Nalae LEE

Master TAL-IL

RAPPORT du STAGE

effectué du 17 Février 2021 au 16 aout 2021

à Viasema

(8 Passage de brûlon, 75012 Paris)

Refactoring de grammaires Jape (Gate)

dans le domaine juridique

Sous la direction de Mme. Taravella

Soutenu le 25/06/2021 à l'UFR Phillia Université Paris Ouest Nanterre La Défense 200 Avenue de la République 92001 Nanterre cedex

Année Universitaire 2020 - 2021

Résumé :

Le stage se déroule à Viasema, qui est une entreprise spécialisée dans l'ingénierie de l'intelligence artificielle. Le sujet du stage est « Refactoring de grammaires Jape (Gate) dans le domaine juridique ».

Remerciements :

Mes premiers remerciements vont au pôle NLP, Mathilde, Fadila, Léa, Robin. Qui m'ont guidée tout au long de ce stage. J'ai beaucoup appris à vos côtés.

Et je remercie particulièrement Hugh de m'avoir chaleureusement accueilli à Viasema et Jamal pour ses conseils précieux.

TABLE DES MATIERES

INTRODUCTION

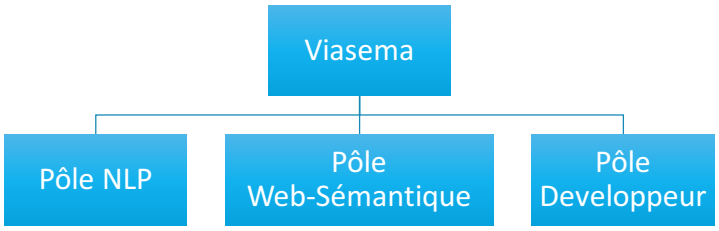
Le projet du stage s'est déroulé à Viasema, une entreprise spécialisée dans l'ingénierie de l'intelligence artificielle. Le sujet du projet est le Refactoring des grammaires Jape dans le domaine juridique. Des grammaires Jape sont des ressources de traitement automatique des langues (TAL) que Viasema développe en vue de l'analyse du texte. Nous avons voulu décomposer et reconstruire ces ressources afin d'en faciliter l'usage et d'améliorer finalement les performances du TAL. Cela était un bon sujet pour comprendre profondément des ressources dans un projet et le processus nous a permis d'avoir une perspective critique et vaste du domaine professionnel. Le projet a commencé le 15 Février et il est prévu de l'achever le 16 Août. Dans ce rapport, nous allons présenter le projet et montrer l'état d'avancement et la perspective d'évolution.

ENVIRONNEMENT DU TRAVAIL

PRESENTATION DE L'ENTREPRISE : VIASEMA

Viasema est une entreprise spécialisée dans l'ingénierie de l'intelligence artificielle. Viasema se voit confier par de grandes sociétés l'analyse des données internes. Andromeda, moteur de l'intelligence artificielle, est le service principal que Viasema offre aux plus de 500 clients[1].

Viasema est divisé en trois pôles : Développeur, Web-sémantique et NLP.



[1] https://www.viasema.com

Le pôle NLP se concentre sur le traitement de corpus brut et trie les informations nécessaires. Le schéma du web sémantique est basé sur l'analyse de NLP et il est déployé afin d'automatiser l'ajout des nouvelles informations. Pour cela, le pôle NLP collabore étroitement avec le pôle web-sémantique.

PRESENTATION DU POLE NLP

Le Pôle NLP de Viasema se constitue de TAListes et de linguistes de corpus. Il effectue chaque projet en trois étapes :

1) Pré-étude
2) Mise en oeuvre
3) Amélioration continue

La pré-étude sur des données brutes est a pour but de comprendre et relever des besoins des projets. En analysant les données, il faut cibler des contenus qui seront davantage exploités. Le cas échéant, une typologie des contenus est établies. Pour cela, l'analyse statistique aussi est une étape importante pour rendre compte des tendances des contenus. Le logiciel textométrique, TXM est utilisé au cours de cette phase.

Après la pré-étude, s'ensuit l'étape de mise en œuvre. Lorsque le besoin a été exprimé, il faut développer les outils qui permettront de répondre à la demande. Du côté du pôle NLP, cela passe souvent par l'élaboration d'outils et de

ressources linguistiques pour permettre l'extraction d'informations. Sachant que le besoin peut évoluer à tout moment, il faut aussi tenir compte de la possibilité de flexibilité des ressources et outils.

Les outils développés par le pôle NLP s'intègrent dans une chaine de traitement complète qui implique les trois pôles – NLP, web-sémantique et développeur – et qui va de la récupération des contenus jusqu'à la modélisation de leur valorisation. Étant donné cela, il est important de tenir compte de l'interopérabilité des outils et d'assurer une bonne communication entre les pôles.

Enfin dernière étape, l'amélioration continue, qui arrive lorsque que le projet est livré en production. Cela consiste au réajustement de petits bugs ou bien la prise en compte de nouveaux contenus injectés ou la mise en œuvre de nouveaux besoins exprimés.

ECOSYSTEME AUTOUR DU POLE NLP

1.3.1 GATE

Gate est un logiciel open source du traitement des langues (TAL) en Java que le pôle NLP utilise principalement. En tant qu'interface, GATE permet d'importer et de déployer des modules variés. L'utilisateur peut importer des modules *Collection of Reusable Objects for Language Engineering* (CREOLE)[2] : Tokeniser, Gazetteer, Jape Transducer, etc. Le Tokeniser, dont on peut paramétrer les critères de découpage, génère des tokens qui sont à la base des analyses de texte qui suivent. Un gazetteer est une liste de vocabulaire dont les occurrences seront trouvées dans le texte. Et le Jape Transducer porte sur la grammaire Jape, il permet de l'appliquer sur le texte pour annoter des patterns plus complexes que ceux annotés par le gazetteer.

[2] https://gate.ac.uk/sale/tao/splitch4.html

Une liste des modules constitue un « pipeline ». Le pipeline est une chaîne de traitement composé de diverses processing ressources permettant d'exploiter le contenu. Pour illustrer le fonctionnement d'un pipeline, nous prendrons l'exemple de ANNIE, un pipeline que Gate fournit par défaut et qui permet d'extraire des entités nommées en corpus anglais. Dans la boîte de *Selected Processing ressources*, chaque module qui constitue ANNIE est aligné dans un ordre défini (figure 1).

