Centre de Recherche en Informatique
Ecole des Mines de Paris
35, rue Saint-Honoré
77305 FONTAINEBLEAU

E/271/CRI

Rapport d'activité

Analyse et développement d'un outil de filtrage de données.

Directeur de thèse :
Robert Mahl (Ecole des Mines)
Katarzyna Wegrzyn-Wolska (ESIGETEL)

Auteur :
Grzegorz Dziczkowski

Laboratoire d'Accueil :
ESIGETEL

Table des matières

Remerciements

Je tiens a remercier tout particulièrement Mme Katarzyna Wegrzyn-Wolska enseignante à ESIGETEL et M. Robert Mahl professeur à l'Ecole des Mines, mon directeur de thèse.

A M. Eric Laporte, professeur à l'Université de Marne-la-Valée, pour ses précieux conseils.

A M. Zygmunt Vetulani, professeur à l'Université de Adam Mickiewicz de Poznan pour me fournir le dictionnaire des mots polonais sans le quelle ce travaille ne pourrait pas être effectuer.

A l'ensemble de l'ESIGETEL, pour m'avoir accueilli pendant la cette anée.

Enfin j'exprime toute ma gratitude envers les enseignants, administratifs, étudiants en thèse et autres stagiaires pour leur accueil cordial et leur bonne humeur.

INTRODUCTION

Durant mon premier année de thèse à l'Ecole des Mines j'ai commencé à travailler dans une domaine de linguistique informatique. Mon laboratoire de recherche se trouve à l'ESIGETEL (Ecole Supérieur d'Ingénieurs en Informatique et Génie des Télécommunications).

Mme Katarzyna Wegrzyn-Wolska m'a proposé de travailler sur la recherche d'informations en utilisant les connaissances linguistiques que j'ai eu l'occasion d'apprendre durant ma formation en DEA.

L'outil développé sera dans le futur utilisé dans le système d'expertise de l'intelligence économique et de la veille technologique pour la recherche d'informations dans un domaine de connaissance précis. J'aurai donc la possibilité de continuer ce travail dans le cadre de ma thèse a l'Ecole Des Mines de Paris.

Ce rapport d'activité est donc là pour vous présenter le travail que j'ai pu effectuer pendant premier année de ma thèse. Il est organisé de la manière suivante : la première partie présente un état de l 'art de la recherche en linguistique dans le domaine de la recherche d'informations, ainsi que l'application sur laquelle j'ai travaillée qu'est l'Unitex. Dans la deuxième partie, je présenterai la problématique de mon projet avec les réponses que j'ai pu apporter. La problématique est divisée en deux parties : la partie linguistique et la partie informatique. Je terminerai enfin par la présentation des différents aspects qui nécessitent des recherches plus approfondies, et les pistes permettant de les mener à bien.

PREMIERE PARTIE

ETAT DE L'ART

1.1 Les systèmes de compréhension de textes

Mon travail se situe dans le domaine de filtrage et d'extraction d'informations. L'extraction d'information consiste à identifier de l'information bien précise d'un texte en langue naturelle et à la représenter sous forme structurée [18]. Elle consiste à la recherche documentaire, qui vise à retrouver dans une corpus un ensemble de documents pertinents au regard d'une question [2].

L'extraction nécessite des lexiques et des grammaires spécialisées. La mise au point de telles ressources est une tâche longue et fastidieuse, qui demande le plus souvent, une expertise du domaine abordé et des connaissances en linguistique informatique. Parmi ces connaissances, on peut citer les techniques de filtrage, de catégorisation de document et d'extraction d'informations.

Au départ, le développement du domaine linguistique concerne les systèmes de compréhension traditionnels. La compréhension de textes est un domaine qui est exploré depuis le début du Traitement Automatique des Langues [3]. Dans les années 70, sont apparus les systèmes « KWIC » qui effectuent la recherche statistique des mots les plus significatifs [4]. Dans les années 80, des systèmes plus perfectionnés pour l'interrogation de bases de données en langage naturel ont vu le jour. L'exemple d'un de ces systèmes est le système « Lunar ». Grâce a ce dernier, les géologues pouvaient interroger en anglais la base des minéraux collectés sur la lune après le retour des missions Apollo [5].

Les systèmes de compréhension de texte ont, pour la plupart, été conçus comme des systèmes génériques de compréhension, mais ils se sont révélés peu utilisables dans des applications réelles. La compréhension est vue comme une transduction qui transforme une structure linéaire, c'est-à-dire que le texte (ie. la structure linéaire) est transformé en une représentation logico-conceptuelle intermédiaire qui est ensuite utilisée pour faire des inférences comme par exemple répondre à des questions.

Pour comprendre l'ensemble du texte il faut effectuer l'analyse syntaxique et l'analyse sémantique. L'analyse syntaxique est la plus large possible a cause des ambiguïtés. L'analyse sémantique vise a produire une structure représentant le plus fidèlement possible l'ensemble de la phrase, avec ses nuances et sa complexité, puis a intégrer l'ensemble des structures produites en une structure textuelle. A la fin, on obtient une représentation logico-conceptuelle du texte. La représentation sémantique varie d'un système a l'autre. On peut voir dans le système « *Core Language Engine »* des formes dites logiques inspirées en partie de la grammaire de Montague [6]. Dans le système « *Kalipsos »,* la représentation sémantique est effectuée par les graphes conceptuels [7] alors que dans le système « *Acord »* possède des structures de représentation discursive [8]. Les structures semantico-conceptuelles peuvent être plus ou moins larges, riches et complexes, plus ou moins ambiguës.

L'adaptation de ces systèmes pose le problème classique de la réutilisation des systèmes et des bases de connaissances qu'ils intègrent. L'adaptation d'une nouvelle tâche à un nouveau domaine nécessite la reconstruction d'une grande partie des bases de connaissances notamment le lexique sémantique.

1.1.1 Solutions proposées

L'échec relatif des systèmes de compréhension générique est aujourd'hui bien connu. Il faut cependant rappeler que ces systèmes issus des travaux de traitement automatique des langues des années 1980 ont réellement permis d'explorer cette approche générique de la compréhension de texte. Les chercheurs essayent d'avoir des dictionnaires électroniques relativement complets avec la syntaxe et la sémantique.

Ceci a poussé un grande nombre des chercheurs à décrire les langues naturelles de la même façon que les langages formels. Maurice Gross entreprit avec son équipe du LADL l'examen exhaustif des phrases simples du français, afin de disposer de données fiables et chiffrées sur lesquelles il

serait possible de faire des expériences scientifiques rigoureuses. Pour cela, chaque verbe fut étudié de manière a tester s'il vérifie ou non des propriétés syntaxiques comme le fait d'admettre une proposition complétive en position sujet. 6000 verbes ont été examinés à l'aide d'environ 300 propriétés. Le résultat est que pour 6000 verbes, nous avons environ 15000 emplois différents, qui présentaient un comportement syntaxique différent. On s'aperçoit qu'on ne peut pas décrire le français avec des règles générales. La même situation vaut pour toutes les autre langues. Les résultats de cette étude ont été codées dans des matrices appelées tables de lexique-grammaire. La table montre une description précise du comportement syntaxique de chaque verbe du français. L'objectif est d'utiliser toutes les ressources des tables lexique-grammaire pour obtenir un système capable d'analyser n'importe quelle structure de phrase simple.

L' unité minimale de sens, d'après Maurice Gross, est la phrase, et non le mot. Le principe est donc d'étudier les transformations que les phrases simple peuvent subir. Les phrases simple ont été indexée par leurs verbes. Pour un verbe on peux avoir plusieurs emplois différents. C'est grâce a des propriétés syntaxique qu'on peux distinguer les emplois d'un verbe. Il n'existe pas deux verbes possédant exactement le même comportement syntaxique. On ne peux donc pas formuler des règles générales qui pourraient expliquer la langue.

Il existe deux solutions pour le traitement automatique des langues. La première solution consiste à utiliser les systèmes statistiques. Les descriptions de ces techniques sont dans [9]. La seconde solution, construite à partir de systèmes basés sur des ressources linguistiques, donnent de meilleurs résultats.

1 .2 Le système UNITEX

J'ai travaillé pendant sur l'application Unitex. L'application a été crée au (Laboratoire d'Automatique Documentaire et Linguistique (LADL) sous la direction de M. Maurice Gross. L'auteur de cette application est M. Sébastien Paumier.

L'application Unitex est basée sur les outils linguistiques comme AGLAE [10] et INTEX [11]. Unitex [12], [13] est un environnement de

développement utilisé pour construire des descriptions formalisées a large couverture des langues naturelles et appliqué a des textes de taille importante en temps réel. Les descriptions des langues naturelles sont formalisées sous la forme de dictionnaire électroniques, de grammaires représentées par des graphes à nombre fini d'états et de lexiques-grammaires. Unitex permet de traiter en temps réel des textes de plusieurs mega-octets pour l'indexation de motifs morpho-syntaxiques, la recherche d'expressions figées ou semi-figées, la production de concordances et l'étude statistique des résultats.

1.2.1 Les dictionnaires

Les dictionnaires électroniques utilisés par Unitex utilisent le formalisme des DELA (Dictionnaires Electronique du LADL). Les dictionnaires électroniques décrivent les mots simples et les mots composes d'une langue en leur associant un lemme avec une série de codes grammaticaux, sémantique et flexionnels. Les dictionnaires ont été élaborés par des équipes de linguistes pour plusieurs langues comme le français, l'anglais, le grec, l'italien, l'espagnol, l'allemand, le thaïlandais, le coréen, le norvégien, le portugais. On peut distinguer deux sortes de dictionnaires électroniques. Les premiers, les dictionnaires de formes fléchies, qui est le type le plus utilisé, sont le DELAF (DELA de formes fléchies) et le DELACF (DELA de formes composées fléchies). Les programmes d'Unitex ne font pas de distinction entre les dictionnaires de formes simples et composées. Les deuxièmes, qui sont le dictionnaires de forme non fléchie, sont le DELAS (DELA de forme simple) et le DELAC (DELA de forme composées).

Pour associer un lemme et une information linguistique à un dictionnaire de formes non fléchies, on utilise des transducteurs lexicaux. Par exemple, dans le cas où il faut reconnaître 5000 chiffres romains, il est impossible de construire un dictionnaire DELAF. Il est plus simple de construire le transducteur correspondant a l'aide de 5 graphes simples [11].

Maintenant, nous allons présenter le format des dictionnaires. Le dictionnaire DELEF d'une langue contient toutes les formes fléchies de la langue et les associe au lemme. En plus on a le code morpho-syntaxique et éventuellement les codes syntaxique, sémantique et flexionnel. Voici l'exemple d'entrée du DELAF :

avions,avion.N+CONC:mp

La première ligne représente le fait que la forme *avions* est associée au lemme *avion.* Puis la lettre N signifie que c'est un nom. CONC nous apporte l'information que la classe distributionnelle est Concret. A la fin, nous avons un code flexionnel :mp qui représente le masculin pluriel. Nous pouvons trouver plus de détails dans [12]. Le dictionnaire DELAF contient toutes les formes fléchies du français, donc environ 680.000 formes fléchies différentes. Pour le dictionnaire DELACF, la seule différence est que la forme fléchie et le lemme peuvent contenir des séquences de lettres et de séparateurs. Voici un exemple d'entrée du dictionnaire DELACF :

Pomme de terre, pomme de terre,N+NDN+Conc:fp

Le DELACF contient 250.000 formes de noms composes, 8.000 adverbes figés, 15.000 formes figées utilisées avec le verbe *être* et 1 600 conjonctions de subordination.

Le format des DELAS est similaire a celui des DELAF. La différence est qu'il ne donne qu'une seule forme canonique suivie de codes grammaticaux et sémantiques. Voici un exemple :

Cheval,N4+Anl

Le premier code est interprété par le programme de flexion comme le nom de la grammaire à utiliser pour fléchir l'entrée. L'entrée de l'exemple ci-dessus indique que le mot *cheval* doit être fléchi avec une grammaire nommée N4[12].

1.2.2 Le réseaux des transitions récursives

Les grammaires sont des représentations de phénomènes linguistiques par de transitions récursives (RTN), un formalisme proche de celui des automates a états finis. De nombreuses études ont mis en évidence l'adéquation des automates aux problèmes linguistiques. Un transducteur a nombre fini d'états est un graphe qui représente un ensemble de séquences en entrée, et leur associe des séquences produites en sortie. Généralement une grammaire représente des séquences de mots et produit des informations linguistiques comme par exemple des informations sur la structure syntaxique. Un dictionnaire représente des séquence de lettres et produit les informations lexicales associées. Le transducteur d'un texte représente les séquences de mots qui forment chaque phrase et leur associe des informations lexicales ou syntaxiques – des résultats produits par les différentes analyses. Les grammaires sont représentées au moyen de graphes que l'utilisateur peut créer et mettre a jour. L'application des dictionnaires à un texte consiste a construire l'union des transducteurs de chaque dictionnaire, et à construire l'union de ce transducteur avec le transducteur du texte.

Les corpus de texte sont représentés par des automates, dans lesquels chaque chemin correspond a une analyse lexicale. Les phénomènes linguistiques sont représentés par les grammaire locales, qui sont traduits en automates finis afin d'être aisément confrontés avec les corpus de texte

Une grammaire locale [14] est une représentation par automate de structures linguistique difficilement formalisables dans des tables de lexique-grammaire ou dans des dictionnaires électroniques. Les grammaires locales, représentées sous formes de graphes, décrivent des éléments qui relèvent d'un même domaine syntaxique ou sémantique. Les descriptions linguistiques décrites sous la forme de grammaires locales sont utilisées pour une grande variété de traitements automatiques appliqués sur les corpus de texte. Ainsi différentes méthodes de désambiguïsation lexicale ont été développés pour mettre en oeuvre des contraintes grammaticales décrites a l'aide de ce type de graphe.

Voici l'exemple de grammaire locale qui a trouvée une utilisation dans le système de filtrage d'informations CORAIL[15]. CORAIL est un moteur de filtrage d'informations intégrant des contraintes d'ordre linguistique par le

biais de dictionnaires, de grammaires locales, et de table du lexique-grammaire. Les grammaires locales présentent des propriétés intéressantes pour une application au filtrage d'informations.

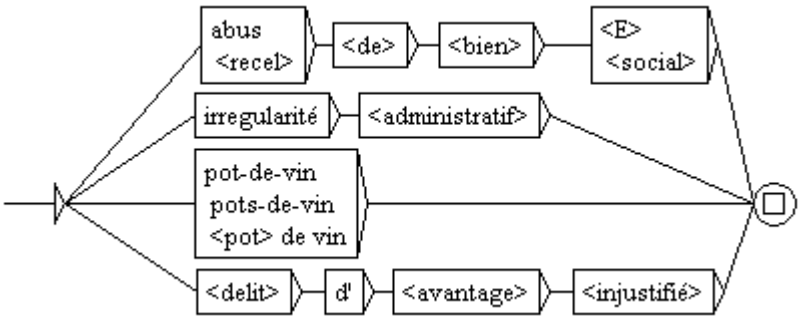


Figure 1. L'exemple d'une grammaire locale.

1.2.3 Les tables de lexique-grammaire

Ces tables lexique-grammaire, que nous avons précédemment définies dans cette partie sont des matrices décrivant les propriétés de tous le verbes simples du français dont elle décrivent les propriétés syntaxiques. Chaque mot ayant un comportement quasi unique, les tables donnent la grammaire de chaque élément de lexique, d'ou le nom de lexique-grammaire. On peut grâce a Unitex construire des grammaire a partir de telles tables. Le lexique-grammaire de Maurice Gross est une description systématique des propriétés syntaxiques et sémantiques des foncteurs syntaxiques du français, c'est a dire les verbes, les noms prédicatifs et les adjectifs. Il est organisé en groupes de tables, qui sont associés a une catégorie syntaxique donnée comme verbes pleins, verbes supports, noms, etc... Une table correspond a une construction syntaxique particulière et rassemble tous les mots qui entrent dans cette construction. Par exemple, la table 1 des verbes contient tous les verbes qui admettent en plus d'un sujet un complément infinitif mais pas un complément qui soit une complétive finie ou non. Une table est divisée en ligne selon les mots qu'elle contient et en colonnes selon les propriétés syntaxiques ou sémantique qui s'appliquent à ces mots et leurs arguments. A l'intersection d'une ligne et d'une colonne, un signe + ou – indique que la propriété indiquée en entête de la colonne s'applique

positivement ou négativement au mot placé en entête de la ligne. Cette propriété est soit un ajout d'information sur le mot ou un de ses arguments, soit une transformation du cadre de sous-catégorisation de base associée a la table. Actuellement le lexique-grammaire est surtout développé pour les verbes et les locutions prédicatives. Ce lexique contient actuellement 15.000 entrées de verbes simples. En outre, 25.000 locutions prédicatives ont été décrites, de même que 20.000 locutions construites avec ê*tre* ou *avoir*[16].

