Université de Genève

Knowledge Organisation Systems

D400006

Knowledge-based Access to Historical Texts

Auteurs :

Muhammad Azeem Arshad

Fabrice Hategekimana

Mohsen Hassam Al-Naeni

Courriels :

Muhammad.arshad.2@etu.unige.ch

Ganza.Hategekimana@etu.unige.ch

Mohsen.Hassan@etu.unige.ch

Github project repo :

https://github.com/azeemarshad97/kos-project

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique

Table des matières

Avant-Propos

Le fichier *.zip* contient le projet, télégargé depuis github (lien disponible dans la page de garde ou ici). Le fichier *.zip* contient également le fichier owl que nous avons mit apart pour vous faciliter l'accessibilité.

1 Introduction

Avec plus d'une centaines de volumes recueillant les archives électronique genevoises, devoir simplement rechercher une personne devient vite une tâche complexe auquel les historiens genevois sont confrontés tous les jours. L'organisation et la catégorisation des ces données devient dès lors un problématique important.

Une solution envisagée par le groupe de l'université de Genève spécialisé dans l'ingénierie de connaissances est de créer une ontologie à partir des indexes des volumes. Cela permettrait aux historiens de faire des recherches rapide et efficace.

Dans le cadre de notre cours de Knowledge Organisation Système, notre but était donc d'extraire de ces indexes les informations les plus importantes et de les classer dans une graphe de connaissances, sous la forme de triplets RDF. Étant donné le temps qui nous était impartit, notre travail se concentrait donc sur une petite partie de l'indexe. Notre travail pourrait néanmoins se généraliser sur l'ensemble des indexes.

1.1 Etat de l'art

Nous avons fait une recherche sur les travaux connexes et avons trouver des informations intéressantes sur des outils allants dans la même direction que nous. Nous entendons par là des outils capable de traiter des données semis structurées pour pouvoir obtenir des données sous la forme de triplets. Un premier article, *Review of Tools for Semantics Extraction : Application in Tsunami Research Domain* [1] , compare les différentes outils créés pour concevoir une ontologie automatiquement à partir d'un texte. Pour la plupart des outils :

- elles ne peuvent plus être installé ou télégargé
- elles ne sont plus maintenues
- seule la langue anglaise marche avec
- Les outils marchant avec la langue française :
  - elles ne sont pas accessible, voire pas de possibilité de télégargement
  - l'ontologie généré n'est pas utilisable. Par exemple, elle classe un même déterminant différemment !

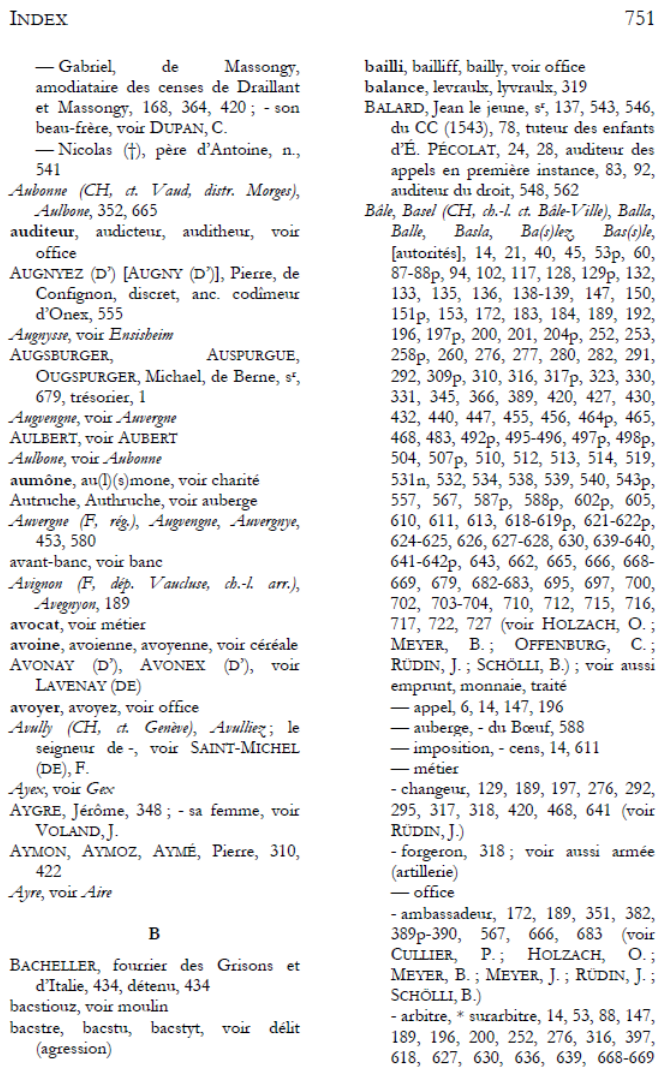
Un autre travail qui se rapproche de notre projet est celui des d'un groupe d'une université chinoise [2] qui utilise des outils de NLP et des bots de web-scraping pour collecter, organiser et enfin créer une ontologie appelé GAO. Cette dernière regroupe les archives électroniques de différentes provinces (le but étant de regrouper les archives des différentes provinces car il devient difficile d'accéder aux archives de chaque province séparément) ce qui facilite au niveau de l'accès et ainsi que de l'intéroptabilité. les outils utilisé ont été une source d'idées pour améliorer notre parser ainsi pour les améliorations qui pourrait être apporté dans le future.

