III – Exploiter son édition numérique

La partie de cours que je vais vous présenter aujourd'hui vise à montrer les différentes méthodes que vous avez en votre possession pour exploiter votre édition numérique. Pour ce faire, je présenterai des outils qui sont systématiquement liés à la XML. Je parlerai de la fouille de textes avec le XPath ou XML Path Langage, qui permet de désigner des chemins et portions de documents XML ; mais aussi, avec le XQuery ou XML Query Langage, qui est un langage de requête pour des fichiers XML. Je parlerai de transformation de texte, avec le XSLT ou eXtensible Stylesheet Langage Transformation, qui est un langage de transformation écrit en XML. Enfin, je parlerai de publication de corpus, avec TEI Publisher qui est un outil basé en exist-db, qui est un système de gestion de base de données en XML. Ainsi, je vais vous montrer l'étendu du XML-TEI et de ses applications.

A – Fouiller dans ses données (XPath et XQuery)

Le XPath

Le XPath est un langage de requête qui permet de rechercher des informations au sein d'un fichier XML, qui s'utilise de multiples manières, tel qu'avec le XQuery, qui fera de l'extraction d'information, ou la XSLT, qui fera de la transformation. Comme son nom l'indique, le XPath s'utilise en fournissant un chemin, similaire aux chemins de fichier que l'on peut trouver au sein des systèmes de navigation des ordinateurs (et notamment dans la navigation au sein d'un terminal). Ces chemins de localisation se composent de plusieurs étapes qui permettent de descendre au fur et à mesure dans l'arbre XML. Ces étapes sont séparées par un slash (/) : un seul signifie que l'on descend sur l'élément directement en dessous dans la hiérarchie alors que deux signifie que l'on saute une ou plusieurs étapes. La requête pourra également être affinée à l'aide de prédicat, que l'on retrouvera entre crochets et qui permettra de filtrer un peu plus les résultats. Ensuite, lorsque l'arbre est correctement connu, les noms de balises pourront être utilisés pour décrire l'étape, mais dans le cas contraire, le XPath se compose de plusieurs axes de navigation au sein de l'arbre :

- parent ou ancestor : cela permet de remonter dans l'arbre, soit avec l'élément directement au-dessus pour le premier, soit pour des éléments plus hauts dans la hiérarchie pour le second
- child ou descendant : cela permet de descendre dans l'arbre, soit avec l'élément qui sera directement en dessous pour le premier, soit pour des éléments plus bas dans la hiérarchie pour le second
- preceding ou following : cela désignera l'élément juste avant ou juste après dans l'arbre
- preceding-sibling ou following-sibling : cela désignera, pour un élément avec un parent spécifique, tous les enfants précédents ou suivants de ce parent

- attribute : cela désignera les attributs de l'élément appelé de l'arbre

Quelques exemples en XPath

- ❏ **//book/title** → tous les titres de livre
- ❏ **//book/title[@xml:lang="fr"]** → tous les titres de livre en français
- ❏ **//book/@type** → les types de chacun des livres
- ❏ **//book[@type="fantasy"]/author** → les auteurs des livres de type "fantasy"
- ❏ **//book/date[preceding::author="Agatha Christie"]** → les dates des livres où l'auteur est Agatha Christie
- ❏ **//book[@type="fantasy"]/title[@xml:lang="en"]** → les titres anglais de livres fantastiques
- ❏ **//author/following-sibling::element()** → ce qui suit l'auteur dans le parent livre, soit la date

Exercice

Ouvrez le fichier "index_places" placé dans le dossier "xpath_xquery" du dépôt du cours. En vous aidant des exemples précédemment cités et de la documentation, trouvez les éléments suivants :

- ❏ Les noms de lieux qui sont des villes → //listPlace/place[@type="city"]/placeName[1] (350 (avec le crochet de fin) ou 368 (sans le crochet de fin) résultats)
- ❏ Les coordonnées géographiques des lieux qui sont des pays → //place[@type="country"]//geo (53 résultats)
- ❏ Les identifiants des lieux qui sont des mers → //place[@type="sea"]/@xml:id (8 résultats)
- ❏ Les noms de lieux des régions françaises contenus dans l'index → //place[@type="region"]/country[text()="France"]/preceding-sibling::placeName ou //place[@type="region"]/placeName[following-sibling::country="France"] (14 résultats)