Bien qu'il soit aujourd'hui clair que le lexique est une composante essentielle des systèmes TAL, les ressources disponibles sont rares. Pour l'anglais, COMLEX Syntax [17] contient une information détaillée de 38.000 mots dont 6.000 verbes. VerbNet, lui, décrit 4.000 sens verbaux à partir de 191 classes sémantique et 52 cadres syntaxiques. Pour le français, plusieurs lexiques sont disponibles mais la plupart concernent la morphologie plutôt que la syntaxe. Ainsi le lexique LEFFF (Lexique des Formes Fléchies du Français) contient 5.000 verbes et 200.000 formes fléchies mais l'information associée est purement flexionnelle. Comme nous l'avons précisé, le lexique-grammaire de Gross contient une information détaillée et exhaustive et a été numérise par le Laboratoire d'Automatique Documentaire et Linguistique (LADL) et il est maintenant partiellement disponible sous une licence LGPL-LR. Tout ceci facilite la constitution d'une ressource lexicale appropriée au TAL.

Table 38LH

N0 source	*N0 destination*	*N1 V*	*N2 V N1*		*Pfx nég / source*	*Pfx nég /nv dest*	*N1 =: V-n*	*N0 V N2 (de N1)*	*source / destination*	*Prép =: de*	*autre Prép source*	*Prép =: dans*	*Prép =: sur*	*Prép =: contre*	*Prép =: à*	*Prép =: vers*	*N2 =: V-n*	*Ppv =: y*	*Ppv =: en*	*N1 est Vpp*	*N1 =: N-hum concret*	*mot Loc texte*	*idée Loc esprit*	*Nhum Loc Nabs*	*N1 =: Qu P*	
-	-	-	-	immiscer	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	+	+	+	+	+	-	*Max ~ sa soeur dans les affaires de Luc*
-	-	-	-	impliquer	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	+	-	-	-	+	-	*Max ~ Luc dans un scandale*
-	-	-	+	incarcérer	-	-	-	-	-	-	-	+	-	-	+	-	-	+	-	+	-	-	-	+	-	*On ~ Max à la prison de Dax*
-	-	-	+	incorporer	+	-	-	-	-	-	-	+	-	-	+	-	+	+	-	+	-	-	+	-	-	*On ~ Max dans la marine*
-	-	-	-	infiltrer	-	-	-	+	-	-	-	+	-	-	-	-	-	+	-	+	-	+	+	-	-	*Max ~ un agent dans ce réseau*
-	-	-	-	inhumer	-	-	-	-	-	-	-	+	-	-	-	-	+	+	-	+	-	-	+	-	-	*On ~ Max dans le cimetière*
-	-	-	-	inscrire	-	-	-	-	-	-	-	+	-	-	+	-	-	+	-	+	-	-	-	-	-	*Max ~ Ida dans un club de yoga*
-	-	-	-	interner	-	-	-	-	-	-	-	+	-	-	-	-	+	+	-	+	-	-	-	-	-	*On ~ Max dans un asile*
-	-	-	-	introduire	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	+	+	-	-	-	-	*Cette lettre ~ Léa auprés de Max*
-	-	-	-	introduire	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	-	-	-	-	-	-	*Le valet ~ Bob dans le boudoir*
-	-	-	-	jeter	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-	-	+	-	*Ce malheur ~ Max dans le désespoir*
-	-	-	-	jeter	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	-	-	-	-	-	-	*Le patron ~ Max de son boulot*
-	-	-	-	lever	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-	-	+	-	*Max ~ Léa de son lit*
-	-	-	-	libérer	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-	-	+	-	*On ~ Max de sa prison*
-	-	-	-	licencier	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	*Cette entreprise ~ 1000 ouvriers*
-	-	-	-	limoger	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	*On ~ Max de son poste*
-	+	+	+	loger	+	-	-	-	-	-	-	+	+	+	+	-	+	+	-	+	-	+	+	-	-	*Max loge chez lui des amis*
-	-	-	-	lourder	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	+	-	+	+	+	-	*On ~ Max de son poste*
-	-	-	-	mander	-	-	-	-	-	-	-	-	-	-	+	-	-	+	-	-	-	-	-	-	-	*César ~ Caïus chez lui*
-	-	-	-	masser	-	-	-	-	-	-	-	+	+	+	+	-	-	+	-	+	-	-	-	-	-	*Le spectacle ~ les gens sur la place*
-	-	-	-	mener	-	-	-	-	+	+	-	+	-	-	+	+	-	+	+	-	-	-	-	-	-	*Max ~ Luc chez lui*
-	-	-	-	mobiliser	+	+	-	-	-	-	-	+	-	-	-	-	-	+	-	+	-	-	-	-	-	*On ~ Max dans la marine*
-	-	-	+	murer	+	-	-	-	-	-	-	+	-	-	-	-	+	+	-	+	+	-	+	+	-	*L'éboulement ~ Max dans la grotte*
-	-	-	-	muter	-	-	-	-	+	+	-	+	-	-	-	-	-	+	+	+	-	-	-	-	-	*On ~ Max à Gap*
-	-	-	-	nommer	-	-	-	-	-	-	-	+	-	-	+	-	-	+	-	+	-	-	-	-	-	*On ~ Luc à la présidence*
-	-	-	-	noyer	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	+	-	+	+	+	-	*Max ~ les chatons dans la rivière*
-	-	-	-	parachuter	-	-	+	-	-	-	-	+	+	-	+	-	-	+	-	-	-	+	+	-	-	*On ~ Max dans cette entreprise*
-	-	+	+	parquer	+	-	-	-	-	-	-	+	-	-	-	-	+	+	-	+	+	+	+	+	-	*Max ~ les boeufs dans l'enclos*
-	-	-	-	pelotonner	-	-	-	-	-	-	-	+	+	-	-	-	-	+	-	+	-	-	-	-	-	*Ida ~ sa grande taille sur le divan*
-	-	+	-	pendre	+	-	-	-	-	-	-	-	-	-	+	-	-	+	-	+	-	-	-	+	-	*On ~ Max au gibet*

Figure 2 Echantillon de la table 38LH du lexique grammaire

DEUXIEME PARTIE

APPORTS PERSONNELS

Le domaine linguistique informatique est un domaine récent et en pleine expansion. Des différents travaux de recherches ont été effectués dans ce domaine pour apporter des solutions aux problèmes inhérents à ce domaine tels que la compréhension du texte écrit. Le filtrage et l'extraction des informations sont des outils bien connus et sont utilisés par plusieurs applications comme par exemple les moteurs de recherche, les translater automatique, le système de traitement de texte. Les travaux réalisés durant concernent la recherche automatique des informations dans un domaine précis en utilisant des connaissances linguistiques.

Après la présentation détaillée de cette problématique, nous proposerons les méthodes pour résoudre les problèmes trouvés. En effet, nous avons souhaité à la fois effectuer une étude théorique de la problématique, mais nous avons également décidé d'avoir une approche pratique, en implantant les solutions proposées dans l'application Unitex.

Après la présentation détaillée de ces solutions, nous effectuerons pour chacune d'elles des mesures de performances, et nous commenterons ces résultats. Nous comparerons ainsi les solutions, et présenterons leurs avantages et leurs inconvénients majeurs.

Enfin nous verrons comment améliorer les solutions proposées pour avoir un système assez complet.

2 .1 Problématique

L'objectif de mon travail est l'amélioration d'un analyseur de corpus Unitex en ajoutant des nouvelles fonctions. L'analyseur amélioré permettra la recherche automatique des occurrences de la requête dans le texte choisi par l'utilisateur. Le travail est fait en langue polonaise et couvre la domaine électronique.

Dans le chapitre précédent, nous avons vu la possibilité d'effectuer une recherche efficace dans un texte grâce à des outils linguistiques. Dans l'application Unitex, on peut rechercher les correspondances en utilisant les graphes, les dictionnaires et les grammaires. Pour trouver la correspondance avec la requête que l'utilisateur souhaitera appliquer, l'utilisateur était obligé de construire les graphes lui-même. Pour construire des graphes, une connaissance des bases linguistiques s'avère nécessaire. C'est pour cette raison que j'ai eu l'idée d'améliorer l'application pour pouvoir faire la recherche automatiquement.

Dans une recherche automatique, l'utilisateur n'a qu'à choisir le ou les textes qu'il veut parcourir ainsi que la requête avec les mots recherchés.

Dans le but d'enrichir UNITEXT, et vu que je maîtrise la lange polonaise d'utiliser, j'ai effectuer tout mon travail en polonais. Il est nécessaire de préciser que l'application réalisée peut facilement fonctionner avec d'autres langues: pour une langue bien définie, il suffit juste de refaire les graphes.

La recherche automatique est faite pour un domaine bien précis. Le domaine considéré est celui d'*électronique* pour lequel j'ai généré une base de graphes. Ma formation d'électronicien justifie mon choix de domaine.

Pour pouvoir rechercher dans le texte les occurrences des mots demandés par l'utilisateur, on procède de la manière suivante :

1- intégration de la langue polonaise dans le logiciel Unitex.

2- adaptation du dictionnaire pour couvrir le maximum des mots du domaine considéré.

3- création des graphes pour pouvoir rechercher les occurrences les plus intéressantes.

Cette partie est la partie linguistique. La seconde partie, informatique, consiste à :

1- choisir les différents paramètres pour effectuer une recherche via l'interface utilisateur. Ces paramètres sont: la saisie du texte, le choix de la méthode de la recherche et la saisie de la requête.

2- retrouver le bon graphe à partir de la requête d'utilisateur. Deux méthodes différentes présentées dans la section 2.3.2 nous permettent de lier les graphes avec les requêtes d'utilisateur.

3- créer des graphes correspondant à la requête si, pour cette requête, aucun graphe de la base ne convient.

4- générer les résultats.

2.2 La partie linguistique

2.2.1 Préparation du logiciel Unitex pour travailler en langue polonaise

Pour utiliser l'analyseur de corpus Unitex avec une nouvelle langue un processus de préparation est obligatoire. Il faut préciser l'alphabet, le dictionnaire, les graphes, choisir le corpus. Eventuellement, on peut appliquer les règles grammaticales pour avoir les formes fléchies de formes canoniques du dictionnaire. Pour appliquer les méthodes que nous expliciterons ultérieurement, nous avons besoin des quelques fichiers de configuration.

L'application Unitex était déjà préparé pour travailler en plusieurs langues: anglais, français, grec, finnois, italien, norvégien, portugais, russe,

espagnol, thaïlandais. La langue polonaise n'était pas inclus. C'est pour cette raison qu'il fallait adapter l'application. Tout d'abord on a entré les lettres nécessaires à la langue polonais. Voici l'alphabet : a, ą, b, c, ć, d, e, ę, f, g, h, i, j, k, l, ł, m, n, ń, o, ó, p, q, r, s, ś, t, u, v, w, x, y, z ,ż ,ź.

Ensuite il faut ajouter le dictionnaire. Dans notre cas le dictionnaire est composé de quatre fichiers. Chaque fichier est un sous-dictionnaire. On a donc un dictionnaire des lieux géographiques, un dictionnaire des noms propres, un dictionnaire des mots polonais avec les formes fléchie, et un dictionnaire avec les mots typique pour la domaine choisi, qui est dans notre cas le domaine électronique. Plus de détails sur les dictionnaires vont être donnés dans le chapitre suivant (2.2.2).

Comme un dictionnaire de formes fléchies sera implémenté, nous n'avons pas besoin des graphes décrivant la flexion automatique. Mais partant du principe que notre dictionnaire n'est pas complet, on a ajouté dans le répertoire Inflection les graphes qui permettent de passer du dictionnaire DELAS - dictionnaire non fléchi - au dictionnaire DELAF - dictionnaire fléchi -.

Dans l'application Unitex, chaque texte traité doit d'abord passer l'étape du pré-traitement. Cette opération sur le texte consiste à lui appliquer les opérations suivantes : normalisation des séparateurs, découpage en unités lexicales, normalisation de formes non ambiguës, découpage en phrases et application des dictionnaires.

Pour la langue polonaise, la partie normalisation de formes non ambiguës n'a pas de sens. Dans le langue polonaise, l'apostrophe n'est pas utilisée, de même que les contraction des mots, à l'inverse de la langue française.

Exemple :

J'ai → Je ai

Au → à le

Malgré qu'en langue polonaise, nous ne sommes pas censés utiliser cette partie, nous avons tout de même laisser le traitement qu'on retrouve en langue française, car nous pouvons trouver assez souvent des mots empruntés de la langue française dans les textes polonais.

Pour la partie découpage en phrases, nous nous sommes inspirés du graphe de la langue française qui est le suivant. Dans cet exemple, nous pouvons constater les difficultés posées pour la compréhensions du caractère « . ». Il faut faire la distinction si le « . » désigne la fin de la phrase ou si il indique une abréviation comme par exemple M. J. Dupont.

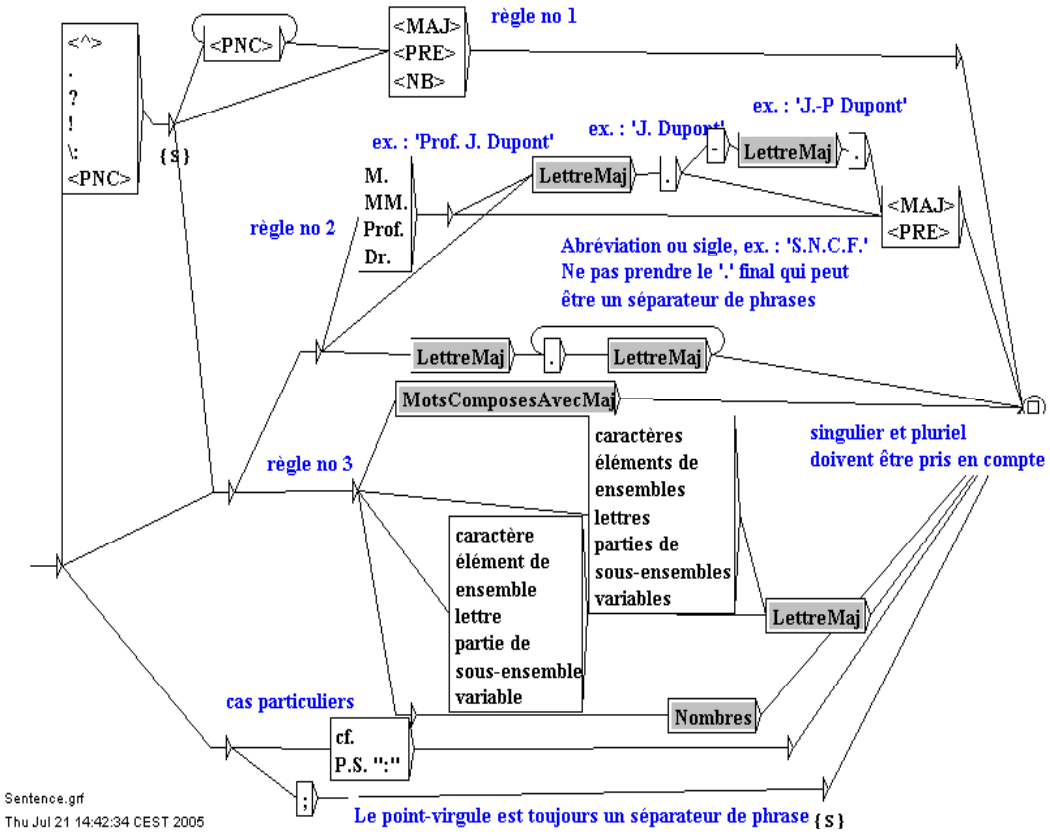


Figure 3 Le graphe de decoupage des phrases.

Nous pouvons retrouver plus de détails sur ce graphe dans [3].