2 Nos objectifs

Notre objectif est donc 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.

2.1 Sources

Les historiens nous ont fournis une liste intéressante de 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. Une page de l'index est montré dans la figure 1



INDEX 751

— Gabriel, de Massongy, amodiataire des censes de Draillant et Massongy, 168, 364, 420 ; - son beau-frère, voir DUPAN, C.
— Nicolas (†), père d'Antoine, n., 541
Aubonne (CH, ct. Vaud, distr. Morges), *Aulbone*, 352, 665
auditeur, audicteur, auditheur, voir office
AUGNYEZ (D') [AUGNY (D')], Pierre, de Confignon, discret, anc. codimeur d'Onex, 555
Augnysse, voir *Ensisheim*
AUGSBURGER, AUSPURGUE, OUGSPURGER, Michael, de Berne, s^r, 679, trésorier, 1
Augvengne, voir *Auvergne*
AULBERT, voir AUBERT
Aulbone, voir *Aubonne*
aumône, au(l)(s)mone, voir charité
Autruche, Authruche, voir auberge
Auvergne (F, rég.), *Augvengne*, *Auvergnye*, 453, 580
avant-banc, voir banc
Avignon (F, dép. Vaucluse, ch.-l. arr.), *Avegnyon*, 189
avocat, voir métier
avoine, avoienne, avoyenne, voir céréale
AVONAY (D'), AVONEX (D'), voir LAVENAY (DE)
avoyer, avoyez, voir office
Avully (CH, ct. Genève), *Avulliez* ; le seigneur de -, voir SAINT-MICHEL (DE), F.
Ayex, voir *Gex*
AYGRE, Jérôme, 348 ; - sa femme, voir VOLAND, J.
AYMON, AYMOZ, AYMÉ, Pierre, 310, 422
Ayre, voir *Aire*

B

BACHELLER, fourrier des Grisons et d'Italie, 434, détenu, 434
bacstiouz, voir moulin
bacstre, bacstu, bacstyt, voir délit (agression)
bailli, bailliff, bailly, voir office
balance, levraulx, lyvraulx, 319
BALARD, Jean le jeune, s^r, 137, 543, 546, du CC (1543), 78, tuteur des enfants d'É. PÉCOLAT, 24, 28, auditeur des appels en première instance, 83, 92, auditeur du droit, 548, 562
Bâle, Basel (CH, ch.-l. ct. Bâle-Ville), *Balla*, *Balle*, *Basla*, *Ba(s)lez*, *Bas(s)le*, [autorités], 14, 21, 40, 45, 53p, 60, 87-88p, 94, 102, 117, 128, 129p, 132, 133, 135, 136, 138-139, 147, 150, 151p, 153, 172, 183, 184, 189, 192, 196, 197p, 200, 201, 204p, 252, 253, 258p, 260, 276, 277, 280, 282, 291, 292, 309p, 310, 316, 317p, 323, 330, 331, 345, 366, 389, 420, 427, 430, 432, 440, 447, 455, 456, 464p, 465, 468, 483, 492p, 495-496, 497p, 498p, 504, 507p, 510, 512, 513, 514, 519, 531n, 532, 534, 538, 539, 540, 543p, 557, 567, 587p, 588p, 602p, 605, 610, 611, 613, 618-619p, 621-622p, 624-625, 626, 627-628, 630, 639-640, 641-642p, 643, 662, 665, 666, 668-669, 679, 682-683, 695, 697, 700, 702, 703-704, 710, 712, 715, 716, 717, 722, 727 (voir HOLZACH, O. ; MEYER, B. ; OFFENBURG, C. ; RÜDIN, J. ; SCHÖLLI, B.) ; voir aussi emprunt, monnaie, traité
— appel, 6, 14, 147, 196
— auberge, - du Bœuf, 588
— imposition, - cens, 14, 611
— métier
- changeur, 129, 189, 197, 276, 292, 295, 317, 318, 420, 468, 641 (voir RÜDIN, J.)
- forgeron, 318 ; voir aussi armée (artillerie)
— office
- ambassadeur, 172, 189, 351, 382, 389p-390, 567, 666, 683 (voir CULLIER, P. ; HOLZACH, O. ; MEYER, B. ; MEYER, J. ; RÜDIN, J. ; SCHÖLLI, B.)
- arbitre, * surarbitre, 14, 53, 88, 147, 189, 196, 200, 252, 276, 316, 397, 618, 627, 630, 636, 639, 668-669

