A Knowledge-based Architecture for Free Text Processing

Jawad Berri
Department of mathematics and computer science
U. A. E University
P. O. Box 17551, Al-Ain
United Arab Emirates
j.berri@uaeu.ac.ae
Tel: (971 3) 7064414
Fax: (971 3) 7671291

Abstract

In this paper we present an architecture that allows the implementation of specific morphological analyzers dedicated to process free text. The architecture includes a key component: the GMA (Generic Morphological Analyzer) that is language independent and contains the rule models and the software components used to develop morphological analyzers. The architecture relies on a flexible linguistic method namely the *contextual exploration method* to describe the morphological rules of a language and implement them as a knowledge-based system. We focus on the Arabic language and provide the description of the implemented system. Then we show how generic the approach is so as to extend it to other languages.

Keywords: Contextual exploration method, generic morphological analyzer, exception rule, matching algorithm, free text.

1. Introduction

One of the major challenges facing research in computational linguistics is the implementation of tools that can process free text regardless of the language in which texts are written and the domain considered by the texts content. Implementing systems that can cope with these two barriers would provide solutions for many real-life applications. They would allow users to access dynamic and heterogeneous sources of information, as the Internet, in order to retrieve information of interest. Unfortunately, due to the inherent complexity of natural language, most of traditional language understanding systems available in the research community as well as in the industry limit the field of investigation to restricted text. In theory, they could be adapted to different languages and domains. However, experience has shown that they lack a level of flexibility to allow effortless adaptation. The main shortcoming of these systems is the need for language and domain dependent resources such as lexicons and conceptual structures which are very demanding in time and effort [1]. Moreover, these systems lack a very high degree of robustness and efficiency in order to be able to cope with free text.

To fulfill the increasing demand of systems capable of processing free text, researchers have turned from the theoretical view of the problem towards more practical approaches (Desclés et al., 1995, ARPA, 1996, Mollá et al., 1998, Mädche et al., 1999). These approaches seek primarily to implement systems which convey solutions to linguistic problems with less cost by avoiding a complete description of the problem. They make use of heuristics instead of large and complex combinatorial algorithms, and rely on some language regularities that are observed in a corpus.

1 Wordnet (Miller et al., 1991) that is an English lexicon, is an example of these systems for which the development required many researchers during the past 15 years.

This paper is a step toward this concern. The architecture presented allows the implementation of specific morphological analyzers dedicated to process free text. The architecture includes a key component: the GMA (Generic Morphological Analyzer) that is language independent and contains the rule models and the software components used to develop morphological Analyzers. The architecture relies on a flexible linguistic method namely the *contextual exploration method* to describe the morphological rules of a language. In order to achieve the required robustness for a system dealing with free text, the morphological analyzer does not make use of a full-form dictionary as in the case of classical systems. Indeed, the coverage of such tool is usually restricted and cannot cope with free text from dynamic information sources such as the Internet.

In this paper we focus on the Arabic language and provide the description of the rules that have been implemented, then we show how generic the approach is so as to extend it to other languages

The remaining sections of this paper are organized as follows: Section two gives a description of the contextual exploration method. Section three presents the description of the architecture. Section four describes in details the implementation of the system. In section five we discuss the advantages and limitations of the architecture as well as its generic features. Finally, section six concludes this paper with future directions to extend this work.

2. The Contextual Exploration Method

In order to implement the morphological analyzer, we used the *contextual exploration method* (Desclés, 1990, Berri et al., 1992, Desclés et al., 1995). It is a decision-based method that is able to resolve with less processing cost some classical computational linguistics problems. It has been implemented successfully to resolve classical linguistic problems such as tense and aspect (Berri et al., 1992), automatic abstracting (Berri et al., 1995) and knowledge acquisition and modeling (Jouis, 1995). The principle of this method is to scan a given linguistic marker and its context (surrounding tokens in a given text) looking for linguistic clues that guide the system to make the suitable decision. This method simulates the behavior of a reader who aims at taking a decision regarding a given linguistic problem. In our case, the method scans an input token and tries to find the required affixes in order to isolate the root-form and to associate the corresponding morpho-syntactic information. In the case of irregular words, the method looks-up a knowledge base of word-lists to process and associate the right root-form to the input token. Contextual exploration method aims to build a knowledge-based system.

