SAFAR: vers une plateforme ouverte pour le traitement automatique de la langue Arabe

Seddik SIDRINE, Younes SOUTEH, Karim BOUZOUBAA, Taoufik LOUKILI
EMI, Université Mohamed V-Agdal,
Avenue Ibnsina B.P. 765 Rabat Maroc

Résumé
Différents travaux en traitement automatique de la langue font apparaître la nécessité d'articuler et faire coopérer des traitements et des ressources au sein de chaînes complexes. Une plateforme est le moyen d'assembler au sein du même processus des traitements et des ressources de natures et de provenances diverses. Au delà des plateformes de traitement de la langue qui sont basées sur les langues indo-européennes, des solutions adaptées doivent être proposées pour résoudre les contraintes spécifiques à la langue Arabe. La plateforme SAFAR ('Voyage' en Arabe) propose une architecture et un ensemble d'outils visant à faciliter la mise en œuvre de tels « assemblages », afin de faciliter le développement des applications destinées au traitement de la langue Arabe. Cela est rendu possible à travers une plateforme intégrée avec un ensemble de services et des ressources préexistantes, et une architecture ouverte et extensible qui supporte des implémentations tiers. Dans cet article, nous allons présenter notre approche pour le développement de l'architecture de la plateforme SAFAR et notre état d'avancement pour sa réalisation.

Mots clés : *Traitement automatique de la langue Arabe, plateforme de traitement de la langue, analyse morphologique.*

I. INTRODUCTION

L'intérêt au traitement automatique de la langue Arabe (TALA) ne cesse de se développer depuis les années 80. Plusieurs recherches ont été conduites dans ce domaine :

- analyse morphologique pour définir la structure des mots [2] [3][7].
- désambiguïsation morphologique pour le choix d'une seule solution par mot et étiquetage automatique (POS Tagging) pour déterminer la catégorie et les attributs grammaticaux d'un mot [10][12].
- restauration des voyelles d'un texte (Diacritization) [14][15].
- analyse syntaxique pour permettre de déterminer la structure d'une phrase ou d'un texte selon une grammaire formalisée [5].
- analyse sémantique qui permet de déterminer le sens des mots du texte [21].
- récupération d'informations (Information Retrieval) [22].
- les systèmes de question/réponse qui permettent de donner une réponse exacte à une question plutôt qu'une liste de passages comme le font les moteurs de recherche. La question et la réponse sont formulées en langage naturel [20].

Ces différentes recherches ont donné lieu à des propositions d'approches, d'algorithmes, de solutions voire aussi d'applications. Cependant, la maturité de ces recherches n'a touché que le niveau morphologique (analyse et génération) de la langue. Il y a donc un besoin général d'améliorer la maturité du TALA et de développer des applications avancées plus robustes (moteurs de recherche, systèmes de question/réponse, traduction simultanée, apprentissage à distance de la langue Arabe, etc.). Pour cela, les chercheurs ont besoin :

- de modules de base de traitement automatique de la langue Arabe comme l'analyse morphologique, syntaxique et sémantique
- de ressources gratuites (dictionnaires, corpus, base de données lexicales, …)
- de ressources et modules de comparaisons et d'évaluation
- d'utilitaires de traitement de la langue (outils de recherche de texte, outils statistiques sur les textes et corpus annotés, etc.)

Bien que quelques uns de ces modules et ressources existent, nous pouvons souligner les limitations suivantes :

- beaucoup de modules et ressources sont à finalité purement commerciale
- il est très souvent difficile de coupler un module (même libre) avec un autre (appeler par exemple, une analyse morphologique depuis un module de syntaxe). La raison étant que les modules sont développés de manière isolée
- la majorité des ressources ne respectent pas de standard de représentation
- il y a un manque de langage de programmation dédié au traitement automatique de la langue en général, et au TALA en particulier

Le besoin de la communauté TALA est donc de disposer d'une plateforme intégrée, disposant de ces modules, ces outils et ressources, avec les caractéristiques suivantes :

- sources libres
- flexibilité, pour appeler les modules souhaités
- ouverture pour pouvoir utiliser des modules ayant fait leur preuve. Par exemple, l'analyseur morphologique de Buckwalter[2] est très populaire auprès de la communauté TALA et il serait intéressant de continuer à l'utiliser dans le cadre d'une nouvelle plateforme

- extensibilité pour développer de nouvelles applications selon le besoin
- adéquation parfaite avec la langue Arabe

L'existence d'une telle plateforme ouverte, offrant les bases d'intégration des solutions de traitement de la langue Arabe, est à notre sens une voie efficace pour la standardisation, l'optimisation des efforts, la collaboration et l'accélération des développements dans le domaine. Des plateformes de traitement des langues existent déjà : GATE (General Architecture for Text Engineering) [25], NOOJ (Environnement de développement linguistique) [26], (UIMA : applications pour la gestion des informations non structurées) [27]. Ces plateformes offrent un espace de développement intégré, et surtout des mécanismes et ressources dédiés au traitement de la langue.

Cependant, ces plateformes ne répondent pas aux besoins de la communauté TALA, car non seulement elles n'intègrent pas des modules et des ressources pour le traitement de la langue Arabe, mais présentent certaines limitations que nous détaillerons dans la section 2 de ce papier.

Ainsi, notre objectif est de répondre aux différents besoins cités ci dessous, en développant une plateforme intégrée que nous baptisons **SAFAR** et qui constitue une continuité de travaux présentés précédemment.

Le reste de cet article se présente comme suit. La section 2 décrit les plateformes GATE, UIMA et NOOJ comme étant les plus connues dans le domaine de traitement des langues. La section 3 est dédiée à la description de la plateforme SAFAR. La section 4 concerne les ressources présentées par la plateforme SAFAR. La section 5 fait un focus sur l'intégration de l'analyse morphologique à la plateforme SAFAR. Enfin, en dernière section, nous concluons le travail et discutons quelques horizons futurs.

II. PLATEFORMES EXISTANTES

Cette section présente brièvement des architectures et plates-formes de référence de traitement du langage, en positionnant SAFAR par rapport à celles-ci.