FIGURE 1 – Indexe extrait de "Index_1543.docx"

Nous avons d'abord une première section écrite avec en texte clair. Cette première section nous introduit le do-

cument 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.

2.2 Méthodologie

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

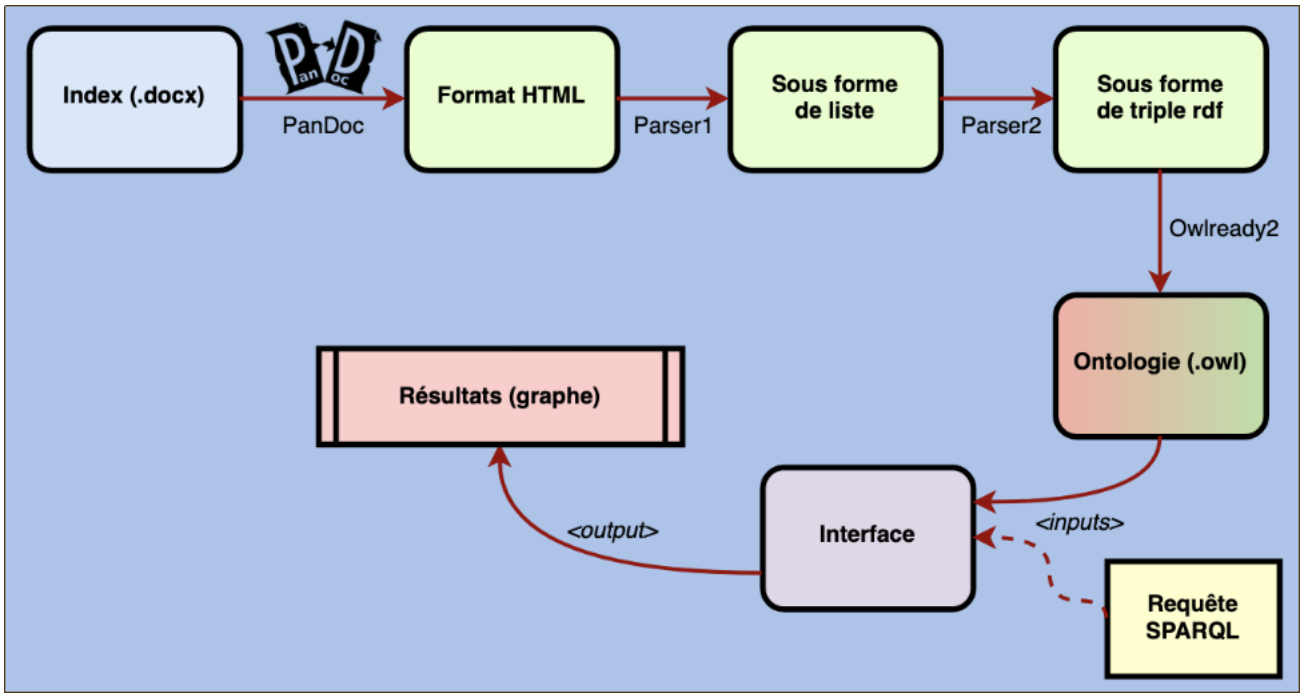


FIGURE 2 – Comme présenté dans ce flowchart, notre solution va extraire et formater les données présentes dans l'index à l'aide de quelques outils et quelques étapes.

