Master 1 Informatique
Université des Sciences et Technologies
Lille I

Projet individuel

"PJI n°54 : Mais que font les députés ? Une sociologie informatique du travail parlementaire"

Encadrant universitaire : Samuel Hym
Encadrant : Etienne Ollion
Étudiant : Ouamar Sais

Table des matières

Remerciements

Merci à Samuel Hym et à Etienne Ollion pour leur accessibilité, pour leur suivi, leur réponses à mes questions, et pour avoir fourni ce sujet très intéréssant.

Introduction : Description générale du projet

Projet personnel

Le PJI est un projet personnel qui se déroule au second semestre de la première année de Master informatique à Lille 1. L'étudiant choisi un sujet puis le développe sur l'ensemble du semestre. Tout le code du projet peut être trouvé à cette adresse : `https://github.com/ouams/PJI_project`

Ce rapport récapitulera le travail effectué sur ce projet ainsi que ses résultats. Dans un premier temps, il s'agira d'une description concrète du projet choisit, puis une présentation des outils qui ont été utilisés pour le développement de celui-ci ainsi que la raison pour laquelle il ont été choisis. Ensuite, nous aborderons la manière dont les données nécessaires au projet ont été récupérées afin de pouvoir continuer sur leur nettoyage et analyse. Enfin dans une dernière partie, il s'agira de la façon dont les données récupérées par l'analyse ont été stockées.

Description du sujet

Le projet est le numéro 54 dans la liste des PJI, celui-ci se nomme "Mais que font les députés ? Une sociologie informatique du travail parlementaire" et vise à exploiter les comptes rendus intégraux de l'assemblée nationale afin mieux connaître son fonctionnement. Il est encadré par Samuel Hym, enseignant chercheur au laboratoire CRYStAL de Lille 1 ainsi que par Etienne Ollion, chercheur CNRS au laboratoire SAGE de Strasbourg.

Plus particulièrement, durant les séances publiques, les discussions donnent parfois lieu à des échanges plus ou moins vifs entre parlementaires, les comptes rendus font état de certaines de ces manifestations. Le but ici est de réussir à récupérer les indications de ces "humeurs" (applaudissement, protestation ...).

Le problème qui se pose dans la démarche d'analyse de ces comptes rendus est que ceux-ci, de par leur lisibilité moyenne, ne facilitent pas la récupération de ces informations de manière humaine, il faudrait en effet lire tout ces comptes rendus, compter et noter les informations voulus, il est clair que ce n'est pas une tâche agréable. C'est pourquoi le projet demande la création d'un outil qui permettra d'analyser automatiquement chaque compte rendu de séance et d'en extraire les informations voulues.

1 Description générale de l'architecture du programme

L'application est séparée en deux entités, l'une est écrite en Scala est concerne la récupération des comptes-rendus intégraux nécessaire à la réalisation de la suite du projet. L'autre entités est écrite en Java et concerne l'analyse et l'extraction des informations demandées.

L'application est organisée comme suit :

```
|_PJI_Project/
    |_src/
    |    |_ csv/
    |    |   |_CSVHandler.java
    |    |_analyse/
    |    |   |_TalkAnalyser.java
    |    |   |_Main.java
    |    |_cleaning/
    |    |   |_Cleaner.java
    |    |_util/
    |       |_DateHandler.java
    |_scala/
    |    |_src/
    |    |_main
    |    |_scala/
    |    |_download/
    |       |_URLManager.scala
    |       |_HTMLDownloader.scala
    |_cri/
    |   |_X/
    |     |_\{Année\}
    |           |_\{Type\}
    |               |_fichier
    |                   |_\{Fichier.html\}
    |_lib/
    |_data/
    |    |_X/
    |     |_\{Année\}
    |           |_\{Type\}
```

```
|                |_fichier
|                    |_infoSeance
|                        |_\{Fichier.csv\}
```

2 Outils utilisés

Pour la réalisation du projet, aucun impératif concernant les technologies utilisés n'ont été donnés,

2.1 Java/Scala

Java et Scala sont les deux languages de programmation qui ont été utilisés pour mener à bien le projet, ils ont chacun servis dans deux parties distinctes.