GATE (General Architecture for Text Engineering)[25] est une infrastructure permettant le développement et le déploiement de composants pour le traitement de la langue naturelle. Développée depuis 1995 à l'Université de Sheffield, elle est largement utilisée sur des tâches de fouille de textes et d'extraction d'informations. GATE propose une architecture, un framework en Java (incluant de nombreux modules) et un environnement de développement intégré. Cependant, elle présente quelques limitations: son utilisation nécessite un coût d'apprentissage (le langage JAPE) et un certain niveau d'expertise technique. En plus, elle ne supporte pas des graphes d'exécution complexes car elle se base sur une architecture de pipeline pour l'exécution des traitements. En effet celle-ci limite le périmètre d'utilisation de GATE dans le cas où nous avons besoin d'adopter d'autres approches, telle que celles des multi-agents.

UIMA (Unstructured Information Management Architecture) [27] est une architecture logicielle pour le développement et le déploiement d'outils d'analyse de l'information non structurée. Son objectif est de décrire les étapes de traitement d'un document de type texte, image ou vidéo afin d'en extraire de façon automatique des informations structurées. En revanche, UIMA ne décrit ni comment ces informations doivent être extraites, ni la façon de s'en servir. La visée très générale de cet environnement en fait une architecture abstraite relativement de bas niveau ne proposant, en tant que telle, aucun module d'analyse de traitement automatique de la langue immédiatement utilisable. La mise en œuvre des traitements en vue d'une tâche donnée reste ainsi à la charge du concepteur, qui doit se munir de composants d'analyse développés par lui-même ou par des tiers, ces derniers restant à l'heure actuelle relativement peu nombreux et très spécifiques.

NOOJ[26] est un environnement de développement linguistique permettant de construire, de tester et de maintenir des descriptions formalisées à large couverture des langues naturelles (sous forme de dictionnaires et de grammaires électroniques), et de développer des applications du traitement de la langue. Cependant il adopte un formalisme unique (un modèle d'analyse), à base d'automates, et se base sur une architecture pipeline pour constituer des traitements complexes. De plus, il est basé sur la technologie .NET ce qui limite son utilisation en dehors d'un environnement Windows.

Au vu des caractéristiques de ces plateformes, nous dégageons les limitations communes suivantes:

- Absence d'un modèle en couches avec des servies clairement définis
- Absence des API des services à vocation métier
- Couplage fort entre les ressources et les traitements
- Une faible modularité des ressources et des services qui complexifie leurs réutilisations en dehors de la plateforme
- Limitation pour le traitement de la langue Arabe

SAFAR vise à palier à ces différentes limitations et se présente comme une plateforme ouverte et flexible pour les divers besoins de la communauté TALA

III. LA PLATEFORME SAFAR

SAFAR (**S**oftware **A**rchitecture **F**or **A**rabic language p**R**ocessing) est une plateforme dédiée au TALA. Elle est open source, multi plateforme, modulaire, et propose un environnement de développement intégré (IDE). Elle intègre les composantes nécessaires suivantes :

- Modules des niveaux de base d'une langue, plus particulièrement ceux de la langue Arabe, à savoir la morphologie, la syntaxe et la sémantique

- Ressources TALA nécessaires à différents traitements
- Les applications concernant le TALA

Notre approche pour le développement de SAFAR est centrée sur l'intégration des modules et des ressources existants plutôt que sur un développement à zéro. Lorsque la ressource ou le module n'est pas disponible ou n'est pas satisfaisant, nous nous chargeons d'en effectuer le développement.

Notre approche est également de pouvoir fournir à travers la plateforme différents services pour toutes les composantes (niveaux de base, ressources, applications) de diverses manières, et cela selon le besoin de l'utilisateur. Ainsi, un service peut être accessible soit directement par une interface graphique dédiée, soit à travers une API, soit finalement à travers un appel à un service web. Un tel travail impose, entre autres, les défis suivants :

- Bien assurer l'intégration des modules et ressources existantes car ces derniers sont pour la majorité hétérogènes
- La réutilisation de modules et de ressources existants impose des contraintes spécifiques relatives à l'ingénierie logicielle et au traitement linguistique
- Proposer une API standardisée tout en respectant les règles linguistiques qui s'appliquent à la langue Arabe.

SAFAR est organisé en couches: Chaque couche contient un ensemble de services qui partagent les mêmes préoccupations. La figure 1 donne un aperçu de l'architecture.

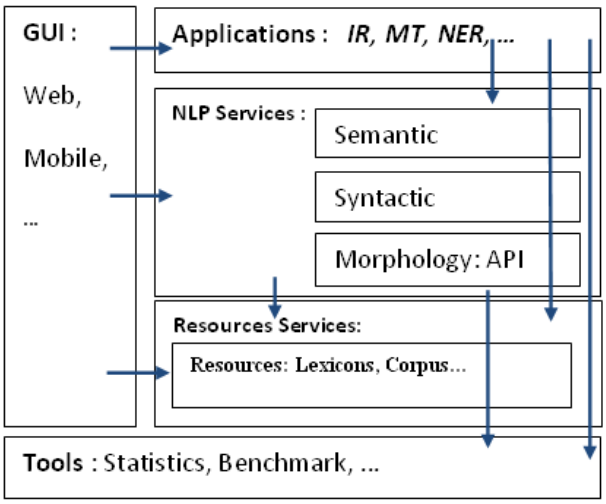

Figure 1: Architecture de la plateforme SAFAR

Chaque couche est développée comme un ensemble d'API Java réutilisables:

- *Tools (Outils):* La boîte à outils de notre plateforme, elle regroupe une série des services techniques (fonctions statistiques, outils de test, etc.)
- *Ressources Services:* Fournit des services de consultation des ressources linguistiques telles que les lexiques et les corpus. Les servies de cette couche utilisent les services de la couche Tools
- *Services NLP:* Contient les trois couches de traitement de langue régulières (morphologie, syntaxe et sémantique). Cette couche est organisée en sous couches, où chaque couche supérieure peut utiliser les services de la couche inférieure. Par exemple, la sous couche sémantique peut utiliser les services de la sous couche de morphologie
- *Applications:* Contient des applications de haut niveau qui utilisent les couches énumérées ci-dessous. Par exemple, pour mettre en œuvre un système de questions/réponses [30], un « light stemmer » (segmenteur léger) est suffisant. Ainsi, en utilisant notre plateforme, le développeur de cette application peut appeler un sous-ensemble des fonctions de la couche de la morphologie qui remplit le rôle d'un « light stemmer ».
- *GUI:* Dans le cas où l'utilisateur a besoin d'utiliser directement les services d'une couche, il n'est pas nécessaire d'appeler les API correspondants: des interfaces graphiques (GUI) pour chaque couche sont intégrées dans la plateforme.