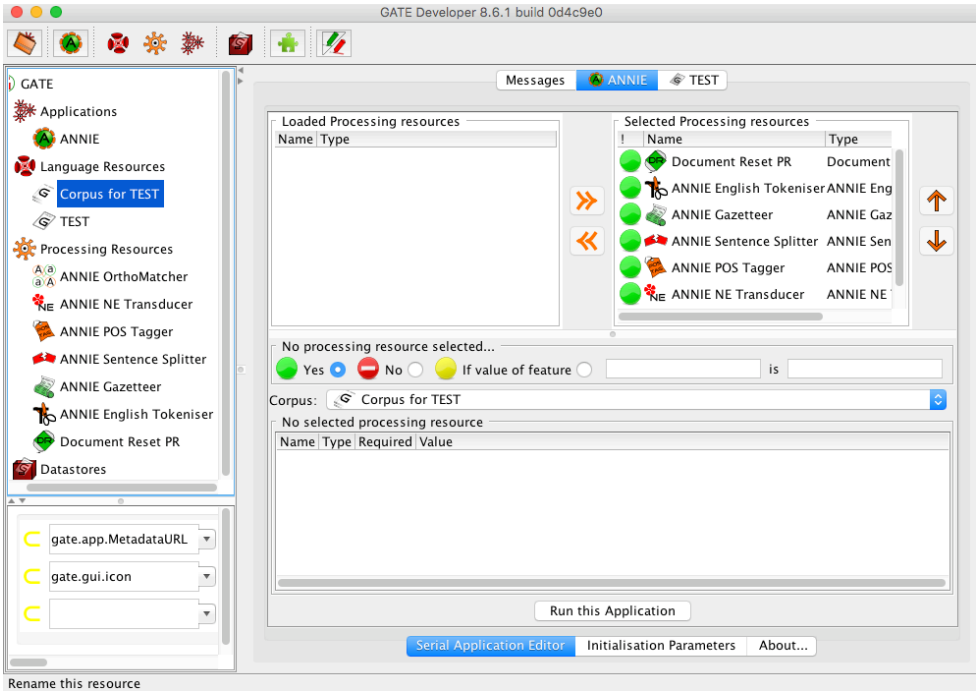


Figure 1 Pipeline ANNIE sur GATE

Gate et ses modules permettent la constitution d'un pipeline qui assure l'exploitation complète d'un contenu, de sa tokenisation à l'extraction d'information, afin de répondre à un besoin spécifique. Au pôle NLP pour repérer des entitées nommées, nous composons un pipeline pour chaque projet nécessitant du NLP, de l'étape de tokenisation à celle de la constitution des grammaires. Ces pipelines sont faits sur mesure.

1.3.2 TXM

TXM est un outil de textométrie permettant de faire de l'analyse de contenu et de la fouille de données. Il est utile pour visualiser le corpus en format XML et mesurer des co-occurrences d'un certain mot. Donc, au sein de Viasema il est généralement utilisé au niveau de la pré-étude.

1.3.3 GITLAB

GitLab est un système de contrôle de versions des ressources qui enregistre les modifications pendant la production des ressources et aide les utilisateurs à visualiser et à réviser les modifications apportées.

L'aide d'un « système de contrôle de version » est nécessaire dans les situations suivantes.

- Lorsque vous supprimez accidentellement un code
- Lorsque plusieurs ingénieurs modifient le même module en même temps
- Lorsqu'un problème survient dans une ancienne version qui a déjà été publiée et doit être corrigée
- Lors de l'ajout de nouvelles fonctionnalités sans interrompre le déploiement actuel

Même si une erreur apparaît sur le code qui est actuellement déployé dans un service, le lancement doit être maintenu. GitLab permet de maintenir le code dans une branche et corriger le même code sur une autre branche sans que cela nuise au bon fonctionnement du code.

Un répertoire de git est une base de données de fichiers qui gèrent les versions du code source. Ce répertoire est un stockage en cloud. On l'appelle aussi le répertoire distant. Le répertoire local est l'espace dans l'ordinateur

individuel. On peut appeler dans le répertoire local une des versions qui est enregistrée dans le répertoire distant. Et chaque version se voit attribuée un ID afin d'être distinguée des autres.

Des versions peuvent être branchées sous la forme d'une arborescence. Il peut y avoir plusieurs branches, mais il n'y a qu'une seule racine. Et les branches peuvent être fusionnées si nécessaire.

Le pôle NLP utilise GitLab en vue de suivre des versions, de collaborer avec plusieurs membres sur un même projet et pour stocker nos ressources de façon sécurisée.

PROJET DE STAGE : REFACTORING DES RESSOURCES ISSUES DU PROJET ASSIGNATIONS

Dans l'informatique, le « Refactoring » est un processus de restructuration des codes informatiques existants sans modifier les fonctionnalités offertes aux utilisateurs finaux aux utilisateurs. Il vise à améliorer principalement la lisibilité et à faciliter la maintenance. Le changement cherche à améliorer la logique interne ou la structure interne. Cependant il ne faut pas modifier l'aspect fonctionnel global.

Nous avons adapté le processus de Refactoring pour le projet de notre stage en respectant nos besoins.

2.1 PROJET ASSIGNATIONS

Le projet du stage est le « Refactoring de grammaires Jape dans le domaine juridique » (ci-dessous, Refactoring). Le projet Assignations est un des premiers projets clients au sein de Viasema impliquant autant de NLP. Viasema fournit au client un processus automatique de traitement des assignations, qui sont un type de document juridique par le biais duquel une personne physique ou morale est citée à comparaître en justice. Avant la mise en place de l'automatisation du traitement des assignations, les étapes de traitement manuel des assignations se déroulaient comme suit.

Le processus des tâches commence dès qu'un assistant juridique de la société cliente de Viasema souhaite traiter une assignation. L'employé numérise l'assignation, il identifie le juriste qui est chargé de traiter l'assignation et la lui distribue. Ensuite, le juriste lit tout le document pour vérifier les informations clés. Si l'employé distribue le document à la mauvaise personne, le même processus se répète.

Andromeda automatise des tâches simples et répétitives des humains pour augmenter la productivité chez le client. Andromeda procède de la manière suivante :

- Analyse de l'assignation
- Extraction des informations clés
- Identification du juriste en charge de l'assignation et envoi de l'assignation au juriste

Le texte du document est analysé par le biais de méthodes de traitement automatique des langues. Les informations clés sont ensuite annotées et permettent :

- d'identifier le juriste en charge de l'assignation
- d'extraire des informations clés à destination du juriste

Les juristes, en fonction du département dans lequel ils travaillent, sont amenés à maîtriser plusieurs spécialités du droit, par exemple, droit de l'immobilier, droit fiscal. Ainsi, l'assignation doit être transmise au bon juriste dont le département ou la spécialité le concerne en prenant pour point de repère le contenu de l'assignation.

En vue d'aiguiller le texte vers le bon service, nous identifions le thème de l'assignation à l'aide des champs sémantiques modélisés sous la forme de gazetteers. Le thème de l'assignation correspond à un service au sein de l'entreprise client. L'identification de l'entité responsable se fait par le biais de règles côté web-sémantique. Cette identification se base sur une liste de mots-clés. Par exemple, un vocabulaire lié au thème de la relation clients nécessite de

vérifier si l'assignation concerne un client de type personne physique ou de type entreprise. En fonction du type de client, le texte sera aiguillé vers la bonne entité.

Ainsi, la tâche du pôle web-sémantique est étroitement liée à celle du pôle NLP. Alors que le pôle web-sémantique s'occupe de la classification des assignations, le pôle NLP se charge de l'analyse du texte et l'extraction des informations. Puisque la classification opérée par le pôle web-sémantique dépend des données extraites par le pôle NLP, il faut que les retours du pôle NLP soient homogènes, stables et maintenables.

CORPUS

Les documents fournis par le client sont des documents internes. Comme ce sont des documents confidentiels, on ne partage pas d'exemple dans ce rapport mais on explique son format. Les documents bruts sont classifiés par le pôle NLP selon leurs formats et leurs besoins. L'assignation est un format de document juridique dans lequel une personne physique ou morale est citée à comparaitre. Elle comporte des informations nécessaires car attendues par le client.

En vue d'extraire ces informations, le document en papier est numérisé en pdf et ensuite il est océrisé pour reconnaitre correctement des textes comme des chaines caractères et pour ne pas perdre des informations de la mise en forme et il est sorti en format Alto XML. Etant donné que Gate ne traite que des textes qui sont sortis de balises, nous transformons Alto XML encore une fois par une feuille xslt de façon à ce que le texte soit sorti des textlines.

