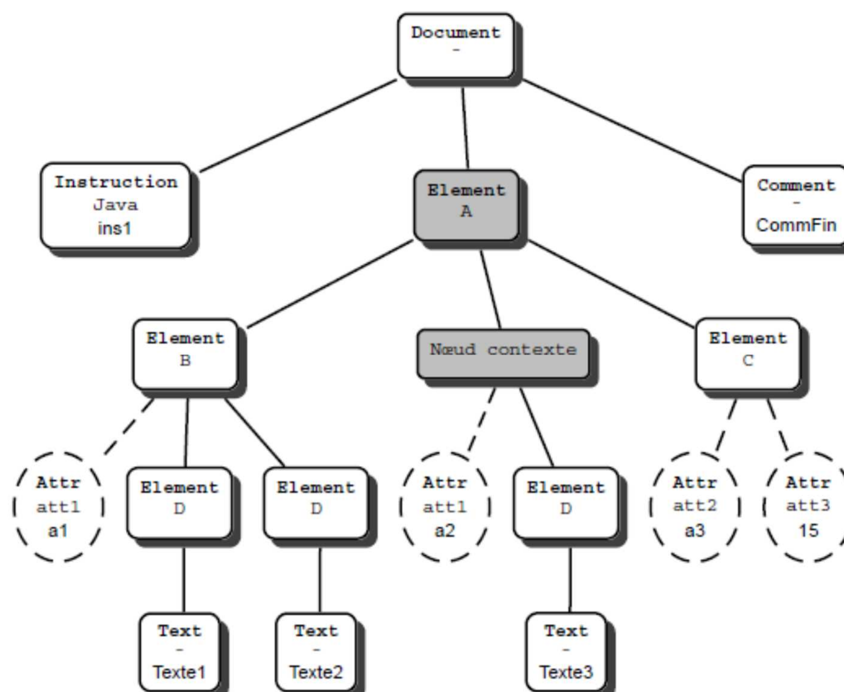
Exemple: `child::D` (ou `D`)



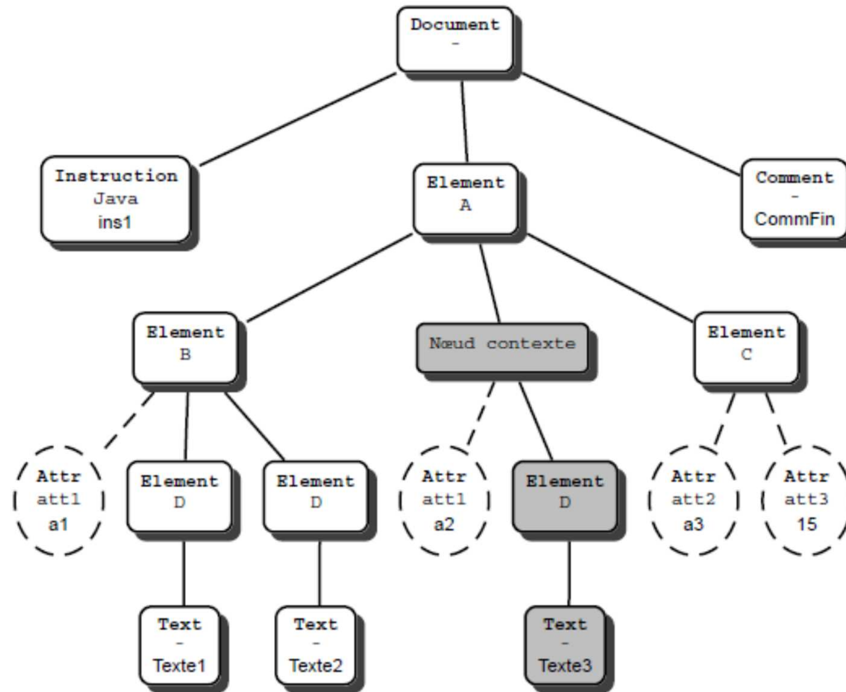
attribute::att1 (ou @att1)