Chaque couche utilise les services des couches inférieures. Cependant, une couche inférieure peut être utilisée seule, sans les couches supérieures: la couche de morphologie (à savoir les API associées) peuvent être utilisées directement dans n'importe quelle application sans les autres couches, la couche syntaxique (i.e. les API associées) peut être utilisée directement aussi. Parmi les objectifs qui ont influencé la conception de la plateforme était le but à atteindre un niveau de modularité et d'indépendance entre ses composantes. En outre, les flux échangés entre les couches sont au format normalisé XML.

La plateforme présente les caractéristiques suivantes:

- Elle garantit la portabilité car elle est développée en Java
- Elle pourrait être utilisée et évaluée par la communauté car elle est open source
- Elle est souple, car l'utilisateur peut utiliser l'une des interfaces graphiques ou effectuer des appels d'un ou de plusieurs des bibliothèques existantes
- Les ressources sont standardisées et présentées sous format XML, ce qui facilite l'importation des services dans un autre système
- La réutilisation des composants, à savoir que certaines fonctions API peuvent être réutilisées dans de nombreux contextes. Par exemple, la fonction qui extrait la racine d'un mot est utilisée dans la l'interface graphique dédiée à la morphologie mais pourrait aussi être utilisée dans un système de questions/réponses
- Des services web pour exposer les services de la plateforme à la communauté à travers l'Internet
- Capacité à intégrer d'autres composants

- Environnement de développement intégré pour la compilation, le développement et le déploiement de services.

Etant donné sa dimension élargie, nous nous sommes fixés comme objectif, de développer la plateforme en plusieurs étapes. Jusqu'à maintenant, nous avons réalisé:

- La structure logicielle de l'architecture, ce qui facilite l'organisation de développement sous la plateforme : chaque sous projet correspond a une couche, et pour chaque service de la couche, nous définissons une API avec éventuellement plusieurs implémentations
- Le développement d'une API pour la consultation de l'Alphabet Arabe avec un GUI web pour interroger le contenu de la ressource.
- Le développement d'une API pour la consultation des mots vides (stopwords) de la langue Arabe[39].
- Le développement de la couche d'analyse morphologique à base d'une API standardisée

La structure logicielle est organisée selon l'architecture décrite dans la figure 2:

- Chaque couche est représentée par un projet. Par exemple le projet « *NLP_Services* » représente la couche « *NLP Services* »
- La relation entre les projets respecte les relations d'utilisation entre les couches
- Un projet contient un ensemble de services, chacun définit une API avec plusieurs implémentations. Par exemple le service *Morphology* propose une API « *IMorphologyAnalyzer* » avec deux implémentations

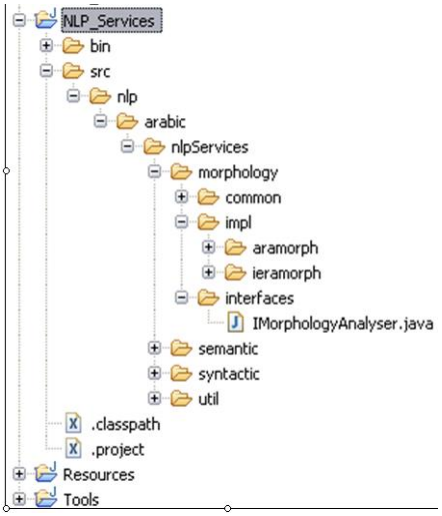

Figure 2: Structure et API de la plateforme SAFAR

IV. RESOURCES LINGUISTIQUES

Les traitements automatiques des langues naturelles sont basés principalement sur les ressources linguistiques. Ces derniers sont réparties en ressources orales et ressources écrites. Nous nous intéressons dans nos travaux à cette deuxième famille qui se compose principalement de :

- Caractère : symbole graphique utilisé comme unité dans l'écriture. Cette ressource peut être exploitée pour connaître la langue à traiter
- Lexique : ensemble des mots d'une langue. Ce type est utilisé dans l'analyseur syntaxique
- Dictionnaire : ensemble des termes généralement présentés par ordre alphabétique et fournissant pour chacun une définition, une explication ou une correspondance. Cette ressource peut trouver sa place par exemple dans les applications de l'apprentissage de la langue
- Glossaire : liste des termes relatifs à un domaine particulier (e.g. glossaire d'informatique, de médecine, etc.)
- Corpus : ensemble de documents regroupés pour un objectif précis. Un corpus peut être utilisé pour évaluer des applications TALA. Il peut également être exploité lors du développement d'une application de la traduction automatique
- Ontologie : ensemble structuré des termes et concepts représentant le sens d'un champ d'informations (e.g. Arabic Wordnet-AWN). Une ontologie peut servir à étendre les mots d'une requête d'un moteur de recherche ou d'un système de questions/réponses

Par ailleurs, la diversité structurale des ressources disponibles a rendu l'échange et la fusion de leurs données incontestablement difficiles et complexes. De toute évidence, une représentation unifiée et standardisée est nécessaire et préalable à toute exploitation de ces ressources hors de leur propre contexte de conception.

Afin de respecter l'un de nos objectifs de départ, à savoir la possibilité d'offrir à la communauté TALA, des ressources libres et unifiés, nous adoptons une approche de conception des ressources structurées et standardisées. Nous avons commencé par l'alphabet Arabe comme étant l'unité de base de la langue. Evidement, l'un de nos objectifs ultérieurs est de pouvoir travailler sur les autres types de ressources à savoir : les lexiques, les dictionnaires, etc.

Nous avons choisi le format XML pour la ressource de l'alphabet Arabe, pour rester complètement indépendants des plates-formes, des logiciels, des systèmes d'exploitation, etc.

