Résolution de la coréférence à l'aide des ontologies

Adrian Petru Dimulescu

LRI Université Paris-Sud - 91405 Orsay Cedex France, dadi@lri.fr

Résumé

Ces travaux traitent de la résolution de la coréférence (aussi appelée *anaphore*) et spécialement du cas particuler des anaphores sortales, qui utilisent le type sémantique des leurs antécédents pour y faire référence. Une étape intermédiaire de construction d'ontologies par instanciation de concepts dans les textes a été explorée. Ce rapport rend également compte de nos expériments sur des outils de stockage d'ontologies et des outils de fouille de textes.

1 Introduction

Le but de la résolution des coréférences (anaphores) est la détection et la résolution des éléments lexicaux qui pointent vers des entités lexicales ou sémantiques antérieures dans le texte[1]. Un exemple habituel d'anaphore est celui réalisé par les pronoms personnels. La résolution de la coréférence est une étape importante dans la fouille de textes car elle augmente la chance de trouver des relations porteuses de signification entre les termes du texte.

Nous nous sommes concentrés sur les coréférences qui avaient particulièrement du sens pour la catégorisation des textes de la compétition de catégorisation de texte TREC Genomics Track[2]. Les anaphores qui se produisent dans ce genre de textes montrent quelques traits particuliers par rapport aux textes de journaux sur lesquels est centrée une partie importante de la littérature sur la coréférence. Un aspect spécifique est la présence plutôt discrète de l'anaphore pronominale qui se compose dans ce cas principalement de pronoms *we (nous)* non-intéressants qui pointent vers les auteurs des articles et des pronoms *it*, *its*, *itself* qui seuls restent intéressants à résoudre. Ceci est en fait une simplification de la présence pronominale plus complexe de l'anaphore dans les textes d'actualités. Dans ces textes la gamme entière des pronoms personnels est déployée et doit faire l'objet de la résolution mais il apparaît également la question des citations – avec ses problèmes spécifiques.

Il existe une littérature assez riche qui traite de la résolution des anaphores pronominales [15]. Des bons résultats ont été obtenus [14] sur des textes d'actualités en utilisant exclusivement des méthodes de surface (basées sur l'étiquetage POS (part of speech - partie du discours), sans utilisation de relations syntaxiques ou de connaissances extérieures). Nous avons expérimenté l'algorithme de surface implémenté dans Gate/ANNIE [9] pour la résolution du *it*.

Un autre type de coréférence est ce qu'on appelle l'anaphore sortale. Elle fait souvent référence à ses antécédents en utilisant leurs types sémantiques. Quelques exemples d'anaphores sortales sont : « these genes », « both proteins », « the promoter ». Du fait de la nature sémantique de ce type de coréférence nous avons décidé d'utiliser une source externe de connaissance qui peut réduire la résolution de la coréférence à la satisfaction de certaines contraintes sémantiques entre l'anaphore et ses antécédents. Un effort dans la direction de la résolution de la coréférance a été aussi réalisé pour le concours TREC Novelty 2004 par [1]. Ce travail envisage la résolution de la coréférence pronominale mais aussi sortale (telle que réalisée par des fonctions sociales comme *the president, the father*) à l'aide de taxonomies construites manuellement. Il existe le travail intéressant de [4] sur l'instanciation de concepts dans les textes (sect 4.2) portant lui aussi sur des textes d'actualités.

[1] Il existe aussi un genre moins fréquent de coréférence, appelé *cataphore* qui se réfère à des entité apparaîssant après l'anaphore dans le texte.

[2] http ://ir.ohsu.edu/genomics/

2 Web sémantique et ontologies

Notre solution logicielle de résolution de la coréférence doit disposer d'un support intégré de techniques d'interrogation, création et modification d'ontologies. Les ontologies sont des spécifications partagées de conceptualisations. Si la meilleure définition pour une ontologie constitue encore objet de controverse, dans notre projet, nous les utilisons comme des manières de spécifier et de stocker des concepts ainsi que les relations entre eux. Pour des petites structures hierarchisées on peut se limiter à un stockage en fichiers au format XML ou textuel. En revanche, pour des solutions scalables nous avons dû inspecter l'état de l'art dans le domaine des formats utilisables pour la description des ontologies ainsi que des technologies de stockage des ontologies. Nous allons donc porter par la suite un regard sur les spécifications et sur les outils que nous jugeons importants pour l'utilisation des ontologies dans notre projet.

Les observations sur les outils n'ont pas la prétention d'être exhaustives, elles sont le reflet de nos expériences dans le but de mettre en place une solution utilisable de stockage d'ontologies. Une comparaison intéressante sur les moteurs de stockage de triplets[3] existants se trouve dans [12].

2.1 RDF et OWL

RDF *(Resource Description Framework)* est vu parfois comme une spécification permettant l'expression de métadonnées. Souvent confondu avec le XML ou avec les flux RSS des sites d'actualités, RDF n'est pas toujours compris même par ceux qui sont assez familiers avec XML[4]. RDF commence à être pourtant une partie de plus en plus présente sur le Web. Le développement d'interfaces XUL (utilisé par Mozilla, Firefox par exemple) nécessite de connaître le RDF car celui-ci est utilisé pour la partie « model » (en parler MVC - Model/View/Controller) de ses éléments graphiques[5]. On peut se demander assez légitimement à quoi sert RDF, qu'est-ce qui fait l'unité de cette spécification utilisée pour des applications si diverses. Une bonne définition pour RDF, donnée sur le site officiel est celle de *langage général permettant la représentation de l'information sur le Web*. La définition est déficitaire pourtant par son manque de précision et par un certain excès de généralité — car il peut s'avérer difficile pour un débutant de comprendre précisément à quoi sert vraiment RDF à partir de cette description faute de lire un didacticiel ou la spécification même. De notre point de vue, RDF est intéressant pour la possibilité qu'il offre de représenter des propositions (Statements) et des taxonomies de concepts, bref, en tant que spécification de base pour la représentation des ontologies.

Pour le stockage de l'ontologie extraite nous avons utilisé la spécification OWL basée sur RDF. Il s'agit, plus ou moins, du standard établi pour le stockage des ontologies probablement pour plusieurs raisons. Une des raisons est celle que RDF et son supplément RDFS permettent l'expression de « triplets » ou *statements*, affirmations. Une autre est sans doute le fait que la spécification propose une façon standardisée de stockage en tant que XML ce qui a contribué à sa popularisation du faite de la nature indépendante de plate-forme de ce dernier.

Voici une manière de spécifier en RDF la date de naissance de la ressource #adrian :

```
<rdf :Description
  rdf :about=
  "http ://example.net/people#adrian">
   <s :born>June 18, 1976</ns :born>
</rdf :Description>
```

RDFS, ou *RDF Schema* est un vocabulaire de base qui permet d'exprimer en RDF des taxonomies (`rdf :type` permettait déjà l'hiérarchisation des concepts) ou bien des propriétés de base. Souvent RDFS est suffisant pour l'expression d'ontologies hiérarchisées et essentiellement taxonomiques.

OWL, version perfectionnée de son ancêtre, DAML+OIL, ajoute des niveaux supplémentaires de raffinement pour l'expression d'ontologies.

```
<owl :Class
```

[3] « Triplets » dénote une succession sujet - prédicat -objet, structure qui est à la base de RDF.