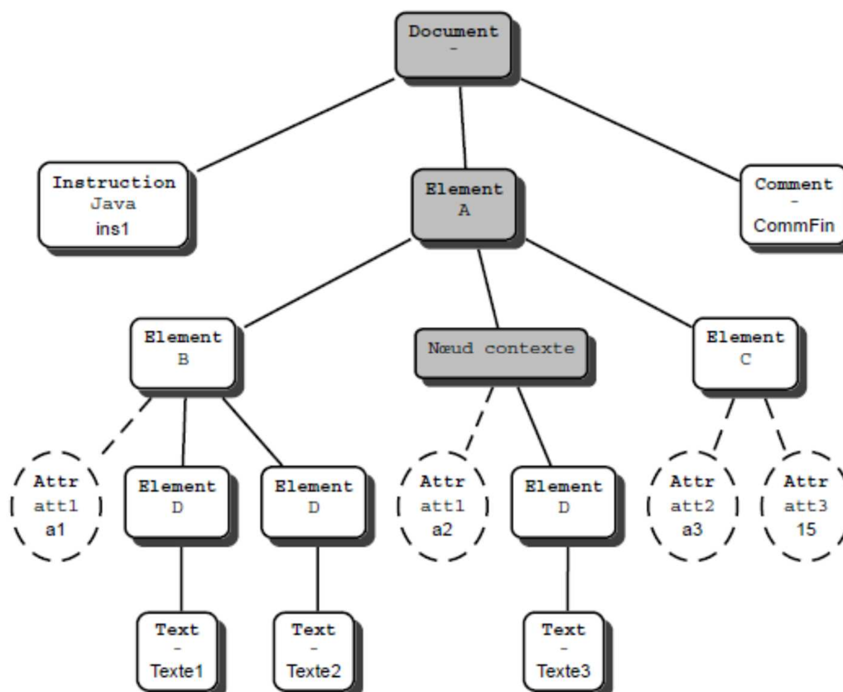
parent::node() (ou ..)



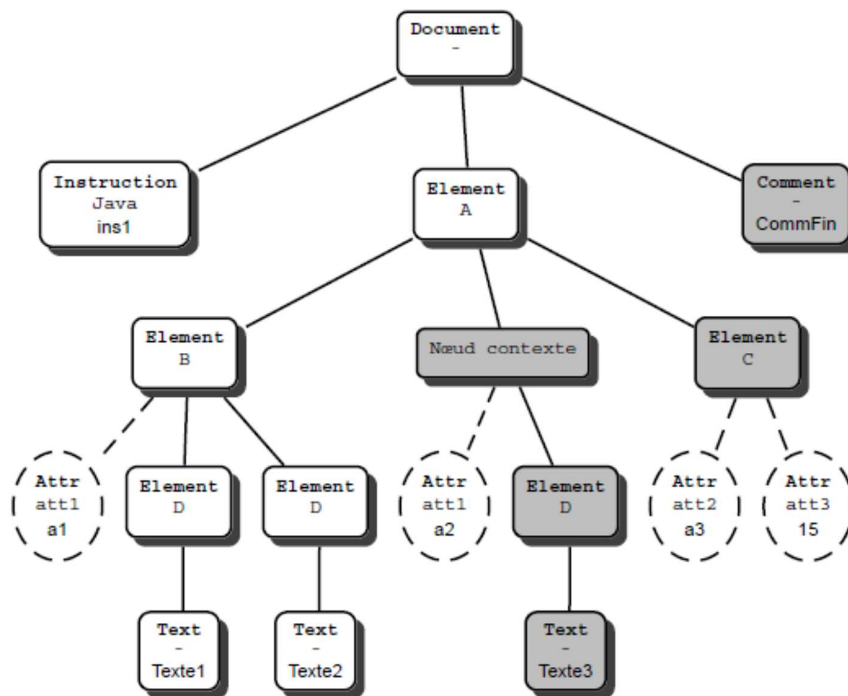
descendant::node()