Un caractère Arabe peut être dans une des trois catégories suivantes (les figures 3, 4 et 5 illustrent un extrait de ces trois catégories) :

- Lettre : représente les lettres de base
- Voyelle : représente les symboles de diacritization
- Ponctuation : indique les signes de ponctuation qui sont propre à l'Arabe comme le point d'interrogation (؟)

Chaque caractère de l'alphabet est caractérisé par :

- Description : pour la prononciation du caractère en Arabe et sa transcription en anglais

- Forme contextuelle (display) pour présenter les variantes contextuelles et qui prend les valeurs suivantes : isolée, finale, médiane et initiale
- Encodage : celui d'Unicode étant donné sa large adoption
- Translitération : Il existe plusieurs systèmes de translittération de l'Arabe mais nous avons choisi de mettre celles proposées par Buckwalter[36] et Wikipedia[37] étant donné leur large adoption et utilisation par la communauté TALA.

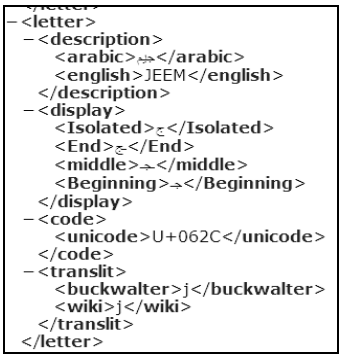

Figure 3 : présentation XML de la lettre Arabe

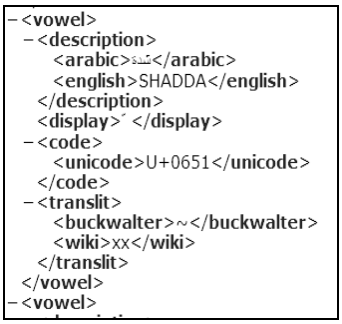

Figure 4 : présentation XML des voyelles Arabes

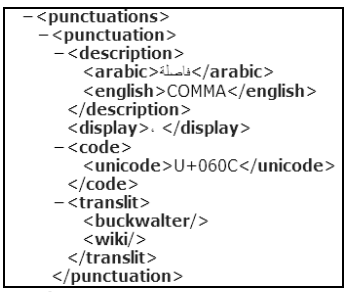

Figure 5 : présentation XML des ponctuations Arabes

La ressource de l'alphabet Arabe peut être récupérée et utilisée d'une manière indépendante[1]. Aux côtés de cette ressource basique des caractères Arabes, et afin d'illustrer

- D'une part son utilité
- D'autre part la flexibilité, la modularité et l'utilité de la plateforme SAFAR

nous avons développé trois autres composants à des endroits diverses de SAFAR. Ainsi à côté de la ressource sur les caractères Arabes, les autres composants sont (figure 6) :

- L'API pour exploiter la ressource (Alpharabic_API)
- Une application sur l'apprentissage de la langue Arabe Abou Alhorouf (lettres et voyelles seulement) en faisant appel à Alpharabic_API
- Une sortie sous format web de cette application d'apprentissage

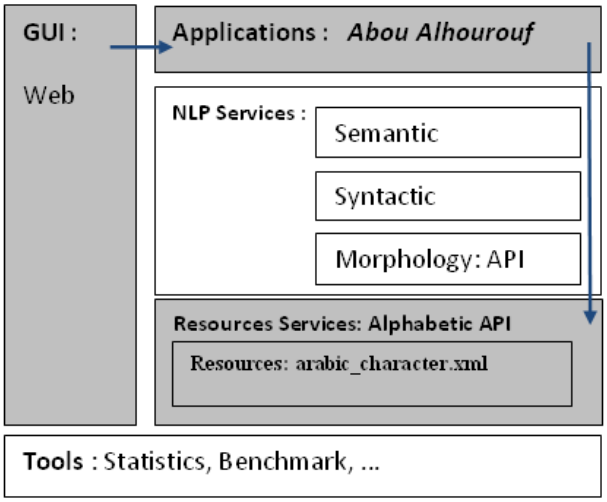

Figure 6 : Arabique alphabet dans SAFAR

L'API Alpharabic expose différents services pour pouvoir exploiter la ressource sur les caractères Arabes. Les services proposés sont sous forme de getters, l'appel d'un service retourne un objet (`Letter`, `Vowel`, `Punctuation`) qui encapsule toutes ses propriétés décrites ci-dessus. La figure 7 présente un extrait des services de l'API. Par exemple le service `getLetterByMiddleDisplay(String trans)` reçoit en entrée la forme contextuelle médiane du caractère et fournit en sortie l'objet `Letter` recherché.

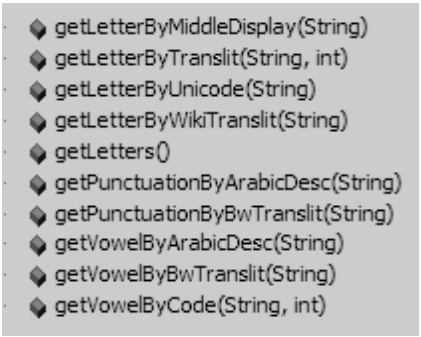

Figure 7: Extrait des services de l'API de l'alphabet Arabe

L'application Abou Alhorouf et sa sortie web sont basées principalement sur notre API de l'alphabet Arabe. Elles permettent d'aider les personnes désireuses d'apprendre la langue Arabe à avoir accès à un site web afin de se familiariser avec les caractères de la langue Arabe (avec et sans diacritization)[2]. Abou Alhorouf (figure 7) se décline en quatre niveaux. Dans chaque niveau, l'utilisateur est invité à reconnaître ce qui lui est affiché à l'écran en saisissant au clavier le même contenu :

[1] Téléchargeable à partir du site www.emi.ac.ma/bouzoubaa/download/

[2] http://sibawayh.emi.ac.ma:8081/alpharabic

- **Niveau 1** : l'utilisateur est devant une interface qui lui présente le caractère en image accompagnée de sa prononciation
- **Niveau 2** : Pour ce deuxième niveau, l'image du caractère disparaît et l'utilisateur est amené à le reconnaître à l'aide de sa prononciation uniquement
- **Niveau 3** : Ce niveau est le même que le premier, sauf que le caractère est voyellé
- **Niveau 4** : Ce niveau est le même que le deuxième, sauf que le caractère est voyellé

En passant d'un niveau à un autre plus supérieur, la complexité s'incrémente et permettra à un utilisateur apprenant de la langue Arabe de se familiariser avec les caractères et voyelles de la langue de manière graduelle.