[4] Stefano Mazzochi, *A No-Nonsense Guide to Semantic Web Specs for XML People*, http ://www.betaversion.org /~stefano/linotype/news/57/

[5] Adrian Dimulescu, *Experimenting rich web clients with Mozilla and Cocoon*, http ://adrian.dimulescu.free.fr /article.php3 ?id_article=4

```
 rdf :about="#permissive_receptor">
 <rdfs :subClassOf>
  <novel_mechanism
    rdf :about="#receptor">
     <rdf :type>
       <owl :Class
         rdf :about="#ion_channel"/>
     </rdf :type>
     <rdfs :label
       xml :lang="en">receptor
     </rdfs :label>
...
```

La possibilité d'expression d'OWL au format XML est sans doute une manière très utile de consulter le contenu d'une ontologie par une personne, sans devoir forcément faire appel à un outil de visualisation complexe. D'autant plus qu'une manière encore plus simplifiée nécessitant moins d'écriture, appelée N-Triples, a été précisée. Elle ressemble à ceci :

```
:LXRalpha
   a :gene ;
   rdfs :label "LXRalpha"@en .
:integrin_complexe
   a owl :Class ;
```

Le stockage en XML lui-même est sujet à plusieurs manières de sérialisation[6]. Une façon immédiate serait le stockage en RDF « pur » celui dans lequel chaque `Statement` est représenté en tant qu'une balise `<rdf :Description about="...">`. Malgré les bienfaits apportés par sa standardisation, cette écriture est quand-même assez volumineuse et finalement difficilement lisible pour l'humain. Une notation plus dans l'esprit du balisage XML serait la suivante :

```
<target
 rdf :about="...#alpha_dystroglycan">
 <rdfs :label
   xml :lang="en"
   >alpha_dystroglycan
 </rdfs :label>
</target>
```

Cette écriture porte le nom de XML-ABBREV et a la signification suivante : la ressource indiquée par l'attribut XML `rdf :about` est de type *target* et elle a une propriété *rdfs :label* de la valeur indiquée. Cela reviendrait à plusieurs déclarations RDF, résumées grâce à cette notation familière.

Mais le stockage en tant que fichier XML ou N-TRIPLES pose des problèmes sérieux de passage à l'échelle. La raison en est qu'un fichier doit être chargé la plupart de temps dans son intégralité en mémoire afin d'effectuer, par exemple, une simple requête qui ramène un seul concept. Ceci constitue une menace importante pour la mise en pratique d'un projet car il se peut qu'un prototype fonctionne très bien en conditions de test seulement pour se voir invalidé au moment de l'utilisation effective à cause de la quantité de données à intégrer.

2.2 Jena

Jena (http ://jena.sf.net) est une librairie Java initiée par les laboratoires HP qui permet la manipulation de RDF et de OWL. Jena a support intégré de OWL par l'intermédiaire de la classe `OntModel` et des classes associées. De cette manière, l'utilisateur OWL de Jena peut utiliser une API dédiée et ne pas se retrouver obligé d'exprimer des constructions spécifiques OWL en utilisant des appels standards RDF.

Jena propose aussi, en dehors des modèles classiques « en mémoire », des modèles stockés en base de données, ayant le choix entre MySQL, PostgresSQL et Oracle. Ceci est un trait intéressant si l'ontologie à traiter devient trop importante pour être stockée en mémoire. Le coté moins séduisant est que Jena ne propose pas d'interface graphique web ou swing pour la gestion de ses ontologies ce qui rend son utilisation dépendante de l'écriture de code Java.

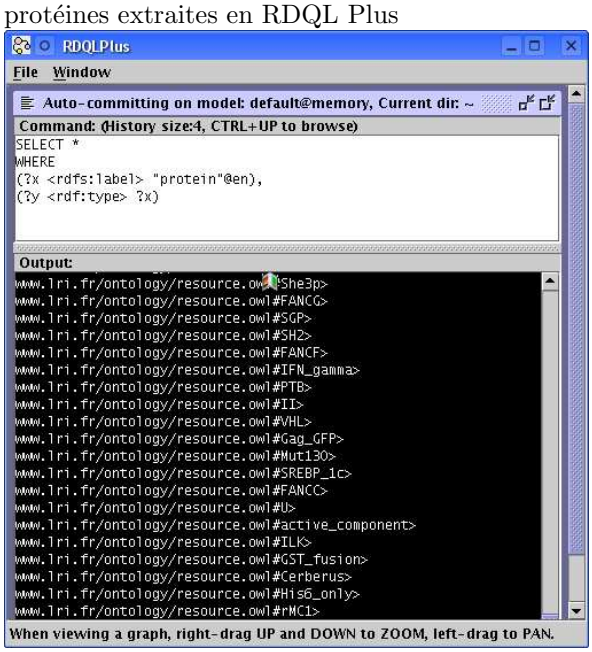
2.2.1 RDQL

Le langage d'interrogation préféré par Jena s'appelle RDQL *(RDF Data Query Language)*. RDQL fait partie d'une poignée de langages d'interrogation qui se disputent le rôle de standard pour l'interrogation du web sémantique, c'est à dire à peu près le rôle que SQL détient dans le monde des bases relationnelles.

RDQL essaye pour des bonnes raisons de ressembler autant que possible à SQL. Il garde le squelette de base `SELECT ... WHERE` renonçant à `FROM` probablement pour des raisons liées au schéma fixe de RDF.

[6] La sérialisation est l'écriture d'un modèle de données dans un flux persistant (comme un fichier).

FIG. 1 – Requête RDQL pour récupérer les protéines extraites en RDQL Plus



Voici un exemple, si on voulait récupérer les ressources RDF qui sont des sous-classes de la ressource etiqueté avec le mot « *protein* » on pourrait utiliser la construction suivante :

```
SELECT *
WHERE
  (?x <rdfs :label> "protein"@en),
  (?y <rdf :type> ?x)
```

L'exécution proprement dite de la requête peut se faire en environnement de programmation pour les spécialistes ou en utilisant RDQL-Plus (http ://rdqlplus.sourceforge.net/), une interface Swing open-source pour l'exécution de RDQL et visualisation graphique des résultats des requêtes. RDQL-Plus utilise Jena à l'intérieur. Une autre solution encore serait d'utiliser l'interface web de Sesame qui, comme détaillé par le sous-chapitre 2.4, supporte RDQL comme un des langages d'interrogation possibles.

2.3 Redland

Redland (http ://librdf.org/) est une librairie écrite en C qui permet la manipulation, sérialisation et désérialisation de RDF. Redland ne dispose pas de support OWL au niveau de son API. Par contre, Redland dispose d'une série impressionnante de *bindings* pour les langages de script les plus utilisés : Perl, Php, Python, Ruby et bien d'autres. Même si la documentation pour ces derniers laisse quelque peu à souhaiter, Redland fournit du coup un support RDF à quantité d'utilisateurs de l'environnement très répandu LAMP (Linux/Apache/Mysql/Php-Perl-Python) ce qui en fait un choix important.

2.4 Sesame

Sesame (http ://www.openrdf.org) est, d'une part, une alternative à Jena dans le sens où il propose une API de manipulation de RDF complète sans utiliser les bibliothèques Jena. D' autre part Sesame propose le support de certaines bases de données SQL ainsi qu'un langage de requête, SeRQL et une interface web conviviale bien qu'incomplète (permet l'importation et la visualisation des données mais non pas leur modification ou effacement).

2.4.1 SeRQL

Sesame, comme Jena, supporte aussi RDQL mais ce n'est pas RDQL qui est vraiment son point fort. Le langage d'interrogation de préférence proposé par Sesame s'appelle SeRQL (lire comme *circle* en anglais), c'est le langage dont les requêtes sont le plus optimisées.

La récupération d'un concept se fait selon sa forme lexicale. Ainsi, `http ://...#protein` a un libellé *(rdfs :label)* qui est « protein@en ». En parenthèse, la convention de Wordnet est différente, il existe une propriété de chaque concept appelée `wn :wordForm` qui contient une forme lexicale du concept (il peut y en avoir même plusieurs formes lexicales).