2.3 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 des outils déjà existant.

2.4 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 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

2.5 Conversion des structures python en triplets

Dans cette nouvelle étape, les données structurées (sous forme de liste python) vont être traités 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'informations 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.

2.6 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 contentant 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 graphique qui va faire des requêtes de type SPARQL pour obtenir le contenu de ces données. Données qui vont être présentées sous forme de graphe ou de tableau.

3 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 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. De plus, chaque terme qui est utilisé pour les classes et n'est pas présent dans l'index aura un tiret-du-bas "_" avant le dit terme.

Hiérarchie des classes.

3.1 Classes liées aux personnes

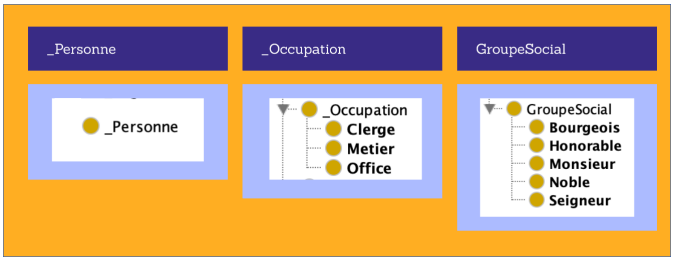


FIGURE 3 – classes liées aux personnes

Nous avons jugé pertinent de prendre les classes liées aux personnes. Nous avons créé les classe Personnes, Occupation et GroupeSocial. Cela permet une liberté aux historiens de pouvoir accéder à une quelconque information qui relaterait à des personnes mentionné dans les textes.

3.2 Classes liées aux localisations

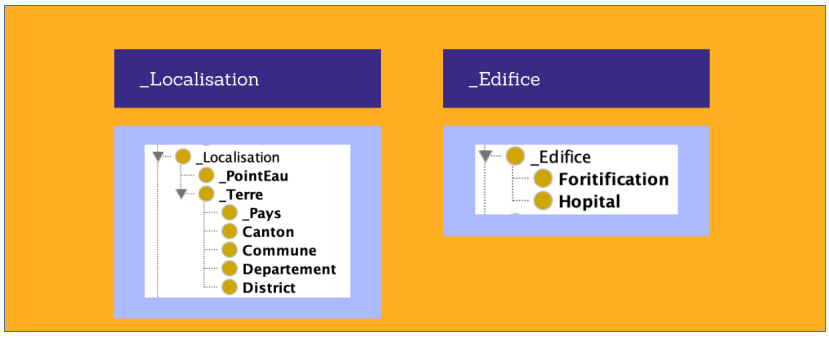


FIGURE 4 – classes liées aux localisations

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

3.3 Classe liée à l'état

Tout données lié à l'état se retrouverait dans cette classe. Une sous-classe `Finance` a été créé afin de pouvoir catégoriser séparément tout ce qui lie à l'économie/finance de l'époque. Statistiquement parlant, l'imposition et la monnaie été énormément mentionné dans l'indexe. De plus, plusieurs sous-classes dans ces catégories étaient présentes dans l'indexe, d'où la nécessité de créer cex deux sous-classe dans `Imposition`

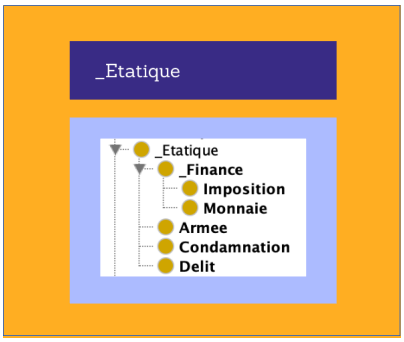


FIGURE 5 – classes liées à l'état

3.4 Classe lié à la page

Une des classes les plus importante car elle représente l'objectif majeur : elle relie l'ontologie et le numéro de la page de l'archive qui permettrait de guider l'historien et trouver l'information désirée :

