

Knowledge-based Access to Historical Texts

Mohsen Hassan Naeini, Fabrice Hategekimana, Azeem Arshad

Contents

Introduction	2
Contexte	2
Nos objectifs	2
Méthodologie	2
Sources	2
Conversion du fichier word en fichier html	2
Conversion du fichier html en structure python	2
Conversion des structures python en triplets	3
Conversion des triplets en base rdf	3
Travaux connexes:	3
Ontologie	4
Classes liées aux personnes	4
Classes liées aux localisations	5
Classe liée à l'état	5
Les autres pages	6
Algorithmes	6
Données	6
To_HTML	6
To_Triplet	8
To_Owl	8
Résultats	8
Interface	8
Conclusion	8
Difficultés rencontrées	8
Amelioration et la suite	8

Introduction

Contexte

motivations

Nos objectifs

Notre objectif est de créer une interface de lecture thématique pour les textes historiques. Plus spécifiquement notre objectif se réalisera sur ces deux fronts. Premièrement, il nous faut créer un graphe de connaissances (ontologie) pour indexer sémantiquement les textes. Deuxièmement, nous allons développer un système de “requête” pour naviguer dans le graphe de connaissances et accéder aux textes.

Méthodologie

Sources

Les historiens nous ont fournis une liste intéressante documents. Mais le type de fichier qui nous intéresse le plus est l’index. Nous nous sommes concentré sur le fichier d’index “Index_1543.docx”. Le fichier est assez simple dans sa composition.

Nous avons d’abords une première section écrite avec en texte clair. Cette première section nous introduit le document parlant de son contenu et de la structure adopté dans l’écriture de l’index. Nous y avons trouvé des informations précieuses pour l’élaboration de nos parseurs qui extrairons par la suite le contenu de ce document. Nous avons ensuite l’index. Celui-ci relate les éléments classés par ordre alphabétique. Si la première partie contient du texte non structuré, nous nous retrouvons avec une deuxième partie qui contient des informations que nous somme capable d’extraire.

La figure 1 illustre le pipeline utilisé pour le traitement des donnée ainsi que la partie concernant notre interface de requête.

Conversion du fichier word en fichier html

Premièrement, le format du fichier d’index est changé du format word (.docx) au format html. Nous avons décidé d’utiliser l’outil Pandoc qui est une sorte de convertisseur universel de différents type de fichier de documentation. Cette étape est la plus facile du processus puisque nous utilisons un outils déjà existant.

Conversion du fichier html en structure python

À partir du fichier HTML obtenu, des données structurés de python (sous la forme de liste) sont alors extraits pour des traitements supplémentaires. Ici nous

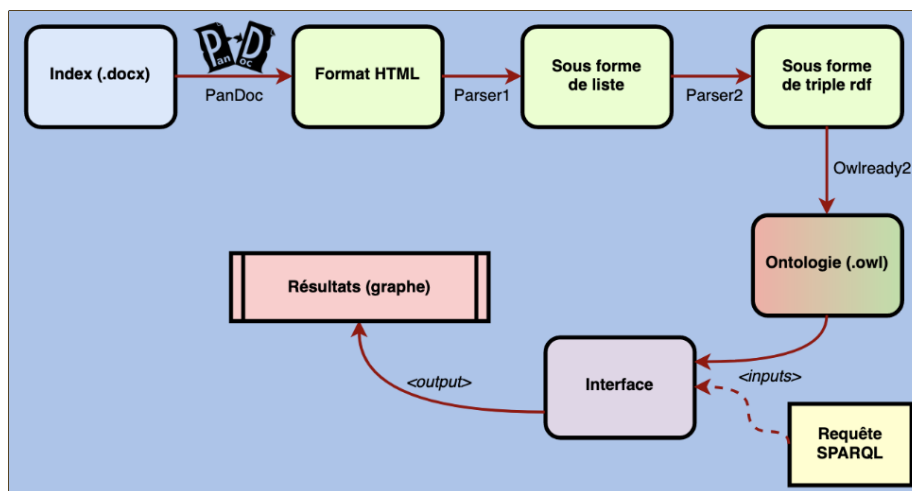


Figure 1: Comme présenté dans ce flowchart, notre solution va extraire et formater les données présentes dans l'index à l'aide de quelques outils et quelques étapes.

utilisons un parseur construit par nos soins pour obtenir ces structures. Notre tentative est seulement de transformer le typage syntaxique en typage structurel. Ainsi les éléments stylistiques comme la mise en gras ou l'usage d'italiques vont laisser place à des étiquettes qui définiront les termes de façon plus régulière et donc traitable par une machine. Nous avons utilisé les définitions données au début de la fiche d'index. Voici un tableau qui illustre les correspondance établies.

style	étiquette	signification
gras	common_new	nom commun contemporain
italique	Place	Lieux et cours d'eau
petite majuscule	Personne	Nom de personne