Le XQuery

Le XQuery ou XML Query Language est un langage de requête pour l'extraction d'information dans un arbre XML. Il se compose de divers éléments qui permettent de faire une requête et ainsi d'obtenir des informations extractibles. Comme on l'a vu précédemment, on utilise le XPath pour créer les requêtes, puisque l'on aura besoin de préciser ce qui nous intéresse dans l'arbre afin d'extraire nos informations. Pour ce faire, on retrouvera plusieurs chemins de fichier, puisque la requête permet de se promener dans plusieurs endroits de l'arbre pour obtenir ce que l'on recherche. On peut également utiliser des fonctions, pour spécifier certaines choses, comme *concat()*, que je présenterai dans les exemples. Cette fonction permettra de combiner plusieurs résultats en même temps, comme avoir dans le résultat deux enfants d'un même parent. Enfin, l'élément le plus important pour travailler le XQuery sera *FLOWR*, qui sont les initiales des cinq clauses qui font la syntaxe

du XQuery. Cela signifie *For*, *Let*, *Order by*, *Where* et *Return*. Toutes ces clauses ne sont pas obligatoires : seuls la première et la dernière le seront. *For* est nécessaire pour initier la requête et *Return* est nécessaire pour la clore et pour produire les résultats à extraire. Ainsi, *For* servira à itérer sur une séquence. Il permet de parcourir une séquence par l'intermédiaire d'une variable, qui sera indiqué par le signe $ et qui prendra successivement la valeur de chaque élément de la séquence. Ainsi, si vous définissez la variable comme une balise contenue dans un arbre XML, si cette balise est présente plusieurs fois, la variable contiendra toutes les instances de cette balise contenue dans l'arbre. *Let* est utilisé pour définir une constante, qui prendra pour valeur l'information qui lui sera donnée, que cela soit une séquence, un fichier, etc. Cela peut être utile notamment lorsque l'on veut récupérer plusieurs fichiers en même temps, ceux-ci pourront ainsi être appelés dans deux constantes que l'on pourra utiliser par la suite, comme je le montrerai dans un des exemples. Pour cette déclaration, *$const* est la variable de stockage et les deux-points, suivi du signe égale (:=) est l'opérateur d'assignation. *Order by* est mis à contribution pour trier une séquence, selon un ou plusieurs critères. Il s'utilise en suivant du *For* et permet de contrôler la manière dont les informations s'afficheront dans la fenêtre de résultats. Ce tri est choisi selon un élément particulier contenu dans la séquence recherché. Il y aura ensuite deux options : on pourra choisir d'opérer un tri croissant avec **ascending** ou décroissant avec **descending**. *Where* s'utilisera similairement à l'option *Order by*, par le fait qu'il se trouve en suivant de *For* et qu'il est défini selon un ou plusieurs critères. Cependant, il servira, lui, à filtrer une séquence. Cela signifie que son objectif est de réduire le nombre de résultats suivant le critère qui est donné. Si l'on spécifie avec *Where*, que l'on ne veut que les instances avec un certain élément, attribut ou valeur, les résultats seront filtrés pour ne retenir que ces instances. Enfin, *Return* permet de clore une expression FLOWR : celle-ci ne sera pas considérée comme juste et complète tant qu'un élément *Return* n'aura pas été ajouté. Il permet également de choisir les éléments que l'on veut voir afficher dans les résultats. C'est à ce moment-là que l'on pourra d'ailleurs utiliser certaines fonctions, avant de choisir le type d'affichage que l'on veut pour les résultats.