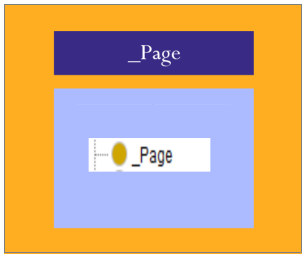


FIGURE 6 – Classe liée à la page

3.5 Ecrit

Si un historien voudrait trouver un quelconque écrit (procès verbale, registre, taxe, lettre, correspondances,...), la recherche se fera dans cette classe :

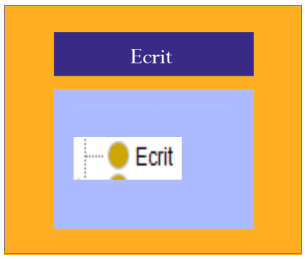


FIGURE 7 – Classe liée aux écrits

4 Algorithme de parsing

Dans cette partie, nous donnons un peu plus de détails concernant l'algorithme utilisé dans notre projet. **Le projet pourrait être récupéré plus tard**.

4.1 Fichiers de l'algorithme de parsing

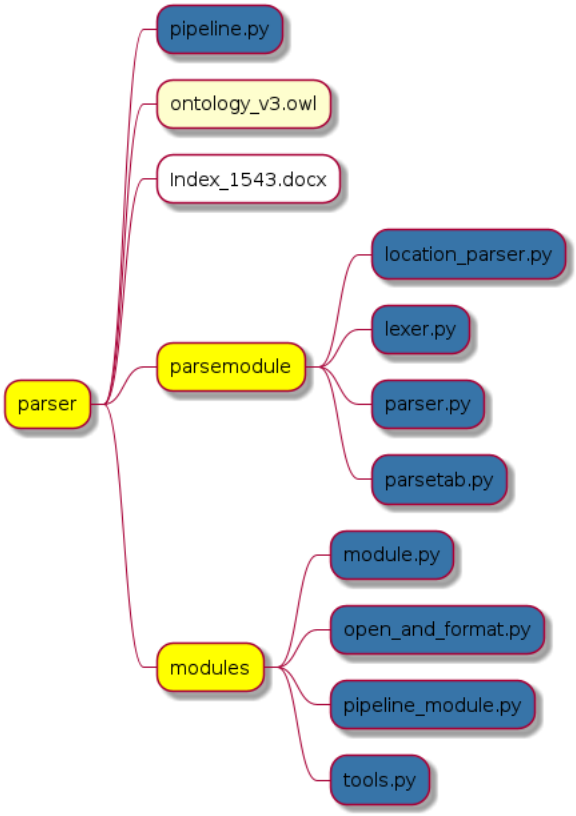


FIGURE 8 – Le code de l'algorithme de parsing est possède un fichier principal (pipeline.py) ainsi que deux module : un pour le parsing et un module pour les fonctions plus gnérales

L'algorithme de parsing se trouve dans le dossier parser. Il y a le fichier principal "pipeline.py" accompagné de deux modules qui disposent du code nécessaire au parsing. Le premier dossier "parsemodules" est fait pour contenir tout le code nécessaire au parsing de l'index. C'est là où peuvent être logés les futures parsers. Il y a aussi le dossier "modules qui contient les fonctions d'ordre plus général ou spécifique à l'exécution du programme.

Nous avons aussi les fichier d'index "Index_1543.docx" (qui est le fichier d'index par défaut) et le fichier "ontology_v3.owl" qui est notre fichier contenant nos classes. À la fin de l'exécution du parsing, quelques fichiers seront générés. Il y a notemment le fichier "final.owl" qui contiendra la combinaison de notre ontologie et des éléments parsés dans l'index. Il y a aussi le fichier intermédiaire "Index.html" qui sera créé et finalement le fichier "failed.html" qui contiendra toutes les entrées qui ont obtenues une erreur de parsing et qui pourront être traités ultérieurement.

4.2 Code

Le fichier de départ est pipeline.py. C'est lui qu'il faut lancer en premier. On peut mettre en paramètre le nom du fichier d'index qu'on veut, sinon, il choisira le fichier "Index_1543.docx" par défaut. Le fichier pipeline.py est équipé de 3 fonctions par défaut :

— `docx_to_html()`

— `html_to_triple()`

— `triple_to_owl()`

Ces fonctions ont des noms explicites pour suivre le processus de conversion entreprit.

4.3 docx_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 9 – La fonction html to triplet est la seconde étape de notre pipeline. Comme son nom l'indique, elle permet de convertir les données html en triplet.