Conversion des structures python en triplets

Dans cette nouvelle étape, les données structurées (sous forme de liste python) vont être traitées sémantiquement par un bloque de code pour produire une suite de triplets rdf. Ici nous faisons en sorte de déplier toutes les sous structure pour avoir un rendu plus linéaire. C'est aussi dans cette phase de dépliage que nous pouvons appliquer d'autre parseurs pour extraire d'avantage d'information sur les entités. Par exemple nous pouvons obtenir plus d'information sur les éléments contenant une parenthèse ou des information entre crochet. Après cela, nous finissons avec un groupe d'élément (terme+type).le premier terme va être lié aux autres termes par des propriétés établies au préalable pour donner des triplets.

Conversion des triplets en base rdf

Dans cette dernière partie, nous finalisons notre ontologie. Nous utilisons le module python owlready2. Il suffit maintenant de joindre ces triplets avec l'ontologie que nous avons déjà créé pour obtenir une nouvelle ontologie contenant les classes, ainsi que les propriétés et entités. Les entités, propriétés et les liens entre entités sont construit à mesure que les triplets sont scannés.

Cette nouvelle ontologie va servir de base de données pour notre interface

Noms de l'Article: Review of Tools for Semantics Extraction: Application in Tsunami Research Domain

Le cas de cette études s'intéresse aux recherches axées sur les Tsunamis.

Ontologie

Après une observation et une étude minutieuse de la fiche d'index, nous avons pu dégager quelques outils pratiques pour l'élaboration d'une ontologie représentative. Ici nous nous intéressons plus particulièrement aux classes. Nous avons établies différentes classes importantes à nos yeux. Nous avons dégagés les classes liées au personnes, les classes liées aux localisations ainsi qu'une classe lié à toutes les unités étatiques et une classes concernant les écrits par lequel l'index fait ses références.

Hierarchie des classes.

Classes liées aux personnes

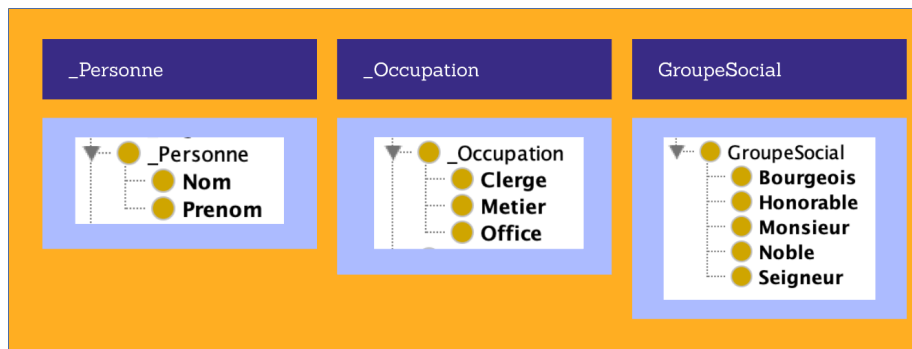
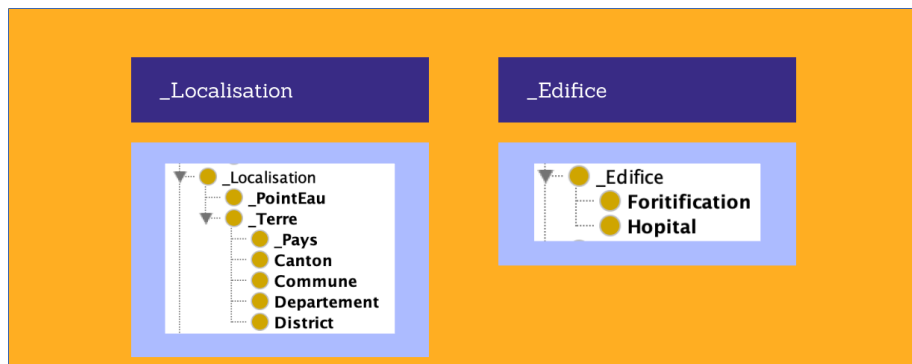


Figure 2: classes_liées_aux_personnes

Nous avons jugé pertinent de prendre les classes liées aux personnes. Nous avons créé les classe Personnes, Occupation et GroupeSocial.

Classes liées aux localisations



La localisation et les lieux sont aussi un aspect intéressant de ces registres, en effet, nous sommes en mesure de pouvoir répondre à la question “où”. Nous avons construit la classe localisation et la classes Edifice.