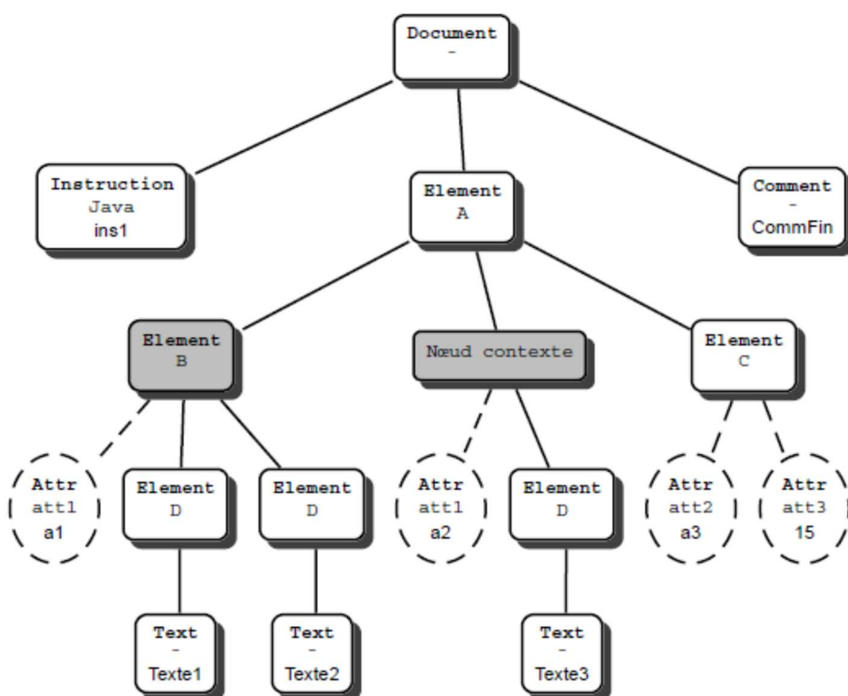
ancestor::node()



following::node()



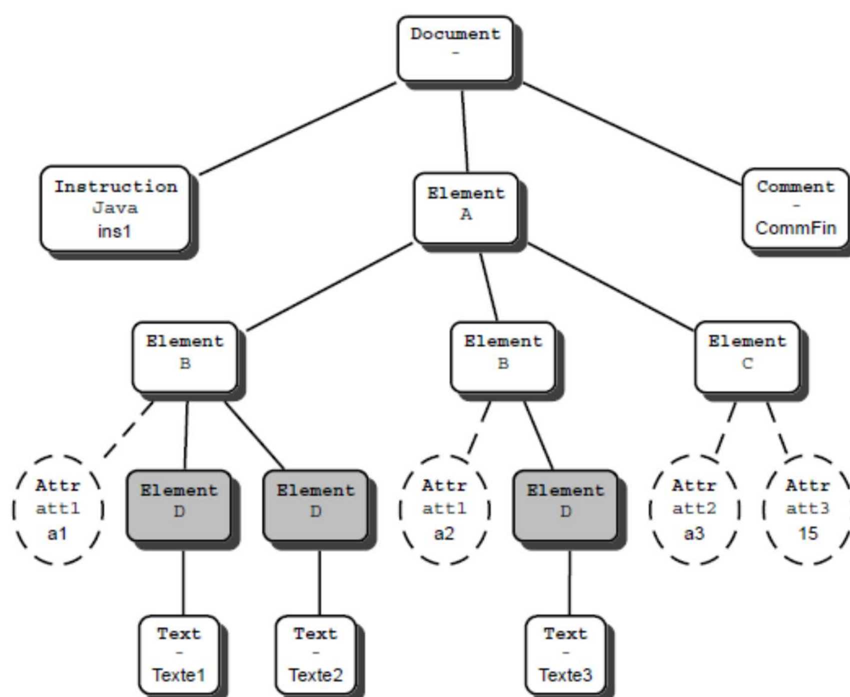
preceding-sibling::node()