Scala a été utilisé pour effectuer la récupération des fichiers nécessaire pour la suite. C'est un langage qui associe les deux paradigmes de programmation objet et fonctionnel quand que ceux-ci sont rarement combinés. Il a aussi l'avantage d'avoir un typage statique et surtout d'être compilable sur la JVM ce qui permet de pouvoir utiliser les bibliothèques Java ou même de mixer du code Java avec du code Scala. Il est utilisé avec l'outil open source SBT qui facilite la construction d'application scala.

Le language Java est utilisé pour la seconde partie du projet, l'analyse. C'est le language objet que je maîtrise le mieux et avec lequel je me sens le plus à l'aise. Il a donc été plus facile pour moi de manipuler les librairies utilisées dans ce langage.

2.2 HTMLCleaner/Jsoup/OpenCSV/juniversalchardet

Plusieurs librairies Java ont été nécessaires dans la réalisation du projet. A noter que les librairies utilisées sont toutes open source.
La première librairie à avoir été utilisée est juniversalchardet, celle-ci est intervenu lors de la récupération des fichiers HTML, ceux-ci ayant parfois des encodage différent, cet outils a permis de récupérer l'encodage de chaque fichier de façon simple.

La seconde librairie utilisée est HTMLCleaner, elle a été utilisée uniquement pour nettoyer le code HTML d'un point de vue "visuel" pour pouvoir se rendre compte de l'organisation de celui-ci.

Jsoup est la librairie qui a le plus servi dans le projet, elle permet d'extraire et de manipuler des données sur du code HTML. Cette bibliothèque offre une API simple d'utilisation et très bien documentée ce qui rend son utilisation aisée. Seule la partie concernant l'extraction de donnée a été utilisée mais il faut savoir qu'elle permet également de créer/modifier directement du code HTML.

Enfin, la librairie OpenCSV a servi à la manipulation des fichiers csv sont stockés les informations récupérées. C'est une librairie simple dont les deux méthodes de sauvegarde ont été utilisées.

3 Récupération des données

Avant d'envisager une quelconque analyse des données, il a tout d'abord fallu récupérer les fichiers nécessaires sur le site de l'assemblée nationale.

3.1 Organisation des fichier sur le site

Sur le site de l'assemblée Nationale, les comptes rendus intégraux sont organisés par législature, par année, et par type de séance. Chaque page d'index est de la forme *http ://www.assemblee-nationale.fr/X/debats/index.asp* où X est le numéro de la législature. Sur cette page sont répertoriés les liens vers la liste des séances par année.
Par exemple la hiérarchisation est effectuée comme suis pour la législature 14 :

```
14/debats/index.asp
    14/cri/2014-2015/
        14/cri/2014-2015/20150225.asp
        .
        .
    14/cri/2013-2014/
        14/cri/2013-2014/20140255.asp
        .
        .
    14/cri/2013-2014-extra/
        14/cri/2013-2014-extra/20141027.asp
        .
        .
```

Pour le projet, on commence par la législature actuelle puis on essaye de remonter au maximum dans le temps. En réalité le programme fonctionne jusqu'à la moitié de la douzième législature.

3.2 Choix du format

Deux formats de fichiers on été envisagés, le premier était le pdf en téléchargeant toutes les versions pdf des journaux officiels puis d'utiliser la librairie Java PDFBox afin de pouvoir les manipuler. Seulement après réflexion et lecture du code html des pages de compte rendu, il a finalement

été jugé qu'il était plus facile de travailler avec ce code html en utilisant JSoup. En effet, la retranscription écrite d'une séance en html suit un schéma clair et répétitif en ce qui concerne les interventions, du moins pour les législatures les plus récentes.

3.3 Téléchargement des données

Le programme qui effectue cette récupération est écrit en Scala, c'est une modification du code que Quentin à utilisé pour les PDF, il comporte deux fichiers : *URLManager.scala* et *HTMLDownloader.scala*. Les modifications apportée au code concerne les liens pour *URLManager.scala* et la récupération de l'encoding et la manière de télécharger les fichiers dans le fichier *HTMLDownloader.scala*.