Figure 7 : Application Abou Alhorouf

V. ANALYSE MORPHOLOGIQUE

Comme annoncé dans la section 3, une des couches que nous avons implémentée dans la plateforme SAFAR est la couche morphologique (voir figure ci-dessous).

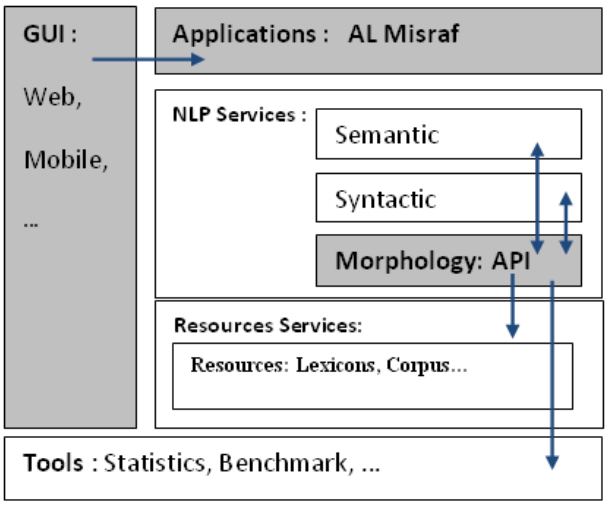

Figure 8 : la couche morphologique de SAFAR

La couche morphologique respecte les principes d'architecture de SAFAR :

- Elle offre les modules de base pour l'analyse morphologique de la langue Arabe
- Elle offre une sortie standard (output) déclinable en une interface API et simplifiant les tâches d'intégration, d'évaluation et de benchmarking
- Elle permet une intégration fluide et standardisée des analyseurs morphologiques existants

Dans la suite de cette section, nous allons commencer par rappeler le rôle d'un analyseur morphologique, puis nous allons définir un standard de la structure de la sortie (output). A partir de cette sortie standard, nous allons découler l'interface API à fournir par la couche morphologique. Nous allons aussi montrer comment l'analyseur de Buckwalter[2] a été intégré en tant qu'implémentation possible de cette interface. Finalement nous allons présenter le niveau applicatif (interface utilisateur) permettant d'utiliser l'API morphologique.

A. L'analyse morphologique

Un analyseur morphologique est un composant essentiel des systèmes de traitement automatiques de la langue. D'une part, il permet de supporter diverses applications destinées à l'utilisateur final (correcteur d'orthographe, dictionnaires en ligne, etc.), et d'autre part il constitue une base pour les niveaux supérieurs d'analyse (syntaxique et sémantique).

La fonction principale de l'analyseur morphologique est de fournir la structure des mots. Considérons par exemple la phrase suivante :

(1) : وأشارت واشنطن تايمز إلى أن اتفاقية 2005 أنهت سنوات من الحرب بين الشمال والجنوب السوداني، وأن الجنوبيين سيدلون بأصواتهم في استفتاء بشأن انفصالهم من عدمه

Et a indiqué le Washington Times que l'accord de 2005 a mis fin aux années de guerre entre le nord et le sud du Soudan, et que les sudistes voteront lors d'un référendum sur l'opportunité de la séparation.

Le tableau suivant présente une analyse morphologique du mot «بأصواتهم» de la phrase (1).

Type	Segment	Propriétés
Prefixes	ب	POS : Particle > Preposition GLOSS : By/With
Stem	أصوات	POS : Noun > Common, ROOT : ص و ت NUMBER : Plural, GENDER : Masculine FORM : أفعال, GLOSS : Voices
Suffixes	هم	POS : Noun > Pronoun NUMBER : Plural, GENDER : Masculine

Figure 9: exemple d'analyse morphologique.

La première colonne présente les types des segments. Un segment (ie : unité morphologique minimale) est :

- Soit un préfixe (enclitique) comme بِ, فَ, سَ, كَ
- Soit le stem du mot (ie : le corps ou la partie principale du mot)

- Soit un suffixe (proclitique) généralement un pronom personnel comme هُمْ, هَا, نَا …

La deuxième colonne présente les différents segments de la solution relatif au mot «بأصواتهم», soit : un préfixe بِ un stem أصوات et un suffixe هم.

La troisième colonne contient des informations détaillées sur chaque segment. Il y a deux types d'informations associées à chaque segment:

- **POS** (part of speech), partie de discours ou catégorie grammaticale. La langue Arabe supporte trois catégories principales : les verbes (الأفعال), les noms (الأسماء) et les particules (الحروف)
- **Attributs** : Chaque segment, possède un ou plusieurs attributs qui spécifient un état ou une fonction. Par exemple l'attribut GENDER spécifie le genre (féminin, masculin ou neutre). L'attribut NUMBER désigne le nombre (singulier, dual, pluriel). La liste des attributs possibles pour un segment est déterminée par sa catégorie grammaticale (POS). L'attribut TENSE (temps) par exemple s'applique aux verbes mais pas aux noms. ROOT est un attribut commun aux verbes et noms et désigne la racine (الجذر) du nom ou du verbe, c.à.d. les trois (éventuellement quatre ou cinq) lettres radicales du mot, abstraction faite de la transformation causée par l'inflexion ou la dérivation. Par exemple, la racine ص و ت est commune aux mots suivants : أصوات (des voix), صَوتان (deux vois), صَوَّتَ (a voté), صيت (renom).

Le mot «بأصواتهم» supporte une seule solution (segmentation). Par contre le mot «عدمه» du texte (1) supporte (hors contexte) plusieurs solutions dont :

- عَدَمِ+ ِه : Stem(عدم) = {POS: Nom ; GLOSS : null/negation} + Prefix(ه) = {POS : Noun > Pronoun; GLOSS : him}
- عَدِمَ+ ُه Stem(عدم) = {POS : Verb ; GLOSS : miss} + Prefix(ه) = {POS : Noun > Pronoun ; GLOSS: him}

B. Analyseurs existants :

Plusieurs recherches ont été conduites dans ce domaine, allant du simple « Stemmer » (ie : un extracteur du Stem ou de la racine Root)[31] à des analyseurs plus avancés [1][5][7]. Nous pouvons résumer nos remarques sur les analyseurs les plus connus par les points suivants :

- Leur non disponibilité : Bien que l'analyseur BAMA [1][2] soit le plus répandu et utilisé, sa seconde version [3] n'est plus libre. Les analyseurs des compagnies Sakhr [29] et RDI [30] sont à finalité commerciale
- Leur non accessibilité sur Internet
- Leur non portabilité : certains sont développés en utilisant des langages de programmation non portables tels que le cas de l'analyseur Elixir [7] développé en Haskell
- Difficulté de les faire fonctionner avec d'autres modules, car ils sont développés de manière indépendante
- Il n'a y a pas de consensus sur les sorties et les fonctionnalités que doit fournir un analyseur morphologique, ni sur les corpus de test et d'apprentissage, ni finalement sur les méthodes d'évaluation et de comparaison[1].

