# Licence 3 d'Informatique — XML — TP 1

Début : mardi 13 février 2024

## 0 Objectifs

Cette première séance de travaux pratiques vise principalement à vous familiariser avec l'outil xmllint, qui va nous servir pour *parser* un document XML [1], le valider par rapport à un fichier DTD [2] et tester des chemins écrits dans le langage XPath. Puis, après un détour par une introduction aux espaces de noms, nous abordons l'utilisation des attributs de types `ID` et `IDREF(S)`. Tous les fichiers mentionnés ci-après ont été réunis en un fichier d'archive .tar[3] for-lc-1.tar, qui comporte les quatorze fichiers suivants :

| | | |
|---|---|---|
| automne-0.xml | films-plus.xml | sf-0.dtd |
| automne-1.xml | lc-1.pdf [4] | sf-1.dtd |
| books-0.xml | namespace-example-3.dtd | teacher-3.xml |
| books-1.xml | poemefr0.dtd | traverse-plus.xsl |
| cine-club-plus.dtd | poemefr1.dtd | |

quelques-uns ayant déjà été abordés dans le cadre des cours ou des travaux dirigés. Ce fichier for-lc-1.tar se trouve sous moodle, à l'endroit suivant :

Programmation fonctionnelle scripts --- XML > XML - semestre 6 > Travaux pratiques 2024

Enfin, n'hésitez pas à consulter l'annexe L de votre cours polycopié, en particulier à propos des principales commandes de l'outil `xmllint`. Ce programme peut être employé sous le système d'exploitation Windows, mais nous l'utiliserons quant à nous sur le réseau Linux.

## 1 *Getting Started*

Tel qu'il vous a été fourni, le document books-0.xml contient plusieurs erreurs qui le rendent *mal formé*. Tapez la commande `xmllint --shell books-0.xml`, ce qui va vous démarrer une session interactive et afficher les erreurs présentes. Ensuite — lorsqu'elles seront corrigées —, essayer de *valider* ce document par rapport au fichier DTD sf-0.dtd (là aussi, quelques corrections pourraient s'avérer nécessaires dans ce fichier books-0.xml). Puis toujours avec `xmllint`, entraînez-vous à effectuer les actions suivantes :

— démarrer une session interactive mais lançant la validation du document XML dès qu'il est *parsé* ;

1. *e**X**tensible **M**arkup **L**anguage.*
2. ***D**ocument **T**ype **D**efinition.*
3. ***T**ape **AR**chive.*
4. Il s'agit du présent fichier, en format PDF (***P**ortable **D**ocument **F**ormat*).

— afficher les caractéristiques du nœud racine, puis la structure à partir de ce nœud racine ;

— bien voir la différence entre les commandes `cat` et `xpath` de `xmllint` ;

— vous promener dans l'arbre d'un document XML, afficher les informations des nœuds rencontrés ;

— vérifier que la commande `cd` de `xmllint` ne s'applique qu'à des séquences ne comportant qu'un seul nœud ;

— vous promener dans un tel arbre XML au moyen d'une succession de commandes `cd` du programme `xmllint` appliquée à des chemins *relatifs* de XPath, puis essayer la commande `pwd` de `xmllint`.

Donnez le sens des commandes suivantes :

— `cat //sf-0/book`

— `cat /sf-0/node()[2]`

— `cat /sf-0/*[2]`

— `cd //sf-0/book[1]`

## 2 Découverte d'XPath

Les fichiers utilisés pour ce § 2 ont été déjà vus en cours et sont :

— deux fichiers DTD pour des poèmes, la première — poemefr0.dtd — n'utilise que des éléments dans une organisation *ramifiée*, la seconde — poemefr1.dtd — utilise des attributs tant dans son préambule que dans le marquage des strophes et des renfoncements en début des lignes, l'organisation des lignes successives étant plus *linéaire* ;

— deux exemples de documents XML — automne-0.xml et automne-1.xml — donnés par notre poème favori, et correspondant à chacune des deux organisations. En ce qui concerne ces deux spécifications du même poème de Paul Verlaine, ne *jamais* perdre de vue — sous peine de méchamment vous embrouiller — que tout ce qui se rapporte à la version ramifiée porte le **n° 0**, tandis que ce qui se rapporte à la version plus linéaire porte le **n° 1**. Nous suivrons scrupuleusement cette convention tout au long des séances de l'unité qui utiliseront cet exemple.

En ce qui concerne les expressions XPath, vous pouvez essayer quelques chemins d'apparence assez complexe au premier abord, par exemple :

— sur le fichier automne-0.xml, d'abord : `cd poemefr0/corps/strophe[1]/ligne[1]` ;

  ⋆ puis `cat following-sibling::node()` ;

  ⋆ puis `cat following-sibling::ligne` ;

  ⋆ puis `cat following-sibling::ligne[1]` — rappelons que « `[1]` » est mis pour la contrainte complète « `[position() = 1]` » — ;

— en passant au fichier automne-1.xml, taper d'abord : `cd poemefr1/corps/ligne[1]` ;

⋆ puis `cat following-sibling::ligne[1][@strophe = "oui"]` ;

⋆ puis `cat following-sibling::ligne[@strophe = "oui"]` ;

⋆ puis `cat following-sibling::ligne[@strophe = "oui"][1]`.