Exemples

Requête 1 → Types de lieux (A-Z) avec des valeurs uniques et les "_" compris dans @type sont remplacés par un espace

```
let $doc := doc("/Users/fchiffol/Downloads/index_place.xml")
for $type in distinct-values($doc//place/@type)
order by $type ascending
return replace($type, '_', ' ')
```

Requête 2 → Nom/prénom (A-Z) des femmes françaises

```
let $doc := doc("/Users/fchiffol/Downloads/index_person.xml")
for $person in $doc//person
where $person/sex/@value="2" and $person/nationality="French"
order by $person/persName/text ascending
return $person/persName[2]/text()
```

Requête 3 → Lieux et de leur type (Z-A), sous forme d'une phrase

```
let $doc := doc("/Users/fchiffol/Downloads/index_place.xml")
for $place in $doc//place
order by $place/@type descending
return $place/placeName[1] || " is a " || $place/@type
```

Requête 4 → Personne française avec leur première occupation et leur lieu de naissance (A-Z)

```
let $doc := doc("/Users/fchiffol/Downloads/index_person.xml")
for $person in $doc//person
where $person/nationality = "French"
order by $person/persName[1]/text() ascending
return concat($person/persName[1], " was a ", $person/occupation[1], " and was born in ",$person
/birth/placeName)
```

Requête 5 → Dans les cas où un lieu de l'index des lieux correspond au lieu de naissance d'une personne de l'index des personnes, une phrase nous donnera le nom de cette personne, avec sa première occupation et le pays de son lieu de naissance

```
let $doc := doc("/Users/fchiffol/Downloads/index_place.xml")
let $doc2 := doc("/Users/fchiffol/Downloads/index_person.xml")
for $place in $doc//place
for $person in $doc2//person
where $place/placeName[1]=$person/birth/placeName
return $person/persName[1] || " was a " || $person/occupation[1] || ", born in " || $place/
country
```

Exercice

Après avoir ouvert l'application Oxygen, ouvrez les fichiers "index_places" et "index_person" placé dans le dossier "xpath_xquery" du dépôt du cours. En utilisant les exemples précédemment donnés et la documentation, faites les requêtes suivantes :

- ❏ Depuis l'index de personnes, récupérer le nom de toutes les personnes qui étaient journalistes

  let $doc := doc("/Users/fchiffol/Downloads/index_person.xml")

  for $person in $doc//person

  where $person/occupation = "journalist"

  return $person/persName[1

  → 91 résultats

- ❏ Depuis l'index des lieux, récupérer le nom de toutes les villes allemandes, classé par ville de A à Z

```
let $doc := doc("/Users/fchiffol/Downloads/index_place.xml")
for $place in $doc//place
where $place/@type="city" and $place/country="Germany"
order by $place/placeName[1] ascending
return $place/placeName[1]/text()
```

→ 14 résultats

- ❏ Depuis l'index des personnes, récupérer les personnes qui ont reçu le prix Nobel ("Nobel Peace Prize"), en faisant une phrase qui donne leur nom et leur nationalité

```
let $doc := doc("/Users/fchiffol/Downloads/index_person.xml")
for $person in $doc//person
where $person/event/p="Nobel Peace Prize"
order by $person/persName[1]
return $person/persName[1] || " was a " || $person/nationality || " " ||
$person/occupation[1] || " Nobel Peace Prize"
```

→ 18 résultats

- ❏ En joignant les deux index, récupérer les instances où le lieu de mort correspond à un des lieux de l'index de lieux et faites une phrase qui donne le nom de la personne, sa date de mort et son lieu (ville + pays) de mort, en classant le résultat de la mort la plus ancienne à la plus récente

```
let $doc := doc("/Users/fchiffol/Downloads/index_person.xml")
let $doc2 := doc("/Users/fchiffol/Downloads/index_place.xml")
for $person in $doc//person
for $place in $doc2//place
where $place/placeName=$person/death/placeName
order by $person/death/date/@when-iso ascending
return $person/persName[1] || " died in " || replace($person/death/date/@when-iso,
'-[0-9][0-9]-[0-9][0-9]', '') || " in " || $place/placeName[1] || ", " || $place/country
```

→ 317 résultats

B – Transformer ses données (XSLT)