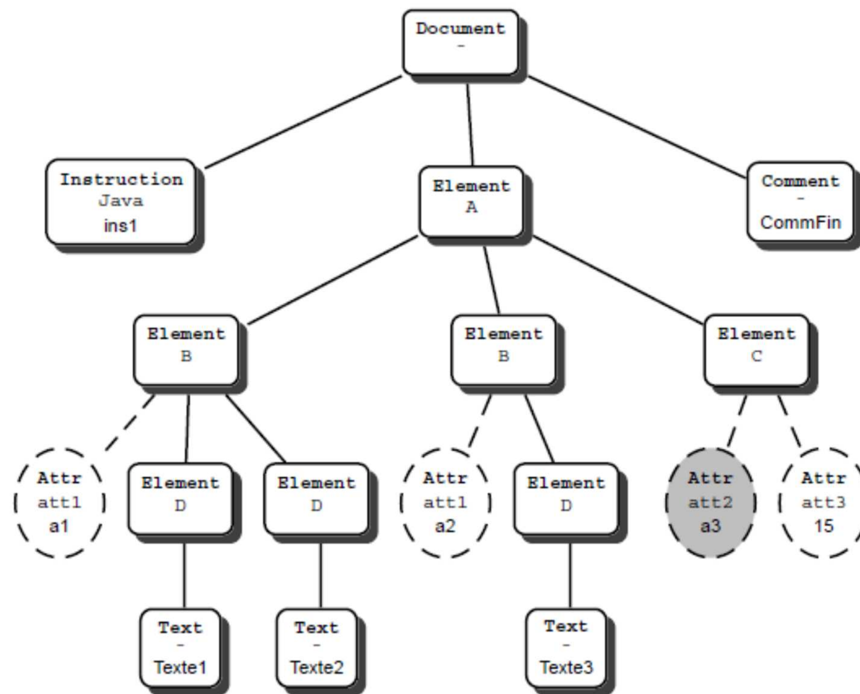
Abréviations

- Permettent de simplifier l'écriture des expressions de chemin
  - `nom` : signifie `child::nom` (élément `nom`)
    - `child` est l'axe par défaut
  - `*` : signifie `child::*` (tous les éléments fils du nœud contexte)
    - `ns:*` : tous les éléments fils du nœud contexte qui ont pour espace de noms `ns`
  - `@nom` : signifie `attribute::nom` (l'attribut `nom`)
  - `@*` : signifie `attribute::*` (tous les attributs du nœud contexte)
  - `.` : signifie `self::node()` (le nœud lui-même)
  - `..` : signifie `parent::node()` (le nœud parent)
  - `a//b` : signifie `a/descendant::b` (les descendants `b` de `a`)
    ou `a/descendant-or-self::node()/b`
    - `descendant-or-self::node()` est l'étape par défaut
  - `//b` : signifie `/descendant::b` (les éléments `b` de tout le document)
    ou `/descendant-or-self::node()/b`
  - `//@nom` : signifie `/descendant-or-self::node()/attribute::nom` (les attributs `nom` de tout le document)