Nous avons déjà précisé pour la recherche automatique la création d'une base des graphes pour le domaine électronique. Tous les graphes se trouvent dans le répertoire Graphs. Les graphes crées ne sont utilisables que pour le domaine de l'électronique et nous les utilisons pour effectuer la recherche des occurrences dans le texte.

Nous allons présenter quelques méthodes qui nous permettront de faire la correspondance entre la requête de l'utilisateur et les graphes. La forme des graphes est présentée dans le chapitre 2.2.3.

Pour chaque langue utilisée par Unitex, nous avons un fichier qui nous montre quels dictionnaires sont utilisés pour notre langue. C'est le fichier *user_dic* et dans notre cas quatre dictionnaires sont présents.

En suivant ce raisonnement, nous avons ajouté deux fichiers de configuration pour satisfaire nos besoins de recherche automatique. Le premier fichier est celui qui indique quels graphes sont utilisés pour notre domaine. C'est le fichier *user_grph*. Le second, *ele_dic,* nous montre quels dictionnaires possèdent les mots typiques pour la domaine choisi. Ces deux fichiers sont nécessaire pour effectuer la recherche automatique.

2.2.2 Le dictionnaire

Nous allons maintenant présenter le dictionnaire créé. Le dictionnaire de langue polonaise est composé de quatre sous-dictionnaires. L'ordre des information dans les dictionnaires est formel [1] est le suivant :

źródłem,źródło.N+T:nsGn

źródłem – c'est la forme fléchie de l'entrée

źródło – c'est la forme canonique de l'entrée, pour les noms et les adjectifs, il s'agit de la forme au masculin singulier ; pour les verbes on a l'infinitif. Il faut préciser qu'en langue polonaise, nous avons des déclinaisons qui changent complètement les adjectifs et les noms par rapport au genre pour lequel elles sont appliquées.

N+Gn+T – c'est la séquence d'informations grammaticales et sémantiques.

:nsGn – c'est la séquence d'information flexionnelles qui décrivent le genre, le nombre, le temps, la déclinaison

Nous avons quatre fichiers de dictionnaire. A certains mots nous pouvons associer une ou plusieurs classes sémantiques. Grâce a ces classes sémantiques, nous pouvons savoir à quel domaine appartient le mot.

Dans le premier fichier, *dico_pl1,* nous trouvons les lieux géographiques, les noms des organisations et le noms propres. Les classes sémantiques pour ce dictionnaire sont : organisation, location country, city, person, firstname, lastname.

Dans le second, *dico_pl2,* nous avons accès aux jours, aux mois, aux titres, ainsi qu'aux noms propres. Dans les classes sémantiques, nous retrouvons les suivants: titre, day, month, person, firstname, location, country.

Le troisième, *slownik2,* qui est le plus grand, contient les mots en langue polonaise, avec les formes fléchies. Dans ce fichier, nous ne trouvons pas des classes sémantique.

La dernier fichier est le dictionnaire pour le domaine électronique. Le fichier est nommée *ele*. Pour notre recherche c'est le dictionnaire le plus intéressant.

C'est grâce a ce dictionnaire et les graphes que nous effectuons la recherche des occurrences de la requête d'utilisateur dans les textes. Dans ce dictionnaire on a plusieurs classes sémantiques.

Nous avons partagé le domaine électronique en plusieurs parties. Tout le domaine est représenté sous forme d'arbre. Cette construction peut être visualiése sur la figure 1. A chaque feuille, nous associons une classe sémantique.

Un mot peur évidemment avoir plusieurs classes sémantiques. Comme par exemple une *photodiode* est un composant optoélectronique, mais possède pourtant les caractéristiques d'une diode qui se situe dans la catégorie des semi-conducteurs.

C'est en nous basant sur cette structure d'arbre que nous avons créé la base des graphes.

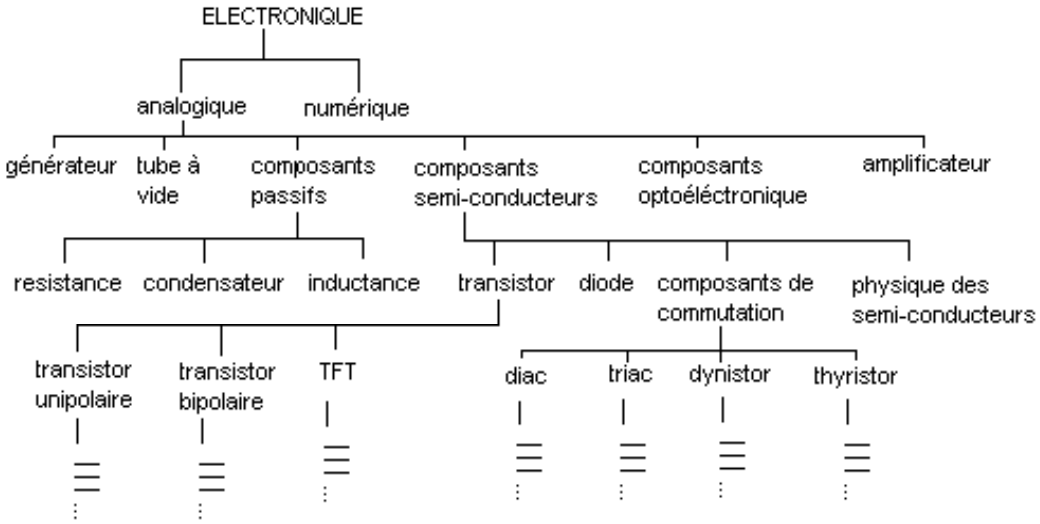


Figure 4 Exemple de partage du domaine sous forme d'arbre.

Remarque : Pour votre compréhension, l'exemple de l'arbre est montré en langue française, pourtant tout le projet a été effectué pour la langue polonaise.

Les déclinaisons sont présentes en polonais, nous allons présenter un exemple pour comprendre leur fonctionnement.

Les déclinaisons sont présentes pour les adjectifs et pour les noms. Il existe sept cas: nominatif, genetif, datif, accusatif, ablatif, locatif et vocatif.

En langue polonaise, le genre singulier peut être masculin, féminin, ou neutre, le pluriel masculin ou féminin. Le neutre devient masculin au pluriel.

Voici un exemple qui montre les déclinaisons existantes en langue polonaise :

- pour le genre masculin adjectif *miły* (fr. gentille), nom *pies* (fr. chien).
- pour le genre féminin adjectif *miły* (fr. gentille), nom *pani* (fr. femme).
- pour le genre neutre adjectif *miły* (fr. gentille), nom *dziecko* (fr. enfant).

Tableau 1La déclinaison.

	ms	mp	fs	fp	ns
Nominatif	miły pies	miłe psy	miła pani	miłe panie	miłe dziecko
Genetif	miłego psa	miłych psów	miłej pani	miłych pań	miłego dziecka
Datif	miłemu psu	miłym psom	miłej pani	miłym paniom	miłemu dziecku
Accusatif	miłego psa	miłe psy	miłą panią	miłe panie	miłe diecko
Ablatif	miłym psem	miłymi psami	miłą panią	miłymi paniami	miłym dzieckiem
Locatif	miłym psie	miłych psach	miłej pani	miłych paniach	miłym dziecku
Vocatif	miły psie	miłe psy	miła pani	miłe panie	miłe dziecko

A cause de la présence des déclinaisons, la recherche automatique est beaucoup plus compliquée. Car en fonction du genre, nous devons rechercher plusieurs mots qui en langue français sont constitués d'un seul mot.

Ce problème est réglé grâce au dictionnaire. On effectue la recherche avec des formes canoniques. Le dictionnaire possède la forme canonique des mots, ainsi que leurs formes grammaticales. En parcourant le dictionnaire, nous trouvons facilement tous les mots qui ont la même forme canonique. Comme résultat, nous allons retrouver toutes les occurrences du mot recherché indépendamment du genre du mot dans le texte.

2.2.3 La création des graphes

Pour pouvoir effectuer la recherche automatique, nous avons besoin d'une base des graphes.

Le domaine choisi est le domaine de l'éléctronique. On a partagé tout le domaine en classes sémantiques. Un exemple est montré sur la figure 1.1.

Apres avoir partager le domaine, nous pouvons associer les classes semantiques aux graphes que nous avons construits pour chaque feuille. Il se trouve qu'un mot peut avoir plusieurs classes semantique.

Voici l'exemple d'un graphe simple:

Electronique → analogique → compsants semi-conducteurs → diode.

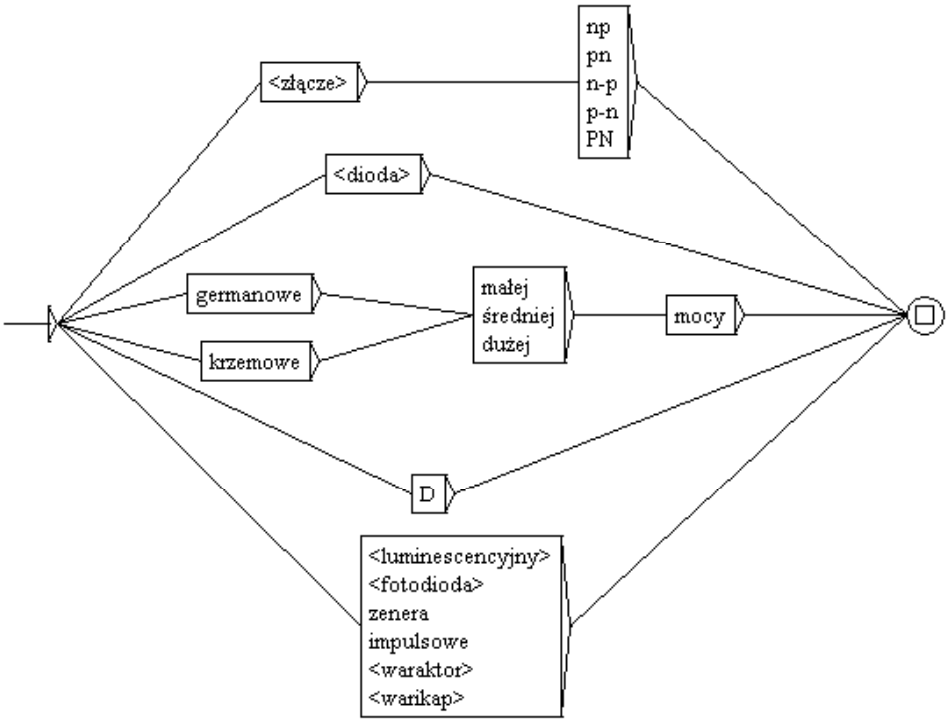


Figure 5 L'exemple d'un graphe

Le graphe est en langue polonaise. Nous avons crée environt vingt graphes pour notre domaine. Dans le graphe, nous avons tous les synonymes possibles de chaque mot, ainsi que tous les éléments possédant

des paramètres similaires, les composants qui font partie de la même famille, etc…

Quand un mot dans le graphe est entre parenthèses <*mot*>, nous recherchons tous les mots qui ont la forme canonique (*mot)*. Nous avons donc la possibilité de retrouver dans le texte toutes les formes grammaticales mais uniquement si le dictionnaire couvre bien le domaine.

L'application associe des graphes aux mots entrés par l'utilisateur lors de sa requête grâce a des methodes qui vont être présentées dans le chapitre suivant.

Un graphe peut appeler d'autre graphe. Nous avons utiliser cette possibilité pour structurer la partage du domaine. Voici l'exemple :

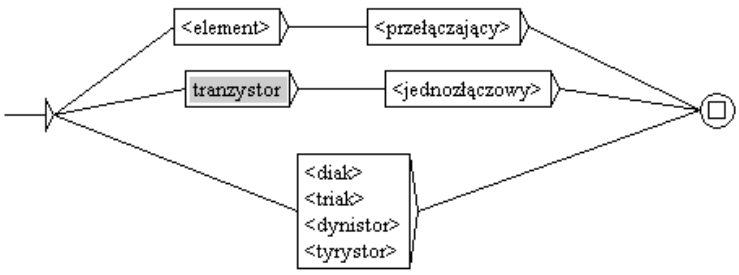


Figure 6 L'appel des sous graphes

Sur l'exemple nous peuvons voir le graphe des composant de commutation analogique. Le graphe appele un sous graphe *tranzystor* qui est le suivant :

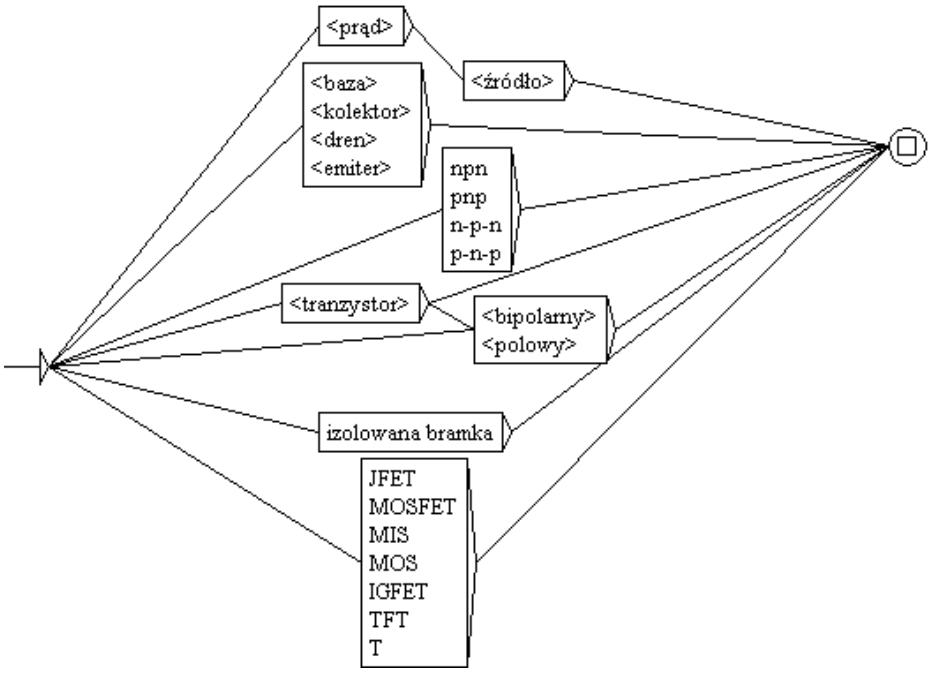


Figure 7 L'exemple de sous graphe

Nous pouvons constaté que la sélection des graphes est la dernière etape de notre recherche. Apres l'étape de sélection des graphes par l'algorithme suite à la requête de l'utilisateur, nous parcourons le texte en recherchant les occurences. C'est pour cette raison que la structure de chaque graphe est primordiale.

Malgré différents essais au niveau des constructions des graphes, nous retrouvons encore plusieurs occurrences qui ne sont pas pertinentes par rapport à la requête de l'utilisateur. L'idée était plutôt de retrouver plus d'occurences que nécessaire, même si certaines d'entre elles ne se révélaient pas pertinentes pour l'utilisateur, que de ne retrouver que des occurences pertinentes, au risque pour l'utilisateur de passer à coté de résultats intéressants.

Le graphe va etre consideré pour la recherche automatique s'il se trouve dans le fichier *user_grph*.

Comme nous l'avons précisé, la recherche automatique consiste à lier un ou plusieurs graphes à la requête utilisateur. Malheureusement, il n'est pas possible de couvrir tout le domaine de l'*éléctronique* avec les graphes.,

même si nous pouvons constater que ce domaine est relativement "clos". Dans ce cas, nous avons la possibilité de generer le graphe automatiquement, directement à partir de la requête d'utilisateur.

La generation de ce graphe est simple mais loin d'être idéale. C'est pour cette raison que l'utilisation d'une base de graphe est recommandée. Comme aucun graphe n'a été retrouvé, nous utilisons la génération automatique d'un graphe.

Cette methode consiste a retrouver tous les mots de la requête d'utilisateur qui sont dans les dictionnaires qui couvre les mots du domaine. Dans notre cas, ces dictionnaires sont réécris dans le fichier *ele_dic*.