Le premier fichier contient plusieurs fonctions qui vont permettre de stocker dans une variable toutes les URL pointant vers une page html contenant le compte-rendu d'une séance pour un nombre de législature fixé.

Sur le site de l'assemblée nationale, on commence à partir de l'URL *http ://www.assemblee-nationale.fr/X/debats/index.asp* où X est le numéro de la législature. À partir de cette page, on récupère tout d'abord les liens qui pointent vers la liste des compte-rendus d'une année (ex :*http ://www.assemblee-nationale.fr/14/cri/2013-2014/* pour les comptes-rendus de la 14e législature et pour l'année 2013-2014 Figure 3.1).

FIGURE 3.1 – urlSession 2013-2014

Une fois que toutes ces URL sont récupérées, le même procédé est appliqué avec les nouveaux liens afin de récupérer les liens des pages contenant les comptes rendus html (ex : *http ://www.assemblee-nationale.fr/14/cri/2013-2014/20140255.asp* 3.2).

Une fois cette première partie effectuée, une des deux fonctions *downloadAll* et *downloadGroupNb* du fichier *HTMLDownloader.scala* vont se charger respectivement de télécharger tous les fichiers html correspondant aux URL précédemment récupérées ou alors de télécharger un nombre de paquet de 100 donné. Le classement se fait par législature, années, et type de séance.

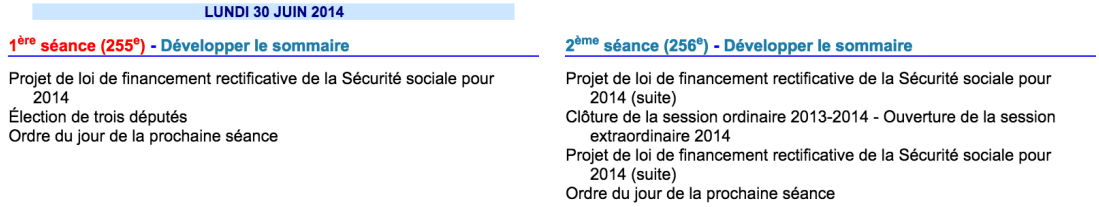

FIGURE 3.2 – urlCRI 2013-2014

3.3.1 Difficultées

Une grosse difficulté s'est manifestée après avoir récupéré les comptes rendus de plusieurs législatures de façon basique en téléchargeant le texte HTML directement. En effet, il a été possible de s'apercevoir que l'encodage de certains des fichiers était en UTF-8 et d'autres en iso-8859-1, il a fallu trouver une solution pour éviter les caractères étranges.

J'ai tout d'abord pensé que c'était uniquement un problème de lecture lors de l'analyse, j'ai donc réalisé un premier script bash qui lit dans chaque fichier la valeur de l'attribut "charset" de la la balise meta concernant l'encodage, et qui ajoute celui ci à la fin du nom du fichier afin de pouvoir y accéder et le préciser à l'InputStreamReader qui lira le fichier. Seulement il s'est avéré que le problème ne venait pas de la lecture uniquement mais débutait lors du téléchargement des fichiers qui étaient tous lu en UTF-8 par défaut. C'est ici que la plus grosse modification du code Scala à eu lieu.

J'ai donc trouvé la librairie juniversalchardet, qui permet d'utiliser la librairie détectrice d'encodage de Mozilla en Java. Le principe est simple, l'objet UniversalDetector lit quelques octets de la source, et "devine" l'encodage de celle-ci. J'ai donc ajouté une fonction *getEncoding* (Fig 3.1)qui se charger d'utiliser cette librairie pour fournir l'encodage de chaque fichier avant leur téléchagement.