GRAMMAIRE JAPE

La grammaire Jape est un module de Gate pour manipuler le corpus, mais à la fois l'objet de notre projet de stage.

Dans le projet de l'assignation, le pôle NLP se charge de repérer et d'extraire des chaines de caractères dans le corpus en XML ALTO. Puisque les contenus repérés sont différents selon le corpus et selon le but du projet, nous reconstituons chaque fois un nouveau pipeline qui se constitue notamment des grammaires Jape.

Un module Jape se constitue d'une main grammaire qui appelle une liste des grammaires qui seront exécutées dans l'ordre dans lequel elles sont appelées par le main grammaire.

Ensuite, une grammaire Jape se constitue d'une partie gauche (LHS) et d'une partie droite (RHS). La LHS décrit des contenus à repérer à l'aide d'expressions régulières et de tokens qui sont déjà annotés. La RHS s'occupe de la partie d'étiquetage qui se compose d'instructions de manipulation d'annotations en langage Java avec ses librairies d'une manière limitée.
Le figure () est une grammaire Jape simple extraite du Thakker et al. (2009). LHS repère un pattern composé des chiffres et des point et RHS l'a annoté comme *Address* et lui distribue le feature dont la valeur est *ipAddress*.

```
Phase:TokenUse
Input: Token
Options: control = brill
Rule: IPAddress
(
   ({Token.kind == number}{Token.string == "."})[3]
   {Token.kind == number}
):ipAddress
-->
:ipAddress.Address = {kind = "ipAddress"}
```

LHS

RHS

La grammaire Jape s'appuie sur des tokens pour repérer des informations. Donc il faut faire précéder les grammaires d'une étape de la tokenisation. Après la tokenisation, les grammaires sont appelées dans le pipeline dans l'ordre dans lequel elles doivent être exécutées.

Par exemple, dans une assignation il est attendu que l'acte de procédure, la partie assignée et la partie qui assigne, le jour de comparution, etc. soit extraits. Pour cela, un pipeline a été construit. Il commence par le module de tokenisation et il enchaîne des modules Jape afin de repérer la date, des noms de personnes, des mots clés propre à l'assignation, etc.

GAZETTEERS

Le gazetteer est aussi un module de Gate qui repère des tokens et crée des annotations. En vue d'annoter, le gazetteer utilise soit

- une liste de chaines de caractères simple
- des requêtes au format, LUCENE, plus précisément en Query Parser surround.

Les gazetteers qui suivent la syntaxe Lucene sont interprétés dans Gate grâce à un plugin Luwak. Ces gazetteers sont nommés « gazetteers Luwak » au sein de Viasema tandis que les gazetteers « simples » sont appelés « gazetteers ». Les gazetteers Luwak permettent d'annoter des patterns plus riches et de façon plus souple. Un gazetteer simple autorise moins d'ambiguités mais permet malgré tout des variations liées à la case.

2.2 OBJECTIF DU REFACTORING

Le besoin du projet de Refactoring vient de l'importance des ressources et le manque de documentation du projet Assignations. Etant donné que le projet

Assignations est un des premiers projets clients au sein de Viasema impliquant autant le pôle NLP, tous les autres projets clients pour lesquels le NLP est impliqué s'appuient sur des ressources issues du projet Assignations. Dans ce projet ont été développées des ressources qui répondent à des besoins élémentaires et qui peuvent être commun à de nombreux projets (comme des grammaires pour repérer une adresse française ou une date).

Malgré la dépendance et la fréquence d'utilisation des ressources issues du projet, il manque la documentation du projet car plusieurs Talistes ont travaillé sur le projet à des périodes différentes sans communiquer entre eux et avec une documentation très maigre. Le manque de documentation ralentit la compréhension de certaines grammaires, notamment au niveau du but de certaines grammaires et des résultats attendus en sortie. Et cela provoque des interprétations hétérogènes sur une grammaire.

Nous vous expliquons cette problématique avec un exemple d'une grammaire que nous avons rencontrée dans le projet Assignations. Il s'agit d'une grammaire otherAddress qui s'intègre dans une main grammaire dont le but est de repérer des adresses. La main grammaire est composée de notamment quatre grammaires de repérage d'adresses, dont otherAddress).

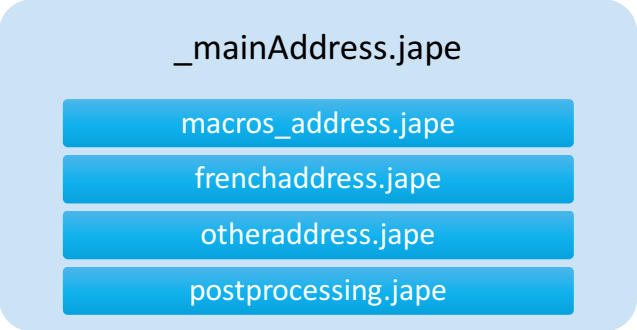
D'abord, l'interprétation de la grammaire otherAddress était différente selon les membres du pôle NLP : l'un l'interprète, en se basant sur le nom de la grammaire, comme une grammaire qui repère des adresses hors de France, l'autre l'interprète comme une grammaire qui repère des adresses belge, l'interprétation étant basée sur le contenu des ressources liées à la grammaire. Si l'interprétation est différente, l'utilisation des grammaires aussi serait différente à cause des sorties attendues différentes. De plus, l'étape qui suit la grammaire et qui permet d'enrichir les grammaires serait aussi différente.

Comme nous l'avons mentionné, la qualité du service fourni au client dépend des données extraites par le pôle NLP, il est important que les sorties de chaine de traitement soient homogènes. Et pour cela, nous avons besoin aussi que les ressources aient une interprétation unique.

En outre, l'enchainement des grammaires est également un problème pour leur réutilisation. Par exemple, la grammaire otheraddress.jape ne fonctionne

correctement que dans le contexte de ce bloc de grammaires de repérage d'adresses. Pour utiliser otheraddress.jape, il faut utiliser les trois autres grammaires aussi.

En dernier, le problème de otherAddress est qu'il ne repère que quelques adresses françaises.



Cela étant, nous avons besoin de refaçonner les ressources issues du projet Assignations en vue d'être compréhensible, réutilisable et productible. Ce qui interrompt la compréhension des grammaires est la présence de noms d'annotations ambigües et le manque de documentation. Également, la dépendance des grammaires entre elles bloque la réutilisation d'une grammaire spécifique indépendamment des autres. Enfin, une grammaire qui ne génère pas d'annotations cohérentes ralentit la chaine de traitement.