Apres avoir retrouver les mots d'un domaine, l'algorithme crée le graphe. Nous n'allons pas prendre en compte les mots indépendants du domaine lors de la requête mais juste ceux en rapport avec l'éléctronique. Nous pouvons ésperer que ses mots soient les plus interessants pour l'utilisateur. Voici un simple exemple pour expliciter l'idee :

La requête d'utilisateur : *Dans quels domaines d'applications les équations relatives aux ondes magnétiques sont-elles nécessaires?*

On imagine qu'aucun graphe ne decrive les ondes magnétiques. Les seuls mots qui vont etre pris en compte sont *ondes* et *magnétique .*

Dans ce cas, le graphe va etre le suivant :

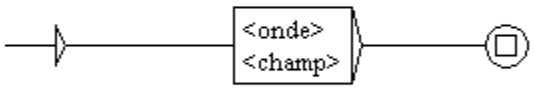


Figure 8 L'exemple de création d'un graphe.

Le but est de ne pas prendre en compte des mots de la requête comme *domaine, nécessaire* ou *dans* car nous pourrions retrouver plusieurs occurences de ces mots dans des textes qui n'ont aucun rapport avec la requête de l'utilisateur.

Cette methode sera présentée plus en détail dans la partie informatique.

2.2.4 Conclusions

La partie linguistique consiste à :

- intégrer du langue polonaise dans le logiciel Unitex.
- créer des graphes, par lesquels nous allons retrouver dans le texte les occurrences en adéquation avec la requête d'utilisateur.

La préparation des ressources linguistiques consiste en la création de dictionnaires qui doivent être formalisés précisément. Le dictionnaire, ne pouvant contenir tous les mots (avec déclinaisons) de la langue polonaise, devrait se concentrer sur la couverture du domaine choisi, qui est dans notre cas le domaine de l'*électronique*.

Pour les mots du dictionnaire appartenant au domaine considéré, les correspondances avec les graphes sont effectués par les classes sémantiques. Pour cela, un partage du domaine fut obligatoire.

Grâce a ces classes sémantiques nous avons donc la possibilité de lier les graphes avec les requêtes d'utilisateur. Pour cela, un fichier contient les correspondances entre classes sémantiques et graphes.

La construction des graphes est liée au partage du domaine (ici électronique).

La construction des graphes est un travail complexe pour l'obtention de bons résultats, chaque graphe étant le résultat de plusieurs essais. Malgré ces inconvénient, les graphes créés sont suffisants pour la recherche

automatique et permettre de trouver des résultats satisfaisants pour l'utilisateur.

Pour conclure cette partie je tiens a préciser que la recherche peut être améliorée en complétant le dictionnaire et en ajoutant des graphes, ou éventuellement en réalisant un partage du domaine plus précis. Cette amélioration va diminuer le nombre des occurrences non pertinentes.

2.3 La partie informatique

2.3.1 Présentation

Tout d'abord, nous allons présenter l'interface utilisateur que nous avons développer. Le projet est écrit en langage Java, et est un ajout au code source de l'application Unitex 1.2 beta, qui est une application Open-Source, qui a été originellement développée à l'Université de Marne-la-Valée.

Nous avons améliorer les possibilités d'Unitex en ajoutant plusieurs fonctions permettant d'effectuer la recherche des occurrences dans les textes directement via la requête d'utilisateur. Dans le menu principal, nous pouvant remarquer une nouvelle option *Automatic search*. C'est grâce à ce menu que l'utilisateur a la possibilité de paramétrer sa recherche. Sur la figure 4, nous pouvons voir ce menu :

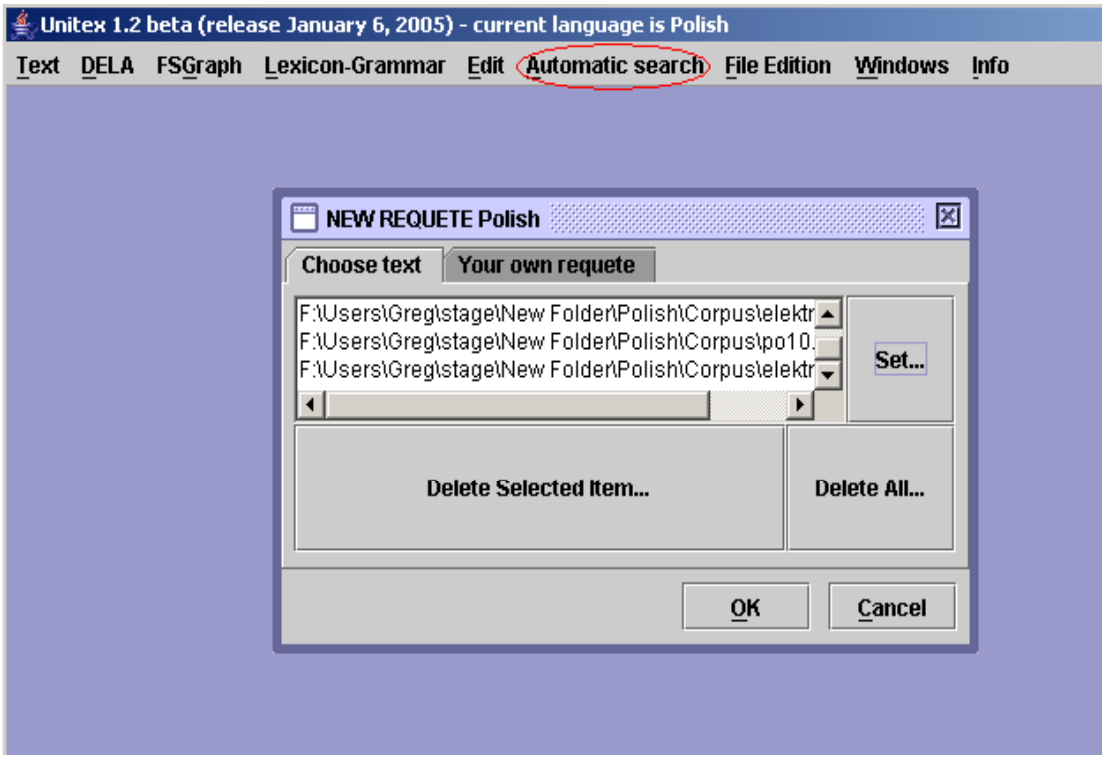


Figure 9. Menu de la recherche automatique.

Apres avoir choisir l'option *Automatic search*, la fenêtre de paramétrage apparaît. Les paramètres sont les suivants:

- le choix du corpus sur lequel la recherche sera effectuée
- le choix de la méthode de recherche
- l'emplacement pour saisir la requête.

La fenêtre s'intitule *NEW REQUETE*. Elle est divisée en deux sous-fenêtres. La première, *Choose text,* présentée sur la figure 4, nous permet de choisir un ou plusieurs textes dans lesquels nous effectuons la recherche. Nous ajoutons les textes désirés dans la liste en appuyant sur le bouton *Set…* Les deux boutons *Delete Selected Item…* et *Delete All…* permettent de supprimer respectivement le texte sélectionné ou tous les textes de la liste.

La deuxième sous fenêtre de la fenêtre *NEW REQUETE* est présentée ci-dessous:

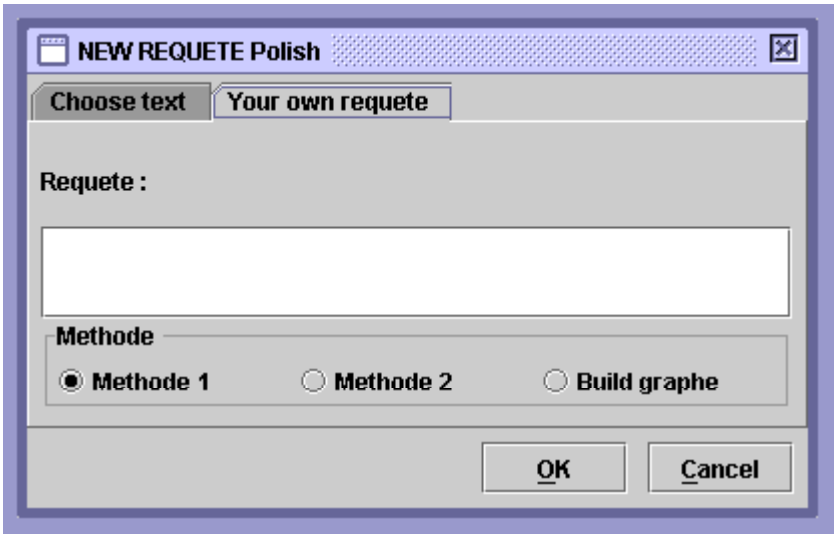


Figure 10. La sous fenêtre *Your own requete.*

Dans cette fenêtre, nous choisissons la méthode de la recherche et nous écrivons les mots de la requête. Nous avons le choix entre trois méthodes :

- les deux premières utilisent la base des graphes déjà crée,
- la troisième *Build graphe* crée le graphe automatiquement à partir de la requête.

La troisième méthode n'est utilisée que dans le cas si où aucun graphe de la base ne convient à la requête d'utilisateur.

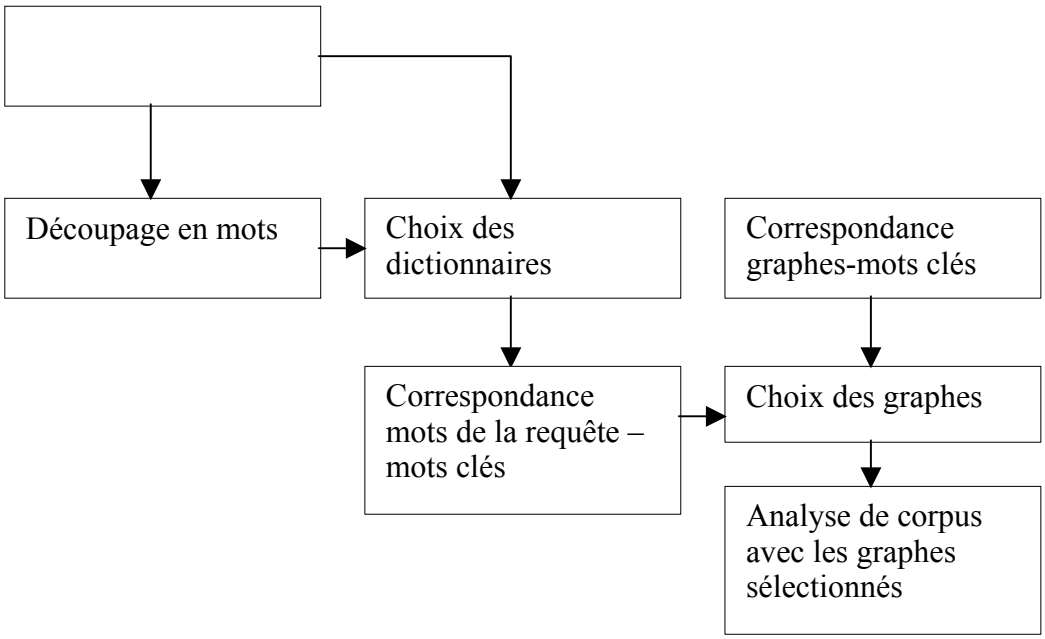
2.3.2 Les méthodes de la recherche automatique

Maintenant, nous allons présenter les méthodes des algorithmes pour effectuer la recherche automatique. Comme nous l'avons décrit dans la partie linguistique pour la recherche, nous allons utiliser des dictionnaires polonais (trois dictionnaires de langue polonaise : *slownik2.bin, dico_pl1.bin, dico_pl2.bin*

et un dictionnaire qui décrit les mots du domaine électronique : *ele.bin*) et les fichiers de configurations (*user_dic.def* – les dictionnaire utilisés, *user_grph.def* – la base de graphes utilisée*, ele_dic.def* – les dictionnaires du domaine utilisé).

2.3.2.1 La méthode des classes sémantiques (la premier méthode)

Dans la méthode des classes sémantique nous disposons de correspondances entre graphes et mots-clés d'une part, et entre entrées du dictionnaire et mots-clés d'autre part ; à partir des mots de la requête, nous sélectionnons les mots-clés correspondants puis les graphes. Les mots clés sont appelés les classes sémantique. La recherche est effectuée de la manière suivante :



1. On ouvre le fichier *user_dic.def* pour prendre tous les dictionnaires utilisés pour la langue courante – ce sont les dictionnaire dans le répertoire DELA

2. Pour chaque mot de la requête de l'utilisateur, on va parcourir tous les dictionnaires pour retrouver les classes sémantiques liées avec le mot recherché.

3. On ouvre le fichier *user_grph.def* dans lequel on associe les graphes avec les classes sémantiques, donc nous récupérons finalement les graphes qui ont été choisis pour trouver les informations concernant la requête. La correspondance entre les graphes et les classes sémantique dans le fichier *user_grpf.def* est la suivante :
   <nom de la classe sémantique 1> <les noms des graphes>
   <nom de la classe sémantique 2> <les noms des graphes>

4. On exécute la recherche dans le corpus en utilisant les graphes sélectionnées.

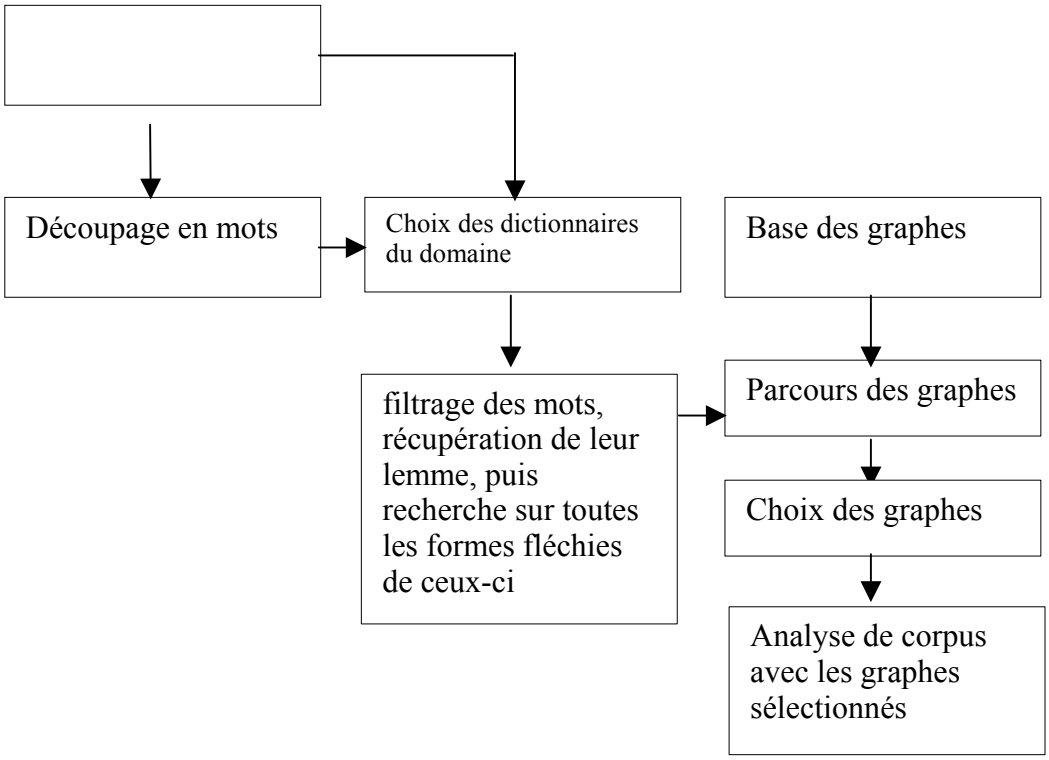
A la fin de la procédure on obtient tout les occurrences retrouvées dans le corpus choisi avec les graphes sélectionnés de la base de graphe. On espère retrouver de bons résultat grâce a la construction de la base des graphes. Pour avoir des résultats satisfaisants, nous devons posséder la base des graphes complets et détaillés. Le dictionnaire doit lui avoir les classes sémantiques détaillées pour tous les mots du domaine et enfin le fichier de configuration doit avoir tout les correspondances possibles.