Listing 3.1 – Fonction getEncoding()

```
private def getEncoding(url: URL): String= {
    val buf = new Array[Byte](1024)
    val detector = new UniversalDetector(null)
    val inStream =  url.openStream()
    var nread = inStream.read(buf)

    while (nread > 0 && !detector.isDone()) {
      detector.handleData(buf, 0, nread);
      nread = inStream.read(buf)
    }
    inStream.close(); detector.dataEnd();

    val encoding = detector.getDetectedCharset();
    if (encoding != null) {
```

```
      encoding
      } else{
        "UTF-8"
      }
    }
```

Le deuxième problème qui s'est manifesté est qu'au départ l'inputStreamReader n'était pas capable de lire tout les fichier d'un coup, même si la taille du buffer correspondait bien à la taille qu'il fallait lire. On se retrouver avec des fichiers incomplets. C'est pourquoi le téléchargement utilise un Scanner qui lit chaque lignes une par une avec *nextLine()* 4.1, cette méthode est bien sûr plus lente mais fonctionne et évite de consommer trop de mémoire.

Listing 3.2 – Fonction download()

```
val sc = new Scanner(inStream, encoding)
        val out =  new BufferedWriter(
        new OutputStreamWriter(new FileOutputStream(path + name),
        encoding))

        while(sc.hasNextLine()){
          out.write(sc.nextLine())
        }
        inStream.close()
       sc.close
        out.close()
```

4 Analyse et extraction des données

4.1 Agencement des fichiers HTML

Après une lecture de plusieurs fichiers compte rendu, il est possible de se rendre compte de la façon dont ceux-ci sont agencés. En effet à première vus, les fichiers peuvent paraître brouillon mais, grâce à la ré-indentation faite à l'aide de HTMLCleaner l'analyse visuelle a pu être facilitée et il a été possible de se rendre compte que les fichiers suivaient en fait un schéma répétitif. Une fois ce schéma detecté, il a été plus ou moins aisé de localiser les informations utiles pour le projet. Tout l'intérêt du html est de pouvoir avoir un document donc l'organisation du fond peut être faite de façon minutieuse, c'est de cela que nous voulons tirer profit ici.

4.2 Différences notables

Bien que les fichiers suivent un schéma similaire, ils ne sont pas tous identiques pour autant. En effet les plus récents sont les plus "propres" et également les plus claires : le code html utilise des class utiles dans les balises qui indiquent clairement le contenu ("intervention", "somprésidence"...), les intervenants ont un lien vers une fiche d'identification et chaque paragraphe correspond à l'intervention d'une personne différente. (ex Figure 4.1)

FIGURE 4.1 – 2015

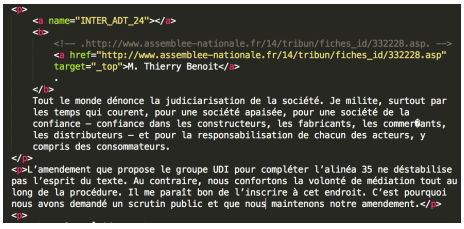

FIGURE 4.2 – 2013

Cependant dès que l'on commence à remonter un peu dans le temps, la présentation diffère et devient de moins en moins claire, les class disparaissent, le schéma n'est parfois pas respecté dans un même fichier ou d'autres modifications (ex Figure 4.2). C'est pourquoi il a fallut faire en sorte que le code du programme s'adapte en fonction du fichier à traiter.

4.3 Extraction des informations

Le processus d'extraction d'informations se fait via la classe *TalkAnalyser*, c'est elle qui s'occupe de filtrer et rechercher les informations nécéssaires dans les fichiers puis de les sauvegarder. Une fonction main parcour l'arborescence des fichiers html et appelle la fonction *process* de la classe *TalkAnalyser* pour chaque fichier, c'est la méthode qui se charge de récupérer toutes les informations et se déroule comme suit :

— lit le fichier : récupère le contenu html du fichier sous forme d'une chaîne de caractère.

— récupère les données : appelle la fonction *getData()* qui va se charger de récupérer toutes les informations nécessaires de la bonne manière en fonction du fichier.

— sauvegarde : enregistre les données récupérées dans des csv.