En vue de satisfaire correctement nos besoins de Refactoring, il faut répondre à trois objectifs :

1) Mettre en place des bonnes pratiques pour rendre les grammaires compréhensibles

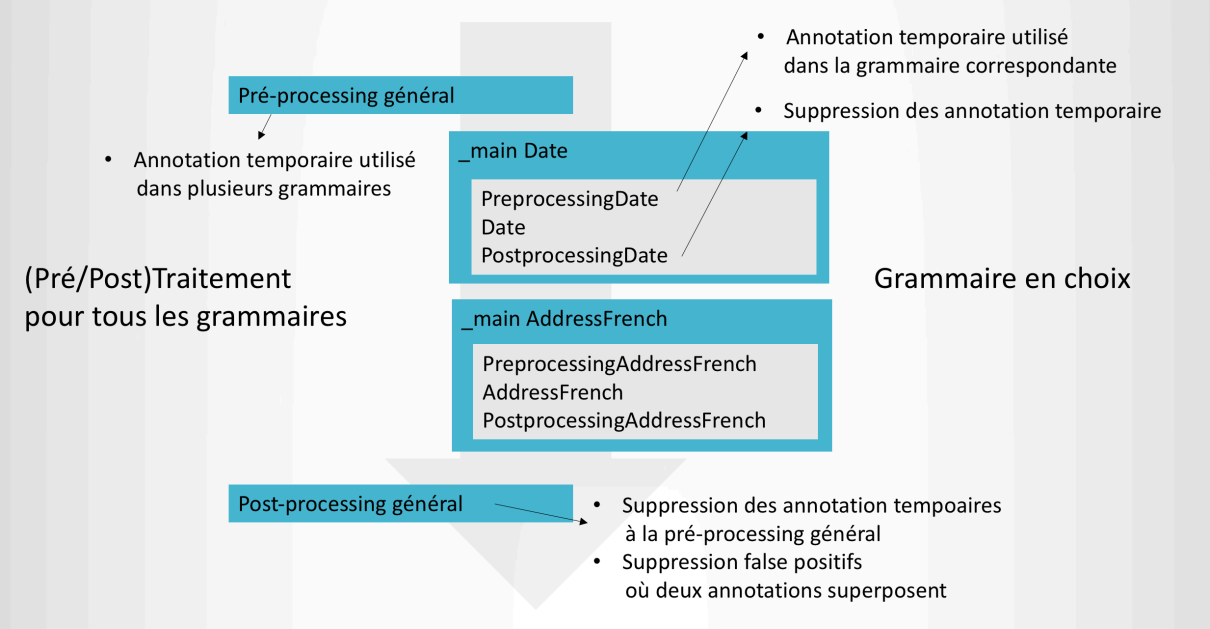
2) Nettoyer les grammaires en supprimant des règles qui ne donnent pas d'annotation

3) Faire la distinction entre ce qui est spécifique au projet Assignations et ce qui est générique à tous les projets, et factoriser pour faciliter la réutilisation

Nous allons atteindre ces trois objectifs ci-dessous grâce à trois phases qui seront expliqués dans le chapitre suivant.

2.3 PLANNING

2.3.1 SCHEMA



Pour atteindre nos trois objectifs, nous procédons par étapes. Tout d'abord avant de démarrer le Refactoring, nous élaborons un schéma de grammaires Jape que Viasema souhaite, une grammaire indépendante qui peut s'appliquer à n'importe quel projet.

Donc, notre objectif de Refactoring est de séparer chaque grammaire et de remodeler chaque grammaire afin qu'elles puissent être utilisées de façon indépendante et qu'elles restituent une sortie propre, quel que soit le projet dans lequel les grammaires ont été intégrées. Et en tenant compte des besoins, il faut également améliorer la lisibilité et la performance.

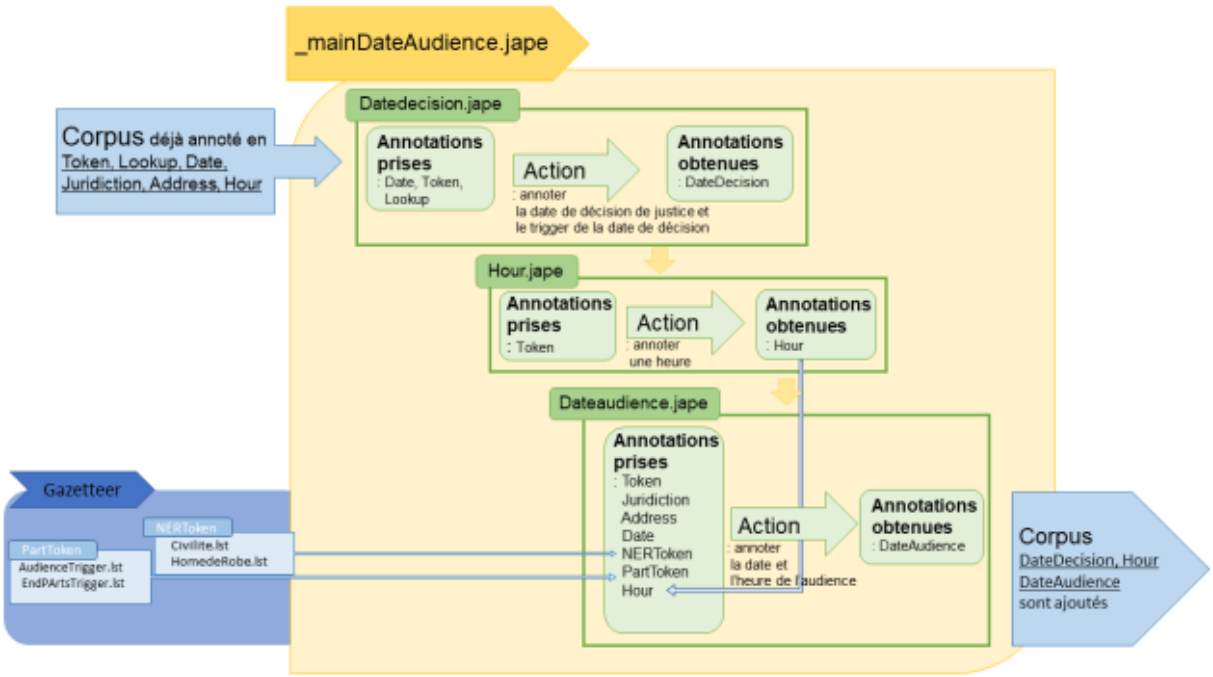
Dans l'état initial du pipeline du projet Assignations, le résultat d'une grammaire est souvent dépendu de celle précédemment exécutée. Donc, il est difficile de ne pas modifier le comportement final du pipeline après Refactoring. En outre, ce pipeline est profondément lié au besoin du projet Assignations. Cela veut dire qu'il est sur-appris sur certain type de corpus. Étant donné cela, une modification partielle des résultats obtenus en sortie du pipeline est inévitable.

2.3.1 ROADMAP

Le projet de Refactoring des ressources issues du projet Assignations est divisé en deux parties : Refactoring des grammaires et Refactoring des gazetteers. Le projet de stage ne prend en charge que le Refactoring des grammaires, mais suggère des propositions à propos des gazetteers qui sont utilisés dans les grammaires correspondantes.

Comme le projet est divisé en deux parties, la communication sur l'état d'avancement est important. Donc, nous avons créé une RoadMap, une page de wiki interne pour partager des objectifs et des étapes avec tous les membres du pôle NLP. Le RoadMap contient une description des ressources sur lesquelles porte le Refactoring, des objectifs, des étapes et des points d'attention à garder en tête pendant le processus. En vue de résumer les analyses, une figure et deux listes sont présentées.

FONCTIONNEMENT DANS GATE



Le figure a pour but de faciliter la compréhension de la relation entre les Gazetteers et les grammaires. Ce figure explique la façon dont une main grammaire fonctionne. Dans l'exemple présenté ci-dessus, la main grammaire se constitue de trois grammaires. Chaque grammaire prend en entrée des tokens annotés par des grammaires Jape et des tokens annotés par des gazetteers. En vue de décomposer des grammaires dans une main grammaire, il faut comprendre les impacts de l'un à l'autre.

MODE D'EMPLOI DES GAZETTEER

Lorsque la grammaire prend des gazetteers comme entrée, elle appelle le gazetteer par le biais du type d'annotation généré par le gazetteer. Par exemple, quand une grammaire prend le type d'annotation « AddressToken » comme une entrée, cette annotation appelle trois gazetteers – une liste de types de voies, une liste de types d'édifices, une liste de compléments d'adresses.