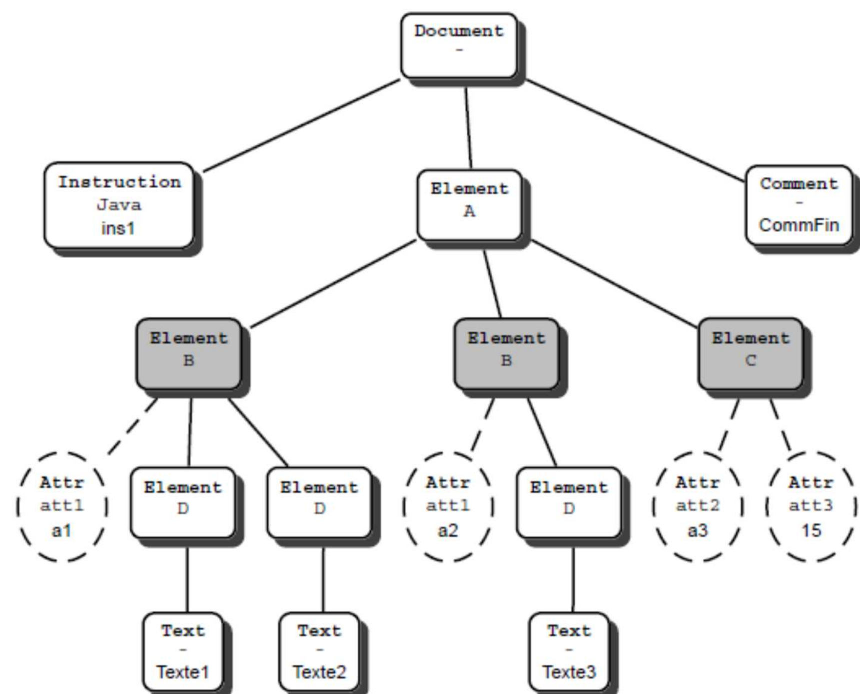
Exemple: `/A/B/D`



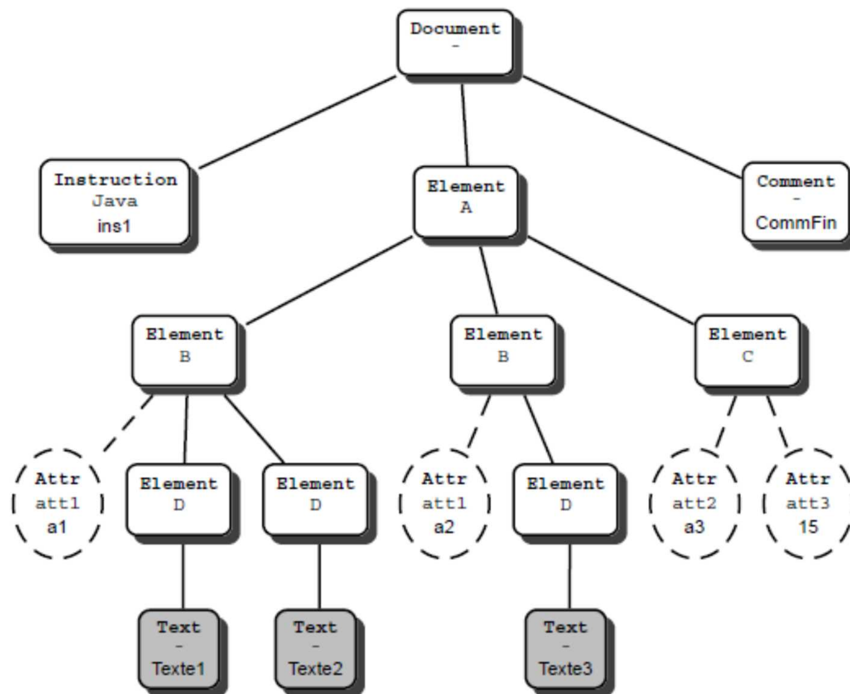
//@att2



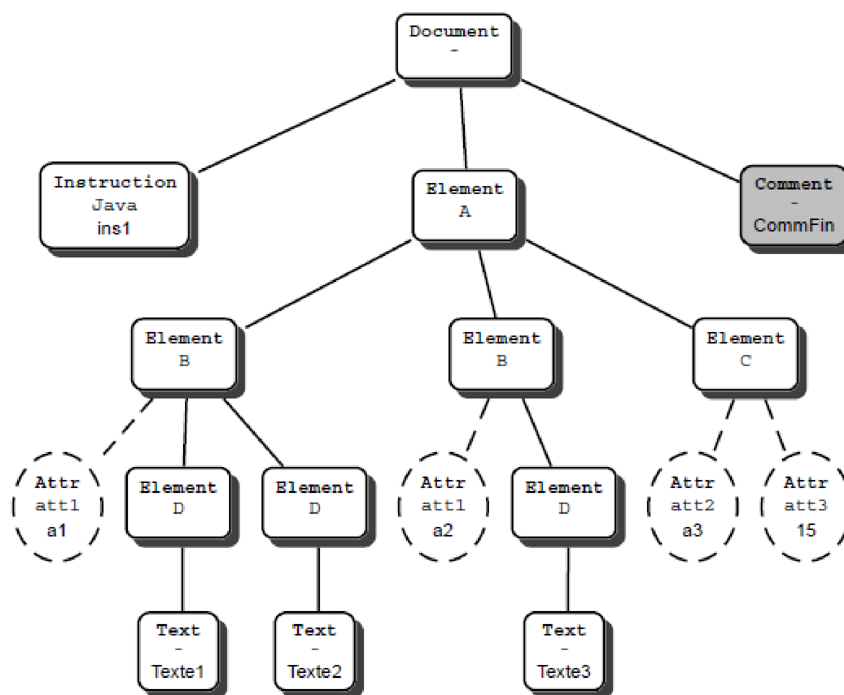
/A/*



/descendant::text()