La plateforme SAFAR est une réponse à ces limites, elle adopte une démarche de standardisation, de modularité, d'intégrabilité, de flexibilité et de compatibilité avec la langue Arabe. La partie suivante, présente une proposition de standardisation de la sortie de l'analyse morphologique.

C. Les parties de discours et sorties standards

Les analyseurs existants proposent des ensembles de propriétés (POS + attributs) différents. La majorité de ces propositions s'inspirent des ensembles conçus pour les langues européennes avec certaines adaptations.

Dans notre cas, nous nous sommes basés sur la proposition de Khoja [31] pour les raisons suivantes :

- La langue Arabe ne suit pas les recommandations EAGLES[3] réservées aux langues européennes
- Les parties de discours de la langue Arabe présentent un aspect important d'héritage, ou chaque sous classe de mots, hérite les propriétés de la classe parente
- La langue Arabe est riche en dérivation et sous catégorisation. Ces caractéristiques constituent le cœur de la grammaire Arabe tel que enseignée depuis plusieurs siècles

Nous avons cependant apporté quelques modifications à l'arbre de Khoja (voir figure 10). Ces modifications sont de deux natures :

- Extension de certaines catégories (sous catégorisation) pour répondre à des besoins syntaxiques ou sémantiques
- Repositionnement, remplacement ou modification de certaines catégories de manière à sauvegarder la conformité avec la langue Arabe

Un des avantages présentés par cette modélisation est la structure hiérarchique des catégories. Cette hiérarchie permet :

- D'étendre l'arbre selon les besoins. Nous pouvons donc sous catégoriser une catégorie pour des besoins d'analyse sémantique par exemple
- D'appliquer un système d'héritage d'attributs. Ainsi si la catégorie des noms « Noun » dispose de l'attribut genre « Gender », la catégorie des adjectifs « Adjective » hérite cette propriété
- D'appliquer des règles générales. Par exemple la règle syntaxique <Inna><Noun> implique que la particule « Inna » (إنّ) doit être suivi par un nom.

[3] Expert Advisory Group on Language Engineering Standard. Voir [33].

Cette règle est donc applicable pour toutes les sous catégories de la catégorie «Noun » (ie : adjectifs, noms propres, pronoms,…)

- De restreindre, ou adapter la catégorisation aux capacités de l'analyseur morphologique. Par exemple, un analyseur qui ne gère pas les sous catégories des nombres « Numeral », peut se limiter à annoter le mot analysé en tant que nombre
- La même remarque s'applique sur la gestion des corpus annotés avec plus au moins de détails. Grâce à cette structure, il est possible de ramener plusieurs types de sorties, à une sortie commune pour des besoins de tests, de comparaisons ou de traitement de corpus.

De plus, les sorties utilisées par d'autres analyseurs morphologiques peuvent être facilement transposées vers la sortie que nous proposons.

La sortie « standard » est donc déterminée par les points avancés ci-dessus, à savoir :

- Déterminer la segmentation du mot : liste des solutions: préfixes+stem+suffixes.
- Déterminer pour chaque segment de la solution, la catégorie grammaticale et ses attributs.

Il en découle, les fonctions de l'interface API que nous décrivons dans la partie suivante.

D. L'interface morphologique : API

L'API morphologique est directement découlée de la sortie standard décrite dans la partie précédente. Pour la segmentation des mots nous avons des fonctions qui permettent de récupérer les préfixes : getPrefixes`()`, le stem : `getStem()` et les suffixes : `getSuffixes()`.

Pour récupérer toutes les solutions possibles avec une analyse complète des segments, nous avons la fonction `getMorphoAnalysis()`.

Pour les parties de discours, nous avons des fonctions qui permettent de vérifier la catégorie du segment: `isVerb(),isNoun(),...` ou de récupérer cette catégorie `getPOS()`.

Pour les attributs associés au segment, nous avons des fonctions qui permettent de récupérer leur valeur : `getGender(),getNumber(),...`

La combinaison de ces fonctions, permettent de développer d'une manière simple et triviale des applications plus évoluées.

Ces fonctions s'appliquent à un texte, à un mot ou à un objet structuré (ie : un objet retourné par une autre fonction). La figure 11 présente une partie de l'interface JAVA représentant cette API.