Trouvez maintenant, les expressions XPath qui permettent de capturer les parties suivantes des documents automne-0.xml et automne-1.xml :

— la deuxième ligne de la deuxième strophe,

— la deuxième ligne de chaque strophe,

— la dixième ligne depuis le début du poème, sans recommencer la numérotation à chaque strophe,

— toutes les lignes précédant une ligne avec renfoncement (sans tenir compte de la subdivision par strophes) ;

— toutes les lignes sauf la deuxième ligne de chaque strophe.

## 3 Découverte des attributs de type `ID` et `IDREF(S)`

Considérons les deux fichiers films-plus.xml et cine-club-plus.dtd, abordés dans le cadre des travaux dirigés. Ce sont des versions *erronées* qui ont été jointes à la distribution : le document XML n'est *pas* valide par rapport au fichier DTD. Identifier les erreurs et montrer qu'il est possible de les corriger en :

— supprimant toute déclaration et utilisation de l'attribut `with-vhs` — une autre possibilité étant de le déclarer de type `CDATA` — ;

— de modifier un attribut `for-film` d'un élément `opening`.

## 4 *Intermezzo* : espaces de noms

Considérer le fichier teacher-3.xml, ainsi que le fichier DTD namespace-example-3.dtd qui lui est associé. Remarquer comment les espaces de noms sont organisés. Pour voir quels sont les espaces de noms qui sont accessibles et dans quelle partie du document, vous pouvez faire tourner le programme XSLT [5] traverse-plus.xsl. Pour appliquer une feuille de style XSLT 2.0, nous utiliserons l'outil Saxon, plus précisément la *home edition* [6] de ce programme, dans sa version en Java [7]. Effectuer d'abord l'initialisation suivante :

```
export CLASSPATH=${CLASSPATH}:/opt/saxon/saxon-he-11.5.jar
```

— cette initialisation est à effectuer une fois par session, il peut donc être utile de recopier cette ligne dans un fichier exécutable au moyen de la commande UNIX `source` — et le lancement du programme s'effectue par :

```
java net.sf.saxon.Transform -s:teacher-3.xml -xsl:traverse-plus.xsl
```

5. *e**X**tensible **S**tylesheet **L**anguage **T**ransformations.*
6. . . . ainsi que l'indique le logo « `he` ».
7. . . . ainsi que l'indique le suffixe du fichier .jar (*Java **AR**chive*) utilisé.

Par la suite, vous utiliserez Saxon comme processeur XSLT pour vos exercices ; l'annexe L déjà mentionnée vous donnera toutes indications utiles quant à son fonctionnement.

Revenant au programme traverse-plus.xsl, bien remarquer en particulier que le changement d'espace de noms par défaut ne se transmet *pas* aux attributs. Vous pouvez voir aussi les espaces de noms en utilisant le programme xmllint, mais uniquement en utilisant les fonctions `name()`, `local-name()` et `namespace-uri()` de XPath : cela tient au fait que l'axe « `namespace::` » ne peut pas lui-même être utilisé dans un motif (*pattern*) de XPath :

```
xmllint --shell teacher-3.xml
/ > cd /namespace-example-3/*[1]
l3:teacher > xpath name()
Object is a string : l3:teacher
l3:teacher > xpath local-name()
Object is a string : teacher
l3:teacher > xpath namespace-uri()
Object is a string : http://l3.univ-fcomte.fr
l3:teacher >
```

# 5 *Finale* : travail sur des références bibliographiques

Nous retournons à votre version corrigée du fichier books-0.xml (*cf.* § 1). Rappelons qu'il contient pour la plupart des spécifications d'œvres de science-fiction — livres (*books*) ou nouvelles (*inproceedings*) extraites d'une anthologie — et que le fichier DTD correspondant est sf-0.dtd.

## 5.1 *Intrada* : critique de la version primitive

Examinez les choix opérés dans le fichier DTD sf-0.dtd, ainsi que dans le fichier correspondant de données books-0.xml et discutez-les. Par exemple, examinez dans quelle mesure certains éléments — donnant accès à une information que l'on peut qualifier d'« atomique » — pourraient être modélisés par des attributs.