/comment()



Prédicats

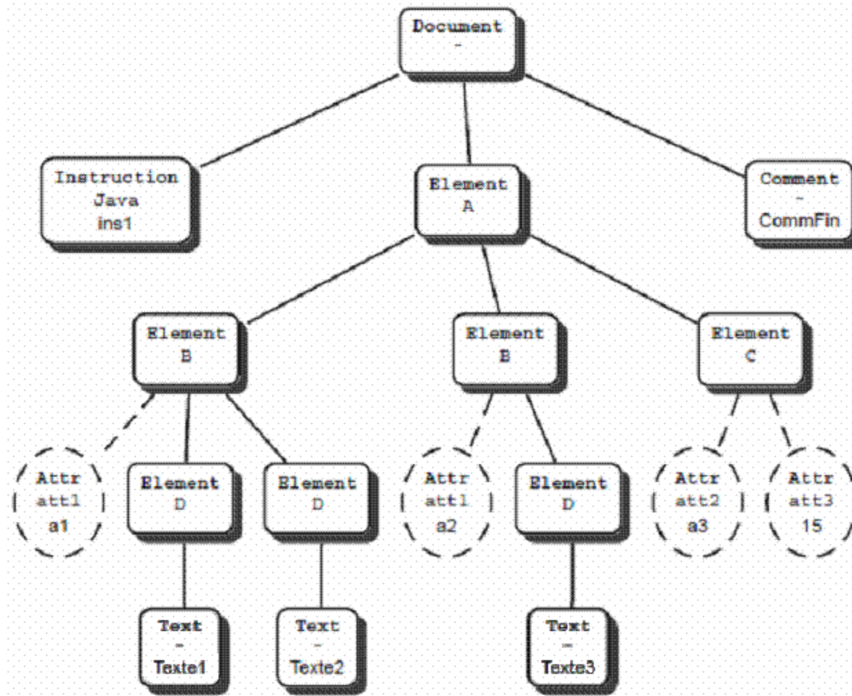
- *Prédicat*: expression booléenne constituée de *tests* connectés par les opérateurs logiques ET et OU
  - La négation: par une fonction (*not()*)
- *Test*: expression booléenne élémentaire
  - Comparaison
  - Appel de fonction booléenne
  - Expression de chemin convertie en booléen
    - Ensemble de nœuds: `false` si l'ensemble est vide, sinon `true`
    - Numérique: `false` si 0 ou NaN (« not a number »), sinon `true`
    - Chaînes de caractères: `false` si chaîne vide, sinon `true`