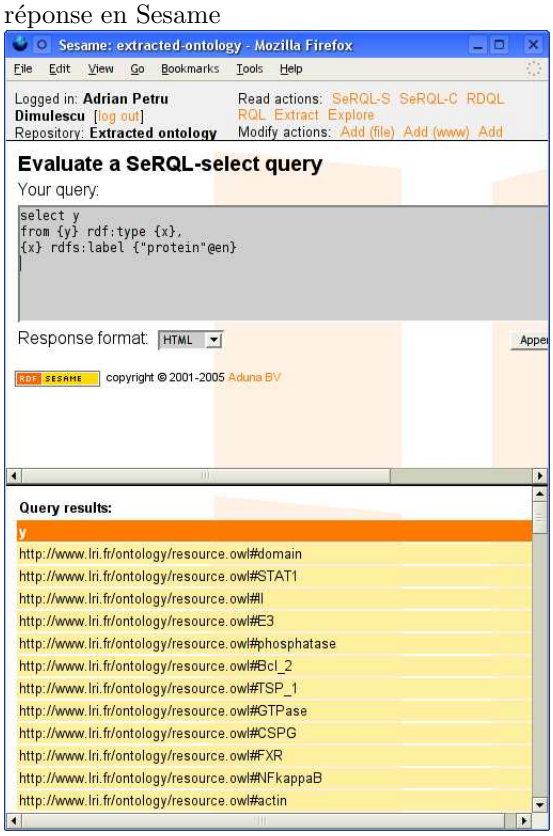
Voici une requête SeRQL qui récupère les instances et les sous-classes d'un concept étant donnée sa forme lexicale anglaise, « protein » :

```
select y
from
  {y} rdf :type {x},
  {x} rdfs :label {"protein"@en}
```

Sésame permet l'exécution de cette requête dans une application web qui tourne dans la même ma-

chine virtuelle Java que le serveur RDF proprement dit.

FIG. 2 – Requête SeRQL et la visualisation de la réponse en Sesame



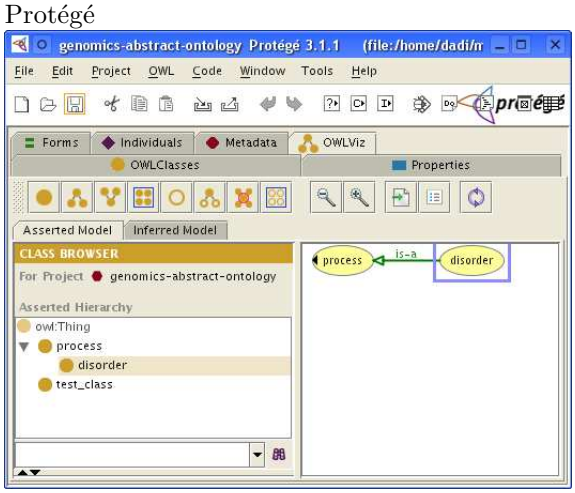
L'interface web proposée par Sésame permet aussi l'exploration de ressources RDF. Chaque ressource est un hyperlien et en cliquant dessus une page nous est présentée dans laquelle figurent toutes les relations dans lesquelles se trouve impliquée la ressource en question.

Une version de Sesame intégrée avec des facilités de contrôle des versions, de gestion de l'accès et sécurité, accès par SOAP, ainsi qu'une intégration avec OntoEdit est implémentée dans le cadre du projet OMM (Ontology Middleware Module), décrite en [3].

2.5 Visualisation d'ontologies

En général les grandes ontologies posent le problème de leurs accessibilité. Leur visualisation doit prendre en compte :

FIG. 3 – Edition et visualisation d'ontologies en Protégé



- la formulation de requêtes adéquates pour l'extraction de sous-ensembles intéressants,
- la mise en page de cette dernière pour une consultation facile par le non-spécialiste.

Le premier critère peut être atteint par la formulation attentive des requêtes en des langages comme RDQL ou SeRQL. Les résultats peuvent être sauvegardés et visualisés à l'aide d'utilitaires comme IsaViz (http ://www.w3.org/2001/11/IsaViz/), VisualRDF (visualrdf.sourceforge.net), OwlViz, plugin de Protégé basé sur GraphViz (www.graphviz.org).

2.5.1 Protégé

Mentionner Protégé seulement au chapitre « visualisation » est quelque peu une injustice car l'outil est plus complexe, néanmoins, c'est un des usages que nous lui avons donné.

Nous avons aussi utilisé Protégé pour ébaucher des petites ontologies de test ainsi que pour son plugin OwlViz qui permet la représentation graphique de OWL.

*

Afin de pouvoir d'utiliser ces outils du web sémantique dans la fouille de textes, nous avons cherché à utiliser des logiciels déjà fonctionnels. Nous en donns en compt-rendu par la suite.

3 Quelques outils de fouille de textes

3.1 Gate

Gate (http ://gate.ac.uk)[6] est un outil de traitement du langage naturel développé par l'université de Sheffield. L'outil est disponible en licence libre, les sources y comprises. Il consiste de quelques sous-systèmes que nous allons détailler par la suite.

3.1.1 Le cadre général Gate

Gate propose un cadre (framework) puissant de développement d'applications de fouille de textes qui permet la représentation abstraite des annotations, des couches d'annotations, des documents, des corpus de documents et des applications en *pipeline* qui s'exécutent sur ces corpus.

Les corpus des documents sont des ensembles de documents qui peuvent être stockés sur le disque ou bien dans une base de données SQL. Le support de stockage est transparent, les manipulations sont les mêmes dans tous les cas.

Un document Gate consiste principalement du texte proprement dit et d'une liste de couches d'annotations. Chaque couche d'annotations (AnnotationSet) contient un ensemble d'annotations qui peuvent s'enchevêtrer. Une Annotation a un nom et un ensemble de traits *(features)*. Chaque trait de l'annotation peut contenir tout type Java disponible ce qui permet de rajouter aux annotations, en dehors de leur noms, de l'information supplémentaire très utile.

Ainsi, une annotation qui marque une coréférence peut consister d'un nom descriptif ; comme ANAPHOR ayant un trait appelé *antecedents* contenant l'ensemble des annotations qui forment les antécédents de cette anaphore. Il peut avoir un autre trait appelé *type* qui spécifiera de quel type d'anaphore il s'agit (pronominale, sortale), d'une autre encore qui s'appelle *cost* qui contiendrait le temps de calcul de cette anaphore et ainsi de suite.

Le modèle Gate fournit donc une abstraction pour la représentation des corpus. Il implémente des facilités de manipulation des annotations plus avancées par rapport au système « classique » de type Brill. Ce dernier annote un corpus texte ASCII en rajoutant des étiquettes à la fin de chaque mot et en le séparant du mot par une barre oblique (« / »). Le système est particulièrement bien adapté pour POS-tagging (l'étiquetage des parts du discours), cas ou seulement une annotation doit être fait pour chaque mot. Mais s'il s'agit de rajouter des annotations pour des zones de texte plus étendues il faut recourir à des artifices comme le groupement de plusieurs mots dans un seul. Prenons comme exemple, la suite *smart/JJ little/JJ girl/NN*. Pour des raisons diverses, on peut vouloir passer un détecteur de groupes nominaux *(noun phrases)* qui détectera cette séquence entière en tant que groupe nominal ou terme. On souhaitera marquer toute la séquence avec une annotation NP. Si on reste dans la convention Brill, on peut utiliser comme artifice la notation suivante : *smart_little_girl/NP* (on unifie les 3 mots dans un seul). Le problème est qu'on perd ainsi l'information initiale sur l'étiquetage de chaque mot. Gate permet d'éviter cela, il permet le rajout d'un nombre indéfini d'annotations superposés.

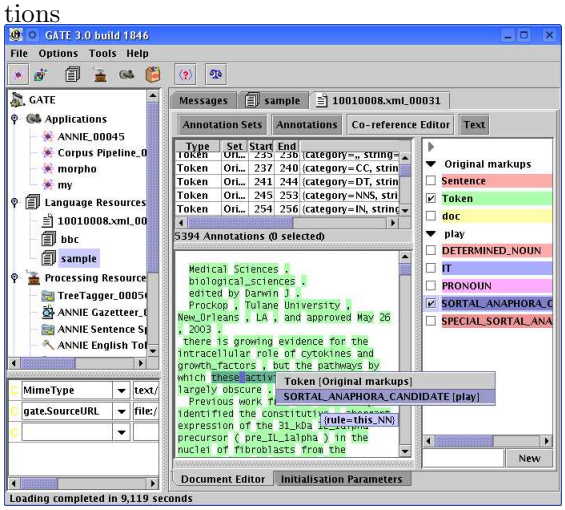
Gate peut également importer des documents HTML, Word, PDF, XML tout en gardant les annotations originelles.