Classe liée à l'état

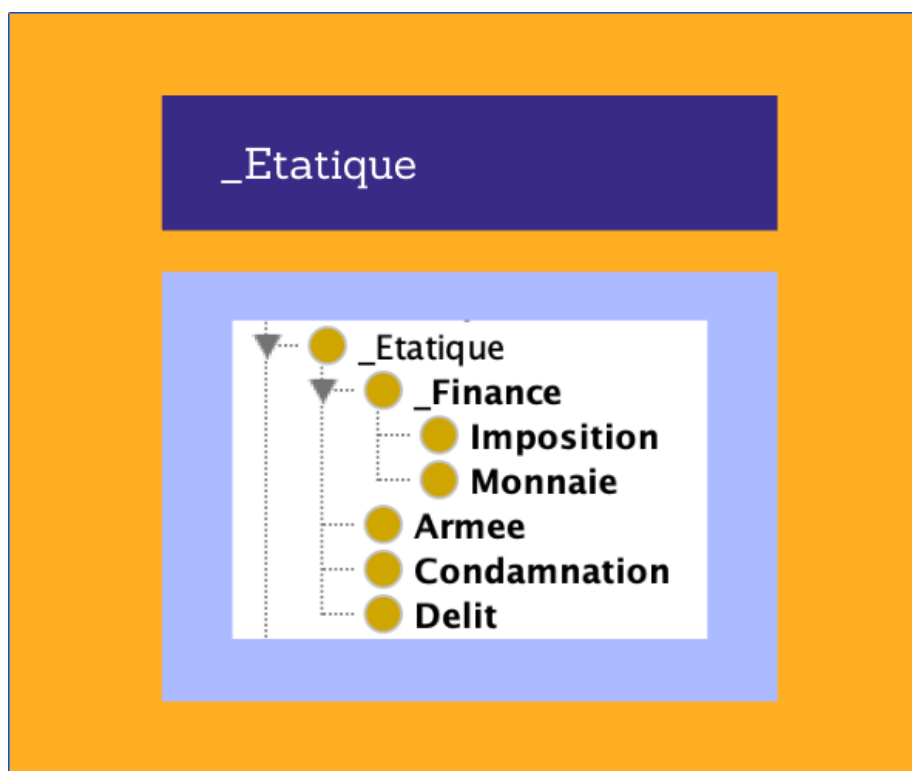


Figure 3: Classe_liee_à_l_état

Les autres pages

Ecrit _Page

Algorithmes

Données

```
<p>INDEX</p>
<p>Cet index recense de façon exhaustive tous les noms de lieux et de personnes mentionnés dans les Registres, ainsi qu'un
<p>L'index suit l'ordre alphabétique continu. Pour une même orthographe, nous présentons les entrées selon l'ordre suivant
<p>Les noms communs sont classés selon leur orthographe contemporaine (en gras), avec, le cas échéant, des précisions de se
<p>Les noms de lieux et les cours d'eau, en italique, sont indexés suivant leur appellation actuelle, suivie de l'appellati
<p>Les noms de personnes, en petites majuscules, sont classés sous le patronyme actuel, selon la graphie du <em>Dictionnair
<p>Les prénoms qui sont encore en usage de nos jours ont été modernisés.</p>
<p>Toutes les entrées matière se référant aux localités autres que Genève sont indexées sous le nom de la localité à laquel
<p>Abréviations utilisées : anc. = ancien ; arr. = arrondissement ; CC = Grand Conseil ou des Deux Cents ; ch.-l. = chef-li
<h5 id="a">A</h5>
<p><strong>abattoir</strong>, eschorcherie, 417, 420 ; voir aussi boucherie</p>
<p><strong>abbaye</strong>, voir couvent</p>
<p><strong>abbé</strong>, abbel, voir clergé</p>
<p><em>Abondance (F, dép. Haute-Savoie, arr. Thonon-les-Bains, ct. Évian-les-Bains)</em>, <em>Habundance</em></p>
<blockquote>
<p>- couvent [chanoines réguliers de saint Augustin], 200</p>
</blockquote>
<p>absenter, voir condamnation (bannissement, interdiction)</p>
<p>accoustrément ; accoustré, accoustree, voir habillement, ornement liturgique (habillement religieux)</p>
<p>accoutumé, accoustumee, voir coutume</p>
<p>acculpacion, acculpation ; accusé, aculpez, voir procès</p>
<p><span class="smallcaps">Achard</span>, Barthélemy, 419</p>
<p>acord, voir traité</p>
```

Figure 4: données

To_HTML

```
def docx_to_html():
    # prend le nom du fichier
    if len(sys.argv) == 1:
        name = "Index_1543.docx"
    else:
        name = sys.argv[1]
    newname = name[:-4]+"html"

    # conversion en html
    subprocess.run(["pandoc", name, "-o", newname])

    # conversion en triplets
    text = open(newname, "r").readlines()
```