De manière plus détaillée, la fonction *getData()* utilise la librairie Java Jsoup afin de filtrer le code html et extraire les informations. Il faut tout d'abord parser le code html lu précédemment, cela retourne un objet Document qui est une représentation hiérarchique du code html sur lequel Jsoup est capable d'effectuer diverses recherches/filtres.

Ce filtrage s'effectue en utilisant la syntaxe CSS en temps que "requête", tous les éléments qui correspondront à la spécification CSS seront retournés sous forme d'*Elements* . Par exemple pour récupérer toutes les interventions d'un fichier récent contenu dans les paragraphes des balise div de classe "intervention", il suffit d'utiliser la fonction select sur l'objet Document comme suit : *doc.select("div.intervention > p")*, il est ensuite possible d'itérer sur chaque paragraphe récupéré afin de les traiter séparément et d'en extraire intervenant, intervention, date et réaction (Voir exemple Annexe A) si présentes en utilisant les méthodes suivantes fournies par Jsoup () :

— hasAttr(*attr*) : Fonction qui retourne vrai si l'element contient l'attribut spécifié, faux sinon.

— hasText() : Retourne vrai si l'Element sur lequel la méthode est appellée contient du texte.

— select() : Selectionne le contenu de l'attribut passé en paramètre.

— text() : retourne une String contenant le contenu de l'attribut/Element sur lequel la méthode est appellée.

Pour les réactions, un fichier Reac.txt contient les mots clés des didascalies contenant les réactions. Une fois les didascalies récupérées, on reconnaît une réaction ou non si celle-ci contient un des mots clés. Il est bien sûr possible de modifier ce fichier.

La date de la séance quant à elle est écrite dans un format texte (“jj mois année"), pour la convertir en format jj/mm/aa, on utilise un csv qui associe à chaque mois à sa valeur numérique (ex : janvier avec 01), il est ensuite facile de remplacer le format de la date format texte vers un format jj/mm/aa grâce à une manipulation de replace()/replaceAll(). Le csv n'est pas indispensable mais il intervient dans un soucis de réduction de code pour DateHandler(Listing 4.1).

Listing 4.1 – Fonction download()

```
public static String dateConverter(String date) {
                final List<String[]> listMois =
                CSVHandler.read("Mois.csv", ',', "UTF-8");
```

```
            final String mois = date.split("␣")[2];

            for (final String[] strings : listMois) {
                if (mois.equals(strings[0])) {
                    date = date.replace(mois, strings[1]);
                }
            }

            date = date.replaceAll("␣", "/");
            return date.substring(date.indexOf("/") + 1);
        }
```

5 Stockage et format des données

5.1 Format de Stockage

Le format choisis pour l'enregistrement des données est le format csv, en effet c'est un format simple à enregistrer et à manipuler par la suite, il existe de nombreuse bibliothèque qui permettent de gérer les csv.

5.2 Forme des données

Les informations dont nous avions besoin étaient :

— Le nom du/de la président(e)

— la date de la séance

— toutes les interventions par intervenant

— les réactions liées à ces interventions (si il y en a)

— le nombre de mots par intervention

Pour ce faire, le csv se construit comme suit :
"Présidence de ...

Intervenant1	Intervention1	NbMots	Reaction	Date

..."

Voir l'exemple ci dessous :

Présidence de Mme Catherine Vautrin				
Mme Catherine Vautrin	L'ordre du jour appelle la suite de la discussion, en no	135		19/05/2015
Mme Catherine Vautrin	La parole est à M. André Chassaigne.	6		19/05/2015
M. André Chassaigne	Madame la présidente, monsieur le secrétaire d'État cl	609		19/05/2015
M. Jean-Claude Buisine	Exact !	1		19/05/2015
M. André Chassaigne	La température moyenne globale à la surface de la pla	989	« Ah ! » sur les bancs du gi	19/05/2015
M. Julien Aubert	Cliché !	1		19/05/2015
M. André Chassaigne	De tels propos ont été entendus au cours des débats a	48		19/05/2015
M. Julien Aubert	Pas à Tricastin !	3		19/05/2015
M. André Chassaigne	Si nous ne mettons pas en cause votre volonté de fair	30		19/05/2015

FIGURE 5.1 – Extrait d'un CSV

6 Mode d'emploi

Le projet se compose de deux parties distinctes, celle qui concerne la récupération des données et celle qui concerne l'analyse et l'extraction de celle-ci.