Voici le résultats obtenu par cette méthode en traitent un texte d'exemple avec la requête d'utilisateur : *Co to jest dioda ? (fr. qu'est ce que c'est une diode)*.

tów wywołuje prąd pomijalny (kilka mA w diodach mocy).{S} Przekroczenie granicznej wartości nap
go przykładu wynika, że możemy obciążyć diodę przez 0,1s prądem nawet 2kA. {S}Jednak po przekro
go przykładu wynika, że możemy obciążyć diodę przez 0,1s prądem nawet 2kA. {S}Z powyższego przy
. {S}W 1958 roku Leo Esaki skonstruował diodę tunelową. {S}Ten japoński fizyk badał zjawiska za
erystykę prądowo-napięciową (statyczną) diody. {S}Będziemy przykładać różne napięcia (regulowan
iody). {S}Napięcie przemienne za pomocą diody zamieniamy na napięcie o dodatniej biegunowości (
cze znaczenie w przypadku wykorzystania diody tunelowej przy bardzo wielkich częstotliwościach.
0,6÷1V to napięcie progowe przewodzenia diody), aby nieco ustabilizować zmiany napięcia równole
{S}Natomiast charakterystyka statyczna diody tunelowej będzie przebiegała tak, jak na rys.{S}
ujemna, będąca cechą charakterystyczną diody tunelowej.{S} Charakterystyka taka stanowi wynik
ncję szeregową, a nawet na indukcyjność diody.{S} Ponadto średnica stożka określa wytrzymałość
Działanie diody jest bardzo proste - przewodzi ona prąd tylko w j
je rys.{S} 1, a schematyczne oznaczenie diody tunelowej przedstawiono na rys.{S} 2.{S} Dalsze z
żdej "połówce" sinusoidy przewodzą dwie diody, co powoduje dwukrotnie większy spadek napięcia.
rzedstawiają rysunki 7 - 9. {S}Planarne diody germanowe, opracowane w Sylvania Electrio Inc. by
apięcia. {S}Ponadto da się zauważyć, że diody nie przewodzą dla małych napięć (poniżej 0,6V) -
acniaczy mikrofalowych wykorzystujących diody tunelowe.{S} Wzmacniacze te pod względem właściwo
j.{S} Należałoby tu wrócić do normalnej diody półprzewodnikowej o małej koncentracji domieszek
już narysowany ... (na stronie z opisem diody). {S}Napięcie przemienne za pomocą diody zamienia
rzyjrzyjmy się poszczególnym parametrom diody. {S}Prąd graniczny - IFAVM (IF(AV)M) - określa ma
zeniu ujemnego napięcia zewnętrznego do diody. {S}Po tym przesunięciu oba poziomy Fermiego zost
można zaobserwować lepsze przewodnictwo diody dla napięć zaporowych niż dla przepustowych. {S}P
średnica ma istotny wpływ na parametry diody: prąd szczytowy, pojemność złącza, rezystancję sz
cznych krzemowych - wynika ono z budowy diody [złącza pn]).{S} Zwiększanie przepływającego prąd
owaną siecią antenową. {S}Wzmacniacze z diody tunelowej są do 10 razy tańsze od wzmacniaczy z 1
i domieszek w obu wartościach P i N. {S}Diody tunelowe są wytwarzane w postaci złączy P+ - N+ n
rzędu 100, 200 i więcej V - napięcie na diodzie podczas przewodzenia jest pomijalne.{S} Analogi
komo mały prąd wsteczny.{S} Napięcie na diodzie w tym stanie jest równe napięciu zasilania. {S}
d Zenera teoretycznie istnieje w każdej diodzie, jednak wartości ich są tak małe w porównaniu d
woduje zmniejszanie się grubości złącza Zenera przy coraz to niższych napięciach wstecznych.{S}
n energetyczny. {S}Prąd tunelowy i prąd Zenera teoretycznie istnieje w każdej diodzie, jednak w
krzemowych - wynika ono z budowy diody [złącza pn]).{S} Zwiększanie przepływającego prądu prakt
ał zjawiska zachodzące w bardzo wąskich złączach PN.{S} Esaki badał emisję pola wewnętrznego w
racji domieszek w warstwach P i N. {S}W złączu PN poziomy Fermiego wyrównują się, gdy na dane w

2.3.2.2 La méthode directe (la deuxième méthode)

Dans la méthode directe à partir du texte de la requête, on sélectionne les graphes par l'intermédiaire des lemmes ou des formes fléchies. Si un mot de la requête a le même lemme ou la même forme fléchie qu'un mot dans un des graphes, on sélectionne ce graphe comme le graphe idoine pour l'analyse du corpus. On ne prend pas en compte tous les mots de la requête mais seulement le mots en rapport avec le domaine. Pour cela on a un fichier de configuration dans lequel nous avons précisé les dictionnaires du domaine – c'est le fichier *ele_dic.def*. La recherche est effectuée de la manière suivante :



1. On ouvre le dictionnaire des mots correspondant au domaine électronique.

2. Pour chaque mot de la requête on va parcourir ce dictionnaire pour retrouver son lemme ainsi que toutes les formes fléchies.

3. Ensuite on ouvre le fichier *user_grph.def* où se situent tous les graphes utilisés. On parcourt les graphes en cherchant les formes retrouvées précédemment – le lemme où les formes fléchies. Quand la forme a été retrouvée, on considère le graphe comme le graphe adéquat pour la recherche.

4. On exécute la recherche dans le corpus en utilisant les graphes sélectionnées.

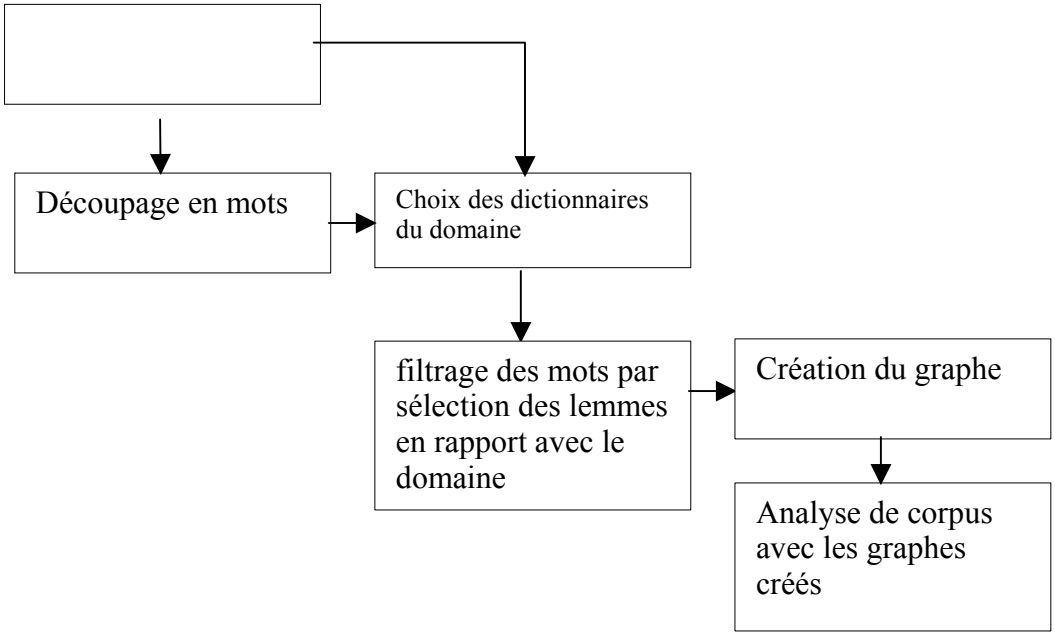
Pour avoir des résultats satisfaisants, on doit avoir la base des graphes complets et détaillés et le dictionnaire doit posséder les classes sémantiques détaillées pour tous les mots du domaine.

Voici le résultats obtenu par cette méthode en traitent un texte d'exemple avec la requête d'utilisateur : *jakie sa rodzaje tranzystorow (fr. quelle sont les types des transistors).*

stora bipolarnego w układzie o wspólnej bazie OB {S}Struktura półprzewodnikowa tranzystora jest
kolektorze OC {S}W układzie o wspólnej bazie - oznaczanym WB lub ob. sygnał jest doprowadzany
ączy możemy traktować jako tranzystor o bazie Pl. doprowadzając niewielkie napięcie między bazę
zone przechodzą łatwo niewielką grubość bazy, dostając się do złącza baza-kolektor, skąd zostaj
prądy IE w gałęzi emitera, IB w gałęzi bazy oraz IC w gałęzi kolektora.{S} Zwroty strzałek prą
} IC = 0,90 ÷ 0,98 IE.{S} Prąd w gałęzi bazy {S}Małym zmianom prądu bazy odpowiadają wielokrotn
ektora oraz mikroamperomierzem w gałęzi bazy. {S}Przy zamkniętym tylko wyłączniku wl płynie jed
śników ładunku oddanych przez emiter do bazy dochodzi do kolektora:{S} IC = 0,90 ÷ 0,98 IE.{S}
ąd w gałęzi bazy {S}Małym zmianom prądu bazy odpowiadają wielokrotnie większe zmiany prądu kole
rądu kolektora DIC i małych zmian prądu bazy DIB, czyli {S}Współczynnik b0 nazywa się małosygna
a.{S} Stosunek prądu kolektora do prądu bazy nazywa się wielkosygnałowym współczynnikiem wzmocn
m wzmacniającym, gdyż małe zmiany prądu bazy powodują duże., zmiany prądu kolektora. {S}W układ
i dziury, o czym świadczy przymiotnik : bipolarny.{S} Możliwe jest przy tym dwojakie uszeregowa
(tranzystory małej mocy). {S}Tranzystor bipolarny jest zatem elementem wzmacniającym, gdyż małe
, obciążenia oraz zwarcia {S}Tranzystor bipolarny jest to element półprzewodnikowy o dwóch złąc
i tranzystory polowe z izolowaną bramką MOS, MOSFET (z ang Metal-Oxide Semiconductor - co znacz
nzystory polowe z izolowaną bramką MOS, MOSFET (z ang Metal-Oxide Semiconductor - co znaczy met
ach tych pokazano strukturę tranzystora NPN wraz z zewnętrznymi źródłami zasilania, miliamperom
ewodnictwa:{S} PNP (pierwszy rysunek) i NPN (drugi rysunek), dające dwa przeciwne typy tranzyst
szarów o różnym typie przewodnictwa:{S} PNP (pierwszy rysunek) i NPN (drugi rysunek), dające dw
znaczy tranzystor wykorzystujący efekt polowy) i tranzystory polowe z izolowaną bramką MOS, MO
.{S} Układ złączy możemy traktować jako tranzystor o bazie Pl. doprowadzając niewielkie napięci
układach elektronicznych na każdy użyty tranzystor przypada statycznie cztery do pięciu rezysto
{S} Field-Effect Transistor - co znaczy tranzystor wykorzystujący efekt polowy) i tranzystory p
ku tysięcy (tranzystory małej mocy). {S}Tranzystor bipolarny jest zatem elementem wzmacniającym
acy: jałowy, obciążenia oraz zwarcia {S}Tranzystor bipolarny jest to element półprzewodnikowy o
współczynnikiem wzmocnienia prądowego> tranzystora w układzie ze wspólnym emiterem OE (WE) {S}
{S}Rysunek przedstawia zasadę działania tranzystora polowego złączowego: przepływ prądu przez p
{S}Rysunek przedstawia zasadę działania tranzystora polowego złączowego: zwężenie kanału i zmia
wodnik). {S}Rozpatrzmy zasadę działania tranzystora FET. {S}Rysunek przedstawia zasadę działani
bazie OB {S}Struktura półprzewodnikowa tranzystora jest umieszczona w hermetycznie zamkniętej
ktor, B - baza. {S}Rozpatrzmy działanie tranzystora bipolarnego kożystając z poniższych Na rysu
ch Na rysunkach tych pokazano strukturę tranzystora NPN wraz z zewnętrznymi źródłami zasilania,
ywane w skrócie wzmocnieniami prądowymi tranzystora.{S} Wartości ich wynoszą od kilkunastu (tra

2.3.2.3 La construction automatique des graphes (la troisième méthode)

La troisième méthode consiste a recréer les graphes directement à partir de la requête utilisateur. Cette méthode est utilisée au cas où aucun graphe de la base ne correspond à la requête utilisateur. Car on ne peut sortir comme résultat qu'aucune occurrence n'a été trouvée alors qu'elle existe, ceci étant du à l'incomplétude de notre base de graphes. La création de graphe à partir de la requête utilisateur ne prend en compte que les mots du domaine choisi. On peut espérer que ce soient les mots les plus significatifs pour la recherche. Le format de la requête utilisateur n'est pas formalisé: on peut trouver toutes les suites des mots. On doit les filtrer pour récupérer uniquement les mots pertinents en regard du domaine choisi. La construction de graphes consiste à retrouver le lemme de tous les mots de la requête qui se trouvent dans le dictionnaire du domaine. La recherche est effectuée de la manière suivante :



1. On ouvre le dictionnaire des mots du domaine électronique

2. Pour chaque mot de la requête on va parcourir ce dictionnaire pour retrouver le lemme.

3. On ajoute tous les lemmes retrouvés dans le graphe.

4. On exécute la recherche dans le corpus en utilisant le graphe créé, la recherche est donc effectuée sur tous les lemmes, ainsi que toutes leurs formes fléchies.

Voici le résultats obtenu par cette méthode en traitent un texte d'exemple avec la requête d'utilisateur : *poszukuje informacji o falach magnetychnych (fr. Je recherche des informations sur les ondes magnétiques).*

ałt zależny od częstotliwości (długości fali l=c/f) - zagadnienie to przedstawiam poniżej.{S} N
y z długością emitowanej lub odbieranej fali elektromagnetycznej. {S}Linie łączące antenę z nad
ówka fali l/2, zatem długość emitowanej fali l=2L. W praktyce należy uwzględnić to, że przewód
ę nadawczą i odbiorczą;{S} K - kierunek fali nadawanej {S}Długość i kształt anteny jest związan
y, że na dipolu półfalowym powstaje pół fali stojącej.{S} Fale napięcia i prądu są przesunięte
nku rozchodzenia się zwanego promieniem fali (prostopadłego do czoła fali) ze stałą prędkością,
asadzie sprzężenia z polem elektrycznym fali elektromagnetycznej.{S} Sprzężenie z polem magnety
asadzie sprzężenia z polem magnetycznym fali elektromagnetycznej.{S}Cewka anteny ramowej jest p
Zatem dipol rzeczywisty - dopasowany do fali o długości l - powinien być krótszy od teoretyczne
gich, falowodach i światłowodach) czoło fali elektromagnetycznej porusza się z inną prędkością
co sprzyja intensywnemu promieniowaniu fali elektromagnetycznej.{S} Na dipolach idealnych (tzn
oną (anteny nadawcze AN) lub do odbioru fali elektromagnetycznej rozchodzącej się w przestrzeni
)l uzyskuje się takie przesunięcie fazy fali wtórnej, że w wyniku sumowania z falą pierwotną na
ją wysłał. {S}Rysunek przedstawia obraz fali bieżącej spolaryzowanej pionowo - obraz uchwycony
li tl ;{S} {S}Rysunek przedstawia obraz fali stojącej - obraz uchwycony w chwili tl ;{S} {S}Rys
, a długość 80÷200 mm.{S} Przenikalność magnetyczna rdzeni i cewek stosowanych w antenach ferry
zie: e, m - przenikalność elektryczna i magnetyczna ośrodka (w próżni e = m =1) {S}Od idealnego
netycznej, ponieważ wytwarza stałe pole magnetyczne wokół przewodnika. {S}Zmienny prąd elektryc
mienne pole magnetyczne, a zmienne pole magnetyczne wytwarza zmienne pole elektryczne itd.{S} C
pole elektryczne wytwarza zmienne pole magnetyczne, a zmienne pole magnetyczne wytwarza zmienn
. {S}Wokół dipola powstaje zmienne pole magnetyczne i zmienne pole elektryczne.{S} Antena promi
cy przez cewkę wytwarza wokół niej pole magnetyczne.{S} Jeśli w tym polu magnetycznym umieścimy
owe kineskopu wyposażono w nabiegunniki magnetyczne, które umożliwiały niezależną regulację poł
na wytworzeniu przemiennego strumienia magnetycznego przez jedno z dwóch sprzężonych magnetycz
ansformatora powodują zmiany strumienia magnetycznego przenikającego zwoje drugiej cewki, a wię
o rozprzestrzenianie się zmiennego pola magnetycznego i elektrycznego, to można ją wytworzyć za
wki L zależy głównie od: przenikalności magnetycznej m i wymiarów (l i d) rdzenia ferrytowego o
rromagnetycznego o dużej przenikalności magnetycznej m.{S} Rdzeń może być stały (np. w transfor
ektromotorycznej przez zmienny strumień magnetyczny.{S} Ten sam dławik włączony w obwód prądu s
ezbędne do wytwarzania korekcyjnych pól magnetycznych zbieżności.{S} W systemie D układy formow
mieni za pośrednictwem zewnętrznych pól magnetycznych.{S} Uproszczenie konstrukcji działa umożl
działają na zasadzie sprzężenia z polem magnetycznym fali elektromagnetycznej.{S}Cewka anteny r
ktromagnetycznej.{S} Sprzężenie z polem magnetycznym wykorzystują anteny ferrytowe, ramowe i fa
j pole magnetyczne.{S} Jeśli w tym polu magnetycznym umieścimy drugą cewkę, to otrzymamy transf

3. Testes

3.1 Mesure de la pertinence

Un document pertinent est un document qui répond exactement à la question posée [19].

