Rappels sur XPath

EThAP 2025

Matthias Gille Levenson

Université de Versailles Saint-Quentin – DYPAC EA 2449 & CIHAM UMR 5648

prénom [point] gille [point] levenson [at] ens-lyon.fr

30 septembre 2025

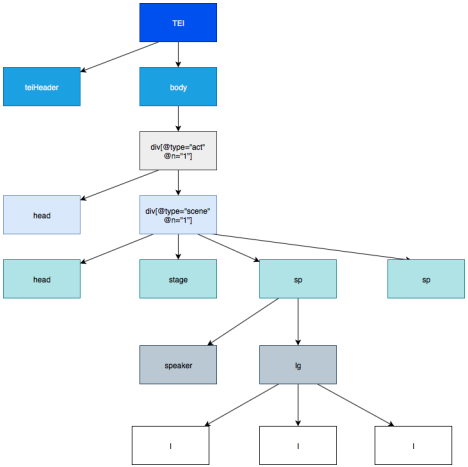
Le document XML comme un arbre

- Le document XML est un arbre qui contient un noeud racine et un certain nombre d'embranchements.
- On passe d'un niveau à l'autre à l'autre de la barre oblique / comme pour n'importe quel système arborescent comme le système de fichiers :
- `Bureau/Travail/Cours_et_formations/Cours_ethap_xpath_xslt_2025/XPath/diapo` représente un système fonctionnant de la même manière que `TEI/body/div/div/head`.

Le document XML comme un arbre

- `/` fait passer d'un noeud à un noeud enfant
- `..` fait remonter au noeud parent
- `//` permet d'accéder directement au noeud visé sans préciser le chemin entier

Application à Andromaque (Exercice A. Pinche)



Exercice 1

- Quels éléments sont sélectionnés par les chemins suivants ?
  - `TEI/body/div/div` ;
  - `TEI/body/div/div/head` ;
  - `TEI/body/div/div/sp`.

Espaces de noms et préfixes

- Les espaces de nommage ou espaces de noms sont un concept propre au XML. Le XML a une double caractéristique quant au contrôle d'un document. Un document XML est en effet **bien formé** quand il respecte les règles fondamentales du XML (pas de chevauchement des éléments, attributs séparés par des espaces, etc), et il est **valide** quand il respecte un schéma donné.
- Le XML étant un format industriel, il est fréquent que les acteurs qui l'utilisent soient de grosses institutions et consortiums qui produisent des **spécifications standards** : ALTO, PAGE, SVG, DublinCore, et bien sûr la TEI ou la MEI.
- L'espace de nommage permet de préciser la spécification du XML que l'on a choisie. Un espace de nommage est représenté par URI (Uniform Ressource Identifier) et par un préfixe (qui sera présenté plus bas).

Espaces de noms et préfixes

- Dans les exemples de requêtes, je présenterai de façon arbitraire des exemples de **requêtes préfixées et non préfixées** :
- `TEI/body/div/div` est équivalent à `tei:TEI/tei:body/tei:div/tei:div`
- La seconde requête précise l'espace de nommage des éléments visés, mais elle est plus verbeuse. En fonction des configuration de votre logiciel et de votre feuille de transformation, il sera nécessaire (ou pas) d'ajouter ces préfixes.
- Si quelque chose ne marche pas, il y a de bonnes chances que ça ait un lien avec les espaces de nommage !

Les axes XPath

- On peut vouloir naviguer dans l'arbre sans connaître exactement sa structure. Pour ce faire, on aura besoin de naviguer selon les grandes relations entre noeud
- Un noeud a en effet 0 ou 1 parent, et il peut avoir 0 ou plus enfants. Il peut avoir des ancêtres et des descendants, ainsi que des *adelphes* (*siblings*).
- Les **axes** (*axis*) XPath permettent de représenter ces relations et aident à la navigation dans l'arbre
- On utilise un axe de la façon suivante « `axe::noeud` ». Ici, le double deux-points est fondamental, il permet de distinguer l'axe du préfixe de l'espace de nommage.