Type d'annotation générée par le gazetteer	Nom du gazetteer
NERToken	LawyerOf.lst
	Civilite.lst
	Prenoms.lst
	HommesDeRobe.lst
	CodesPays.lst
	LieuxEurope.lst
	Pays.lst
	etatsUS.lst
	villesUS.lst
AddressToken	TypesDeVoies.lst
	TypesDEdifices.lst
	ComplementsDAdresse.lst

Il a été constaté que certains types d'annotations sont très larges dans ce qu'ils regroupent. C'est le cas par exemple de NERToken. Cette annotation ne semble pas répondre à un but précis, car NERToken appelle des référentiels de toponyme et à la fois des référentiels de personnes physiques comme des prénoms et des hommes de robe. Nous pouvons comprendre l'utilité de NERToken par analyse des usages dans la grammaire. D'après notre analyse de l'usage des annotations NERToken, il ressort que ces annotations sont souvent utilisées dans grammaires de la manière suivante : !NERToken. Cela signifie que ce qui est cherché ce sont des tokens qui ne sont pas annotés comme *NERToken*Cela est utilisé afin d'exclure des candidats lors de l'annotation.

Ainsi, il faut comprendre le but des gazetteers et leur emploi afin de comprendre et distinguer les grammaires génériques des grammaires spécifiques à un projet.

DISTINCTION DES GRAMMAIRES CLASSEES PAR DOMAINES.

En vue de comprendre des grammaires, nous les avons triés dans une liste. Il s'agit d'une partie de liste qui restitue des grammaires pour repérer des adresses en France.

Grammaires	**Input**	**Output**	**Utilité**
macro_address.jape	NERToken, Token, AddressToken, Date	Annotations temporaires	Annoter temporairement des éléments afin de faciliter l'annotation finale des adresses
frenchaddress.jape	NERToken, Token, Annotations temporaires	Address, SilentToken	Annoter des adresses en France
otheraddress.jape	NERToken, Token, AddressToken, Date, Adress, Annotations temporaires	Address, SilentToken	Annoter des adresses hors France
postprocessing.jape	NERToken, Token, OrganizationToken, TextBlock, PartToken, CourtToken	SilentToken, CleanUpToken	Supprimer des annotations temporaires et des tokens inutiles

D'abord, la grammaire macro_address.jape et postprocessing.jape se situe au début et à la fin de la liste de la main grammaire. Cela repère et supprime des annotations temporaires pour deux grammaires qui repères des adresses différentes.

Ensuite, Otheraddress.jape semble avoir pour but de repérer les adresses hors de France. L'analyse des gazetteers appelés par cette grammaire relève que ce sont les adresses belges qui sont en partie ciblées. D'autre part, le test de cette grammaire indique que cette grammaire permet d'annoter des adresses françaises.

En conclusion, la RoadMap permet de mettre en avant des problématiques à considérer. Les grammaires ont besoin d'être analysées et testées au niveau de chaque règle. Les gazetteers ne sont pas l'objet du stage de Refactoring des grammaires, mais ils sont profondément liés à la tâche de Refactoring des grammaires. Alors des propositions de modification de gazetteers accompagnent inévitablement le Refactoring des grammaires. Et dernièrement, nous avons besoin de construire une convention de formats ou de nommage pour faciliter la compréhension et uniformiser les grammaires.

PROJET REALISE

1ERE PHASE : MISE EN PLACE DE LA BONNE PRATIQUE

Dans la première étape nous avons établi un guide de bonnes pratiques après la discussion et le feedback de la part des membres de pôle NLP qui utilisent les grammaires.

Nous avons commencé par la phase de mise en place de bonne pratique en première, car il faut bien comprendre chaque grammaire au niveau de ses inputs, ses sorties, l'impact des relations entre grammaires. Alors que certaines grammaires étaient inventées et exploitées par un seul TAListe au début, la collaboration et la communication entre les pôles n'avaient pas été rendues compte. Les grammaires marchent correctement, mais elles manquent de commentaires. De plus, Jape n'est pas une grammaire dont un guide de style est fourni comme python.

Donc, on a créé une convention et un guide de bonne pratique standard à partir duquel toutes les grammaires sont transformées.

3.1.1 GUIDE DE BONNES PRATIQUES POUR LA REDACTION DE GRAMMAIRES JAPE

Le guide de bonnes pratiques pour la rédaction de grammaires Jape vise à améliorer la lisibilité et la productivité des grammaires Jape, on propose une forme standard en interne de Viasema.

```
Imports: { import java.text.DecimalFormat; }

//Liste des phases : Phase_Name, Phase_Name2
//Pré-requis : Annotation1 Annotation2

////////////////////////////////////////////////////////////////
//                                                            //
//      Explication de Phase: Au dessus de Phase              //
////////////////////////////////////////////////////////////////

Phase: Phase_Name //Nouveau Phase quand Input/Options change
Input: Annotation1 Annotation2
Options: control = miniscule

Macro: MACRO_NAME //tous les Macro se place après Phase, pas après chaque Rule

/*
Explication de Rule : au dessus de Rule
*/

Rule : RULE_NAME (
):a
-->
:a.TypeName(nomDuFeature="valeur")

/**
DÉSACTIVÉ : raison de la désactivation //S'il y a un Rule désactivé

Rule : RULE_NAME (
):a
-->
:a.TypeName(nomDuFeature="valeur")
**/
```

ÉCRITURE BICAMERALE

Étant donné que la grammaire est sensible à l'écriture bicamérale, Jape propose une convention de l'écriture bicamérale. Tous les noms suivent l'exemple de l'écran ci-dessus à propos de l'écriture majuscule, minuscule et de l'utilisation de symboles pour la combinaison de mots.

COMMENTAIRES

Les commentaires sont plus importants pour faciliter la compréhension. Il faut une description ou une explication simple à chaque phase et règle. S'il y a un commentaire pour faciliter la compréhension de la grammaire, on écrit sous la ligne correspondante avec le symbole deux slashes(//). Par contre, s'il y a un

commentaire pour une discussion ou une suggestion à faire, on écrit dans un autre fichier, un fichier README.

C. META-INFORMATION

Au début de fichiers, nous avons mis une liste des phases au cas où il y ait plusieurs phases. Cela permet de comprendre directement la structure de la grammaire et donc de gagner en efficacité. De plus, nous avons conseillé d'ajouter une liste des annotations que la grammaire demande à priori. Cela permet de savoir facilement quel gazetteer et quelle grammaire doit être exécutée en amont.

3.1.2 CONVENTION DE NOMMAGE

La convention de nommage est appliquée aux grammaires, aux référentiels et aussi aux noms de fichiers pour avoir la cohérence des annotations. Lorsque les noms des annotations venant de gazetteers sont itérativement utilisés dans la grammaire, ils ont besoin d'être instinctivement reconnu. Il faut standardiser le nom pour reconnaitre instinctivement.

La convention de nommage rend compte de 1) une standardisation de la nomination, 2) la sélection de la langue.

STANDARDISATION DU NOMMAGE : GENERIQUE + SPECIFIQUE

D'abord, la classe générique se trouve au début du nom de fichier et la sous-classe spécifique à la fin. Par exemple, après Refactoring, la main grammaire Address est divisée en quatre types : adresse française, adresse aux était-unis, adresse en Angleterre, et autres adresses. En respectant le format générique + spécifique, ces quatre grammaires sont nommées de la façon suivante :

- AddressFrench
- AddressUSA
- AddressUK
- AddressForeign

Dans les exemples, « Address » est générique donc placé au début et ce qui suit est plus spécifique.