3. System Description

The morphological analyzer is a key component in a framework dedicated to process unrestricted text from the Internet. The objective within this framework is to process a text in order to facilitate its usage by a wide range of further applications (Berri, 2000). Thus, the text input passes through a sequence of modules: It is first pre-processed where all useless parts are filtered out of the document, the *html* tags then are identified and coded. The tokenizer identifies all tokens and sentences. Then a module builds an object-oriented representation of the text that highlights all the basic relationships between the different constituents of the document, namely the token, the sentence, the paragraph and the section.

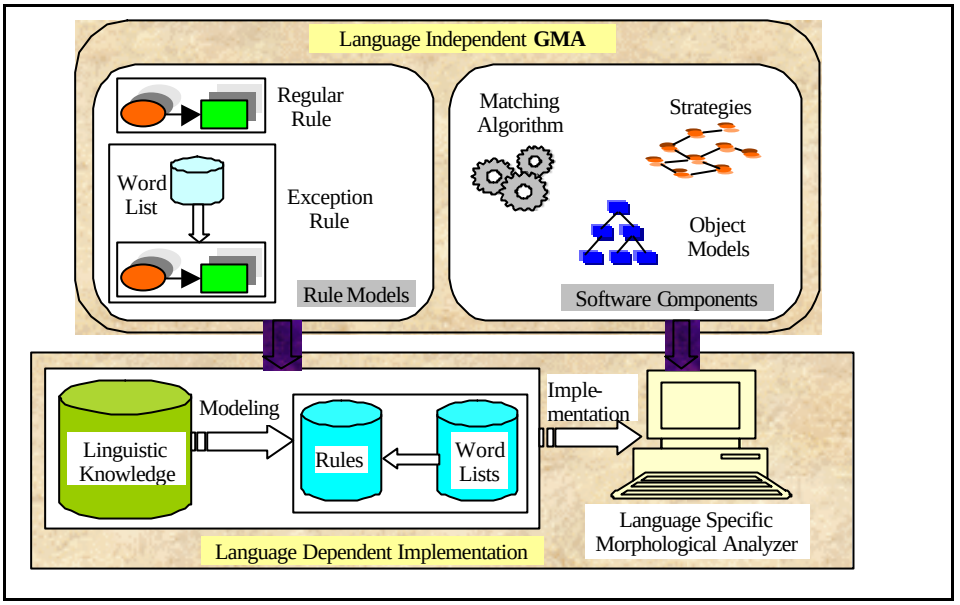

Figure 1. GMA, Generic Morphological Analyzer

Finally, the morphological analyzer finds all word root forms and associates the morpho-syntactic information to the tokens. Within the text representation, a token is an object that includes the following fields: (note that the sentence, the paragraph and the section are also object classes defined within the text representation)

- *Name*, contains the name of the token. Every token is assigned a name that allows the system to identify it.
- *Value*, is the word-form as it appears in the text before any processing.
- *Root*, stores the root of the word that is computed during morphological analysis.
- *Category*: the part of speech to which the token belongs to. It can hold two possible values: verb or noun
- *Tense*: the tense associated to the token in case of a verb
- *Number*: the cardinality of the token consisting of singular, dual or plural
- *Gender*: the gender associated to the token consisting of either masculine or feminine
- *Person*: valid only for verbs, it represents either the first, second or third person.
- *Rule applied*, stores the identifier of the rule that has been fired to analyze the token.
- *Sentence*, a reference to the object "Sentence" containing the token in the text. This is a relationship that holds between the token and its corresponding sentence.
- *Order*, stores the rank of the token in sequence from the beginning of the text. It is used to compare the sequential order of tokens in the text.
- *Positions*, correspond to the offset positions of the token in the text. It is used to highlight a relevant token when displaying the results to the user.
- *Format*, is the associated format (boldface, italics, …) applied to a token in the text.

The architecture that implements specific morphological analyzers is presented in figure 1. It includes the GMA consisting in the rule models and the software components, and the linguistic knowledge modeling process aiming at modeling the

rules and implementing them. The architecture components are presented in the following sections.