La notion de pertinence est très subjective. Elle dépend de l'utilisateur, mais aussi de la méthode utilisée pour sa mesure. Les indices utilisés comme mesure de pertinence en général sont basés sur une approche booléenne.

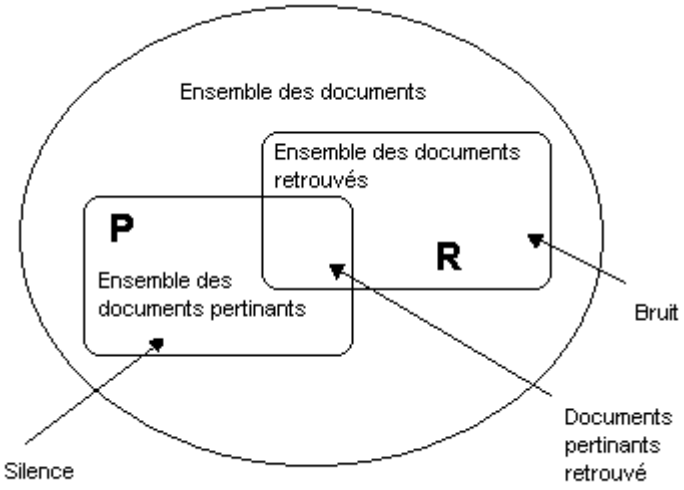


Figure X1 Exemple de représentation du bruit et du silence en recherche de l'information.

L'évaluation de la pertinence dans les recherches documentaires essaie de quantifier le rappel et la précision, avec les indices de bruit et de silence. On les définit comme suit.

Dans les définitions on utilise les symboles suivants :

D_{rp} - l'ensemble de documents retrouvés et pertinents,
D_r - l'ensemble de documents retrouvés,
D_p - l'ensemble de documents pertinents.

Définition de la précision [20] : ratio entre le nombre de documents pertinents trouvés et le nombre total de documents trouvés. Elle mesure le bruit et plus elle est proche de 100%, moins il y a de bruit, et meilleure est la réponse.

$Precision = D_{rp}/D_{r*}$

Définition du bruit [20] : ratio entre le nombre de documents ramenés en réponse, mais qui ne sont pas pertinents par rapport à la question posée, et le nombre total de documents.

$Bruit = 1 - Precision = (D_r - D_{rp})/D_r$

Définition du rappel [20] : ratio entre le nombre de documents pertinents trouvés et le nombre de documents pertinents présents dans la base. Plus il est proche do 100%, moins il y a de silence, et meilleure est la réponse.

$Rappel = D_{rp}/D_p$

Définition du silence [20] : ratio entre le nombre de documents pertinents qui n'apparaissent pas dans le résultat de la recherche et le nombre total de documents.

$Silence = 1 - Rappel = (D_p - D_{rp})/D_p$

3.2 Les testes

3.2.1 Teste 1

La requête : *Jakie sa zastosowania diody w ukladach elektronicznych ?* (fr. quelles sont les différentiels employant d'une diode dans les montage électroniques ?).

Le corpus : 299 phrase, 9445 tokens.

Le numéro de phrase pertinents présents dans la base	Le numéro de phrase retrouver par la 1er méthode	Le numéro de phrase retrouver par la 2eme méthode	Le numéro de phrase retrouver par la 3eme méthode
50, 51, 52, 53, 54, 60, 61, 62, 70, 71, 72, 73, 74, 77, 78, 79, 80, 91, 92, 97, 98, 99, 100, 101, 126, 127, 130, 139, 159, 160, 161, 162, 163, 165, 166, 167, 168, 169, 171, 172, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 192, 193, 194, 195, 197, 198, 199, 200, 202, 203, 205, 207, 208, 210, 211, 212, 213, 214, 215, 247, 248, 249, 250, 251, 252, 253, 262, 263, 264, 265, 296, 297, 298, 299, 301, 302, 303	1, 2, 6, 7, 8, 9, 10, 11, 29, 36, 46, 47, 50, 51, 52, 54, 61, 62, 65, 66, 67, 69, 70, 71, 72, 73, 76, 77, 79, 80, 82, 84, 90, 91, 92, 94, 95, 96, 97, 98, 99, 100, 101, 126, 127, 130, 132, 133, 134, 135, 136, 139, 143, 144, 145, 146, 147, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168 ,169, 170, 171, 172, 173, 174, 176, 177, 178, 179, 180, 181, 182, 184, 191, 192, 194, 199, 200, 202, 208, 210, 211, 213, 214, 215, 247, 248, 249, 250, 251, 254, 255, 258, 259, 262, 263, 296, 297, 298, 299, 304	1, 2, 6, 7, 8, 9, 10, 11, 29, 36, 46, 47, 50, 51, 52, 54, 61, 62, 65, 66, 67, 69,70, 71, 72, 73, 76, 77, 79, 80, 82, 84, 90, 91, 92, 94, 95, 96, 97, 98, 99, 100, 101, 126, 127, 130, 132, 133, 134 , 135, 136, 139, 143, 144, 145, 146, 147, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 176, 177, 178, 179, 180, 181, 182, 184, 191, 192, 194, 199, 200, 202, 208, 210, 211, 213, 214, 215 247, 248, 249, 250, 251, 254, 255, 258, 259, 262, 263, 296, 297, 298, 299, 304	1, 2, 6, 7, 8, 9, 10, 11, 29, 36, 46, 47, 50, 51 ,52, 54, 61, 62, 65, 67, 69, 70, 71, 73, 76, 80, 82, 90, 94, 95, 99, 100, 101, 126, 130, 132, , 133, 134, 135, 136, 139, 143, 144, 145, 146, 147, 159, 160, 161, 162, 163, 164, 165, 166, 169, 170, 171, 172, 173, 174, 176, 177, 178, 179, 180, 181, 182, 184, 191, 192, 194, 199, 200, 202, 208, 210, 211, 213, 214, 215, 247, 248, 249, 250, 251, 254, 255, 258, 259, 262, 263, 296, 297, 298, 299, 304

Rappel : RI = 74.7% RII = 74.7% RIII = 63.2%

Précision : PI= 59.6% PII = 59.6% PIII = 57.3%

3.2.2 Teste 2

La requête : *Jakie sa parametry tranzystorow ?* (fr. quelles sont les paramètres des transistors ?).

Le corpus : 306 phrase, 9748 tokens.

Le numéro de phrase pertinents présents dans la base	Le numéro de phrase retrouver par la 1er méthode	Le numéro de phrase retrouver par la 2eme méthode	Le numéro de phrase retrouver par la 3eme méthode
2, 3 ,4 ,5 ,6, 8, 10, 11, 13, 15, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 40, 43, 44, 45, 53, 224, 231, 232, 233, 234, 235, 236, 238, 239, 240, 241, 250, 252, 259, 260, 261,	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25, 26, 27, 38, 39, 40, 42, 43, 44, 47, 50, 53, 55, 58, 59, 224, 225, 226, 228, 229, 230, 231, 232, 233, 234, 236, 237, 238, 239, 240, 243, 250, 251, 252, 259, 260, 261,	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25, 26, 27, 38, 39, 40, 42, 43, 44, 47, 50, 53, 55, 58, 59, 224, 225, 226, 228, 229, 230, 231, 232, 233, 234, 236, 237, 238, 239, 240, 243, 250, 251, 252, 259, 260, 261,	1, 2, 3, 5, 6, 15, 20, 26, 27, 38, 39, 40, 42, 43, 44, 47, 50, 53, 55, 58, 59, 224, 225, 226, 228, 229, 230, 231, 234, 236, 237, 238, 239, 250, 251, 259, 260, 261,

Rappel : RI = 73.8% RII = 73.8% RIII = 50%

Précision : PI= 58.4% PII = 58.4% PIII = 55.2%

3.2.3 Teste 3

La requête : *W jaki sposob mozemy podzilic rezystory ?* (fr. Comment on peut partage les résisteurs ?).

Le corpus : 297 phrase, 8841 tokens.

Le numéro de phrase pertinents présents dans la base	Le numéro de phrase retrouver par la 1er méthode	Le numéro de phrase retrouver par la 2eme méthode	Le numéro de phrase retrouver par la 3eme méthode
7, 11, 26, 27, 28, 36, 67, 68, 69, 70, 71, 72, 73, 76, 77, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 94, 107, 108, 109, 111, 114, 115, 116, 121, 129, 130, 134, 135, 136, 137, 138, 139, 163, 165, 166, 202, 208,	3, 7, 11, 21, 22, 23, 26, 27, 28, 30, 31, 32, 36, 37, 39, 43, 45, 46, 48, 50, 55, 67, 68, 69, 70, 72, 73, 76, 77, 79, 80, 81, 83, 85, 86, 87, 91, 94, 96, 97, 101, 104, 105, 107, 109, 111, 114, 115, 116, 117, 120, 121, 124, 126, 127, 129, 130, 134, 136, 137, 139, 145, 147, 149, 154, 159, 160, 161, 163, 165, 166, 167, 168, 170, 171, 172, 173, 174, 175, 176, 180, 181, 182, 187, 189, 190, 191, 192, 193, 194, 196, 197, 198, 202, 204, 206, 208, 209, 219, 224, 236, 240, 243, 246, 253,	3, 7, 11, 21, 22, 23, 26, 27, 28, 30, 31, 32, 36, 37, 39, 43, 45, 46, 48, 50, 55, 67, 68, 69, 70, 72, 73, 76, 77, 79, 80, 81, 83, 85, 86, 87, 91, 94, 96, 97, 101, 104, 105, 107, 109, 111, 114, 115, 116, 117, 120, 121, 124, 126, 127, 129, 130, 134, 136, 137, 139, 145, 147, 149, 154, 159, 160, 161, 163, 165, 166, 167, 168, 170, 171, 172, 173, 174, 175, 176, 180, 181, 182, 187, 189, 190, 191, 192, 193, 194, 196, 197, 198, 202, 204, 206, 208, 209, 219, 224, 236, 240, 243, 246, 253,	3, 7, 11, 26, 27, 28, 30, 31, 32, 33, 36, 37, 39, 48, 50, 67, 68, 69, 73, 76, 77, 79, 80, 81, 83, 145, 147, 149, 154, 159, 160, 161, 163, 165, 166, 167, 168, 170, 171, 173, 174, 175, 176, 180, 181, 182, 193, 197, 198, 202, 209, 219, 224, 236, 240, 243, 246, 253,

Rappel : RI = 80% RII = 80% RIII = 38%

Précision : PI= 38.1% PII = 38.1% PIII = 32.7%

3.2.4 Teste 4

La requête : *Projektowanie mikroprocesorow rodziny AVR w jezyku C ?* (fr. La programmation des microprocesseurs de famille AVR en langage C?).

Le corpus : 122 phrase, 2822 tokens.

Le numéro de phrase pertinents présents dans la base	Le numéro de phrase retrouver par la 1er méthode	Le numéro de phrase retrouver par la 2eme méthode	Le numéro de phrase retrouver par la 3eme méthode
1, 23, 24, 25, 26, 29, 30, 31, 33, 34, 35, 36, 37, 38, 40, 41, 42, 44, 51, 63, 64, 65, 71, 72, 80, 89, 100, 117, 118, 119,	1, 2, 6, 9, 16, 18, 23, 24, 25, 26, 31,33, 35, 36, 37, 38, 40, 41, 44, 51, 52, 71, 77, 78, 80, 82, 83, 89, 100, 101, 105, 112, 117, 119,	1, 2, 6, 9, 16, 18, 23, 24, 26, 31, 33, 35, 37, 38, 40, 44, 89, 105, 112, 117, 119,	18, 119

Rappel : RI = 70% RII = 36.7% RIII = 0.03%

Précision : PI= 61.8% PII = 61.1% PIII = 50%

3.2.5 Teste 5

La requête :*Czym cechuja sie mikrokontrolery rodziny PIC, instrukcje, architektura?* (fr. Quelles sont les paramètres de microcontrôleur PIC, les instructions, l'architecture?).

Le corpus : 506 phrase, 14212 tokens.

Le numéro de phrase pertinents présents dans la base	Le numéro de phrase retrouver par la 1er méthode	Le numéro de phrase retrouver par la 2eme méthode	Le numéro de phrase retrouver par la 3eme méthode
4, 42, 53, 54, 55, 56, 81, 82, 83, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96, 97, 101, 102, 103, 128, 160, 161, 167, 199, 221, 258, 285, 286, 288, 289, 290, 291, 292, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 306, 310, 311, 312, 313, 314, 315, 322, 323, 324, 325, 326, 327, 340, 341, 361, 362, 363, 364, 365, 366, 382, 383, 384, 385, 386, 464, 465, 466, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 508,	4, 5, 7, 16, 34, 42, 53, 54, 55, 56, 59, 63, 65, 67, 68, 70, 81, 82, 83, 85, 86, 87, 91, 92, 95, 96, 97, 101, 118, 126 , 127, 128, 135, 136, 161, 163, 167, 221, 258, 285, 286, 289, 291, 295, 296, 297, 298, 304, 306, 311, 322, 323, 325, 326, 327, 333, 351, 358, 363, 364, 375, 382, 386, 434, 465, 473, 490, 492, 493, 495, 496, 499, 501, 505, 508, 509,	4, 5, 7, 16, 34, 42, 53, 54, 59, 63, 65, 67, 68, 81, 82, 83, 85, 86, 87, 91, 92, 96, 97, 101, 118, 126, 127, 128, 135, 161, 163, 285, 286, 289, 291, 298, 304, 306, 311, 322, 323, 333, 351, 358, 363, 364, 375, 386, 434, 465, 473, 490, 492, 493, 495, 496, 499, 501, 505, 508, 509,	4, 5, 7, 16, 34, 42, 53, 54, 59, 63, 65, 67, 68, 81, 82, 83, 86, 91, 92, 97, 101, 118, 126, 127, 128, 135, 161, 163, 285, 311, 333, 351, 358, 363, 364, 490, 492, 509,

Rappel : RI = 55% RII = 40.6% RIII = 20.9%

Précision : PI= 65.9% PII = 60.7% PIII = 50%

3.3 Les résultats

Nous avons effectué plusieurs testes de chaque méthode implémentée. Il faut préciser que les méthodes proposées sont difficile à tester, car il

n'existe pas des testes standards pour pouvoir décrire leur performance exacte. Pour cela, Nous nous sommes intéressé aux deux paramètres que sont:

- Le rappel
- La précision.

D'après les testes sur les deux premières méthodes appliquées (les méthodes qui effectuent les concordances entre la requête d'utilisateur et la base des graphes), nous pouvons constater que la recherche des graphes directement de la requête d'utilisateur donne des résultats satisfaisants. C'est à dire, qu'il est toujours possible de trouver les graphes si les graphes existent – pour les deux méthodes - et si pour la première méthode les mots clés (classes sémantique) sont présents. Si les deux méthodes de recherche donnent des résultats satisfaisants, elles sont loin d'être complets et elles manquent de précisions , car en d'après le résultat de recherche, en retrouve trop d'occurrences dans le corpus qui ne sont pas intéressantes en vu de la requête de l'utilisateur.

Dans nos testes, pour avoir notre corpus, nous avons utilisé de plusieurs textes du domaine (les dépêche de presse, les textes retrouver sur Internet) et pour avoir des requêtes d'utilisateur, nous avons utilisé les questions posées sur les groupes publiques du domaine. Ces testes nous ont permit d'améliorer notre base des graphes. En revanche, il est nécessaire de préciser qu'avec nos deux méthodes, nous trouvons souvent des graphes non permanents ou bien des graphes qui ne sont pas assez détaillés. Nous avons également remarqué que le domaine est beaucoup plus large que nous l'avons imaginer.