Le désavantage majeur du modèle de document Gate dérive des traits même qui en font sa complexité : c'est son accessibilité moins facile. Alors que la modification ou la visualisation d'un corpus etiqueté « Brill » nécessite un simple éditeur de texte et un lecteur attentif pour distinguer le texte des étiquettes, un document Gate doit être visualisé à l'aide de son interface graphique, jolie et puissante certes, mais relativement gourmande en mémoire.

Un détail, pendant le développement nous nous sommes apperçu qu'il existe un bogue dans la version courante, 3.0, qui fait que l'interruption forcée d'une application Gate avant sa fin normale peut mener à la corruption du document sur lequel l'application s'exécute. Comme cette situation apparaît assez souvent en phase de développement il faut penser à sauvegarder son referentiel *(repository)* de corpus afin de pouvoir le restaurer si besoin.

Comment écrit-on une application en Gate afin de traiter ces documents ? Gate propose le concept de ressource de traitement *(processing resource)* qui constitue une unité de traitement. Ces unités peuvent être arrangées en *pipelines* de manières différente. Une telle *pipeline* est une application Gate. L'IHM Gate permet de rajouter et de suppri-

FIG. 4 – Gate : visualisation document et annotations



mer des ressources de traitement à une application, ainsi que leur configuration et exécution visuelle.

3.1.2 ANNIE

La distribution Gate vient avec un paquet de ressources de traitement appelé ANNIE (A Nearly New Information Extraction) qui permet des tâches habituelles de fouille de textes : segmentation en phrases, POS-tagging, dictionnaire de termes *(gazetteers)*, détection des termes *(noun phrases)*, des groupes verbaux, résolution de la coréférence pronominale et orthographique ainsi que d'autres.

Les ressources pré-existantes proposées par Gate contiennent aussi un certain niveau de support pour les ontologies[9]. Ainsi il existe des dictionnaires à support d'ontologies DAML+OIL qui permettent d'annoter des termes en utilisant des références vers des concepts compris par une ontologie. Aussi, on peut exporter certaines annotations dans un fichier au format DAML+OIL.

ANNIE est principalement conçu et testé sur des corpus de journaux. Le système ANNIE fonctionne mal sur des textes biomédicaux comme les nôtres. D'où le besoin d'interopérabilité avec d'autres systèmes, qui, eux, soient spécialement conçus pour les textes biomédicaux.

3.1.3 Jape

Jape est un langage proposé par la plateforme Gate qui permet la manipulation des annotations d'une manière relativement simple. Par exemple il permet la détection des anaphores sortales possibles comme des zones de textes qui commencent avec un déterminant démonstratif (*this*, par exemple) :

```
Rule :this_NN
(
 (  {Token.string == "this" }
  | {Token.string == "these" })
 ({Token.category == JJ})*
 (  {Token.category == NN}
  | {Token.category == NNS} )+
) :textzone
-->
:textzone.SORTAL_ANAPHORA_CANDIDATE
  = {rule = "this_NN"}
```

Jape peut être utilisé pour faire du réetiquetage aussi. Si, suite à un étiquetage défectuex on doit changer l'étiquetage de *that* de déterminant en conjonction de subordination au cas où il est suivi par un verbe, nous pouvons exécuter un script Jape ressemblant au suivant :

```
Rule :fix_badly_tagged_that
(
 (  {Token.category = WDT}
    {Token.string = "that"}
 ) :text_to_retag
 (Token.category = VBZ)
)
-->
:text_to_retag.Token = {
   category = IN,
   rule="fix_badly_tagged_that"
}
```

Le trait *rule* qui est rajouté à l'annotation `Token` n'est pas nécessaire, c'est uniquement une convention pour que les utilisateurs ultérieurs des annotations sachent quelle règle à ajouté cette annotation.

Si un script Jape utilise les mêmes constructions trop de fois, Jape propose un mécanisme de description de macros qui permettent la factorisation du code dupliqué.

3.1.4 Visualisation

Un des points à remarquer dans Gate est la composante de visualisation qui permet une consultation conviviale des annotations à l'aide de couleurs différentes pour chaque type etc. L'annotation manuelle des textes est rendue aussi relativement simple. Il suffit de sélectionner une zone de textes à la souris et la marquer en tant qu'annotation d'un certain type. Gate peut proposer des annotations disponibles si on lui fournit un schéma des annotations. Un manuel d'utilisation pour les annotateurs est disponible sur le site gate.ac.uk.

3.1.5 Interopérabilité avec du traitement non-Gate

Une bonne partie du traitement effectué dans notre projet n'utilise pas le formalisme Gate. La transformation d'un format à l'autre se fait en utilisant la capacité d'importation de XML de Gate. Il est relativement simple d'écrire un script qui transforme le format « Brill » au format XML. Ainsi, la séquence etiquetée *girl/NN* serait transformée dans la balise XML `<Token category="NN" string="girl"/>`. Le fichier résultant sera importé en Gate ce qui permettra de retenir les annotations faites et de travailler dessus.

3.1.6 Limites et inconvénients

Même s'il dispose d'un éditeur visuel de coréférences, le support de Gate pour ceux-ci est marqué par une limite pénible pour notre expérimentation dans le sens où chaque anaphore est censée pointer vers un seul antécédent. Les antécédents multiples, cas fréquent dans la résolution des anaphores sortales ne sont pas supportés par l'IHM de Gate. Le système de manipulation d'annotations au coeur de Gate est capable de représenter cette référence multiple mais l'IHM existant ne pourra pas s'en servir dans sa version actuelle pour le rendre consultable aux utilisateurs.

Le langage Jape présente lui aussi certaines limitations. Parmi les plus importantes, on nombre l'impossibilité d'utilisation de caractères génériques *(wildcards)* sur les étiquettes, l'impossibilité de spécifier des conditions négatives (par exemple déclencher une règle si l'annotation n'est *pas* NN). Les deux limites sont contournables, la première par le réétiquetage successif et l'utilisation des *gazetteers*, la deuxième par l'écriture de règles plus prioritaires qui sont déclanchés en premier par la règle qu'on souhaiterait négative. Le déclenchement de cette règle supplémentaire fait en sorte que la règle initiale ne peut plus elle aussi être déclenchée.

Un inconvénient qui ne tient pas tellement de Gate en soi mais plutôt de la plate-forme Java dans laquelle Gate est développé consiste en ce que dans les milieux de fouille de textes il existe souvent une bonne expertise Perl, mais une moins fréquente familiarisation avec la plate-forme Java. Ce qui apporte un coût supplémentaire d'accoutumance à la nouvelle plate-forme. La maturité de Java ainsi que la grande quantité de logiciels open-source disponibles pour cette plate-forme en font pourtant un candidat envisageable pour complémenter Perl dans ce domaine.

3.2 CorTag

CorTag est une application développée par l'équipe IA du LRI qui permet le réétiquetage de textes au format « Brill » et aussi le déclanchement d'actions suite à la mise en correspondance *(matching)* de patrons sur les annotations. CorTag a beaucoup de points communs avec Jape en ce qu'il permet la manipulation facile des annotations et le déclenchement d'actions suite à la mise en correspondance de patrons sur les annotations. En effet, la configuration particulière des annotations est souvent un indice précieux quant aux actions à entreprendre pour détecter la structure sémantique du texte.

L'exécution du script Perl CorTag sur un fichier de règles produit d'autres scripts Perl qui représentent une version « compilée » des règles-sources. L'utilisateur exécute ces derniers scripts générés sur le corpus afin de démarrer le traitement.