3.1. Linguistic Analysis

Arabic is known to be a highly inflected language, its famous pattern model using the CV (Consonant, Vowel) analysis has widely been used to build computational morphological models (Kiraz, 1998). During the last decade, an increasing interest has been noticed to implement Arabic morphological analyzers (El-Sadany & Hashish, 1989, Alshalabi & Evens, 1998, Beesley 1998, Saliba & Al Dannan, 1989, Smets, 1998). The linguistic analysis of Arabic is based on unvoweled texts since most Arabic texts available on Internet are written in modern standard Arabic that usually doesn't use diacritical marks.

The main idea underlying the linguistic analysis is the distinction between regular and irregular morphological forms of verbs and nouns in a language. Language regular forms are constituted by concatenation of affixes and root-form, they follow the following pattern *prefix+root+suffix*. Their analysis is straightforward: first, identify the affixes in order to isolate the root-form, then associate the morpho-syntactic information to the analyzed token. Language irregular forms express the language exceptions that need more analysis. Indeed, the root-form cannot be obtained by simple isolation of affixes, the identification of the token itself as an irregular form is necessary, then the root-form is processed accordingly. Thus, the constitution of word lists that include these irregular word forms is indispensable in order to: i) identify the input token as an irregular form, ii) process the root-form and iii) associate to the input token the morpho-syntactic information.

3.2. Linguistic Knowledge Modeling

By linguistic knowledge we mean knowledge related to the language morphology as it is provided by linguists and available into grammar books (Arrajihi, 1973, Qabawah, 1994). Modeling the linguistic knowledge is required before the implementation, it consists in representing all the morphological rules by using the two rule models: regular and exception rules [2]. In general this operation is straightforward for regular rules, whereas more consideration is required during the representation of exception rules. In fact, an exception rule handles a set of words sharing common morphological properties. These words must be identified and put into an exception list.

3.2.1. Regular rules

A regular rule is represented as an object, instance of the class *regular-rule*. It models a spelling rule for adding affixes. The structure of *regular-rule* consists of nine fields:

- *Name*: identifies uniquely a rule
- *Class*: is the class of the object
- *Prefix*: a sequence of characters at the beginning of a token
- *Suffix*: a sequence of characters at the end of a token
- *Category, Tense, Number, Gender and Peron* are the same fields as the class token (sec. 3).

For instance, consider the Arabic word (in the active mode: 'they write') that is composed of the three following morphemes: the root of the verb that is (/ktb/, notion of writing), the prefix that denotes both the present tense and the third person, and the suffix that denotes the masculine and the plural.

The rule that analyses this word is represented in (fig. 2 (a)). In (fig. 2 (b)) the token is shown before matching, and in (fig. 2 (c)) the token attributes are updated.

[2] A formal rule specification is given in (Berri et al. 2001).

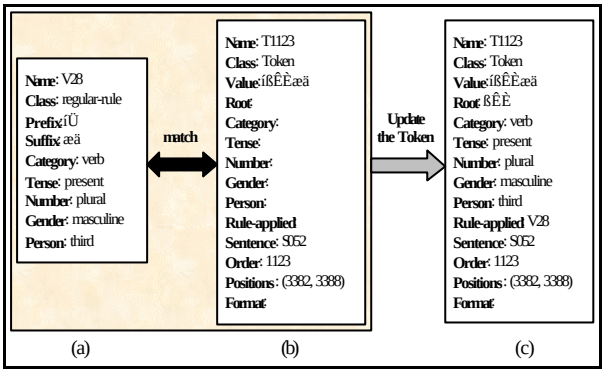

Figure 2. Matching regular rules

3.2.2. Exception Rules

An exception rule is represented as an object, instance of the class *exception-rule*. The structures of *exception-rule* and *regular-rule* are similar, the difference lies in the conditions to fire a rule. Whereas the presence of affixes in a token is enough to fire a regular rule, an exception rule needs more conditions. The adherence of the token to a list of exceptions is generally necessary to fire such a rule, moreover, the length of the token and the presence of consonants are also examined.