Les différents tests que nous avons effectué nous en permis de :

- agrandir la base des graphes en essayant d'avoir les graphes les plus précis,
- ajouter des classes sémantiques,
- agrandir le dictionnaire du domaine.

Malgré ces améliorations que nous avons apportés, nous sommes encore loin du cas idéale. D'après nos résultats de tests, et comme il faut partir du principe que les graphes devrons être plus complexes et plus compliqués, nous avons constaté que la méthode des classes sémantiques donne de meilleurs résultats que la méthode directe. En effet, dans la seconde méthode, nous ne pouvons pas profiter de l'agrandissement de complexité des graphes, c'est a dire, en recherchant directement des mots clés dans les graphes on n'a pas la possibilité de tenir compte du format du graphe, donc souvent, le graphe retrouvé n'est pas permanent alors que dans la première méthode ce problème est réglé grâce a des classes sémantiques.

La précision retrouvées dans le corpus diminue avec l'augmentation du nombre des graphes non permanents utilisé pour la recherche. Nous avons pourtant préciser dans le chapitre précédent, que nous nous sommes basés sur l'idée de retrouver tous les occurrences probables et c'est à l'utilisateur d'effectuer le filtrage.

Malgré tout les inconvénient cités ci-dessus, notre recherche des occurrences directement de la requête d'utilisateur donne des résultats satisfaisants. Nous extrairons les informations liées avec la requête comme par exemple nous donnons des informations avec des synonymes, avec des composants de la même famille ou des composants qui ont les mêmes paramètres ; ainsi on tient compte des difficultés posées par les formes grammaticales comme par exemple la déclinaison.

Les résultats donnés par la troisième méthode qui consiste a la création du graphe directement de la requête sont faciles a supposer. On peut décrire cette méthode comme la méthode du recherche directe des mot des clés en tenant compte de tout les formes grammaticales. C'est pourquoi cette méthode est utilisée seulement en cas où la base ne contient pas des graphes permanents avec la requête d'utilisateur.

D'après les différents testes effectué nous pouvons donner les résultats suivant :

Premier méthode	Deuxième méthode	Troisième méthode
Précision = 70.7% Rappel = 56.76%	Précision = 61.16% Rappel = 55.58%	Précision = 34.4% Rappel = 49.04%

CONCLUSION

Nous nous sommes focalisés sur le travail de la recherche automatique d'information dans un corpus, plus précisément sur l'adaptation des outils linguistique pour l'extraction des informations concernant un domaine précis. Notre étude était faite sur l'application "Unitex" puisqu'elle permet d'effectuer une recherche profonde en utilisant les grammaires, les tables de lexique-grammaire et les dictionnaires. Notre objectif était de :

- adapter le logiciel pour travailler avec la langue polonaise,
- développer des ressources linguistiques pour pouvoir effectuer notre recherche
- ajouter plusieurs méthodes afin de permettre une recherche automatique des occurrences dans le corpus à partir de la requête d'utilisateur.

Nous avons réussi la création et l'intégration de trois nouvelles méthodes. Ces méthodes permettent d'effectuer les correspondances entre la requête d'utilisateur et les graphes. Ces derniers sont utilisées pour la recherche des informations permanentes via la question d'utilisateur. L'adaptation des ressources linguistiques comme la création des graphes ou l'adaptation du dictionnaires faisait partie du notre travaille. Nous avons obtenu des résultats satisfaisant mais il est nécessaire de préciser qu'ils restent plusieurs points a améliorer.

Les solutions de la recherche automatique des informations proposer dans ce rapport donnent une image de la complexité de ce domaine et met en évidence la nécessité de faire des améliorations et surtout ouvre plusieurs portes dans la recherche. Les résultats obtenus pourrons être améliorer en effectuant une recherche plus profonde sur la compréhension de la requête d'utilisateur ou bien en améliorant les ressources linguistiques, comme par exemple la base des graphe. Malheureusement la possibilité d'avoir des solutions idéales qui permettent de retrouver que les informations utiles semble être une utopie.

Au point de vue personnel, ce premier année de thèse m'a permis de m'orienter vers un domaine passionnant celui du filtrage et d'extraction d'information a base de connaissances linguistique. De plus, j'avais la possibilité d'apprendre un nouveau environnement de programmation (JAVA)

qui va me permettrent de développer et de mieux comprendre les applications écrites dans ce langage.

Bibliographie

1. VETULANI Z., WALCZAK B., Obrębski T. – *Unambiguous coding of the inflection of Polish nouns and its application in the electronic dictionaries - format POLEX / Jednoznaczne kodowanie fleksji rzeczownika polskiego i jego zastosowanie w słownikach elektronicznych - format POLEX*, Adam Mickiewicz University Press, Poznań, 1998 (co-authors: B. Walczak, T. Obrębski, G. Vetulani)

2. VOORHEES, E.M. – « Natural language processing and information retrieval ». In M.T. PAZIENZA, *Information extraction, toward scalable, adaptable systems,* Springer Verlag (Lecture Notes in Computer Science), Heidelberg, 1999, pp32-48.

3. SABACH G. – « Sens et traitements automatique des lanque ». In PIERREL J., *Ingenierie des langues. Hermes, Paris, 2001, pp. 77-129*

4. SALTON G. et McGILL M. – *Introduction to modern information retrieval.* McGaw-Hill, New York,1983.

5. WOODS W.A. – « Progress in natural language understanding : An application to lunar geology ». In *The American Federation of Information Processing Saocieties Conference Proceedings.* Montvale (AFIPS'1973), 1973, pp.441-450.

6. ALSHAWI H. – *The core language Engine*. MIT Press (ACL-MIT Press Series in Natural language Processing), Cambigde, 1992.

7. SOWA J. – *Conceptual Structures. Information processing in Mind and Machine.* Addison Wesley Publishing CO., Reading, 1984.

8. KAMP H. – « Evenemts, representations discursives et reference temporelle ». *Langages, nb 64, 1981, pp.39-64.*

9. COLLINS M. – « A new statistical parser based on bigram lexical dependencies ». In *Proceeding of ACL 96*, 1999.

10. PAUMIER S. – *Recherche d'expressions dans de grands corpus : le systeme AGLAE.* Memoire de DEA. Universite de Marne-la-Valee, 2000.

11. SILBERZTEIN M. – *Dictionnaires electronique et analyse automatique de texte, le sesteme INTEX*. Masson, Paris, 1993.

12. PAUMIER S. - *Unitex 1.2 Manuel d'utilisation.* Université Marne la Vallée, 2004.

13. PAUMIER S. – *De La reconnaissance de formes linquistique a l'analyse syntaxique*. These, Marne-la-Valee, 2003.

14. GROSS M. – « The construction of local grammars », *Finite-State Languauga Processing*, E. Roche and Y. Schabes, Cambridge, Mass./London, England : MIT Press, pp 329-354, 1997.

15. Balvet A. – *Grammaires locales et lexique-grammaire pour le filtrage d'information Vers une (re)utilisabilite des ressources linquistique pour la recherche d'information.* Conference TIA, Nancy, 2001.

16. GARDENT C., GUILLAUME B.,Falk I., PERRIER G.- *Le lexique-grammaire de M.Gross et le traitement automatique des langues.* LORIA & ATILF, Nancy, 2005.

17. MACLEOD C., GRISHMAN R., MEYERS A.- « Comlex syntax : Building a computational lexicon. In *Preceedings of COLING'94,* pp 2668-272, 1994.

18. PAZIENZA M.T. – *Information extraction (a multidiscipilinary approach to an emerging information technology).* Springer Verlag (Lecture Notes in Computer Science), Heidelberg, 1997

19. SALTON G., McGILL M,J. – *Information retrieval*, volume 1. McGraw-Hill Book Company, 1983

20. Leloup C. – *Moteur d'indexation et de recherché.* Eyrolles, 61, Bld Saint-Germain, 75240 Paris, 1998

Annexe I. Expressions régulières, automates et transducteurs dans Unitex

Le système Unitex permet de construire des expressions régulières des automates et des transducteurs a nombre fini d'états pour le traitement automatique des langues. Une description sommaire des principales caractéristiques est fournie dans le rapport. Nous présenterons ici certains aspect formels des langages à nombre finis d'états.

1. Rappels sur les langages formels.

Soit un ensemble fini non vide *A*, que l'on appelle *alphabet*. Toute suite finie d'éléments de *A* est un *mot.*

Le *mot vide*, note ε, est une suite qui ne comporte aucun élément.

Etant donne deux mots x et y, on peut leur associer un troisième mot qui est obtenu par la concaténation de x et y. On note xy la *concaténation* de x et y.

L'ensemble de tous les mots composes d'éléments de l'alphabet *A* est le monoïde libre sur *A,* note *A*.*

1.1 Langage.

Tout sous-ensemble d'un monoïde libre *A** constitue un langage formel défini sur *A.* On connaît les opérations suivantes sur les langages :

Intersection : a deux langages L1 et L2, on associe un troisième langage forme par l'intersection ensembliste de L1 et L2. Ce langage est note L1∩L2.

Union :a deux langages L1 et L2, on associe un troisième langage forme par l'union ensembliste de L1 et L2. Ce langage est note L1∪L2 ou L1+L2.

Complément : a un langages L1 et L2, on associe un autre langage note L ou A* - L, compose de tous les mots de A* qui ne sont pas dans L.

Produit : a deux langage L1 et L2, on associe un troisième langage note L1L2 tel que pour tout mot x de L1, pour tout mot y de L2m xy appartient à L1L2.

Opération étoile : à un langage L, on associe un langage note L* tel que, pour tout nombre entier n, si x un mot de L, la concaténation x avec lui même (n-1) fois forme un mot de L*.

1.2 Expressions régulières.

Une expression régulière sur un alphabet A est définie de la manière suivante :

1. le mot vide ε est une expression régulière
2. si a est un élément de A, alors a est une expression régulière
3. si R est une expression régulière, alors R* est une expression régulière
4. si R et S sont des expression régulières, alors RS et R+S sont des expressions régulières (concaténation et union)

1.3 Automates.

Un automate à nombre fini s'états est défini par un quadruple

$$< K, V, Q, \delta >$$

- K est un ensemble fini d'états $\{E_0, E_1...E_n\}$ où E_0 désigne toujours l'état initial
- V désigne un vocabulaire ou alphabet des entrées $\{a_0, a_1...a_k\}$
- Q est un sous-ensemble de K dont les éléments sont appelés états terminaux
- δ est une application, dite fonction de transition qui, à tout couple compose d'un élément q de K et d'un élément a de V, associe un sous-ensemble de K. On écrit $r \in \delta (q,a)$ dans le cas ou r appartient à ce sous-ensemble. (E_0, a_0) --> E_k (*qui se lit: si l'automate est dans l'état E_0 et reçoit l'entrée a_0, alors il passe à l'état E_k*)

Un automate est dit *déterministe,* si, a un couple compose d'un élément q de K et d'un élément a de V, la fonction δ associe plus un élément r de K.

Théorème de Kleene (1956)

Pour tout automate à nombre fini d'états, il existe une expression régulière qui représente le langage reconnu.

Pour toute expression régulière, il existe un automate à nombre fini d'états qui représente le langage représente.

1.4 Transducteurs.

Un transducteur à nombre fini d'états est défini par un sextuplet

$$< K, Ve, Vs, Q, F, \delta >$$

- K est un ensemble fini d'états
- Ve désigne le vocabulaire ou alphabet d'entrée (destine à être reconnu)
- Vs désigne le vocabulaire ou alphabet de sortie (ou de réécriture)
- Q est un élément appelé état initial
- F est un sous-ensemble de K dont les éléments sont appelés états terminaux
- δ est une application, dite fonction de transition qui, à tout couple compose d'un élément q de K et d'un élément a de Ve, associe un sous-ensemble de K. On écrit $r \in \delta (q,a)$ dans le cas ou r appartient à ce sous-ensemble.

Un transducteur permet donc, via le vocabulaire Vs, d'associer une séquence de sortie à une séquence reconnu. On utilise les transducteurs pour annoter les texte (ajouter des information linguistique, baliser les séquences pertinentes, etc.).

2. Unitex et la technologie a nombre fini d'états

2.1 Alphabet et symboles utilises.

Alphabet

Unitex permet à l'utilisateur de définir son propre alphabet pour une langue donne. Voici un exemple d'alphabet défini dans Unitex, avec la correspondance minuscule majuscule.

Aa, Ąą, Bb, Cc, Ćć, Dd, Ee, Ęę, Éé, Ëë, Ff, Gg, Hh, Ii, Jj, Kk, Ll, Łł, Mm, Nn, Ńń, Oo, Óó, Öö, Pp, Qq, Rr, Ss, Śś, Tt, Uu, Üü, Vv, Ww, Xx, Yy, Zz, Żż, Źź

Codes grammaticaux usuels

Code	Signification	Exemples
A	adjectif	fabuleux
ADV	adverbe	réellement, à la longue
CONJC	conjonction de coordination	mais
CONJS	conjonction de subordination	puisque, à moins que
DET	déterminant	ses, trente-six
INTJ	interjection	adieu, mille millions de mille sabords
N	nom	prairie, vie sociale
PREP	préposition	sans, à la lumière de
PRO	pronom	tu, elle-même
V	verbe	continuer, copier-coller

Codes sémantiques

Code	Signification	Exemple
z1	langage courant	blague
z2	langage spécialisé	sépulcre
z3	langage très spécialisé	houer
Abst	abstrait	bon goût
Anl	animal	cheval de race
AnlColl	animal collectif	troupeau
Conc	concret	abbaye
ConcColl	concret collectif	décombres
Hum	humain	diplomate
HumColl	humain collectif	vieille garde
t	verbe transitif	foudroyer
i	verbe intransitif	fraterniser
en	particule pré-verbale (PPV) obligatoire	en imposer
se	verbe pronominal	se marier
ne	verbe à négation obligatoire	ne pas cesser de

Codes flexionnelles usuels.

Code	Signification
m	masculin
f	féminin
n	neutre
s	singulier
p	pluriel
1, 2, 3	1^{ere}, 2^{eme}, 3^{eme} personne
P	présent de l'indicatif
I	imparfait de l'indicatif
S	présent du subjonctif
T	imparfait du subjonctif
Y	présent de l'impératif
C	présent du conditionnel
J	passé simple
W	infinitif
G	participe présent
K	participe passé
F	futur

Symboles spéciaux.

- <E> : mot vide, ou epsilon. Reconnaît la séquence vide;
- <^> : reconnaît un retour a la ligne;
- <$> : séparateur de phrase;
- <L> : reconnaît n'importe quelle lettre;
- <PNC> : reconnaît les symboles de ponctuation;
- <TOKEN> : reconnaît n'importe quelle unité lexicale;
- <MOT> : reconnaisse n'importe quelle unité lexicale formée de lettres;

- <MIN> : reconnaît n'importe unité lexicale formée de lettres minuscules;
- <MAJ> : reconnaît n'importe unité lexicale formée de lettre majuscules;
- <PRE> : reconnaît n'importe unité lexicale formée de lettres et commencent par une majuscule;
- <DIC> : reconnaît n'importe quel mot compose figurant dans les dictionnaire du texte;
- <NB> : reconnaît n'importe quelle suite de chiffres contigus;
- # : interdit la présence de l'espace.

Lemme

Avec Unitex il est possible de faire référence à toutes les formes fléchies d'un mot en décrivant le lemme de ce mot entre chevrons>

2.2 Expressions régulières, automates et transducteurs.

Le système Unitex permet de traiter des expressions régulières, des automates et des transducteurs. L'implantation de ces formalismes est décrite dans le projet. Nous donnons ci-dessous des exemples de chacun de ces formalismes dans Unitex.

Expression régulière

(<automate> + <transducteur>) (<E> + à nombre fini d'états + fini)

Cette expression permet de reconnaître les séquences suivantes:

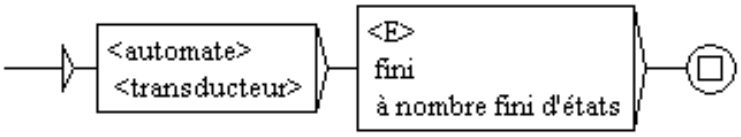
automate; automates, transducteur; transducteurs; automate à nombre fini d'états; automates à nombre fini d'états, transducteur à nombre fini d'états; transducteurs à nombre fini d'états; automate fini; transducteurs fini; etc.

L'étiquette <E> autorise une chaîne vide comme second élément, ce qui permet de reconnaître Automate ou Transducteur de manière isolée.