Les règles de CorTag utilisent des séquences de triplets ayant en première position l'index relatif de positionnement du terme, en deuxième une condition sur la forme lexicale du terme et en troisième une condition portant sur son étiquette. Chacun de ces trois éléments du triplet peut manquer mais évidemment pas tous à la fois. Voici un exemple de règle écrite en CorTag qui cherche des anaphores sortales commençant par *this* ou *these* et qui sauvegarde le contexte des anaphores trouvées dans un fichier :

```
si (0,or(this,This,these,These),)
   (*1,,JJ)
   (*1,,JJ)
   (,or(gene,protein,genes,proteins),
     or(NN,NNS))
alors saveContext(0)
```

CorTag supporte au niveau de sa syntaxe la catégorisation des étiquettes ou des mots. Il permet d'écrire des règles qui matchent des listes d'etiquettes, non pas des étiquettes seules uniquement. Ainsi on peut construire des listes d'équivalence et utiliser la notation `@liste` au lieu de la simple étiquette en troisème place d'un triplet. L'utilité de ces listes est manifeste surtout dans le contexte des critères de catégorisation de la linguistique systémique fonctionnelle[8].

```
si (?,,@have)
   (*1,,ou(CC,VB,JJ,RB))
   (0,,VBN)
alors (0,,VBD)
```

Afin d'utiliser CorTag pour extraire des relations de généralisation du corpus, nous avons rajouté une fonction d'exportation de RDF qui utilise les bibliothèques Perl Redland afin de créer des modèles RDF et de les sérialiser. En voici un exemple d'extraction de relations de généralisation à partir d'une formulation textuelle de type *« A is a B »*.

```
si (-1,,NP) (0,is,) (+1,a,) (+2,,NP)
alors
 rdf_statement(-1, rdfs :subClassOf, +2)
 rdf_statement(+2, rdfs :subClassOf,
                     rdfs :Class)
```

3.2.1 Limites

CorTag utilise comme entrée des fichiers ASCII au format « Brill » ce qui amène les avantages et désavantages discutés de ce format. Il n'est pas possible d'avoir d'annotations superposées, par exemple. On ne peut pas annoter plusieurs sous-parties de la construction détectée à la fois (Jape le permettait, en utilisant un libellé affecté à la sous-partie en question). L'utilisation de CorTag reste quelque peu du ressort des proffessionnels informaticiens dans cette version. Comme les scripts générés utilisent des fonctions décrites dans le corps principal de CorTag, il faut soit tout exécuter dans le même répertoire (CorTag, scripts générés, corpus), soit avoir des connaissances sur le mécanisme d'inclusion de modules Perl afin d'exécuter CorTag dans un répertoire séparé ou l'on ne risque pas par exemple d'écraser les fichiers du programme par erreur.

4 Notre approche

La résolution de la coréférence fait partie d'une succession de traitements qui sont appliqués aux textes. Cette succession commence par des opération de nettoyage du texte, c'est-à-dire de mise en forme spéciale afin de permettre des traitements ultérieurs plus faciles. Suit une étape d'étiquetage en parties du discours qui utilise une version personnalisée de Stanford Log-linear Model Tagger [11]. L'extraction de termes a été faite à l'aide de EXIT (Extraction Itérative de la Terminologie) [13]. La résolution de la coréférence constitue une étape complémentaire à l'extraction de termes qui permet aux constructions anaphoriques de faire partie des termes extraits.

L'idée principale qui est à la base de notre algorithme de résolution de la coréférence réside en ce que les anaphores sortales utilisent le type d'une entité pour y faire référence. Ainsi, dans le texte :

> ...to study the interaction between *pre_IL_1alpha* and *necdin* , we analyzed the association of *these proteins*...

si le programme de résolution dispose déjà de l'information que « pre_IL_1alpha est une protéine » et que « necdin est une protéine », il pourra les choisir comme antécédents valides au détriment des autres antécédents candidats : *interaction* et *association*.

Mais comment disposer de cette information ? Nous avons utilisé le corpus fourni par TREC (450 MB au total, plus de 7000 articles) pour en extraire l'information intéressante.

4.1 Détection des anaphores

Une première tâche est la détection de l'anaphore sortale. En analysant les textes, on s'aperçoit que les anaphores sortales les plus fréquentes sont du type *the gene*, *this protein*, *both alleles*. Nous avons donc utilisé des patrons de surface sur un texte dont les parties du discours ont déjà été étiquetées

(POS tagged) afin de détecter des séquences composées d'un déterminant suivi par des adjectifs et finalement par un groupe nominal (DT JJ* (NN|NNP)+). Cette approche s'est avérée à extraire trop de faux candidats déterminés par *« the »* qui ne faisaient pas partie d'une coréférence. Les faux candidats étaient le plus souvent des constructions fortement déterminées (c'est-à-dire entourées par *« of »* ou par des adjectifs), ce qui nous mène à la conjecture qui reste à expérimenter que seulement les groupes nominaux faiblement déterminés sont contraints à faire référence en arrière. A remarquer aussi que pour ces tests nous n'avons utilisé aucune sélection sur les concepts présents dans le groupe nominal déterminé qui compose l'anaphore. Nous avons décidé de garder seulement les déterminants démonstratifs (*« this »*, *« both »*) dans un premier temps. Des patrons plus complexes comme par exemple la résolution de l'anaphore sortale « the » restent à être développés dans une seconde étape.

Dans la détection des anaphores nous avons orienté nos efforts vers une liste de concepts « intéressants » ; qui apparaissent dans l'anaphore de type *« this »* , et dont la résolution présente un intérêt particulier dans le contexte du concours de génomique TREC. Les concepts les plus fréquents dans l'absolu, récensés sur un segment du corpus sont, dans l'ordre : *this result*, *these data*, *this finding*. Il s'agit ici d'anaphores très complexes qui font référence à des constructions précédentes d'ordre *sémantique* et non pas *lexical* que nous avons décidé d'ignorer dans un premier temps. En revanche nous nous sommes concentrés sur *this protein*, *this cell*, *this gene*, *this region*, *this enzyme*, qui figurent eux aussi parmi les premiers, sur les positions suivantes et dont la résolution est plus utile pour notre but. L'expert en biologie a enrichi cette liste avec d'autres concepts moins évidents comme *activation*, *allele*, *antibody*, *assay* et autres.

4.2 Extraction d'instances

Nous avons remarqué que les textes mêmes contenaient quantité d'information nécessaire pour l'instanciation de l'anaphore sortale, et, qui plus est, configurée en des constructions lexicales relativement faciles à extraire. Une approche similaire est utilisée pour des textes littéraires par[2]. Une construction que l'on peut trouver assez souvent est celle du type *« the p37 and p40 proteins »* . Cette construction permet l'inférence que *p37* et *p40* sont des protéines, information que nous avons mémorisée dans une taxonomie pour une utilisation ultérieure. D'autres constructions lexicales que nous pouvons trouver dans le texte sont : *« the stress inducible proteins hsp60 and GRP94 »* ou *« CML is a clonal disorder »*. Outre l'information sur l'instanciation de l'information, ce dernier exemple permet aussi la récupération de l'information concernant des relations de généralisation : par exemple, *« stress inducible protein »* est une sous-classe de *« protein »* et *« clonal disorder »* est une classe de *« disorder »*. Afin de faciliter la tâche de catégorisation, l'expert en biologie a demandé aussi quelques règles qui permettront de trouver l'origine d'un gène spécifique (homme, souris etc.) et elle a proposé des patrons lexicaux qui permettent cette détection.