XSLT ou *eXtensible Stylesheet Language Transformation* est un langage de programmation, et plus précisément un langage de transformation, qui va consister non plus à simplement

décrire la donnée, mais à la transformer pour lui donner une nouvelle sortie. Ce langage est destiné exclusivement aux fichiers XML, qui seront les seuls documents acceptés comme fichiers d'entrée.
Pour ce faire, XSLT utilise XPath, d'une manière assez similaire à ce que l'on vient de voir, mais cette fois-ci, il ne servira pas qu'à naviguer au sein même de l'arbre XML, mais il aidera aussi à effectuer des transformations.
Il est possible de transformer le fichier XML vers divers types de formats de sortie avec XSLT. Tout d'abord, il est possible de réaliser une transformation de fichier XML à fichier XML. Cela peut être utile dans les cas où on veut mettre à jour un groupe de fichiers, par exemple s'il y a eu des changements dans les règles de balisages pour certains éléments des TEI Guidelines. Au lieu de changer les fichiers un à un, et puisque l'arbre XML devrait généralement suivre le même schéma TEI, la transformation XSLT permet de faire un seul script de transformation qui pourra alors s'appliquer à tous les fichiers. Ensuite, il est aussi possible de transformer son arbre XML pour en faire une page HTML, visible sur le web. Dans ce cas, on pourra aller chercher les différents éléments de l'arbre et les insérer dans une structure HTML pour faire ressortir son contenu. Il est même possible de créer plusieurs pages en fonction du contenu de son arbre et de ce que l'on veut faire apparaitre. Enfin, il est aussi possible de créer une sortie LaTeX, qui est un langage permettant de rédiger des documents avec une mise à page réalisée automatiquement. Avec ce script de transformation, il est possible de travailler à une sortie correctement mise en page du fichier encodé, qui pourra par la suite être aisément transformé en PDF pour un meilleur accès.

Afin de réaliser ces transformations, une série de règles appelées *templates* ont été mis en place. Elles permettent de faire plusieurs types d'action avec l'arbre XML :

- on peut simplement faire apparaitre le contenu des balises,
- faire apparaitre tout un pan de l'arbre, avec ou sans les balises,
- choisir quel partie de l'arbre, on veut afficher,
- sélectionner des modes d'affichages,
- choisir plusieurs types de présentation, etc.

Il existe de multiples règles de transformation en XSLT, qui permettent d'arriver à divers résultats en fonction de ce qui est requêté :

- Requêter au sein de l'arbre :
    - Se situer dans l'arbre XML : <xsl:template>
    - Appliquer les règles de transformations définies pour un élément : <xsl:apply-templates>

    - Appliquer les règles de transformations définies pour un cas donné d'une partie de l'arbre : <xsl:call-templates>
- Faire apparaitre du contenu :
    - Chercher le contenu d'une partie de l'arbre XML, balises et textes : <xsl:copy-of>
    - Chercher le texte contenu dans les balises appelées : <xsl:value-of>
- Appel d'une boucle pour itérer sur les éléments, les uns après les autres : <xsl:for-each>
- Affichage d'un élément sous certaines conditions : <xsl:if>/<xsl:choose>/<xsl:when>/<xsl:otherwise>
- Création d'un mode pour faire apparaitre plusieurs fois la même partie de l'arbre XML, mais selon divers critères : @mode

Exercice

En vous aidant de l'exemple fourni avec les fichiers XML, XSL et HTML de "Orgueil et Préjugés" et en vous servant du *template* XSL fourni pour "Le Tour du Monde en 80 jours" et de votre fichier d'encodage, faites apparaitre les informations suivantes:

- ❏ Créer une variable et un fichier de sortie qui permettront de produire le fichier HTML "Le_tour_du_monde_en_80_jours.html" qui aura pour titre HTML "Encodage"
- ❏ Faites apparaitre, sous forme de liste labellisée, toutes les données contenues dans le <fileDesc>
- ❏ Faites apparaitre le texte après un titre "Texte" où les sauts de lignes apparaissent (balise
 en HTML), où les entités nommées sont en italiques (balise <i> en HTML) et où la foliation est visible comme un titre (balise <h6> en HTML)
- ❏ Afficher le lien du facsimilé avec l'option de cliquer dessus pour y accéder (balise <a> en HTML)