Vous pouvez remarquer aussi que la modélisation donnée souffre de quelques limitations : en particulier, l'impossibilité d'exprimer un sous-arbre pour une personne en fonction d'un autre sous-arbre pour une autre personne. Cette limitation est particulièrement visible lorsque nous spécifions un pseudonyme : les informations concernant la « personne réelle » sont regoupées dans la valeur associée à un attribut, et non données dans un sous-arbre dont l'élément de base serait `personname`. Ce qui n'est pas possible directement, car on ne peut pas exprimer un élément en fonction de lui-même.

## 5.2 *Variations* : nouveau texte, nouveau fichier DTD

L'un des principaux reproches à l'organisation du fichier books-0.xml concerne la gestion des informations rattachées aux personnes : à chaque mention d'une personne, toutes les informations que l'on a pu collecter à son sujet sont rassemblées. Une telle organisation entraîne une redondance d'informations pour les personnes citées plusieurs fois, ce qui en particulier est le cas des auteurs de *plusieurs* ouvrages d'une même bibliographie.

Dans la version révisée books-1.xml que vous propose la distribution, l'arbre global a été réorganisé en deux sous-arbres, l'un consacré aux ouvrages, l'autre répertoriant les informations concernant les personnes. Chaque personne citée — que ce soit dans la bibliographie ou dans les

informations concernant les personnes — a été dotée d'un attribut univoque de type `ID`, et une référence à cette personne se fera par le biais d'un attribut référençant cette information. Un tel attribut sera de type `IDREF`... ou de type `IDREFS`, par exemple, pour modéliser le fait que certains ouvrages ont été écrits par *plusieurs* auteurs. Notez que cette ré-organisation — l'accès systématique à un élément `personname` par un attribut de type `IDREF(S)` — fait également disparaître la limitation signalée plus haut à propos des pseudonymes. Une nouvelle version du fichier DTD, sf-1.dtd, vous a également été distribuée, mais ce fichier est *incomplet* : les parties signalées par le motif `**to-be-completed**` sont à compléter par vos soins. Notez que vous devez parvenir à la validation du document books-1.xml sans apporter la *moindre modification* à ce dernier.

### 5.3 *Sinfonia* : plus de structuration

Une amélioration introduite dans le fichier books-1.xml par rapport à la version primitive books-0.xml concerne les spécifications de *titres*, de façon à ce qu'un tri puisse être fondé sur les premiers mots significatifs, les articles et prépositions situés en début étant à omettre. Ce genre de fonctionnalité est difficile à automatiser complètement car très dépendant de la langue employée et sujet à exceptions. Notre solution repose sur l'utilisation d'un élément supplémentaire :

```
<title><skippable>Le </skippable>policier apache</title>
<title><skippable>The </skippable>Sleeping Sorceress</title>
<title><skippable>Auf der </skippable>Spur des Vernichters</title>
```

(remarquez que le contenu de l'élément `skippable` se termine toujours par un caractère d'espacement, expliquez pourquoi c'est préférable). Puis expliquez pourquoi la spécification de l'élément `title` donnée dans le fichier sf-1.dtd n'est qu'un pis-aller, ne traduisant pas réellement l'utilisation de l'élément `skippable` (il s'agit d'une limitation inhérente au formalisme des DTD).

### 5.4 *Fuga* : fonction `id` d'XPath

Considérez, dans le fichier `books-1.xml`, une valeur d'un attribut déclaré de type `ID`. Remarquez que vous pouvez récupérer l'élément auquel est attaché cet attribut au moyen de la fonction « `id(...)` » de XPath. Ceci suppose toutefois que le programme `xmllint` effectue une validation automatique par rapport au fichier DTD décrivant le document XML, donc l'emploi de l'option `--valid`, comme vous pouvez le constater par vous-mêmes :

```
xmllint --shell books-1.xml
> cat id("bachman")          # (Peau de balle et balai de crin.)
> quit
xmllint --shell books-1.xml --valid
> cat id("bachman")          # (Nous y sommes...)
```

### 5.5 *Postludio* : de nouvelles expressions XPath

Donner des expressions du langage XPath permettant de récupérer les sous-arbres suivants :

— les titres des œuvres en allemand ;

— le nombre d'œuvres qui sont des nouvelles extraites d'anthologies (utiliser la fonction `count` de XPath) ;

— le nombre d'ouvrages qui sont des anthologies — rappelons qu'une *anthologie* est rassemblée par un ou plusieurs *anthologistes* (élément `editor`[8]) et n'est pas le fruit d'un ou plusieurs auteurs (éléments `author`) travaillant en collaboration — ;

— le nombre de notes qui contiennent un texte en allemand ;

— les *clés* — attributs `key` — des auteurs qui ont écrit à deux ou plusieurs ;

— les *prénoms et noms* — éléments `personname` — des auteurs qui ont écrit à deux ou plusieurs ;

— les titres d'œuvres dont au moins un auteur a utilisé un pseudonyme.

8. Rappelons qu'en anglais, « *editor* » a le sens de « rédacteur en chef », le mot « éditeur » étant traduit par « *publisher* ».