La fonction docx_to_html() est la première action du pipeline et appelle seulement l'outil Pandoc pour convertir l'index Word en fichier html. Il est très simple dans sa configuration et n'a pas grand détail à présenter. Dans des travaux futures, on pourrait s'imaginer utiliser des outils plutôt internes à python pour faire la conversion, comme des modules téléchargeable. Car cette méthode demande que l'utilisateur aie Pandoc préalablement installé pour faire la conversion.

4.4 html_to_triple

```python
def html_to_triple():
    text = open("Index_1543.html", "r").readlines()      You, maintenant • Uncommitted changes

    # 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 destructuré.
    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 10 – La fonction html to triplet est la seconde étape de notre pipeline. Comme son nom l'indique, elle permet de convertir les données html en triplet.

Cette fonction est assez particulière, car elle se compose de deux parties importantes. Elle utilise d'une part un parseur pour créer des structure, et d'autre part un processus pour transformer ces structures en code. Cette fonction a été faite pour traiter un fichier d'index complet contenant aussi les textes explicatifs au début du document. L'index n'aura pas besoin d'un pré-traitement manuel, le processus se fera automatiquement.

La fonction va ignorer la première partie de l'index et commencer le parsing qu'à la première référence. Les référence apparaissent ligne par ligne. Si le parseur parvient à parser la référence, alors la référence transformées en structure sera ensuite transformé en triplet. Si le parseur ne parvient pas à parser la référence avec les règles de grammaire données, alors la référence est redirigé dans la fonction d'écriture des références qui ont échouées. Si la référence appartient à une liste (= se trouve au milieux des balise "blockquote" ou est une balise "blockquote") elle sera ignorée, sinon la référence sera écrite dans le fichier "failed.html" permettant une correction et un pré-traitement ultérieur de ces références. En effet, si elles s'y trouve à l'intérieur, c'est qu'il y a eu des erreurs humaines à l'intérieur.

Nous avons actuellement un parseur secondaire ("location_parser.py") qui agit au sein de la fonction de dépliage. D'autres outils de parsing et d'analyses peuvent y être ajoutés.

4.5 triple_to_owl

```python
def triple_to_owl(tab):
    """
    une fonction qui va transformer le contenu html_to_triple
    en structure dans un premier temps puis crée des triplets
    """
    onto = get_ontology_from_file("ontology_v3.owl")
    define_properties(onto)
    create_instance_and_relation(onto, tab)
    onto.save(file="final.owl", format="rdfxml")
```

FIGURE 11 – La fonction triple to owl est responsable de transformer les triplets obtenu en individual et relation en définissant les propriétés admises

La fonction triple to owl fait trois actions simples. Elle importe l'ontologie préalablement créée, elle crée les propriétés et fini par créer les liens. Cette fonction s'appuie essentiellement sur le module owlready2.

L'importation est tout aussi simple. La fonction de création de propriété crées les propriétés désirées. C'est ici que sont définies les propriétés ainsi que leur domain et leur range. C'est aussi dans cette fonction que seront définis les nouvelles propriétés en fonction des triplet obtenus. Il serait intéressant de créer un système qui permet de déterminer automatiquement le domain et le range des propriétés en observant seulement les triplets donnés en entrée. Pour finir, les individuals et les relations sont créées à l'aide des triplets. Les triplet qui ont un lien qui n'a pas été préalablement défini dans la fonction de création de propriétés ne seront pas admis (évitant d'introduire des relations qui ne font pas sens).

5 Résultats

À partir des modules créés, nous obtenons donc les résultats du processus. À partir du mot ***Passeiry ...*** dans la figure 12, nous obtenons après un premier parsing [4.3] des termes sous forme HTML, avec une première classification de chaque terme. Par exemple, dans la figure 13, on voit la séparation entre *Passeiry(...)*, *Passeyrier* et puis le numéro de la page.

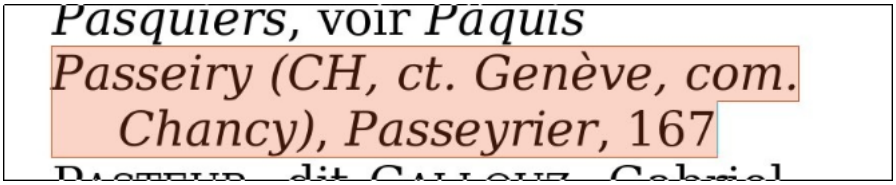


FIGURE 12 – Exemple d'indice

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

FIGURE 13 – HTML

À partir du module html-to-triple [4.4], nous obtenons la structure ci-dessous. À partir de cette structure dans la figure 14, on obtient des listes avec des triplettes.