Automate

Unitex permet de représenter un ensemble d'expressions linguistique sous la forme d'un automate. Dans la représentation proposée, les graphes

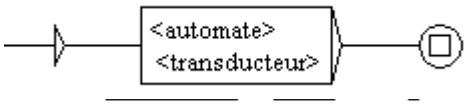
contiennent les éléments du vocabulaire dans des "boites" (correspondant aux états de l'automate, alors que traditionnellement ces éléments figurent



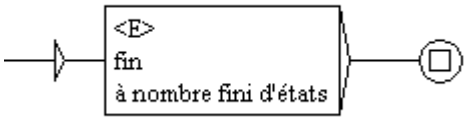
sur les transitions), mais cela ne changent en rien les propriétés des objets manipules.

Cet automate est strictement équivalent à l'expression régulière donnée précédemment. Unitex permet également de modéliser des automates récursifs (RTN), ou un état correspond en fait à un sous-ensemble appelé dynamiquement. L'appel à un sous-graphe apparaît en grise. Le graphe suivant :

est équivalent au précédent s'il existe un automate appelé *automate* qui a la forme suivante :

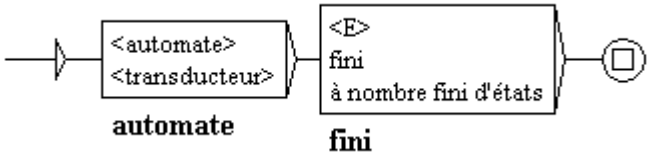


et un appelé *fini* qui a la forme suivante:



Transducteur

Suivant la définition un transducteur Unitex est un automate auquel est associe un vocabulaire de sortie. Le transducteur suivant permet de reconnaître un certain nombre de formes et leur associe systématiquement la sortie *automate fini.*



2.3 Opération sur les graphes

Les automate peuvent subir l'opération étoile, ainsi que l'union, l'intersection et le calcul du complémentaire. Par exemple: l'union consiste en fait essayer d'applique un automate A à partir des états de l'automate B et l'automate A doit être complètement parcouru à partir d'un état quelconque de l'automate B.

Passage des dictionnaires sur le texte

Les dictionnaire Unitex sont compiles sous forme de transducteurs à nombre fini d'états. Le passage d'un dictionnaire sur le texte revient à faire l'union entre le transducteur du texte et le transducteur du dictionnaire. On obtient un graphe ou chaque ambiguïté est représentée par une nouvelle *boite* dans le graphe résultat.

Passage d'une grammaire sur le texte

Une grammaire étant elle-même un transducteur ou un automate, le passage d'une grammaire sur un texte revient à calculer l'intersection de la grammaire avec le texte. A l'inverse de l'union de l'union de graphe, l'opération d'intersection telle qu'elle est défini dans Unitex a tendance à enlever des chemins et à regrouper certains éléments.

Annexe II. Les classes des méthodes en Java

1. Première méthode 1 :

```
public class Methode1{
	Vector dico=new Vector();
	Vector mots=new Vector();
	Vector corespond=new Vector();
	Vector graphs=new Vector();
	Vector existedgraphs=new Vector();
	String requete;
	Vector gf;
	public Methode1(String s,String lang, Vector gf)
	{
		requete=s;
		this.gf=gf;
		try{
		  java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\ele_dic.def");
		  java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file));
		  String line;
		  String renome[]=new String[50] ;
		  while((line=mybuffer.readLine())!=null)
		  {
		    renome=line.split("[.]");
			dico.addElement(renome[0]+".dic");
		//	System.out.println(dico.elementAt(0));
		  }
	    }
	    catch(Exception e)
	    {
		System.out.println(e);
	    }
   }
	// decoupage de la requete en mots --> mots
	public void setMots()
	{
		String test[]=new String[50];
		test=requete.split(" ");
		for(int j=0; j < test.length; j++)
		{
			mots.addElement(test[j]);
		}
	}
	// la recherche des directs
	public void setDirect()
	{
		for(int i=0; i < dico.size(); i++)
		{
		  try{
			System.out.println(dico.elementAt(i)+" w jakich slownikach");
			java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Dela\\"+dico.elementAt(i));
		    java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file,"UTF-16LE"));
		    String line;
		    String line1;
		    String test[]=new String[50];
		    String test1[]=new String[50];
		    while((line=mybuffer.readLine())!=null)
		    {
		      test=line.split(",");
		      if(mots.elementAt(j).equals(test[0]))
		      {	//wez lemme czyli miedzy [,] a [.] i dorzucony lemme do V corespond
			test=line.split("\\p{Punct}");
```

```
                        //      System.out.println(test[1]);
                                if(!corespond.contains(test[1])){
                                  corespond.addElement(test[1]);
                                 // teraz dorzuca wszystkie formy flechie
                                 java.io.FileInputStream file2 = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Dela\\"+dico.elementAt(i));
                                 java.io.BufferedReader mybuffer1 = new java.io.BufferedReader(new
java.io.InputStreamReader(file2,"UTF-16LE"));
                                 boolean arret=false;
                                 //se pointer vers le premier element
                                        while((line1=mybuffer1.readLine())!=null)
                                                { test1=line1.split("\\p{Punct}");
                                                  if(test1.length>1){
                                                        if(test[1].equals(test1[1]))
                                                    {  if(!corespond.contains(test1[0])){
                                                       espond.addElement(test1[0]);
                                                            }
                                                    }
                                                  }
                                        }
                                  }
                                 }

                                }
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
                        }
                        }
                        for(int h=0; h < corespond.size(); h++){
                            System.out.println(corespond.elementAt(h) +" l+ff");
                            }
                        System.out.println("procedure terminee l+ff");
        }
        // la recherche des graphes
        public void setGraph()
        {
                //zebranie wszystkich grafow z pliku user_grph
                        java.io.FileInputStream file;
                   java.io.BufferedReader mybuffer;
                   try{
                   file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\user_grph.def");
                   mybuffer = new java.io.BufferedReader(new java.io.InputStreamReader(file));
                   String line;
                   String test3[]=new String[50];
                   while((line=mybuffer.readLine())!=null)
                   {     test3=line.split(" ");
                        for(int j=1;j<test3.length ;j++)
                        {
                                existedgraphs.addElement(test3[j]);
                       }
                   }
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                for(int i=0; i < corespond.size();i++)
                {
                        try
                        {for(int k=0; k < existedgraphs.size(); k++){
                        java.io.FileInputStream file1 = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Graphs\\"+existedgraphs.elementAt(k));
                        java.io.BufferedReader mybuffer1 = new java.io.BufferedReader(new
java.io.InputStreamReader(file1,"UTF-16LE"));
                        String line1;
                        String test2[]=new String[100];
```

```
                        while((line1=mybuffer1.readLine())!=null)
                        {       test2=line1.split("\\p{Punct}");
                                for(int p=0; p < test2.length; p++){
                                        if(corespond.elementAt(i).equals(test2[p])){
                                                System.out.println(corespond.elementAt(i));
                                                System.out.println(test2[p]);
                graphs.addElement(existedgraphs.elementAt(k));
                                                }
                                        }
                                }
                                }
                 }
                 catch(Exception e)
                 {
                        System.out.println(e);
                 }
                }
                 for(int h=0; h < existedgraphs.size(); h++){
                    }
                 for(int a=0; a < graphs.size(); a++){
                    System.out.println(graphs.elementAt(a)+" znalazlem!!!");
                    }
        }
        // appel des applications unitex
        public void setUnitex()
        {       if(graphs.size()>0){
launchLocate(Config.getUserDir().getAbsolutePath()+"\\Polish\\Graphs\\"+graphs.elementAt(0));
                }
                else
                {
                }
        }
                public int launchLocate(String chemin) {
                MultiCommands commands = new MultiCommands();
                File fst2=null;
                        String grfName = chemin;
                        if (grfName.substring(grfName.length() - 3, grfName.length()).equalsIgnoreCase("grf")) {
                                // we must compile the grf
                                Grf2Fst2Command grfCmd = new Grf2Fst2Command().grf(new File(grfName));
                                commands.addCommand(grfCmd);
                                String fst2Name = grfName.substring(0, grfName.length() - 3);
                                fst2Name = fst2Name + "fst2";
                                fst2 = new File(fst2Name);
                System.out.println("methode2 : ,,,fst2Name="+ fst2Name+" ---
Config.getCurrentSnt()="+Config.getCurrentSnt());
                        }
                if (gf.size()>0)
                                for (int ii=0;ii<gf.size();ii++)
                                {
                                        Config.setCurrentSnt(new File(""+gf.elementAt(ii)));
                                        LocateCommand locateCmd = new
LocateCommand().snt(Config.getCurrentSnt()).fst2(fst2).alphabet();
                                        locateCmd = locateCmd.longestMatches();
                                        locateCmd = locateCmd.ignoreOutputs();
                                        locateCmd = locateCmd.noLimit();
                                        commands.addCommand(locateCmd);
                                        File prevConcord=new File(Config.getCurrentSntDir(),"previous-
concord.ind");
                if (prevConcord.exists()) {
                        prevConcord.delete();
                }
                File concord=new File(Config.getCurrentSntDir(),"concord.ind");
                if (concord.exists()) {
                        concord.renameTo(prevConcord);
                }
                new ProcessInfoFrame(commands, true, new LocateDo());
                        }
                }
                else
                {if(Config.getCurrentSnt()!=null)
```

```
                                {LocateCommand locateCmd = new
LocateCommand().snt(Config.getCurrentSnt()).fst2(fst2).alphabet();
                                        locateCmd = locateCmd.longestMatches();
                                        locateCmd = locateCmd.ignoreOutputs();
                                        locateCmd = locateCmd.noLimit();
                                        commands.addCommand(locateCmd);
                                        File prevConcord=new File(Config.getCurrentSntDir(),"previous-concord.ind");
                                if (prevConcord.exists()) {
                                        prevConcord.delete();
                                }
                                File concord=new File(Config.getCurrentSntDir(),"concord.ind");
                                if (concord.exists()) {
                                        concord.renameTo(prevConcord);
                                }
                                System.out.println("KAKA plus...");
                            new ProcessInfoFrame(commands, true, new LocateDo());
                                }
                        }
                        return 0;
                }
                class LocateDo extends ToDoAbstract {
                        public LocateDo() {
                        }
                        public void toDo() {
                                String res = LocateFrame.readInfo(new File(Config.getCurrentSntDir(),
                                                "concord.n"));
                                String res2 = UnicodeIO.readFirstLine(new File(Config
                                                .getCurrentSntDir(), "concord.n"));
                                res2=res2.substring(0,res2.indexOf(' '));
                                ConcordanceParameterFrame.showFrame(Util.toInt(res2));
                        }
                }
                public void start()
                {
                        setMots();
                        setDirect();
                        setGraph();
                        setUnitex();
                }
}
```

2.Second méthode.

```
public class Methode2{

        Vector dico=new Vector();

        Vector mots=new Vector();

        Vector corespond=new Vector();

        Vector graphs=new Vector();

        String requete;

        Vector gf;

        public Methode2(String s,String lang, Vector gf)

        {

                requete=s;

                this.gf=gf;

                try{
```

```
                java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\user_dic.def");
                java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file));
                String line;
                String renome[]=new String[50] ;
                while((line=mybuffer.readLine())!=null)
                {
                  renome=line.split("[.]");
                dico.addElement(renome[0]+".dic");
                }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
    }
        public void setMots()
        {
                String test[]=new String[50];
                test=requete.split(" ");
                for(int j=0; j < test.length; j++)
                {
                        mots.addElement(test[j]);
                }
        }
        public void setSymentic()
        {
                for(int j=0; j < mots.size(); j++)
                {
                for(int i=0; i < dico.size(); i++)
                {
                  try{
                        java.io.FileInputStream file = new
ava.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Dela\\"+dico.elementAt(i));
                    java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file,"UTF-16LE"));
                    String line;
                    String test[]=new String[50];
```

```
                        while((line=mybuffer.readLine())!=null)
                        {
                          test=line.split(",");
                          if(mots.elementAt(j).equals(test[0]))
                          {
                                test=line.split("[+]");
                                if(!corespond.contains(test[test.length-1])){
                                  corespond.addElement(test[test.length-1]);
                           }
                          }
                        }
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
                        }
                        }
        }
        public void setGraph()
        {
                        for(int i=0; i < corespond.size();i++)
                        {
                                try
                                {
                                java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\user_grph.def");
                                java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file));
                        String line;
                        String test[]=new String[50];
                        while((line=mybuffer.readLine())!=null)
                        {
                          test=line.split(" ");
                          if(test[0].equals(corespond.elementAt(i)))
                          {
                                for(int j=1;j<test.length ;j++)
                                {
                                  if(!graphs.contains(test[j])){
```

```
                                        graphs.addElement(test[j]);
                                    }
                                }
                            }
                           }
                          }
                          catch(Exception e)
                          {
                                System.out.println(e);
                          }
                         }
        }
```

3. La construction automatique de graphe.

```
public class Methode3
{
    Vector dico=new Vector();
    Vector mots=new Vector();
    Vector noun=new Vector();
    Vector graph=new Vector();
    Vector gf;
    String requete;
    public Methode3(String s,String lang, Vector gf)
    {
        requete=s;
        this.gf=gf;
        try{
          java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\user_dic.def");
          java.io.BufferedReader mybuffer = new java.io.BufferedReader(new java.io.InputStreamReader(file));
          String line;
          String renome[]=new String[50] ;
          while((line=mybuffer.readLine())!=null)
          {
            renome=line.split("[.]");
                dico.addElement(renome[0]+".dic");
          }
      }
```

```
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    // decoupage de la requete en mots --> mots
    public void setMots()
    {
            String test[]=new String[50];
            test=requete.split(" ");
            for(int j=0; j < test.length; j++)
            {
                    mots.addElement(test[j]);
            }
    }
    public void setNoun()
    {
            for(int j=0; j < mots.size(); j++)
            {
                    for(int i=0; i < dico.size(); i++)
                    {
                            try{
                            java.io.FileInputStream file = new
java.io.FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Dela\\"+dico.elementAt(i));
                    java.io.BufferedReader mybuffer = new java.io.BufferedReader(new
java.io.InputStreamReader(file,"UTF-16LE"));
                            String line;
                    String test[]=new String[50];
                    while((line=mybuffer.readLine())!=null)
                    {       test=line.split("\\p{Punct}");
                            if(mots.elementAt(j).equals(test[0]))
                            {       if(test[2].equals("N")){
                                            if(!noun.contains(test[1]))
                                            {
                                            noun.addElement(test[1]);
                                            }
                                    }
                            }
                    }
```

```
                        }
                    catch(Exception e)
                    {
                    System.out.println(e);
                    }
                    }
            }
    }
    public void setGraph()
    {       String str = "";
            String out = "";
            int znak = 0;
            try{
            FileInputStream wejscie = new
FileInputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Graphs\\bgt1.grf");
            java.io.BufferedReader wejscie1 = new java.io.BufferedReader(new java.io.InputStreamReader(wejscie,"UTF-
16LE"));
                    znak = wejscie1.read();
                    while( znak != -1 ){
            str += (char) znak;
                    znak = wejscie1.read();
                    }
                    wejscie1.close();
                    System.out.println(str);
            }
            catch(IOException ex)
            {
              System.out.println("Blad zwiazany z odczytem z pliku. "+ex);
            }
            for(int i = 0; i<str.length(); i++){
                    if(str.charAt(i) == 'q')
                            {  for(int p = 0; p<noun.size()-1; p++)
                                    {
                                    out+="<"+ noun.elementAt(p)+">+";
                            }
                            out+="<"+ noun.elementAt(noun.size()-1)+ ">";
                    }
                    else
```

```
                out += str.charAt(i);
        }
          System.out.println(out);
        try{
                FileOutputStream wyjscie = new
FileOutputStream(Config.getUserDir().getAbsolutePath()+"\\Polish\\Graphs\\builed.grf");
                java.io.BufferedWriter wyjscie1 = new java.io.BufferedWriter(new
java.io.OutputStreamWriter(wyjscie,"UTF-16LE"));
                for(int i=0; i < out.length(); i++){
                   wyjscie1.write(out.charAt(i));
                }
                wyjscie1.close();
        }
        catch(IOException ex)
        {
                System.out.println("Blad zwiazany z zapisem do pliku. "+ex);
        }

    }
```