For instance, the *ideal* verb in Arabic belongs to the irregular class of verbs (called the infirmum class) since its first letter is an infirmum letter (either /wa/ or /ya/). The imperative form of the ideal verbs starting with the letter consists in is removing the first letter (that is) and adding the suffix (as in the regular verbs). For instance, the imperative forms of the verb (to describe), are the following:

Person	Verb-form
singular masculine	
dual masculine	
plural masculine	
singular feminine	
dual feminine	
plural feminine	

When one of the above forms is an input token to the system, the regular rules will be applied to isolate the affixes from the root. For instance, if the token is input to the system, a regular rule segments it into the suffix and the root-form . Since the first letter has been removed in the imperative, the root-form obtained is not correct. In order to get the correct root-form of this verb (that is) we need the following exception rule that handles the root obtained:

RV108
IF **Length**(**Root**) = 2
AND X = **Concatenate**(0, , **Root**) **BelongTo** LV8
THEN **Infinitive** is X

LV8 is the exception list containing the following verbs { (to describe), (to held), (to donate), (to fell down), (to put), (to jump), (to inherit), (to trod on), … }

The infirmum ideal verb is a ternary verb that is constituted by three letters. Thus, the first condition of this rule checks if the length of the root equals to two, that is the length of the root in the imperative after processed by the regular rules. The second condition checks whether the concatenation of the root and the letter belongs to the list LV8 that is the exception list containing all the infirmum ideal verbs starting with the letter . If the two above conditions are true, the rule computes the root form of the word by adding the letter to the root as it results from the first analysis by regular rules.

3.3. Generic Morphological Analyzer

GMA (fig. 1) includes the language independent components that can be reused in different implementations of morphological analyzers. GMA consists in the rule models and the software components. The rule models are used to map all the morphological rules related to a given language, they are designed as object classes (as shown in 3.2.1). The software components comprise the object models, the algorithm, and the strategies. These three components are essential in any language specific implementation of a

morphological analyzer, they are described below.

Object models. An input text to the morphological analyzer is a representation that is an instance of a text model. The text model represents the physical structure of any text, it embeds a set of object classes that embody all the text units: the token, the sentence, the sections and all the relationships taking place among them. The other object models in GMA are the object classes that implement the morphological rules: regular and exception rules.

Matching Algorithm. The goal of token-to-rule matching algorithm is to return the regular rule that extracts the root of a given token, and consequently, associates the morpho-syntactic information to the token (fig. 2). The rule is identified if it matches a given pair of suffix and prefix. Thus, first the affixes are extracted from the token, then the matching operation is performed, and finally the token is updated.

Strategies. The strategy we are using for Arabic defines in which order the rules are tested. The first group that is examined is the exception rules for nouns, then comes the regular rules for verbs and nouns, finally, the exception rules for verbs are considered. This strategy seems to be the optimal with regard to the small number of word-forms that must be encoded in the exception lists. An Arabic verb has fourteen form in the present and the past tenses and six forms in the imperative. Indeed, by considering the exception rules for verbs as the last group of examined rules, we avoid to encode all the possible forms, the verb is first considered by the regular rules that remove its affixes, then the exception rules compute its root-form (see 3.2.2). On the other hand the exception rules for nouns are tested first because the irregular form needs first to be identified as such, then it is processed accordingly by the suitable irregular rule.

4. Implementation[3]

We used Java as a programming language to build the GMA and the Arabic morphological analyzer to benefit from several advantages. First, our proposed system is Internet-based, meaning that it will be executed by a web browser. The morphological analyzer runs as a Java applet that is embedded in a web page and stored on a web server. Secondly, Java supports fully Unicode characters, which facilitates the integration of Arabic text as the system input. Finally, the object-oriented features of Java maps easily the system's description presented earlier in this paper.

The rules are stored in vector structures, which are initialized automatically upon start-up of the morphological analyzer from files that store off-line the rules attributes. They are represented as a Java classes which implement the rules attributes as discussed in section three. The rule sub-classes (*RegRulesTable* and *IrregRulesTable*) inherit a set of attributes from a common general class. Additional attributes and methods are added to every sub-class in order to permit the treatment of all tokens.

4.1. Regular rules

The most important attributes of the regular rule class are the *prefix* and the *suffix* used as preconditions to fire a rule.