```
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 14 – triplet

Proche du but, il nous suffit juste de transformer cela dans des triplets RDF grâce au module *triple-to-owl* [4.5]. À partir de là, nous peuplons le fichier OWL principale avec les classes pré-définis avec les individuals appropriés.

```
<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 15 – OWL

6 Interface

Afin de pouvoir accéder au ficher owl et pouvoir faire des requêtes, nous avons décidé de créer un web-app pour pour faire des requêtes. Dans le cas où on déploierait le site sur un serveur, cela répondrait aux besoins des historiens : de pouvoir accéder à l'ontologie directement depuis leur machine personnel.

Nous pouvons visualiser l'interface créée ci-dessous.

FIGURE 16 – interface du web-app

En cliquant sur le bouton "*submit query*", on soumet la requête et cela renvoie donc les résultats sous forme de tableau comme ci-dessous :

Nous obtenons les résultat sous la forme suivante :

FIGURE 17 – Resultats de la requête pour trouver les 20 premières Localisation mentionnés

La création du web-app rendrait ainsi une accessibilité à tout le monde, en tout temps. De plus, si ce projet est repris, il ne sera pas compliqué d'intégrer un wrapper qui évite de faire des requêtes SPARQL. En effet, ayant utilisé la librairie Flask de Python, il est simple de pouvoir créer un "route " qui emmènerait à une nouvelle page pour le wrapper. La génération des résultats resterait la même.

6.1 Visualisation graphique

Grâce à la librarie python `Network`, il a été également possible de générer un graph pour une meilleure visualisation des données. Un exemple est donné ci-dessous :

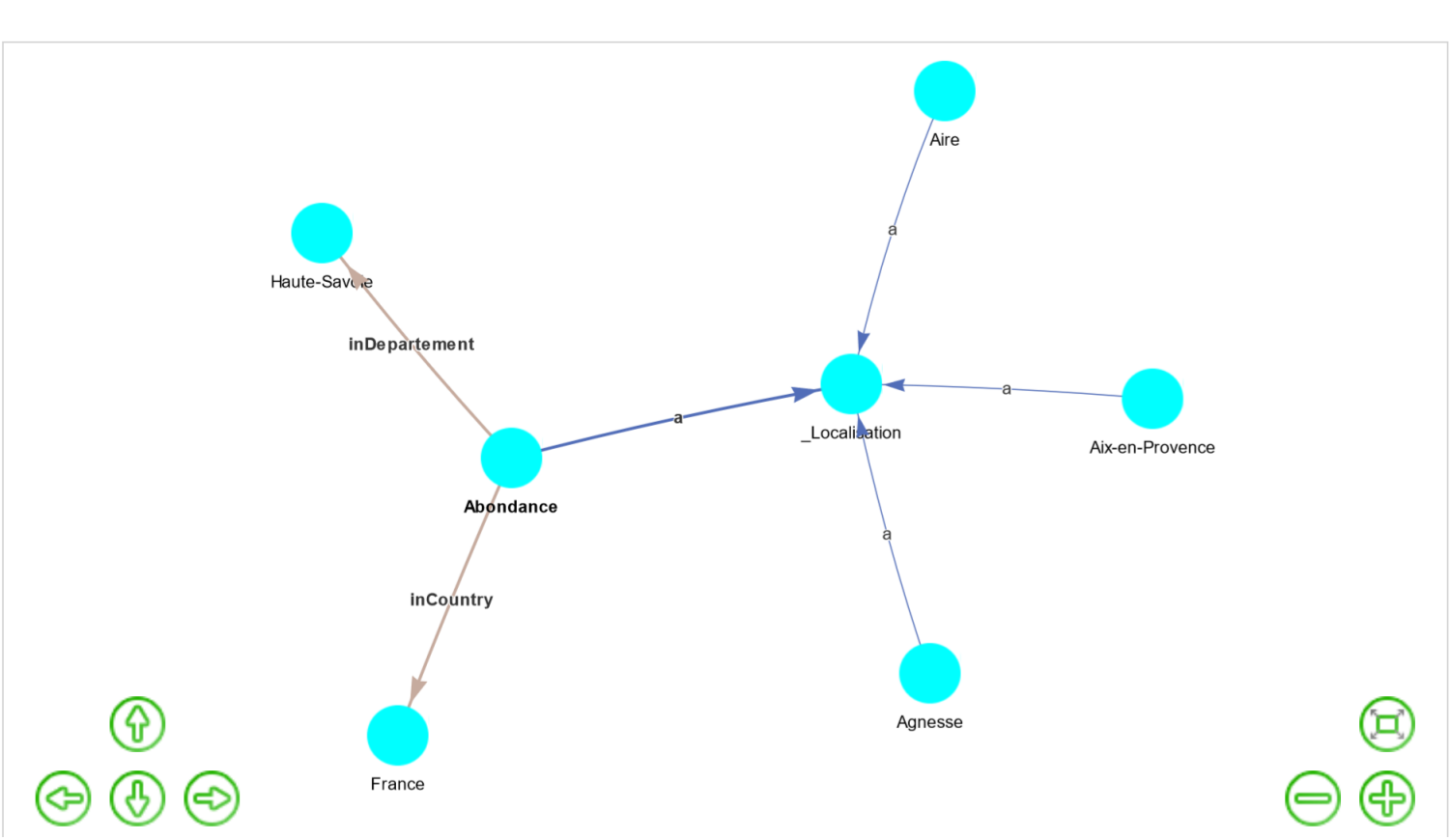


FIGURE 18 – Visualisation graphique d'une requête. La requête demande tout simplement les premières localisations.

Ètant donné que générer un graph requiert des requêtes SPARQL (ajout d'une line *" ?x ?y ?z"* à la fin, ainsi que d'autre modifications), il était préférable de le laisser de côté pour l'instant, et être dans l'optique qu'il soit reprit dans un projet ultérieure. Le module python pour générer le graphe est néanmoins bien documenté et facilement ré-utilisable et avec l'implémentation des paramètres pour avoir un graphe lisible (comme nous pouvons le voir dans la figure ci-dessus) .