En découvrir plus

À partir du fichier "Le Tour du monde en 80 jours" que vous avez encodé et que vous venez de tester, on peut également aller plus loin et produire plusieurs pages HTML en même temps, ce qui pourra nous donner les pages suivantes :

- ❏ Une page d'accueil qui contient les différentes pages
- ❏ Une page de métadonnées reprenant les informations que vous avez encodées précédemment
- ❏ Une page d'index qui présente les personnes et lieux mentionnés dans le texte, avec des liens vers des fichiers d'autorité.
- ❏ Deux versions du texte : une de lecture avec le lien vers le facsimilé et une diplomatique qui reprend les fins de ligne et qui affiche également le facsimilé.

Comme vous pouvez le voir, comme on travaille en HTML, il est également possible de jouer un peu avec le CSS pour modifier la manière dont les éléments apparaissent. Vous pouvez ainsi faire des pages HTML correctes et plus travaillées si vous en avez envie.

C – Publier son corpus (TEI Publisher)

XSLT n'est pas la seule méthode qui existe pour faire une transformation de son fichier, notamment s'il n'y a pas que quelques fichiers à transformer, mais tout un corpus, plus ou moins long. Dans un souci d'aider les chercheurs qui voudraient publier leur corpus afin de le rendre accessible à un plus grand public, mais sans avoir de grandes aptitudes dans le développement et la programmation, un outil a été créé : TEI Publisher.

TEI Publisher est un outil qui dépend d'*exist-db*, un système de gestion de base de données open-source basé sur la technologie XML et qui permet de transformer des fichiers XML-TEI pour une publication web et une exportation en divers formats (LaTeX, ePUB, PDF), avec spécialement la possibilité de générer une application web, où la majorité des transformations (que l'on a notamment vu avec XSLT) seront faites automatiquement par TEI Publisher et notre travail sera simplement de définir les règles de traitement et d'affichage des éléments de notre arbre TEI.

Cet outil présente un intérêt dans la formation dispensée ici, car il a été créé en conformité avec les TEI Guidelines, il s'appuie sur le *TEI Processing Model* (un élément développé dans les guidelines que je n'ai pas encore présenté) et il requiert en majorité, à l'importation, des documents conformes à la XML-TEI.

TEI Publisher s'appuie sur deux éléments principalement pour fonctionner : une ODD et un *template*. L'ODD est utilisée pour documenter son arbre XML-TEI, comme je l'ai présenté précédemment. Ici, elle est utilisée afin de définir le *TEI Processing Model* de chacun des éléments présents dans l'arbre XML TEI. Cela signifie que l'on va définir, pour chacun des éléments, la manière dont ils devront être traités lors de leur transformation et donc lors de leur affichage. On peut décider dans quels conditions l'élément peut apparaitre, quel mode il doit prendre (est-ce un titre ? une liste ? une note ? un élément intégré à un paragraphe ? l'affichage dépendra aussi de ce choix) et même si on veut donner à son contenu une écriture, une taille, une couleur particulière. Il n'est pas nécessaire de tout définir, car il existe certaines règles prédéfinies d'affichage et dans ce cas, on pourra spécifier l'affichage pour un élément si on considère cela nécessaire. Les règles définies dans cet ODD pourront se retrouver dans l'autre élément majeur du fonctionnement de TEI Publisher : le *template*. Un *template* est un modèle de présentation de données, c'est-à-dire la mise en page défini pour l'apparition du texte XML TEI qui sera affiché et des autres éléments d'affichage qui seront choisis. Le *template* vaut pour tous les fichiers présentés sur l'application web (à moins que vous décidiez autrement), ce qui permet une homogénéisation de la présentation du site. Il est fait à base du langage HTML et il est fourni en partie par TEI Publisher, puisque le logiciel propose des *templates* d'exemple assez varié, que l'on peut choisir avant de générer son application. Ainsi, vous pouvez par la suite décider de le modifier à votre

convenance, à l'aide de quelques tutos trouvés sur Internet, ou vous pouvez le garder comme tel, si cela convient à la manière dont vous voulez faire apparaitre votre corpus.
Cet outil a l'avantage d'avoir une documentation longue et exhaustive, qui permet d'aider à la compréhension de l'outil et à son fonctionnement, même pour des novices.
Pour aider à la compréhension :