LANGUE

Lorsque les textes que Viasema traite sont multilingues, le système a besoin d'une langue pour les représenter et l'anglais est choisi. Les quatre grammaires ci-dessus sont écrites en anglais. Néanmoins, la langue locale peut être sélectionnée sous certaines conditions. Si des noms sont liés à un domaine d'activité, il est souvent difficile de trouver la traduction adéquate. Par exemple, *siren*, *rcs* sont des noms propres qui sont liés au monde de l'entreprise en France. Et le vocabulaire autour des *hommes de robe* est spécifique au système juridique en France. Dans ces cas-là on écrit dans la langue correspondante.

3.1.3 CONVENTION SPECIFIQUE A LA PERFORMANCE : MACRO VS. ANNOTATION TEMPORAIRE

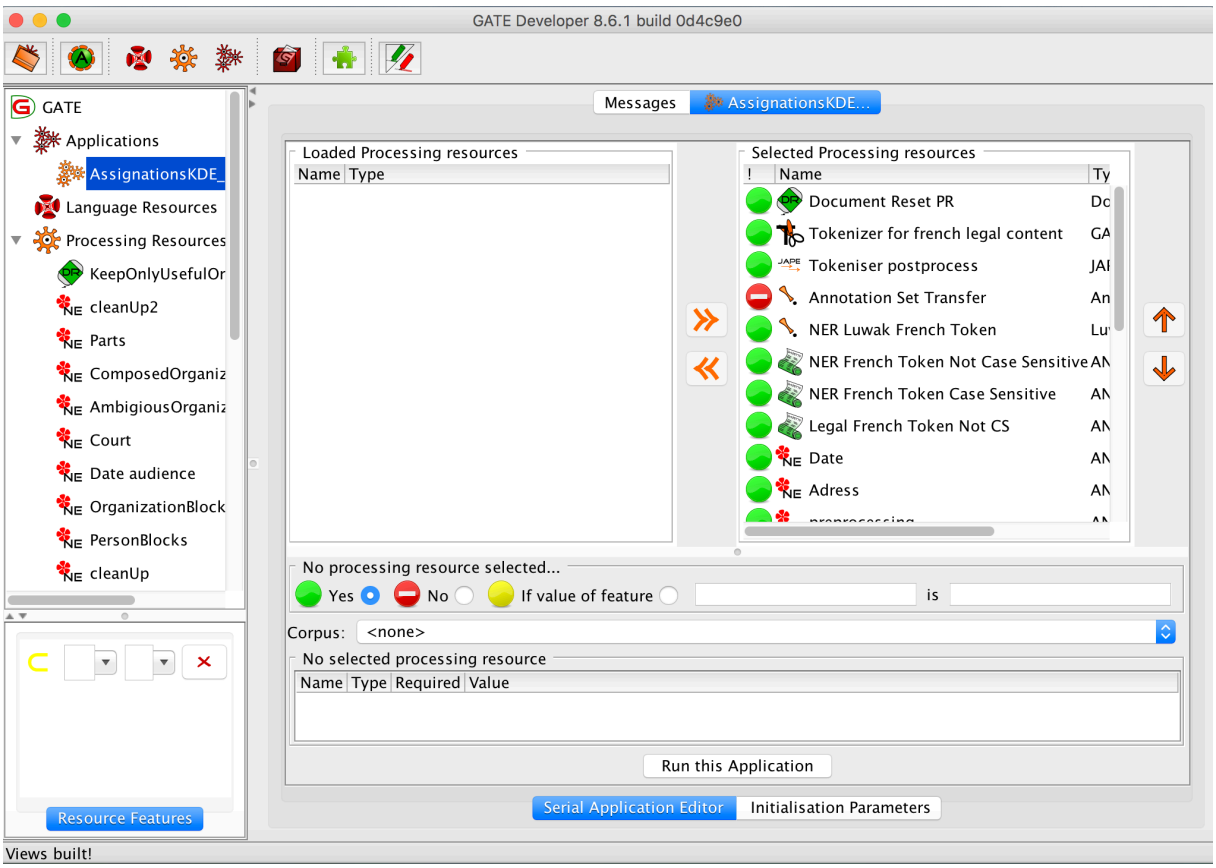
En vue d'extraire un pattern, on peut choisir d'annoter le pattern par fragments ou d'annoter des triggers, ce qui rend parfois la tâche plus aisée. Il y a trois manières différentes d'annoter des parties : en s'appuyant sur les annotations issues de la tokenisation, d'un gazetteer, d'une Macro d'une annotation temporaire ou d'une grammaire.

Si on a une liste de mots que l'on vise, on peut les annoter par le biais d'un gazetteer. S'ils on souhaite utiliser des conditions dépendant d'expressions régulières ou d'autre tokens, on les annote dans les grammaires par le biais de Macro ou d'annotation temporaire. Une macro est une annotation temporaire qui n'est valable que pendant le lancement de la phase correspondante. Il disparait une fois que la phase est finie et il ne génère pas directement une annotation dans le corpus. L'annotation temporaire est un nom que l'on a établi dans le cadre de ce stage. Il s'agit d'une règle ordinaire de Jape, mais qu'on supprime manuellement après son utilisation. L'usage d'une annotation temporaire est moins restrictif que celle d'un Macro. Une macro utilisée dans deux phases différentes devra être créé dans chaque phase tandis que l'annotation temporaire

est exécutée une seule fois et on peut l'utiliser dans toutes les phases jusqu'à ce qu'on la supprime. Par contre, le but d'une Macro est la lisibilité et la simplification du règle. Une Macro semble une factorisation des règles pour lire la règle à la manière de langue naturelle. C'est pourquoi dans les grammaires Assignations avant Refactoring, beaucoup de macro sont utilisées pour donner la lisibilité et pour simplifier la règle.

Ce qui est plus problématique est que l'on ne connait pas la différence de performance et de poids entre la Macro et l'annotation temporaire. C'est pourquoi nous avons organisé un test.

En vue de mesurer la performance de la chaine de traitement, nous avons comparé le runtime de deux pipelines, un original qui est avec des macros et un modifié qui est avec des annotations temporaires. Ce pipeline a été constitué pour annoter l'adresse.



D'abord, le pipeline commence par deux modules et une grammaire Jape pour la normalisation adaptée au projet Assignations. Ensuite, un gazetteer et

trois gazetteer luwak génèrent des annotations dont ont besoin les grammaires qui les suivent : Et en fin de pipeline, deux grammaires se trouvent : La grammaire Date précède la grammaire Address car la grammaire Address exploite l'annotation Date pour éviter de générer le bruit.

Nous avons transformé les trois Macros en annotations temporaires de la grammaire d'Address. Le corpus utilisé pour le test a une taille de 26 assignations et il est aléatoirement choisi parmi le corpus de travail. Gate est un logiciel qui est sensible à l'état de l'ordinateur. Alors on a fait des tests 3 fois et pris la moyenne des runtimes.

	Macro	Annotation temporaire
1er TEST	49.747 s	42.476 s
2eme TEST	48.532 s	41.074 s
3eme TEST	53.062 s	42.142 s
Moyenne	50.447 s	41.897s (-17%)

Les runtimes de version de l'annotation temporaire sont plus courts que celui de Macro. L'annotation temporaire diminue de 20% le runtime. Il faut considérer que les trois Macros étaient répétitivement utilisées dans plusieurs phases et ils ont annotés 1,442 cas. Si le Macro était utilisé une seule fois avec peu de cas à détecter, la différence ne serait pas si grande.