Les axes XPath : navigation sur parent proche

- `self` : le noeud courant
- `child` : l'enfant
- `parent` : le parent
- `preceding-sibling` : l'adelphe précédent (premier enfant précédent du parent)
- `following-sibling` : l'adelphe suivant (premier enfant suivant du parent)

Les axes XPath : navigation sur parent éloigné

- `ancestor` : n'importe quel noeud ancêtre
- `descendant` : n'importe quel noeud descendant
- `preceding` : n'importe quel noeud précédant le noeud actuel
- `following` : n'importe quel noeud suivant le noeud actuel

Expressions conditionnelles ou prédicats

- On peut vouloir cibler des noeuds précis et avoir besoin d'exprimer des conditions pour ce faire. Le **prédicats** permettent d'exprimer ces conditions. Un prédicat se matérialise sous la forme d'une expression entre crochets droits `[condition]`
- Cette condition peut être de n'importe quelle forme : il suffit qu'elle soit vraie pour que le noeud soit retourné.
- On peut ainsi tester l'existence ou l'absence d'un attribut, la valeur d'un attribut, une égalité ou inégalité, comparer des valeurs...
- Ainsi, dans la transcription diplomatique d'un manuscrit, `//tei:text/descendant::tei:lb[@break='no']` permettra de sélectionner tous les débuts de ligne qui coupent un mot, soit toutes les coupures de mot à la ligne.

Exercice 2

- À partir de l'arbre XML simplifié d'Andromaque donner les chemins suivants en partant de l'élément racine TEI
  - Donner le chemin vers `body` ;
  - Donner le chemin vers la `div` dont l'attribut `@type` est égal à `act` ;
  - Donner le chemin vers `stage` ;
  - Donner le chemin vers `speaker` ;
  - Donner le chemin vers les `l`.

Exercice 2

- À partir de l'arbre XML simplifié d'Andromaque donner les chemins suivants en partant de l'élément racine TEI
  - Donner le chemin vers `body` ;
  - Donner le chemin vers la `div` dont l'attribut `@type` est égal à `act` ;
  - Donner le chemin vers `stage` ;
  - Donner le chemin vers `speaker` ;
  - Donner le chemin vers les `l`.
- Vous pouvez tester vos expressions sur le fichier `Andromaque_c.xml`

Enchaînement et juxtaposition des prédicats

- On peut enchâsser les prédicats :
  `preceding-sibling::node()[self::tei:app[descendant::tei:rdg[contains(@wit,`
- On peut juxtaposer les prédicats, ce qui correspond au booléen AND :
  `preceding-sibling::node()[self::tei:w][position() = 1]`
  est équivalent à
  `preceding-sibling::node()[self::tei:w and position() = 1]`

Fonctions basiques : type des noeuds visés

- Noeuds XML : `node()`
- Noeuds textuels : `text()`
- Commentaires : `comment()`

Fonctions basiques : `translate`

Un certain nombre de fonctions pré-établies sont particulièrement utiles ici. **Ce sont des fonctions XPath** : vous verrez avec Ariane d'autres fonctions propres à XSLT, et avec Jean-Paul des fonctions propres à XQuery.

- `translate(text, str, repl)` produit un *mapping* pour remplacer **un à un** les éléments de `str` par les éléments de `repl` dans le texte donné.
- Exemple :
  `translate('Longtemps, je me suis couché de bonne heure', 'aeiou', 'uoiea')`
  produira « Lengtomps, jo mo sais ceaché do benno hoaro. »

Fonctions basiques : `replace`

- `replace(text, str, repl)` remplace par `repl` toutes les chaînes `str` trouvées dans `text`.
- Exemple :
  `translate('Longtemps, je me suis couché de bonne heure', 'couché', 'endorm`
  produira « Longtemps, je me suis endormi de bonne heure. »

Fonctions basiques : `contains()`

- La fonction `contains(string_a, string_b)` permet de vérifier si la chaîne de caractère a contient la chaîne b.