Évaluation d'expressions avec prédicats

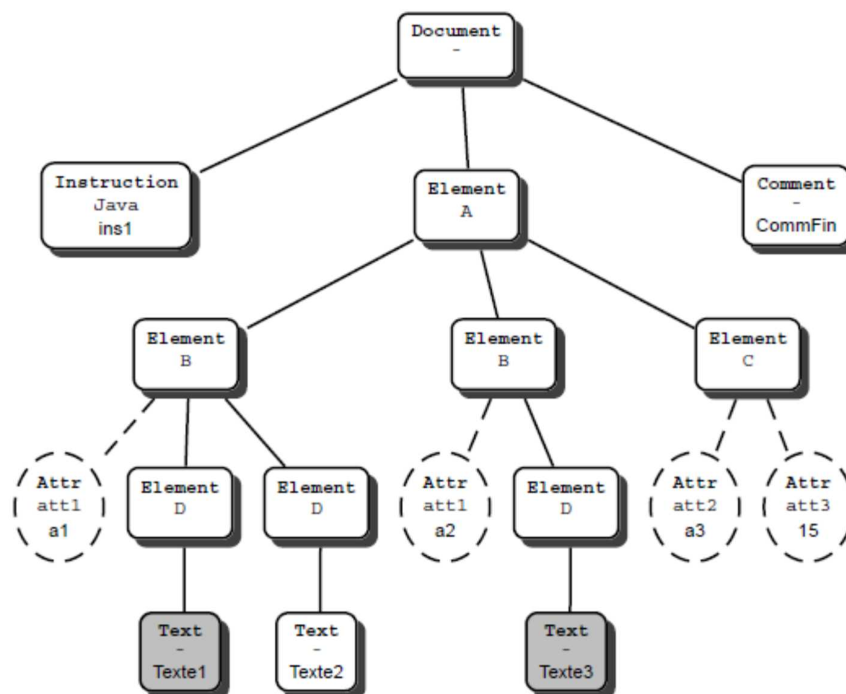
- Exemples
  - `/A/B[@att1="a1"]` : nœuds `/A/B` ayant un attribut `att1` de valeur `a1`
  - `/A/B[@att1]` : les nœuds `/A/B` qui ont un attribut `att1`
  - `/A/B/descendant::text()[position()=1]` : premier nœud texte descendant d'un `/A/B`
  - `/A/B/descendant::text()[1]` : même chose (abréviation)

  Remarque: imbrication de prédicats possible:
  `/A[B[@att1="a1"]/D]/B//text()`
- Évaluation étape avec prédicat (l'étape précédente produit l'ensemble de nœuds N)
  - Pour chaque nœud n de N, l'axe et le filtre de l'étape produisent un ensemble de nœuds N_n
    - Le prédicat est évalué pour chaque nœud n' de N_n
    - On connaît la taille de N_n (*last()*) et la position de n' dans N_n (*position()*)
    - Les expressions de chemin dans le prédicat utilisent n' comme nœud contexte
  - Seuls les nœuds n' qui satisfont tous les prédicats de l'étape entrent dans le résultat de cette étape

Que produit `/A/B/descendant::text()[1]` ?



Résultat



Axes directs et inverses

- L’ordre des nœuds d’un axe:
  - Soit l’ordre du document (axe direct)
  - Soit l’ordre inverse du document (axe inverse)
- Important pour les prédicats utilisant la position
- Axes inverses:
  - `ancestor`, `ancestor-or-self`
  - `preceding`, `preceding-sibling`
- Exemples
  - `ancestor::*[1]`: ancêtre le plus proche (parent), donc le dernier ancêtre dans l’ordre du document
  - `preceding-sibling::*[last()]`: dernier des frères avant le nœud contexte à partir de celui-ci, donc le premier frère dans l’ordre du document

Types et opérateurs XPath

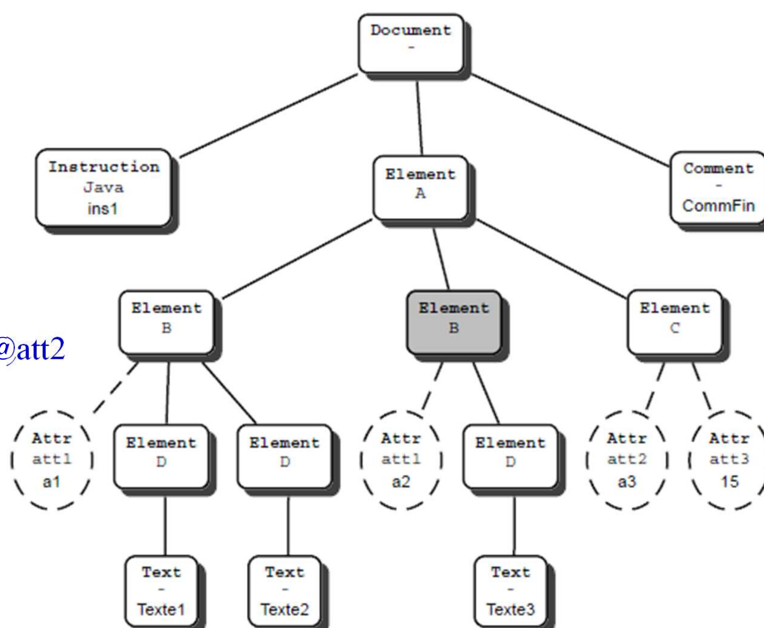
- Quatre types de données
  - Numérique
  - Chaîne de caractères
  - Booléen
  - Ensemble de nœuds (nodeset)
- Opérateurs
  - Comparaison: `<, <=, >, >=, =, !=`
  - Arithmétiques: `+, -, *, div, mod`
  - Booléens: `or, and`
  - Union d’ensembles de nœuds: `|`
- Conversions de types
  - Vers booléen (déjà présentée) + fonction *boolean()* pour conversion explicite
  - Vers chaîne de caractères: fonction *string()* pour conversion explicite
    - Nœud *texte*, *attribut*: la valeur du nœud
    - *Élément*, *document*: la concaténation des textes contenus, dans l’ordre du document
      - N’inclut pas les valeurs des attributs
  - Vers numérique: *number()*, qui peut produire `NaN` si la conversion est impossible