7 Conclusion

Au cours de ce projet, nous avons réussit à créer une ontologie avec le logiciel qui domine ce domaine, *Protégé*. Parallèlement, avec l'indexe nous avons extrait les données principales et transformer ces données en triples RDF, qui s'intègrent en tant qu'*individuals* à l'ontologie créé sur *Protégé*. Á partir de ce fichier OWL, nous avons créé un web-app qui permettrait de d'accéder et faire des recherches dessus pour tout utilisateur ayant accès à internet.

Au cours de ce projet, nous avons rencontrés les difficultés suivantes. Ces difficultés peuvent aider toute personne qui reprend le projet à les appréhender et mieux gérer dès le départ.

- Complications à comprendre l'index : notations, structure, etc.
- Parsing... :
  - Erreurs humaines dans les indexes : 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

7.1 Amélioration et la suite

Ce projet a finalement été nettoyé, avec la plupart des fonctions documenté afin que ce projet puisse être reprit et amélioré. Plusieurs améliorations peuvent être apportés tels que l'ajout d'un wrapper au web-app qui éviterait aux utilisateurs de devoir écrire du SPARQL pour accéder à des données. Pour le parser, la librairie python *SpaCy*, qui utilise du NLP essentiellement et d'autre outils du ML qui pourrait améliorer le parser et ajouter des fonctionnalités en plus. Finalement, étant donné que nous avons travaillé sur une fraction des indexes, une étude plus élargies de l'indexe peut être effectué et cela permettrait de compléter l'ontologie et donc ajouter plus de classes .

Références

[1] František Babič, Vladimír Bureš, Pavel Čech, Martina Husáková, Peter Mikulecký, Karel Mls, Tomáš Nacházel, Daniela Ponce, Kamila Štekerová, Ioanna Triantafyllou, Petr Tučník, and Marek Zanker. Review of tools for semantics extraction : Application in tsunami research domain. *Information*, 13(1), 2022.

[2] Zhiyu Wang, Zhiping Song, Guang Yu, and Xiaoyu Wang. An ontology for chinese government archives knowledge representation and reasoning. 9 :130199–130211, 2021.