Exercice 3

- Trouver toutes les scènes où parlent Andromaque et Pyrrhus.

Exercice 3

- Trouver toutes les scènes où parlent Andromaque et Pyrrhus.
- **Solution :**

```
//tei:div[@type='scene'][descendant::tei:sp[contains(@who, 'andromaque')] and descendant::tei:sp[contains(@who, 'pyrrhus')]]
```

Fonctions basiques : modifier la casse (XPath 2.0)

- `lower-case(text)`
- `upper-case(text)`

Fonctions basiques : identifier des sous-chaînes de caractères

- `substring-before(text, string)`
- `substring-after(text, string)`

Exercice

- Récupérez tous les identifiants ARK du fichier qui contient *Andromaque*

Exercice

- Récupérez tous les identifiants ARK du fichier qui contient *Andromaque*
- **Solution :**

```
//@facs[contains(., 'ark')]/substring-after(substring-before(., '/f'), 'ark:')
```

Exercice

- Récupérez tous les identifiants ARK du fichier qui contient *Andromaque*
- **Solution :**

```
//@facs[contains(., 'ark')]/substring-after(substring-before(., '/f'), 'ark:')
```

- **Ne récupérer que les valeurs différentes :**

```
distinct-values(//@facs[contains(., 'ark')]/substring-after(substring-before(., '/f'), 'ark:'))
```

Fonctions basiques : manipuler des chaînes de caractères multiples

- `concat(text1, text2, text3, ..., textN)` : concatène l'ensemble des chaînes de caractères indiquées par l'utilisateur.ice. Les arguments ne peuvent être que des éléments textuels uniques et non pas des séquences
- `string-join(chemin)` : concatène l'ensemble du texte trouvé dans le chemin. (XPath 2.0)

Fonctions basiques : `compter`

- `count(path)`

Fonctions basiques : la position

- Tous les noeuds ont une position, **qui commence à 1**. Elle peut être décidée à l'aide d'un prédicat et de la fonction `position()` :
  `//library/bookshelfA/book[position() = 3]` ou plus simplement
  `//library/bookshelfA/book[3]`
- Attention : la position est **relative à l'expression XPath qu'elle suit** : l'expression ci-dessus ne récupèrera pas le troisième livre de la bibliothèque, mais le troisième livre de l'étagère.
  - Pour récupérer le troisième livre de la bibliothèque, il faut produire l'expression `//book[3]`
- On peut récupérer le dernier élément d'une liste avec la fonction `last()`

Fonctions intermédiaires : `doc()`

- `doc()` permet de charger **un document complet**
- On la combine à une expression XPath « traditionnelle » pour traiter l'intérieur de l'arbre XML
- exemple : `doc('XPath/Andromaque_c.xml')/tei:TEI` récupère la racine du fichier d'exercice
- Utile si l'on veut comparer deux documents par exemple
- Voir ici `http://web.uvic.ca/~mholmes/dhoxss2013/handouts/doc_function.pdf`

Fonctions intermédiaires : `collection()`

- `collection()` permet de charger une **collection** de documents
- On la combine aussi à une expression XPath « traditionnelle » pour traiter l'intérieur de l'arbre XML
- exemple : `collection('XPath?*.xml')/tei:TEI` récupère la racine de l'ensemble des documents XML dans le répertoire `XPath` du dépôt.
- Attention, la syntaxe ci-dessus est liée au moteur de transformation Saxon et ne fonctionne pas de façon universelle
- Utile si l'on veut itérer sur un ensemble de fichiers.
- Voir ici `http://web.uvic.ca/~mholmes/dhoxss2013/handouts/collection_function.pdf`

Exercice

- Essayez d'appliquer les deux fonctions précédents sur Oxygen (il y a un piège)

Exercice

- Essayez d'appliquer les deux fonctions précédents sur Oxygen (il y a un piège)
- Il faut changer le chemin : les expressions XPATH sont (logiquement) évaluées sur le fichier courant dans Oxygen

Exercice

- Essayez d'appliquer les deux fonctions précédents sur Oxygen (il y a un piège)
- Il faut changer le chemin : les expressions XPATH sont (logiquement) évaluées sur le fichier courant dans Oxygen
- On va donc devoir adapter le chemin en partant du fichier ouvert.