The token-to-rule matching algorithm matches regular rules for verbs and nouns, it makes use of a fast-access data structure, which consists in a hash-code table used as a main-memory repository for the language rules. As rules are continuously looked up to match text tokens, the Java implementation of a hash table is used to speed up the matching process. The rules table implementation is shown below:

[3] This part has benefited of the collaboration of Y. Atif, H. Zidoum and D. Jaouid.

```
class RegRulesTable {
  public Rule[] Table;
  Hashtable Hash;
  public int Total;
//Constructor
  RegRulesTable() {
//Read from file to initialize Table
        . . .
        . . .
// Create a hash table
 Table = new Hashtable();
 for (int i = 0; i <= Total; i++)
   Hash.put( Table[i].Prefix + ' '
   + Table[i].Suffix, i);
}
rule FetchRule (String Suffix, Prefix)
{
return( Table[Hash.get (Suffix + ' ' +
Prefix)] );
}
}
```

Hence, rules are indexed based on their Suffix and Prefix attributes. The Java hash table structure requires that each table entry should have a key and the put method of the HashTable java utility class inserts items in the table based on the following signature: *put(key, item).* The item is an index to the rules table. The chosen key is a concatenation of Prefix and Suffix separated by a blank space to avoid any ambiguity. In case there is no suffix, the actual value of the Suffix attribute will be null and therefore the search for a rule locates only the rules that match the prefix.

The extracted tokens from the source text are represented as objects with distinctive attributes as the value of the token itself and its corresponding root classification inferred by the algorithm. The token class also exhibits two important methods used in the matching algorithm process which are *GetPrefix()* and *GetSuffix()* used respectively to extract the token prefixes and suffixes.

The goal of token-to-rule matching algorithm implemented in method *TokenToRule()* (shown below) is to return the rule that extracts the root of a given token *t*, and consequently, associates the morpho-syntactic information to the token. The rule is identified if it matches a given pair of suffix and prefix. Thus, first the affixes are extracted from the token *t* through *GetSuffix()* and *GetPrefix()* methods, then the matching operation is performed by the method *FetchRule(Suffix, Prefix).*

```
rule TokenToRule (Token t){
String Prefix, Suffix;

 public int i=3, j=1;
 rule Rule;
  do{
    Prefix = t.GetPrefix(i--);
    do{
      Suffix = t.GetSuffix(j--);
      Rule =
RegRulesTable.FetchRule(Suffix,
Prefix);
    }while(Rule==null && j>=0 &&
           t.length.value()-i-j > 1)
  }while(Rule == null && i >= 0 &&
         t.length.value()-i-j > 1);
  return (Rule);
}
```

Note that the length of a prefix for regular rules is at most one character, and the length of a suffix is limited to three characters. The matching algorithm gives the priority to the longest affixes first. Hence, the loops for affix extraction are respectively initialized to 1, and 3 and are gradually decremented as many time as no rule matches and the root still contains at least 2 characters (*t.length.value()-i-j > 1*) due to the fact that for regular tokens no root is less than two characters long.

4.2. Exception rules

Exception rules are activated only if a token is identified as an exception word-form. The system includes two sets of exception rules that are checked according to the order depicted in figure 3. Thus an input token passes first through exception rules for nouns. If it is listed as one of the irregular forms of nouns, the corresponding rule processes it. The system uses a hash table structure consisting of two fields: the token as the index, and the rule which calls the corresponding class methods to process the token.

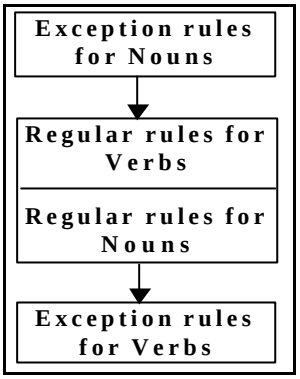

Figure 3. Rules activation strategy

If no rule is activated in the first set, then the regular rules are checked with respect to the order in figure 3 (i.e. first regular rules for verbs then regular rules for nouns) and are fired as described in section 4.1.

Exception rules for verbs are checked either if a token has not been treated, or if a regular verb rule has been activated which means that the affixes has been removed however, sometimes the root-form obtained is not right. In both cases the class methods map the input token to the possible correct root-forms which are then used as indexes to access the hash table. The rule returned is applied, and the token attributes are updated. For example, the token ‘ ’ is mapped to ‘ ’ that is an index to rule RV108 (see section 3.2.2).