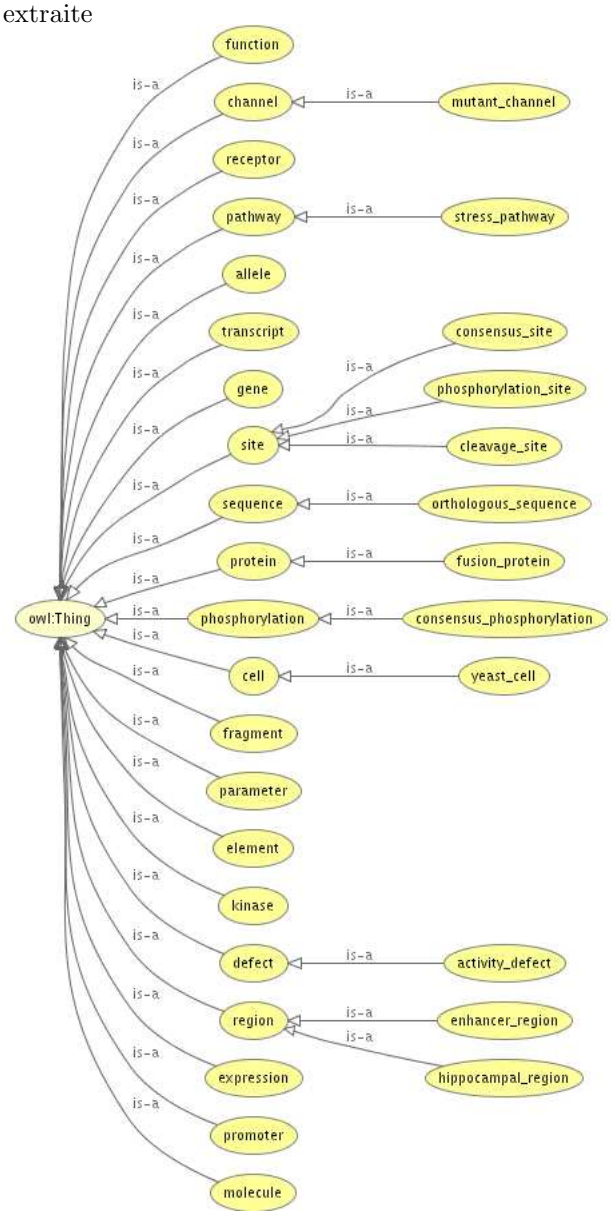
Avant d'avoir à disposition le corpus etiqueté nous avons tenté d'extraire des instances de protéines et gènes à partir du corpus brut, sans étiquettes. A l'aide d'expressions rationnelles nous pouvons détecter les instances qui ressemblaient à des composés chimiques ou biologiques par leur configuration spéciale (chiffres ou majuscules à l'intérieur du mot). Néanmoins, les résultats contenaient des erreurs et les expressions rationnelles construites, devenant de plus en plus complèxes, posaient un problème du point de vue de la maintenabilité et nécessitaient l'écriture de beaucoup de tests unitaires.

L'information taxonomique récupérée après avoir tourné toutes ces règles sur le corpus TREC a été stockée dans un format OWL. Le fichier OWL d'environ ˜30MB généré par le script de détection de patrons lexicaux est stocké dans un référentiel RDF Sesame stocké en MySQL 4.

4.3 Description de l'algorithme

L'anaphore sortale peut pointer en arrière vers plusieurs antécédents. Nous avons utilisé des contraintes quantitatives pour sélectionner le bon nombre d'antécédents : *both* ou *these two* indiquent exactement de combien d'antécédents il s'agit, mais pour *these* nous avons dû utiliser des heuristiques moins simples, préférant les antécédents dans la même phrase, ou s'il n'y en avait pas, les antécédents de la phrase précédente.

FIG. 5 – Visualisation d'une partie de l'ontologie extraite



La sélection des antécédents a été faite parmi tous les noms de la phrase qui précèdent l'anaphore-candidat. Pour l'instant nous ne nous sommes pas occupés de trouver les antécédents qui sont lexicalement impossibles, étape qui pourrait s'avérer utile[10]. En revanche nous nous sommes concentrés sur une sélection positive basée sur une combinaison entre l'information sémantique donnée par des instances que nous avons extrayées automatiquement avec les connaissances fournies par l'expert. Une mesure de saillance a été aussi utilisée pour affecter des scores aux candidates. Un score important a été assigné si le candidate est une sous-classe du concept contenu dans l'anaphore. Pour le cas particulier des gènes et des protéines, le fait que l'antécédent ne soit pas présent dans Wordnet le rend un candidat avec une forte chance de gagner.

4.3.1 Interrogation des ontologies

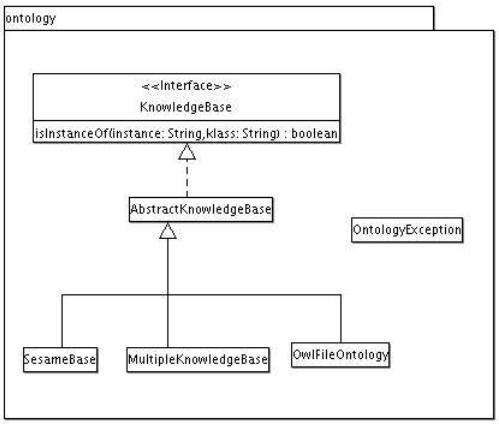
Le support d'ontologies est implémenté par le package Java `fr.lri.coreference.ontologies`. Une interface abstraite `KnowledgeBase` définit le contrat derrière lequel on peut utiliser n'importe quelle implémentation sans rendre l'application dépendente du package particulier choisi. Plusieurs implémentations sont disponibles, `OwlFileOntology` qui utilise Jena pour lire des fichiers OWL, avec le désavantage qu'elle prend assez longtemps à commencer (le temps de désérialiser le fichier d'entrée) ; une autre implémentation, `SesameBase` interroge une base Sesame stockée dans une base SQL avec le désavantage d'un temps d'interrogation supérieur à celui de `OwlFileOntology`. Afin de pouvoir combiner plusieurs ontologies disponibles, nous avons aussi implémenté une `MultipleKnowledgeBase` qui n'est rien d'autre qu'une liste d'autres `KnowledgeBase`s de base qui sont interrogées séquentiellement.

Un appel vers une telle classe se fait de la manière suivante :

```
KnowledgeBase mkb =
  new MultipleKnowledgeBase();
... // rajouter des KnowledgeBases
AssertTrue(
 mkb.isInstanceOf("leukemia",
             "process"));
```

Nous avons essayé ce système en espérant que

FIG. 6 – Classes de support d'ontologie



de cette manière nous pourrions combiner les ontologies extraites automatiquement — de tailles importantes, qui, du fait même de leur taille, doivent être stockées dans des bases SQL — avec des petites ontologies abstraites créées manuellement et stockées d'habitude en des fichiers. Nous avons rencontré le problème de la difficulté (et du temps d'exécution prohibitif) des inférences même simples. Par exemple, pour inférer que *leukemia* is a *process*, étant donné que la connaissance « *leukemia* is a *disorder* » est dans une base Wordnet en Sesame(SQL) et la connaissance « *disorder* est un *process* » se trouve dans une autre `KnowledgeBase` de type `OwlFileOntology`, un algorithme doit prendre itérativement toute l'ascendance du concept *leukemia* dans la première base est la tester contre *process* dans la deuxième. Si les deux ontologies étaient stockées en mémoire ce ne serait encore pas impossible à faire mais comme chaque connexion à une base Sesame se traduit par une requête HTTP suivie par un requête SQL, il faut songer à épargner le nombre d'appels vers Sesame. Un algorithme moins naïf pourrait le faire probablement en récupérant d'un coup toute l'ascendance du concept *leukemia* dans la mémoire ; nous avons considéré plus simple de mélanger toutes les données dans une seule base et utiliser le support d'inférence de la base pour récupérer ce genre d'informations.

4.4 Approche hybride

Nous avons complété notre taxonomie génomique spécifiques extraite avec des versions RDF de Wordnet importées dans notre base Sesame afin d'assurer aussi une couverture des concepts de base de la langue mais les connaissances expertes s'avèrent tout de même importants. L'exemple suivant illustre cette approche hybride. Etant donné le texte :

> *...chronic myelogenous leukemia is a myeloproliferative disorder that, over time, progresses to acute leukemia. Both processes...*

nous voudrions résoudre *both processes* en inférant le fait que *leukemia* est un *disorder* est que *disorder* et un *process*. En explorant le support ontologique nous avons trouvé de la connaissance sur le fait que *leukemia* est un *disorder* mais, dans la version 2.0 de Wordnet, *disorder* n'apparaît pas comme étant un *process*. Même si cette connaissance n'est pas une relation évidente dans le raisonnement humaine normale – pour lequel est conçu Wordnet – elle est néanmoins une information nécessaire dans le domaine technique de la génomique. La solution manifeste de ce problème est d'ajouter manuellement la relation manquante dans une ontologie abstraite : la généralisation de la relation entre les concepts process et disorder.