Figure 5: to_html

```
def html_to_triplet():
    text = open("Index_1543.html", "r").readlines()

    # ignor = True
    ignor = False
    tab = []

    """
    Le fichier index contient du texte d'explication sur la structure de l'index
    Ce texte n'est pas intéressant vu qu'il ne peut pas être déstructuré.
    On doit donc couper le texte du début.
    """

    for line in text:
        # premièrement, on ignore ce qu'il y a avant h5
        if containsH5(line):
            ignor = False
        else:
            if ignor == False and not containsH5(line):
                res = parser.parse(line)
                print(res)
                if res != None:
                    tab.append(createTriplet(res))

    print(tab)
    return tab
```

Figure 6: to_triplet

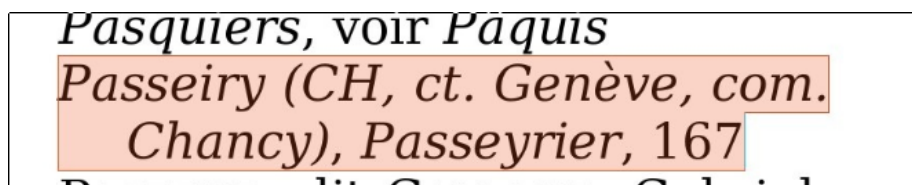
```
if triplet[1] == "inDepartement":
    loc = Classes["Localisation"](triplet[0])
    canton = Classes["Departement"](triplet[2])
    canton.inDepartement = [loc]
```

Figure 7: to_owl

To_Triplet

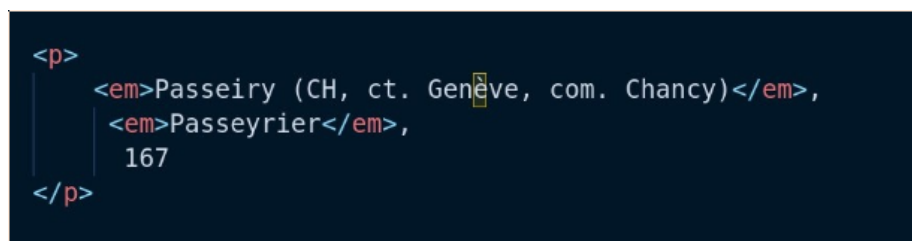
To_Owl

Résultats



Pasquiers, voir Paquis
Passeiry (CH, ct. Genève, com. Chancy), Passeyrier, 167
Passeyrier dit Cully, Cully, Cully

Figure 8: exemple_d_indice



```
<p>
  <em>Passeiry (CH, ct. Genève, com. Chancy)</em>,
  <em>Passeyrier</em>,
  167
</p>
```

Figure 9: html

Interface

Conclusion

Difficultés rencontrées

- Complications à comprendre l'index: notations, structure, etc.
- Parsing...:
 - Erreurs humaines dans les indexs: fautes de frappe
 - Caractères apparaissant en dehors des balises html
- Termes utiliser pour les classes de l'ontologie
- Travail manuel à faire pour parcourir l'index manuellement

Amelioration et la suite

- Parser: ajouter les propriétés manquantes


```

structure = [
    ['Passeiry ( CH , ct. Genève , com. Chancy )', 'place'],
    ['Passeyrier', 'place'],
    ['167', 'page']
]

triplets = [[
    ['Passeiry', 'inCountry', 'Suisse'],
    ['Passeiry', 'inCanton', 'Genève'],
    ['Passeiry', 'inCommune', 'Chancy'],
    ['Passeiry', 'type', 'place'],
    ['Passeiry', 'place', 'Passeyrier'],
    ['Passeyrier', 'type', 'place'],
    ['Passeiry', 'page', '167'],
    ['167', 'type', 'page']
]]

```

Figure 10: triplet

```

<rdf:Description rdf:about="#Passeiry">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <rdf:type rdf:resource="#_Localisation"/>
  <inPage rdf:resource="#167"/>
</rdf:Description>

<rdf:Description rdf:about="#Suisse">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <rdf:type rdf:resource="#_Pays"/>
  <inCountry rdf:resource="#Passeiry"/>
</rdf:Description>

<Canton rdf:about="#Genève">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <inCanton rdf:resource="#Passeiry"/>
</Canton>

<Canton rdf:about="#Chancy">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <inCommune rdf:resource="#Passeiry"/>
</Canton>

<rdf:Description rdf:about="#Passeyrier">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <rdf:type rdf:resource="#_Localisation"/>
</rdf:Description>

<rdf:Description rdf:about="#167">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  <rdf:type rdf:resource="#_Page"/>
</rdf:Description>

```

Figure 11: owl



Figure 12: interface

- web-app:
 - Ajouter une table des résultats pour que l'utilisateur puisse voir les résultats
 - Ajouter un wrapper pour éviter aux utilisateurs de devoir taper des requêtes SPARQL
- Utiliser SpaCy (Bibliothèque Python) pour aider le parser
- Ontologie: Ajout de classes supplémentaire -> Davantage compléter l'ontologie