Pour récupérer les fichiers html contenant les comptes rendus des séances il faut compiler et executer le code Scala contenu dans le dossier *scala* à la racine du projet, pour ce faire taper les commandes suivantes :

```
$ sbt
> compile
> console
scala> HTMLDownloader.downloadAll
```

ou

```
scala> HTMLDownloader.downloadGroup X (paquet de 100 numero      X)
```

Une fois ces commandes éxécutées, un nouveau dossier *cri* est crée contenant tout les comptes rendus HTML classés.

La seconde étape concerne la partie du projet codée en Java, celle qui va réaliser l'extraction des données. Pour l'effectuer il suffit de lancer le jar exécutable à la racine du projet comme suit :

```
$ java -jar PJI.jar
```

Un dossier *data* est créé, celui-ci contient toutes les données extraites classées par législature, année, type de séance et par séance. (ex : *data/13/2010-2011/ordinaire/fichiers/infoSeance/20110220.csv*)

Conclusion

Lors de ce projet, j'ai tout d'abord eu l'occasion de mettre mes compétences en informatique au service d'un domaine qui n'a à priori aucun lien, celui de la Sociologie. Cela à permis de changer des projets habituels moins originaux, j'ai aussi pu lire quelques compte-rendu de séance de l'assemblée nationale très intéressant. D'un point de vue plus technique, j'ai pu apprendre à récupérer fouiller un site internet afin d'y récupérer des documents.
Ensuite j'ai également appris à me servir de plusieurs librairie Java, en particulier JSoup que je ne connaissais pas et qui se révèle être très puissante et intéressante, il est probable que je la réutilise dans d'autre contexte. En résumé, je suis entièrement satisfait du sujet proposé et j'ai pris plaisir à travailler dessus, en espèrent qu'il conviendra à l'utilisation qui en sera faite.

Bibliographie

— `http://www.assemblee-nationale.fr/` (*Site web*)
— `http://jsoup.org/cookbook/` (*Site web*)
— `https://code.google.com/p/juniversalchardet/` (*Site web*)
— `http://opencsv.sourceforge.net/` (*Site web*)

Annexes

Listing 1 – Methode récupération de données

```
private void processDivIntervention(final Elements
listParagraphe) {
// Parcours des paragraphes contenant les interventions
        for (final Element paragraphe : listParagraphe) {
                final String rowData[] = new String[5];

// Si le nom d'un intervenant n'est pas cite avec un lien
                if (!paragraphe.hasAttr("b")) {
                        rowData[0] = paragraphe.select("b").text();
                        // System.out.println(intervenant);
                        paragraphe.select("b").remove();
                } else {
                        rowData[0] =paragraphe.
                        select("a[hrefˆ=/tribun/fiches_id/]").text();
                        paragraphe.select("a").remove();
                }
// Si l'intervenant recuperee est le pr sident/pr sidente
                if (rowData[0].contains("pr sident")
                || rowData[0].contains("pr sidente")) {
         rowData[0] = this.president;
                }

        // Traitement des didascalies
                this.computeDidascalies(rowData,
                paragraphe.select("i"));
                paragraphe.select("i").remove();

// Recuperation de l'intervention contenu dans ce paragraphe
                if (!paragraphe.text().isEmpty()) {
                        rowData[1] = ""
                        + paragraphe.text()
                    .substring(paragraphe.text().indexOf(".") + 1)
                .trim();
```

```
                        rowData[2] = "" +
                        this.countUtilWords(rowData[1]);
// Ajout de la date puis des infos dans les listes
                        rowData[4] = this.date;
                        this.dataSeance.add(rowData);
                    }
                }
}
```