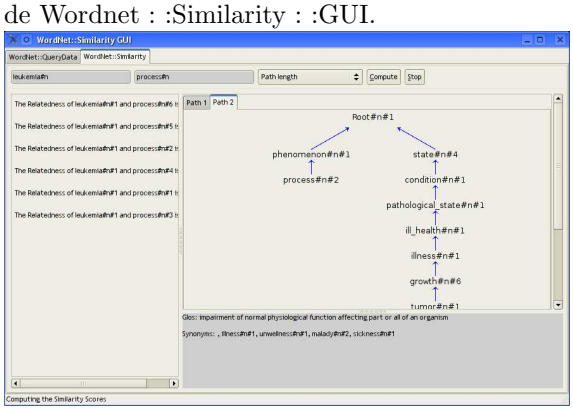
4.4.1 Mesures de similarité en Wordnet

Nous avons besoin souvent d'une métrique de similarité dans des ontologies. Différents algorithmes de calcul des distances entre des mots compris dans Wordnet ont été élaborés[5]. La similarité basée sur Wordnet donne de bons résultats parfois (par exemple ; dans le texte :

> We found that *the N_term extended segment* (KRPT(21)S(22)) of HMG1_Box_A is rapidly and specifically phosphorylated by cAMP-dependent protein kinase (PKA) in vitro. The phosphorylation in *this region*...

l'anaphore *this region* est correctement rapprochée de l'antécédent *segment ;* ce n'est pas toujours le cas pourtant — par exemple, dans le texte donné dans 4.4, le rapprochement *process* - *leukemia* n'est pas bien détecté, l'algorithme préférant le concept

FIG. 7 – Distance Wordnet entre les concepts *leukemia* (branche droite dans l'arbre) et *process* (gauche). On remarque qu'il faut passer par le noeud racine pour créer un chemin entre les deux concepts ce qui revient à ne pas trouver de rapport de généralisation entre eux. Capture d'écran de Wordnet : :Similarity : :GUI.



« *time* » qui se trouve en occurrence parmi les antécédents.

Les mesures de similarité sont relativement diverses, allant du simple comptage du chemin le plus court entre deux concepts jusqu'à des métriques de similarité entre les descriptions lexicales (la définition du dictionnaire) d'un mot. Ces dernières mesures sont loin d'être instantanées. Il faut aussi prendre en compte le fait que la comparaison des deux « mots » en Wordnet implique souvent la comparaison de toutes les combinaison entre les synsets correspondants aux deux formes lexicales. Ainsi par exemple *leukemia* fait partie d'un seul synset mais *process* apparaît en 6 synsets. La situation s'aggrave au fur et à mesure que l'on utilise des mots plus habituels et plus polysémiques.

Il n'y a pas beaucoup d'associations qui sont particulièrement liés à la génomique et il paraît adéquate de les construire manuellement dans un environnement visuel comme Protégé. Pas seulement est-ce une tâche qui est humainement faisable mais en plus l'activité de l'expert est réduite au raffinement des tentatives échoués de résolution.

Les limites de notre expérimentation se retrouvent principalement dans le fait que nous avons uniquement les relations de généralisation en nous appuyant sur des ontologies organisées en tant que taxonomies. Nous avons l'intention d'éteindre notre algorithme en utilisant des interactions fonctionnels et des propriétés entre concepts qui dépassent la simple généralisation.

4.5 Lemmatisation

La lemmatisation est nécessaire pour l'analyse des différentes formes fléchies de mots. Nous avons eu besoin surtout de la lemmatisation des noms en des situations comme la résolution d'une anaphore sortale du type *these proteins*. Afin de pouvoir identifier le concept dans une ontologie existante il faut d'abord en isoler la version au singulier, *protein* suivi d'une recherche dans la base pour le concept selon sa forme verbale. La lemmatisation ne se limite pourtant pas à la résolution des noms, loins de là, la lemmatisation des verbes pose en Anglais probablement plus de problèmes que ceux des noms. Un lemmatiseur relativement matur utilisable et disponible en accès public est Morph [7]. Une autre solution serait d'interroger Wordnet qui englobe un lemmatiseur.

Pour la simple tâche de détection du nombre d'une forme nominale ou bien pour la tâche de récupérer la forme singulière d'un pluriel, nous avons utilisé souvent un « lemmatiseur » naïf qui enlève simplement le *s* final faisant attention a quelques cas plus ou moins rares comme *ies* => *y* ou le pluriel en *es* ou en *en*. Ce simple lemmatiseur est utilisable pour des textes techniques comme ceux de génomique et probablement moins pour l'expression plus libre et variée de textes de journaux ou littéraires.

4.6 Expériments sur des textes TREC Novelty 2004

Un premier domaine d'essai avec la résolution de la coréférence en début du stage a été les textes de journaux pris du corpus TREC Novelty auquel notre équipe a participé en 2004.

Un logiciel de test a été construit pour la résolution d'anaphores dans un texte de « laboratoire » comme par exemple :

> Einstein is smarter than Stalin while the dictator is more powerful than the scientist.

Une ontologie créée à la main a été utilisée dont les concepts sont présentés dans la figure 8. Les ins-

tances en sont EINSTEIN qui est un « Scientist » et STALIN qui est un « Human » mais qui se retrouve dans une relation appelée « dictator » avec la RUSSIE. La RUSSIE est une instance du concept « Country ».

Le programme utilise soit une relation de généralisation soit une relation quelconque pour la résolution de la coréférence. Ainsi, *the scientist* est résolu à EINSTEIN à cause de la relation taxonomique alors que *the dictator* est résolu à STALIN à cause du fait que STALIN est en relation *dictator* avec l'instance RUSSIE. La propriété *dictator* de Stalin en représente en fait un *rôle*. De cette manière STALIN peut se retrouver en tant qu'instance de plusieurs « classes » ou « rôles ». Par exemple, il peut être un *dictator* par rapport à la Russie mais aussi un *husband* par rapport à sa femme ou bien un *consummer* par rapport au système économique. L'utilisation de rôles au lieux de classes mériterait peut-être d'être plus étudiée comme solution à la rigidité des taxonomies. Un parallèle se trouve dans les interfaces dans la programmation orientée-objet. Une interface est un contrat, un rôle qu'une classe peut jouer. Comme l'on sait, le concept d'interface (ou rôle dans notre jargon) permet l'inhéritance multiple en Java, c'est-à-dire des comportements multiples.

4.6.1 Instanciation en Wikipedia

Le même problème se posait, celui de construire une ontologie qui permettrait, par exemple, la résolution, dans le texte :

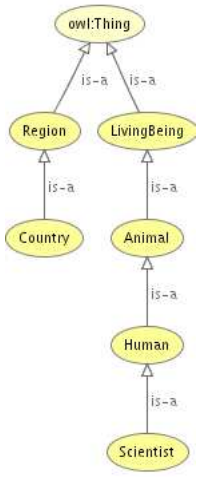
> Human rights campaigners have accused *the general* of exaggerating his health problems to try to win the sympathy of the courts.

de *the general* à *Pinochet*. Une solution explorée a été celle de récuperer la fonction d'une entité nomée (Pinochet dans ce cas) à l'aide de Wikipedia.

Les articles de Wikipedia ont tendance à commencer leur description par des phrases de type « is a », c'est-à-dire par des phrases qui décrivent le sujet en question d'une manière relativement facile à extraire.

Par exemple, le 30 août 2005, l'article décrivant Augusto Pinochet (http ://en.wikipedia.org/wiki/Augusto_Pinochet) :

FIG. 8 – Hierarchie de concepts pour l'expérimentation de la résolution de coréférence. Les relations et les instances des concepts ne sont pas visibles.