En conclusion, on prévoit que l'annotation temporaire peut être plus performante que le Macro. On conseille d'utiliser l'annotation temporaire plutôt que Macro quand on a besoin d'annoter une partie d'annotation visé et répétitive à travers des phrases.

3.1.4 AMELIORATION DE LA PERFORMANCE GENERALE PAR LA MISE EN PLACE DE BONNES PRATIQUES

Outre la transformation des Macro en annotations temporaires, l'application des bonnes pratiques sur les grammaires permet d'améliorer les performances de la chaîne de traitement. Nous avons comparé le runtime de deux pipelines dont l'une avec bonnes pratiques et l'autre sans. Celle avec la bonne pratique nous rend un runtime diminué de 30%.

	Avant la mise en pratique	Après la mise en pratique
1er TEST	69.870 s	49.747 s
2eme TEST	80.558 s	48.532 s
3eme TEST	73.795 s	53.062 s
Moyenne	74.741 s	50.447 s (-33%)

La suppression des macros et des entrées inutiles a aidé à diminuer le runtime.

2EME PHASE : NETTOYAGE DES GRAMMAIRES

Après la réécriture de grammaire pour améliorer la lisibilité, nous nous sommes attelés au nettoyage des grammaires pour augmenter la performance. Les étapes de nettoyage sont les suivantes :

- Suppression des Macros ou des annotations temporaires qui ne sont pas utilisés
- Suppression des règles qui ne rendent pas de résultat
- Suppression des règles qui rendent le même résultat avec le même but

Pour savoir si le nettoyage nous génère du silence, nous avons besoin de tester avec des corpus pertinents. D'abord, le corpus doit inclure un grand nombre d'annotations visées. De plus, le corpus doit représenter un échantillon de population. Lorsque nous traitons des corpus multilingues – français et anglais, la langue aussi est un paramètre à considérer pour construire un corpus.

Ce qui est important dans cette phase, c'est de comprendre la raison du Silence parce qu'il est possible que le corpus inapproprié provoque le manque de retour. De plus, une règle qui vise un contenu avec un pattern rare aussi peut provoquer le même problème. Donc, il faut comprendre la règle et estimer la pertinence en vue de la nettoyer.

3EME PHASE : DISTINCTION DU GENERIQUE, SPECIFIQUE

La 3ème Phase est la constitution de grammaires génériques. D'abord il convient de définir la notion de générique dans le contexte de ce projet de Refactoring.

Transformer une main grammaire en générique, c'est faire en sorte qu'elle soit indépendante et qu'elle puisse être isolée du reste des ressources composant le pipeline. Cela ne signifie pas qu'une grammaire ne sera pas dépendante d'une autre. Par exemple, la grammaire *dateaudience.jape* pour annoter la date de l'audience est exécutée après la grammaire *Address* et utilise la sortie de la grammaire *Address* afin de supprimer des bruits. L'isolement total de la grammaire *dateaudience.jape* n'est pas possible étant donnée sa dépendance avec *Address*. Il convient donc de le renseigner dans un fichier Readme.

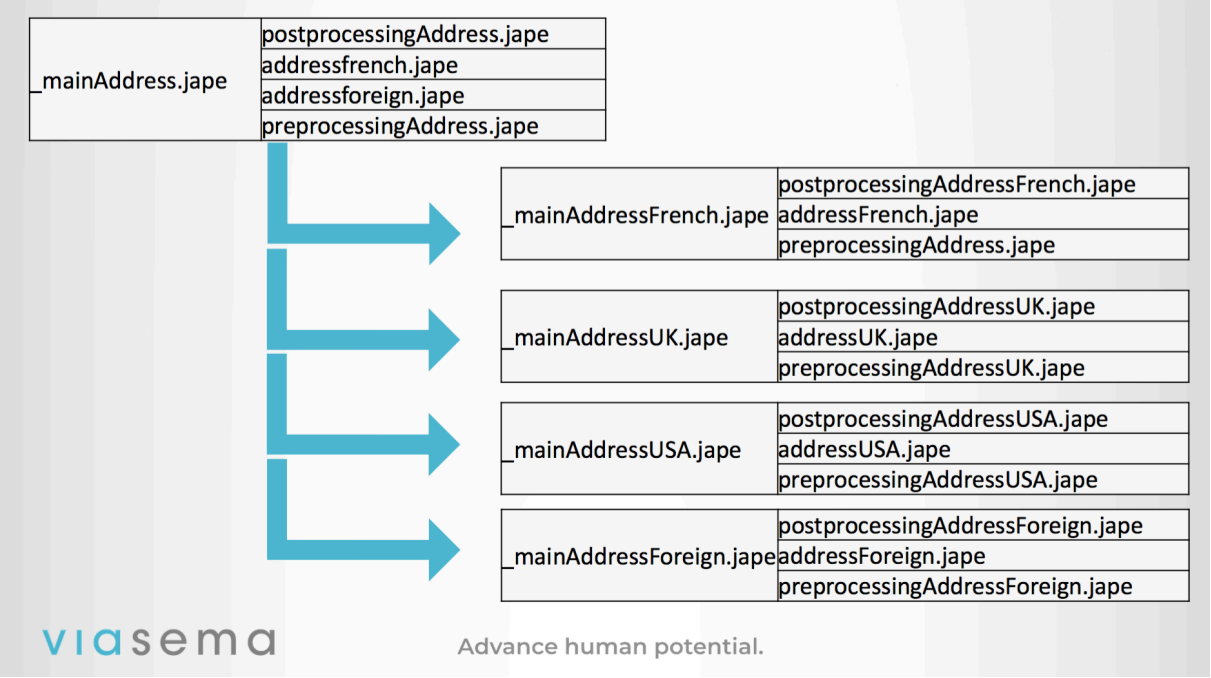
L'autre besoin de Refactoring est de permettre aux grammaires génériques d'être exploitées dans n'importe quel projet. Ainsi, générique signifie que la grammaire ne doit pas dépendre d'un projet en particulier. Dans le projet Assignations, on constate un format d'extraction qui répond à un besoin spécifique exprimé par le client (par exemple, le client souhaite que lorsqu'un nom de société est extrait, la forme juridique de la société en question soit exclue du nom de l'entreprise). D'autre part, dans l'état initial du pipeline Assignations, des grammaires utilisent des ressources spécifiques au client. La grammaire générique doit exclure les annotations demandées et les ressources fournis par le client ou le projet spécifique.

La grammaire générique peut être définie par trois éléments :

1) une grammaire autonome qui fonctionne au maximum toute seule

2) la dépendance avec d'autres grammaires doit être renseignée aux utilisateurs
3) elle n'est pas dépendante de ressources propres à un client en particulier

Le figure en dessous est un exemple de la distinction de la grammaire *Address* :



La grammaire *Address* est divisé en quatre grammaires différentes selon le pays concerné.