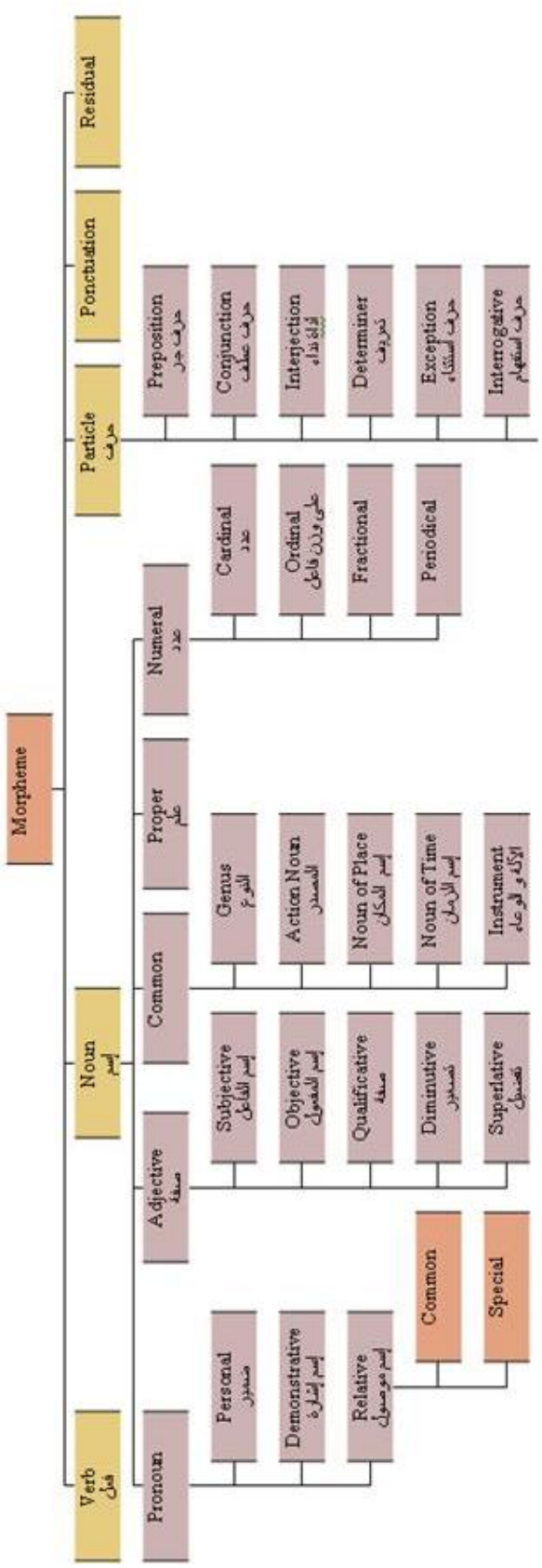

Figure 10: Arbre des parties de discours

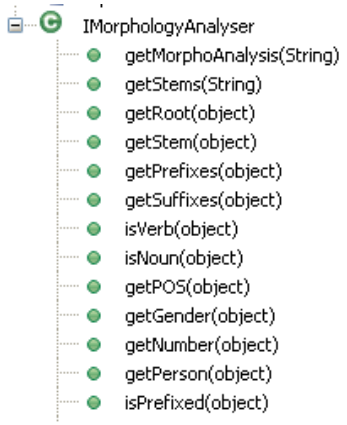

Figure 11: L'interface Java de la couche morphologique.

E. Implémentation de l'analyseur de Buckwalter

La première version de Buckwalter[2] est programmée en langage Perl, un portage vers Java a été réalisé en 2003 : AraMorph[35].

Nous avons implémenté toutes les méthodes de l'interface IMorphologyAnalyser en utilisant le codage réalisé dans [35]. Par exemple, la figure suivante montre une implémentation de la méthode `getMorphoAnalysis(String)` en utilisant la classe AraMorph définie dans [35]. La méthode `wb2std_solutions()` de la classe outil std_morph, transforme la sortie de Buckwalter en sortie standard.

```
public class iaramorph  implements IMorphologyAnalyser{
    public List<Object> getMorphoAnalysis(String word) {
        // Apply Buckwalter analysis of the word
        AraMorph AM =new AraMorph();
        AM.analyzeToken(word);
        HashSet bw_solutions = AM.getWordSolutions(word);
        // Transform Buckwalter output to standard output
        List solutions std_morph.wb2std_solutions(bw_solutions);
        return solutions;
    }
```

Figure 14: Implémentation de l'interface standard.

La figure 16 affiche les résultats au format texte de l'analyse du mot 'بأصواتهم ' par utilisation de la méthode implémentée selon le codage de Buckwalter tout en respectant la sortie standard :

```
Word
    String:بـأصواتهم
    Vocalisation:بِـأَصْوَاتِهِم
    Prefixe :
            String:ب
            POS:Particle>Preposition
            Gloss:by/with
    Stem    :
            String: أصوات
            POS:Noun
            Gloss:voices/sounds
    Suffixe :
            String:هم
            POS:Noun>Pronoun
            Number:Plural
            Gender:Masculine
            Person:Third
            Gloss:their
```

Figure 16: Exemple de sortie standard SAFAR format texte.

VI. CONCLUSION

Cet article présente les plateformes de TALN les plus connues telles que GATE, NOOJ et UIMA et met le focus sur leurs limitations par rapport aux besoins de la communauté de traitement automatique de la langue Arabe. Nous avons décrit de manière succincte et descriptive les caractéristiques de notre plate forme SAFAR. A travers cette description, un certain nombre de caractéristiques de SAFAR ont été soulignées. Ainsi, ses dimensions d'ouverture et de standardisation font d'elle le socle de base pour développer et intégrer des solutions et des services concernant le TALA.

Jusqu'à maintenant, nous avons réalisé la structure logicielle de SAFAR, le développement de la couche d'analyse morphologique, le développement d'une API pour la consultation de l'Alphabet Arabe avec un GUI web pour interroger le contenu de la ressource et le développement d'une API pour la consultation des mots vides de l'Arabe.

Nous comptons dans nos prochains travaux compléter le développement de SAFAR (il est prévu dés que possible de la mettre disponible en open source sur sourceforge), à savoir les ressources linguistiques, les couches de base (syntaxe et sémantique), ainsi que des applications telles que les moteurs de recherche et les systèmes de question réponse.

REFERENCES

[1] Imad Al-Sughayer and Ibrahim Al-Kharashi (2004). "Arabic morphological Analysis Techniques: a comprehensive Survey". In *Journal of the American Society for information Science and Technology* Februry 1, 2004.(pp 189-212).

[2] Buckwalter (2002). "Arabic Morphology Analysis". http://www.qamus.org/morphology.htm.

[3] Buckwalter (2004). "Arabic Morphological Analyzer" Version 2. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2004L02.

[4] Kenneth Beesley (2001). "Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001". In Proceedings of the Arabic Language Processing: Status and Prospect, 39th Annual Meeting of the Association for Computational Linguistics. Toulouse, France.

[5] Mohammed Attia A.(2008). "Handling Arabic Morphological and Syntactic Ambiguity within the LFG Framework with a View to Machine Translation". University of Manchester PhD Thesis.

[6] Mohammed A. Attia (2006). "An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks". The Challenge of Arabic for NLP/MT Conference, the British Computer Society, London.

[7] Otakar Smrz (2007). "Functional Arabic Morphology Formal System and Implementation". Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague. Doctoral Thesis.

[8] Mohamed Attia Mohamed Elaraby Ahmed (2000). "A Large Scale Computational Processor Of The Arabic Morphology, and Applications". Faculty of Engineering, Cairo University. Master of Science Thesis.