Exercice

- Essayez d'appliquer les deux fonctions précédents sur Oxygen (il y a un piège)
- Il faut changer le chemin : les expressions XPATH sont (logiquement) évaluées sur le fichier courant dans Oxygen
- On va donc devoir adapter le chemin en partant du fichier ouvert.
- `doc('Andromaque_c.xml')/tei:TEI`

Exercice

- Essayez d'appliquer les deux fonctions précédents sur Oxygen (il y a un piège)
- Il faut changer le chemin : les expressions XPATH sont (logiquement) évaluées sur le fichier courant dans Oxygen
- On va donc devoir adapter le chemin en partant du fichier ouvert.
- `doc('Andromaque_c.xml')/tei:TEI`
- `collection('../XPath?*.xml')/tei:TEI` ou `collection('?*.xml')/tei:TEI`

Exercice 4

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
2. Trouvez tous les vers dont le schéma prosodique correspond au schéma canonique du sonnet.
3. Comptez le nombre de césures dans le sonnet
4. Identifiez le vers qui comprend la deuxième césure
5. Identifiez le texte du vers qui vient après la première césure

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
   - **Solution :** `//tei:l[tei:seg[@real]]`

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
   - **Solution :** `//tei:l[tei:seg[@real]]`
2. Trouvez tous les vers dont le schéma prosodique correspond au schéma canonique du sonnet.
   - **Solution :** `//tei:l[not(tei:seg[@real])]`

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
   - **Solution :** `//tei:l[tei:seg[@real]]`
2. Trouvez tous les vers dont le schéma prosodique correspond au schéma canonique du sonnet.
   - **Solution :** `//tei:l[not(tei:seg[@real])]`
3. Comptez le nombre de césures dans le sonnet
   - **Solution :** `count(//tei:caesura)`

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
   - **Solution :** `//tei:l[tei:seg[@real]]`
2. Trouvez tous les vers dont le schéma prosodique correspond au schéma canonique du sonnet.
   - **Solution :** `//tei:l[not(tei:seg[@real])]`
3. Comptez le nombre de césures dans le sonnet
   - **Solution :** `count(//tei:caesura)`
4. Identifiez le vers qui comprend la deuxième césure
   - **Solution :** `//tei:caesura[count(preceding::tei:caesura) = 1]/ancestor::tei:l`

Exercice 4 – Corrigé

Nous allons travailler avec le sonnet 17 de Shakespeare, présent dans le répertoire `XPath/` du dépôt. On travaille sur Oxygen ou n'importe quel programme permettant d'évaluer des requêtes XPath sur un corpus.

1. Trouvez tous les vers qui présentent une divergence par rapport au schéma prosodique canonique du sonnet
   - **Solution :** `//tei:l[tei:seg[@real]]`
2. Trouvez tous les vers dont le schéma prosodique correspond au schéma canonique du sonnet.
   - **Solution :** `//tei:l[not(tei:seg[@real])]`
3. Comptez le nombre de césures dans le sonnet
   - **Solution :** `count(//tei:caesura)`
4. Identifiez le vers qui comprend la deuxième césure
   - **Solution :** `//tei:caesura[count(preceding::tei:caesura) = 1]/ancestor::tei:l`
5. Identifiez le texte du vers qui vient après la première césure
   - **Solution :**
     `//tei:caesura[count(preceding::tei:caesura) = 0]/following-sibling::text()`