- Démonstration des divers textes présents dans la collection de démonstration
- Aire de jeux et zone d'annotation où on peut importer ses propres documents et tester des ODD/*templates* ou travailler son annotation d'entités nommées
- Présentation de cinq applications d'exemple de TEI Publisher (cela montre divers types de présentation, de choix d'affichages, etc.)
    - Van Gogh
    - When the wall came down
    - Early English Books
    - Shakespeare plays
    - DiScholEd

Exercice avec TEI Publisher

Tester l'annotation

Pour cette exercice, il vous faudra avoir téléchargé le fichiers "la_reine_margot.xml" situé dans le dossier "teipublisher" du dépôt du cours et vous connecter à TEI Publisher avec les identifiants qui sont donnés sur le site.

- ❏ Aller dans "Annotation Samples"
- ❏ Importer le fichier XML de *La Reine Margot* puis aller dans le fichier
- ❏ Encoder une des instances de "Charles IX" et dites-moi ce que vous observez
  Réponse : Une fois l'encodage d'un seul Charles IX, l'outil va proposer d'encoder de la même façon toutes les autres instances similaires
- ❏ Encoder les autres entités présentes dans le texte

Tester l'ODD

Pour cette exercice, il vous faudra avoir téléchargé le fichier "orgueil_et_prejuges.xml" situé dans le dossier "teipublisher" du dépôt du cours et vous connecter à TEI Publisher avec les identifiants qui sont donnés sur le site.

- ❏ Créer une ODD en mettant comme nom "odd_urfist" et comme titre "ODD URFIST" puis ouvrez cette ODD
- ❏ Ajouter un élément "persName", ajouter un *model* dedans et mettre dans "Renditions" : "color: red;", et enregistrez
- ❏ Aller dans "Aire de jeux"
- ❏ Importer le fichier XML d'*Orgueil et Préjugés* et aller dans le fichier
- ❏ Tester différentes ODD (Dantiscus, Graves, VG), ainsi que la nôtre, et dites-moi ce que vous observez
  Réponse : Avec les différentes ODD, on peut observer que la manière dont les entités nommées apparaissent changent, comme c'est le cas aussi pour notre ODD.

- ❏ Aller dans l'ODD, ajouter un élément "pb", supprimer le *model* existant et en créér un nouveau
- ❏ Mettre dans template "[Page [[number]]]" avec "@facs" comme valeur du paramètre 'facs' et "@n" comme valeur du paramètre 'number', et comme behaviour 'block' et enregistrez
- ❏ Aller voir le fichier XML et tester ce qui est proposé
  Réponse : Avec l'ajout du facs, on voit que l'on peut accéder à l'image en cliquant dessus

<u>En découvrir plus</u>

À partir du fichier "formation_urfist.xar" que vous aurez téléchargé depuis le dossier "teipublisher", vous pourrez observer diverses affichages du fichier que vous avez encodé auparavant.

- ❏ Connectez-vous en tant qu'admin sur votre instance TEI Publisher et importez l'application .xar
- ❏ Entrez les identifiant et mot de passe dans l'application et ajoutez votre fichier encodé du *Tour du monde en quatre-vingts jours*
- ❏ Changez le template d'affichage du fichier avec la ligne suivante (**<?teipublisher template="nomdetemplate.html"?>**) en début de fichier pour voir à quoi ressemble le texte avec un facsimilé, des modes ou un index
  Réponse : Apparition du facsimilé importé depuis Gallica grâce au IIIF, affichage des entités nommées en couleurs, affichage des données dans l'"index" et affichage des lieux sur la carte