[9] Lahsen Abouenour, Said El Hassani, Tawfiq Yazidy, Karim Bouzoubaa, Abdelfattah Hamdani (2008). "Building an Arabic Morphological Analyzer as part of an Open Arabic NLP Platform". LREC 2008 (Sixth International Conference on Language Resources and Evaluation). Marrakech, Morocco.

[10] Nizar Habash and Owen Rambow (2005). "Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop". In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), pages 573–580, Ann Arbor, Michigan, June. Association for Computational Linguistics.

[11] Nizar Habash (2007). "Large Scale Lexeme Based Arabic Morphological Generation". In Proceedings of NAACL HLT 2007, Companion Volume, pages 53–56.

[12] Shereen Khoja (2001). "APT: Arabic Part-of-speech Tagger". Proceedings of the Student Workshop at NAACL-2001, 2001

[13] Shereen Khoja (2001). "A tagset for the morphosyntactic tagging of Arabic". Corpus Linguistics 2001 conference, Lancaster.

[14] Rani Nelken and Stuart M. Shieber (2005). "Arabic Diacritization Using Weighted Finite-State Transducers". In ACL-05 Workshop on Computational Approaches to Semitic Languages, pages 79–86, Ann Arbor, Michigan.

[15] Nizar Habash and Owen Rambow (2007). "Arabic Diacritization through Full Morphological Tagging". In Proceedings of the 8th Meeting of the North American Chapter of the Association for Computational Linguistics/ Human Language Technologies Conference (HLTNAACL07).

[16] Mona Diab Kadri Hacioglu Daniel Jurafsky (2004). "Automatic Tagging of Arabic Text: From Raw Text to Base Phrase Chunks". In 5th Meeting of the North American Chapter of the Association for Computational Linguistics/Human Language Technologies Conference (HLT-NAACL04), Boston, MA.

[17] Abdelhadi Soudi Violetta Cavalli-Sforza Abderrahim Jamari (2001) "Computational Lexeme-Based Treatment of Arabic Morphology". In Proceedings of the Arabic Natural Language Processing Workshop, Conference of the Association for Computational Linguistics (ACL 2001), Toulouse, France.

[18] Chafik Aloulou (2003). "Analyse syntaxique de l'arabe : le système MASPAR". In Actes de RECITAL 2003, Batz-sur-Mer, France.

[19] Kareem Darwish. (2003). "Building a shallow Arabic morphological analyser in one day". In ACL02 Workshop on Computational Approaches to Semitic Languages, Philadelpia, PA. Association for Computational Linguistics..

[20] Lahsen Abouenour, Karim Bouzoubaa, Paolo Rosso (2008). "Système de Question/Réponse dans le cadre d'une plateforme intégrée : cas de l'Arabe". Rencontre Nationale en Informatique :Outils et applications. RNIOA'08 Les 05, 06 et 07 Juin (2008)

[21] Bassam Haddad (2007). "Semantic Representation of Arabic: A Logical Approach to wards Compositionality and Generalized Arabic Quantifiers". International Journal of Computer Processing of Oriental Languages, IJCPOL, Vol. 20. Number 1, 2007.

[22] Ahmed Abdelali, Jim Cowie and Hamdy S. Soliman (2004). "Arabic Information Retrieval Perspectives". Proceedings of JEP-TALN 2004 Arabic Language Processing, Fez 19-22. April 2004.

[23] Mohamed Maamouri, Ann Bies, Seth Kulick (2006). "Diacritization: A Challenge to Arabic Treebank Annotation and Parsing". In Proceedings of the Conference of the Machine Translation SIG of the British Computer Society.

[24] Isaac Silvester de Sacy (1830). *Grammaire Arabe*. *Tome 1*.

[25] Valentin Tablan, Diana Maynard, Kalina Bontcheva,Hamish Cunningham 2004. "GATE: General Architecture for Text Engineering, GATE — An Application Developer's Guide" http://gate.ac.uk University of Sheffield, UK

[26] Max Silberztein (2004). "NooJ: an oriented object approach". In INTEX pour la Linguistique et le traitement automatique des langues. Les Cahiers de la MSH Ledoux. Presses Universitaires de Franche-Comté: Besançon.

[27] Ferrucci D., Lally A., 2004 'UIMA : an Architectural Approach to Unstructured Information Processing in the Corporate Research Environment". *Natural Language Engineering*, vol. 10, p. 327-348,

[28] http://aljazeera.net vendredi 25 décembre 2009 à 10h45.

[29] Sakhr's Multi-Mode Morphological Processor. http://www.sakhr.com/Technology/Morphology/Default.aspx?sec=Technology&item=Morphology

[30] ArabMorpho, RDI's Arabic Morphological Analyzer. http://www.rdi-eg.com/rdi/technologies/arabic_nlp_pg.htm

[31] Shereen Khoja: Arabic stemmer http://zeus.cs.pacificu.edu/shereen/research.htm#stemming

[32] Monica Monachini, Nicoletta Calzolari, Istituto di Linguistica, Via della Faggiola (1996). "*Synopsis and Comparison of Morphosyntactic Phenomena Encoded in Lexicons and Corpora. A Common Proposal and Applications to European Languages*". EAGLES Document EAG-CLWG-MORPHSYN Version of May, 1996.

[33] Leech G, Wilson A (1996). "*Recommendations for the Morphosyntactic Annotation of Corpora".* EAGLES Report EAG-TCWG-MAC/R. http://www.ilc.pi.cnr.it/EAGLES96/annotate/annotate.html

[34] الشيخ مصطفى الغلايني (2008) جامع الدروس العربية

[35] Pierrick Brihaye (2003). http://www.nongnu.org/aramorph/

[36] Buckwallter translitération http://www.qamus.org/transliteration.htm

[37] Wikipedia translitération http://en.wikipedia.org/wiki/Arabic_alphabet

[38] William Black et al (2006). "Introducing the Arabic WordNet Project". In Proceedings of the Third International WordNet Conference, Fellbaum and Vossen (eds).

[39] Karim Bouzoubaa, Hicham Baidouri, Taoufik Loukili, Taoufik El Yazidi (2009). "Arabic Stop Words: Towards a Generalisation and Standardisation". In Proc of the 13th *International Business Information Management Association Conference* IBIMA 2009, Marrakech, Morocco, November, 2009.