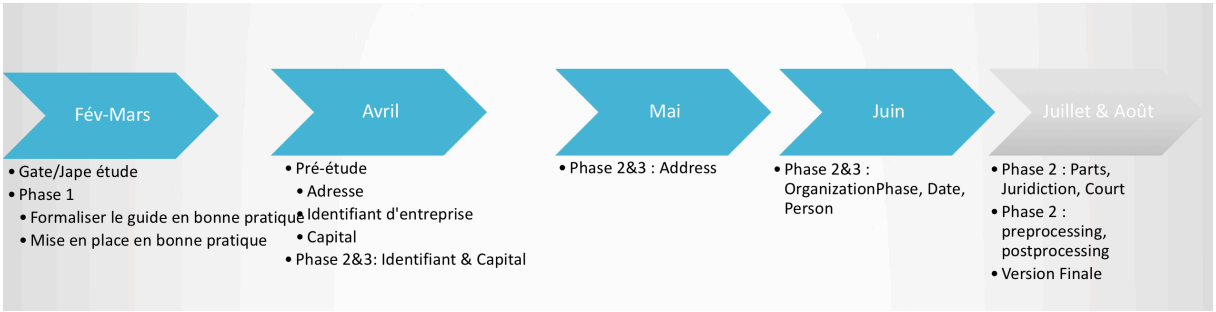
Ces quatre grammaires fonctionnent individuellement dans Gate et n'ont pas d'impact sur les autres grammaires. L'adresse est un contenu général qui peut être utilisé dans de multiples projets. Utilisées ensemble, les grammaires ne génèrent pas d'annotations qui se superposent.

PLAN A VENIR ET DIFFICULTES

Alors qu'il reste encore deux mois à la période de stage, la 3ème phase de Refactoring est en cours et il y a des points que nous n'avons pas encore traités. Nous vous partageons notre plan à venir à la place des perspectives et des difficultés que nous avons confrontées jsuqu'à maintenant et dont nous allons rendre compte en continuant le projet.

PLAN A VENIR

Le projet a commencé dès Février et il s'est poursuivi pendant 4 mois. La mise en place de bonnes pratiques a pris du temps car elle nécessitait au préalable une formation à Gate, une familiarisation avec le projet Assignations, ses ressources et ses enjeux, un constat des problématiques et une veille sur de potentielles bonnes pratiques déjà existantes dans la rédaction de grammaires Jape.



Pendant deux mois, nous allons constituer grammaires génériques et les décomposer comme un facteur autonome. À la fin, après le test du contrôle de la qualité, des grammaires du Refactoring seront injectées dans les chaines de traitement actuelles mise en production.

En vue de recevoir la version finale, il faut rendre compte d'un point. Nous avons toujours besoin de fusionner avec le projet du Refactoring des gazetteers. La modification des gazetteers et du nommage des gazetteers doit être mise à jour. Ensuite, des features des annotations ont besoin d'être triés selon le besoin de connaissance.

DIFFICULTE

CORPUS DE GOLD STANDARD

La première difficulté à laquelle le projet du Refactoring est confronté est celle liée au corpus Gold Standard. Dans le projet, nous avons fait le test d'évaluation avec des centaines voire des milliers de corpus. En vue de l'évaluation, nous avons besoin d'un corpus déjà annoté correctement et cela requiert un processus manuel d'annotation. Ce processus prend du temps.

La constitution du corpus de gold standard aussi est un facteur important. Parce que nous ne pouvons pas tester tous les documents, on constitue un échantillon du corpus entier. Il faut que l'échantillon représente justement la répartition du corpus original. Des facteurs de l'échantillon peuvent être varié selon le corpus et le but du projet. Par exemple, il faut rendre compte des langues, des sous-classifications, et de l'existence de certaine information qui sont indispensables mais rarement obtenues.

LIMITE D'AMELIORATION DE LA PRODUCTIVITE

Le projet de Refactoring exploite des documents issus de plusieurs projets de Viasema. Bien que des projets soient adaptés à certains documents, le projet de Refactoring doit également s'adapter à des variantes de documents.

Par exemple, la date dans le document français a un format spécifique : Jour/Mois/Année et il existe des variantes. Des textes en langue multi-langues ont des nombreuses variantes de format de date. Pour des ressources du projet de Refactoring dont le but est de s'appliquer à n'importe quel projet, l'enjeu est surtour la difficulté d'évaluer le seuil de tolérance qui indique le degré de variance.

Il est clair que d'annoter tous les contenus est impossible. Même si c'est possible, la productivité sera diminuée quand on met un trop grand nombre de règles. Par exemple, nous avons ajouté une règle pour repérer des adresses

étrangères excluant celui de l'Angleterre, des Etats-Unis avec un gazetteer comportant tous les noms de pays. Cette règle permet de résoudre le silence généré avant, mais le gazetteer de noms de pays impacte négativement la performance de la chaîne.

Alors, il faut savoir jauger entre la productivité, le poids du pipeline, et le retour. Jusqu'à maintenant la seule limite à laquelle nous avons été confrontées était le délai de remise de notre projet.

IMPLICATIONS D'UN STANDARD

Pour suivre un standard, il faut être rigoureux. Nous avons établi une convention pour l'utilisation du Macro et de l'annotation temporaire, selon le résultat du test de performance. Et il est ainsi conseillé d'utiliser l'annotation temporaire pour une partie d'annotation précise, qui est répétitive lors des phases.

Cependant, nous avons trouvé qu'un cas où une partie d'annotation est répétitive dans plusieurs règles mais dans une seule phase. Il semblait plus pertinent de générer alors, par une annotation temporaire, que par une macro à l'occasion de nombreuses fois. Comme nous n'avons pas vérifié la différence de performance entre macro et annotation temporaire dans une phase, nous n'avons pas posé de conclusion définitive.

CONCLUSION

Nous avons vu la décomposition des grammaires d'un projet spécifique afin de les rendre génériques, pour afin qu'elles puissent être exploitées dans tous les projets. Pour cela, nous avons analysé des grammaires et des gazetteers existants. Suite à la description des problématiques reconnues, nous avons établi des conventions et un guide de mise en bonne pratique.

Étant donné que toutes les grammaires sont réécrites selon des conventions et selon le guide de mise en bonne pratique, nous avons testé la performance de la chaîne et la lisibilité, et nous avons constaté une incidence : le runtime de grammaire est diminué. De plus, la réutilisation des grammaires nous permet d'avancer plus vite dans l'élaboration d'un nouveau projet.

La distinction des grammaires génériques est la dernière étape du Refactoring. Elle est actuellement lors de mon stage, toujours en cours. Les grammaires vont être reconstruites de façon à être rendues génériques : fonctionner de façon autonome, prévoir la dépendance avec les autres grammaires, ne pas être dépendant d'un projet spécifique.

En vue d'avoir des grammaires qui satisfont ces trois conditions, le Refactoring implique l'analyse d'annotations et de gazetteers et la proposition de modifications.

De manière plus générale, ce projet m'a permis de comprendre l'importance d'une communication et d'une coopération entre pôles car le partage des informations permet de prendre des mesures face à une situation inattendue. Et il permet d'augmenter l'efficacité du pôle NLP. Cela a été une bonne expérience également du point de vue universitaire car cela m'a permis d'appliquer concrètement ce que j'ai appris pendant les différents cours et d'en comprendre les enjeux.

Enfin, ce stage m'a appris sur le plan professionnel la manière de travailler dans une équipe professionnelle et coopérative qui est à l'écoute et qui remet toujours en question ses approches : j'ai beaucoup aimé exprimer mes points de vue et apprendre de ceux des autres. J'ai appris non seulement une vision riche de ce qu'est le traitement automatique des langues naturelles, mais également la capacité de concevoir ses applications concrètes et la manière de les industrialiser.

BIBLIOTHÉQUE

CUNNINGHAM, Hamish, et al. *Developing Language Processing Components with GATE Version 8*. University of Sheffield, 2014.

THAKKER, Dhaval; OSMAN, Taha; LAKIN, Phil. Gate jape grammar tutorial. *Nottingham Trent University, UK, Phil Lakin, UK, Version*, 2009.