Fonctions XPath

- De nombreuses fonctions, ici quelques unes des plus importantes
- Pour nœuds
  - *count(expr)*: nombre de nœuds dans l'ensemble produit par l'expression
  - *name()*: nom du nœud contexte
  - *local-name(), namespace-uri()*: composantes du nom ayant un espace de noms
- Pour chaînes de caractères
  - *concat(ch1, ch2, …)*: concaténation
  - *contains(ch1, ch2)*: vérifie si *ch1* contient *ch2*
  - *substring(ch, pos, l)*: extrait la sous-chaîne de *ch* de longueur *l*, commençant à la position *pos* (les positions démarrent à 1)
  - *string-length(ch)*: longueur de la chaîne
- Pour booléens
  - *true(), false()*: les valeurs vrai/faux
  - *not(expr)*: négation de l'expression logique
- Pour numériques
  - *floor(n), ceiling(n), round(n)*: fonctions d'arrondi arrondi pour la valeur du nœud
  - *sum(expr), avg(expr)*: somme, moyenne des valeurs numériques des nœuds de l'ensemble produit par l'expression

Autres exemples

- ../*
- /A//text()[1]
- //B[@att1="a2"]/D
- //B[@att1]/D
- /A/B[last()]
- //B[count(D)>1]
- /A/C[not(D)]
- /A/B[not(@att1)]
- /A[B[D="Texte1"]/@att1]/C/@att2



XPath 2.0

- Extension de XPath 1.0
  - Compatible avec 1.0
  - Adaptée à XQuery et au typage XML Schema
  - XPath 2.0 est un sous-ensemble de XQuery 1.0
- Principales différences
  - Modèle de données enrichi
    - Type *séquence de nœuds* (ordre, doublons permis)
    - Les types XML Schema peuvent être utilisés dans les tests des prédicats
  - Plus puissant
    - Nouveaux opérateurs (*for*, *if*, *some*, *every*), utilisation de variables
    - Toute expression qui retourne une séquence de nœuds peut être utilisée comme étape → imbrication possible (expressions de chemin en tant qu'étapes)
  - Extensibilité
    - De nombreuses nouvelles fonctions prédéfinies
    - Possibilité de définir ses propres fonctions

Expressions de chemin XPath 2.0

- Nouveaux tests du type de nœud
  - *item()*: tout type de nœud ou valeur atomique
  - *element()*: tout élément (même chose que `child::*` en XPath 1.0)
  - *element(nom)*: tout élément appelé *nom*
  - *element(*, type)*: tout élément du type précisé
  - *attribute()*: tout attribut
- Étapes avec expressions de chemin
  - Une étape ne contient pas forcément un axe
  - Toute expression retournant une séquence de nœuds est acceptable

  Exemple: `/livre/(auteur|editeur)/nom`

Bibliographie spécifique

- Le site W3C:
  - `http://www.w3.org/TR/xpath`
  - `http://www.w3.org/TR/xpath20`
- J. Melton, J. Buxton, *Querying XML*, Morgan Kaufmann