> General Augusto José Ramón Pinochet Ugarte (born November 25, 1915) was *head of the military government* that ruled Chile from 1973 to 1990.

ce qui permet l'inférence Pinochet « is a » head of military government. Ceci, combiné avec des connaissance extérieures statuant que le chef d'un gouvernement militaire est un général pourrait permettre la resolution : *general - Pinochet*.

Similairement, Joseph Stalin est décrit de la manière suivante :

> Joseph Stalin, (December 21, 1879 - March 5, 1953) was the leader of the Soviet Union from mid-1920s to his death in 1953 and General Secretary of the Communist Party of the Soviet Union (1922-1953), a position which had later become that of party leader.

ce qui permet d'extraire l'information *Stalin* est *leader* de *Soviet Union*.

5 Conclusions et perspectives

L'empêchement principal à l'aboutissement d'un taux de détection important d'antécédents a été

principalement dû à la difficulté que nous avons rencontrée dans la manipulation de la base Sesame en ce qui concerne la suppression des éléments faux de la base de connaissance.

Du fait des imperfections des règles de surface ainsi que de l'étiquetage provisoire du corpus sur lequel nous avons exécuté l'extraction des instances, il est arrivé que de la fausse information soit extraite. Par exemple, le concept « association » figure comme une sous-classe du concept « protein » à cause d'une mauvaise correspondance sur le texte :

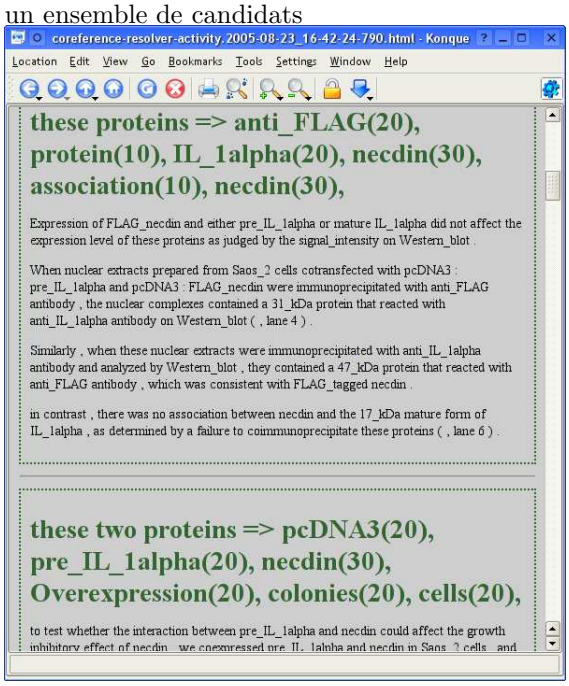
> the/DT surrogate/JJ light/NN chain-GAL1/NNP *association/NN is/VBZ a/DT direct/JJ protein_protein/NNP* interaction/NN

— le programme d'extraction d'instances a « compris » que *association* est une *protein_protein* donc, finalement une *protein*. L'ambiguïté est également générée par la transformation dictée par des besoins de nettoyage du tiret en blanc souligné. Ceci peut évidemment empêcher la résolution correcte de *this protein* dans le cas où il est précédé par *association*. Tant que la fausse information reste rare nous pouvons supprimer les entrées en question par des scripts ou manuellement. Même si on pourrait penser à résoudre cette mauvaise correspondance par la correction des règles utilisées, il n'est pas possible d'exclure la possibilité d'une erreur et donc l'exigence de pouvoir supprimer des connaissances dans la base de connaissances reste une priorité. Sesame ne permet pas la suppression de connaissances dans son interface web et nous sommes en train d'étudier l'utilisation de son API pour exécuter cette suppression ; une piste serait la réinsertion automatique de l'affirmation supprimée par le moteur d'inférence de Sesame à partir d'autres affirmations.

Nous croyons que c'est un empêchement à la fois imprevu et sans répercussions sur le fond de l'algorithme et qui devrait être réglé une fois que nous aurons compris plus en détail le fonctionnement interne de Sesame ou d'une alternative équivalente (le cas échéant Joseki, http ://www.joseki.org ou Kaon, http ://kaon.semanticweb.org aussi pourraient être utilisées à sa place).

Nous avons testé notre implémentation logicielle pour la résolution de coréférences de type *« this protein »* et *« this gene »* sur un petit sous-ensemble du corpus. Dans l'ensemble de tous les antécédents possibles, l'algorithme accorde des scores à un sous-ensemble de candidats afin d'en choisir les gagnants. Alors que la procédure responsable de l'isolation des antécédents dans cet ensemble de candidats doit encore être raffinée, dans 90% des cas, les candidats ayant un score non-nul incluent les vrais antécédents même si les antécédents ne sont pas explicitement isolés pour l'instant.

FIG. 9 – Exemples d'anaphore accompagnées par un ensemble de candidats



La résolution de ces problèmes techniques est une priorité afin de mettre en valeur l'utilisation de connaissances pour la résolution de coréférences.

La principale évolution de l'algorithme devrait à notre sens corriger le caractère purement taxonomique des relations utilisées. Nous envisageons une étape prochaine qui utiliserait une gamme plus large de relations entre concepts ainsi que de rôles joués par les concepts. Ainsi, la présence d'une anaphore de type *« this gene inhibits »* devrait pouvoir indiquer au programme un critère de sélection des antécédents qui ne relève pas uniquement de la généralisation, mais qui prenne en compte aussi la propriété *« inhibits »*.

Références

[1] A. Amrani, J. Azé, T. Heitz, Y. Kodratoff, and M. Roche. From the texts to the concepts they contain : a chain of linguistic treatments. In *In Proceedings of TREC'04 (Text REtrieval Conference), National Institute of Standards and Technology, Gaithersburg Maryland USA, 2004*, pages 712–722, 2004.

[2] E. Alfonseca and S. Manandhar. Improving an ontology refinement method with hyponymy patterns. In *Language Resources and Evaluation (LREC-2002)*, 2002.

[3] Atanas Kiryakov, Borislav Popov, and Damyan Ognyanov. OMM integration. 2002.

[4] Borislav Popov, Atanas Kiryakov, Dimitar Manov, Angel Kirilov, Damyan Ognyanoff, and Miroslav Goranov. Towards Semantic Web Information Extraction. 2003.

[5] Alexander Budanitsky and Graeme Hirst. Semantic distance inWordNet : An experimental, application-oriented evaluation of five measures. 2001.

[6] H. Cunningham. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36 :223–254, 2002.

[7] Guido Minnen, John Carroll, and Darren Pearce. Applied morphological processing of english. *Nat. Lang. Eng.*, 7(3) :207–223, 2001.

[8] M.A.K Halliday and Christian M.I.M Matthiesen. *An Introduction to Functional Grammar*. London : Arnold, 2004.

[9] Kalina Bontcheva, Atanas Kiryakov, Hamish Cunningham, Borislav Popov, and Marin Dimitrov. Semantic web enabled, open source language technology. 2003.

[10] Christopher Kennedy and Branimir Boguraev. Anaphora for everyone : pronominal anaphora resoluation without a parser. In *Proceedings of the 16th conference on Computational linguistics*, pages 113–118, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[11] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. pages 252–259, 2003.

[12] Ryan Lee. Scalability report on triple store applications. 2004.

[13] M. Roche, T. Heitz, O. Matte-Tailliez, and Y. Kodratoff. Exit : Un système itératif pour l'extraction de la terminologie du domaine à partir de corpus spécialisés. In *Proceedings of JADT'04 (Journées internationales d'Analyse statistique des Données Textuelles)*, volume 2, pages 946–956, 2004.

[14] Marin Dimitrov, Kalina Bontcheva, Hamish Cunningham, and Diana Maynard. A light-weight approach to coreference resolution for named entities in text. 2002.

[15] Ruslan Mitkov. Anaphora resolution : the state of the art. 1999.