5. Discussion

The architecture presented in the previous section has been used to implement the Arabic morphological analyzer. Our main concern, when designing the architecture, was to develop language independent models and software components to allow the implementation of morphological analyzers in other languages. The distinction between regular and exception rules can be seen as restrictive to model the morphological rules in a given language. Fortunately, our experience in the implementation of the Arabic morphological analyzer showed that modeling the linguistic knowledge by using the two models of rules turned to be a smooth and effortlessly task. Indeed, morphology as it is described in grammar books always make the distinction between the regular and irregular forms of verbs and nouns. This is, no doubt, one of the major advantages of using GMA. Another advantage is that it is dictionary free, thus it is faster since no need to access a dictionary for every token. It is also robust seeing that all the tokens (even misspelled or in other languages) will be accepted by the system. However, the method lacks some precision by passing over the use of a dictionary. This is why we are currently extending the coverage of the exception lists from dictionaries and corpora: the more the exception lists have a large language coverage, the precise the system is.

We feel confident that the architecture can be applied to a variety of other languages. For instance, English and French morphology make the distinction between the regular and irregular forms of verbs and nouns, they can easily be described by using the two rule models.

The architecture possesses generic components embedded in the GMA. The rule models and the object models are definitely generic, they can be used as they are in any other implementation. The matching algorithm basically matches rules to tokens. However, to speed-up the process, currently it uses a specific heuristic to match Arabic regular verbs, that is, the length of a regular verb root-form cannot be less than two characters, the prefix is at most one character and the suffix is limited to three characters. In the future improvements we will take into consideration to separate the core algorithm that is language independent and the heuristics that have been added for the Arabic language.

The strategy we are using allows the system to organize the order in which the rules are tested (fig. 3), it takes into account somehow Arabic language particularities. It is not completely

language independent, nevertheless it can be reused and adapted in other implementations.

6. Conclusion and Future Work

We described in this paper an architecture that allow the implementation of morphological analyzers in different languages. The architecture includes the GMA that is language independent, it is used to develop language specific morphological analyzers. The architecture highlights two main phases toward the development of a morphological analyzer: First, modeling the linguistic knowledge by using the rule models, then the implementation phase that exploits the software components available in GMA. One direct contribution of the architecture concern multi-lingual processing as sources of information nowadays contain a combination of textual pieces in different languages.

The Arabic morphological analyzer has been implemented by using this architecture. Currently we are working on the extension of the exception lists from dictionaries and corpora in order to enlarge the language coverage of the system. This effort constitute an important ingredient in this work as it is a knowledge-based method.

For the future much work remains to be done. We are planning a study to evaluate the Arabic morphological analyzer using a corpus of suitably collected texts. In addition we want to test GMA with other languages and implement additional morphological analyzers. We are targeting more specifically English and French languages.

7. References

Alshalabi R., and Evens M. 1998. "A Computational Morphology System for Arabic", In *Workshop on Computational Approaches to Semitic Languages COLING-ACL98*, August 16, Montreal.

Arrajihi A. 1973 *The Application of morphology*, Dar Al Maarefa Al Jameeya, Alexandria. (in Arabic)

ARPA. 1996. *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Morgan Kaufmann, Los Altos, CA.

Beesley K. 1998. "Arabic Morphological Analysis on the Internet", In *Proceedings of the International Conference on Multi-Lingual Computing (Arabic & English)*, Cambridge G.B., 17-18 April.

Berri J., Zidoum H, Atif Y. 2001. "Web-Based Arabic Morphological Analyzer", *Lecture Notes in Artificial Intelligence*, Vol. 2004, Gelbukh A. (Ed.), Springer, 389-400.

Berri J. 2000. "A Generic Text Representation for Information Extraction", In *Proceedings of GITIS'2000 Gulf Information Technology & Information Systems Conference*, 130-139, Dubai.

Berri J., Le Roux D., Malrieu D., Minel J.-L. 1995. "SERAPHIN main sentences automatic extraction system", In *proceedings of the Second Language Engineering Convention*, London.

Berri J., Maire-Reppert D., Oh-Jeong H.-G., 1992. "Traitement informatique de la catégorie aspecto-temporelle", *T.A Informations*, Vol. 32, n°1, 77-90.

Desclés. J. –P. 1990. *Langages applicatifs, Langues naturelles et Cognition*, Hermès, Paris.

Desclés J. –P., Minel J. –L., Berri J. 1995. "L'exploration contextuelle des unités linguistiques dans la compréhension de textes", *Internal report*, CAMS (Centre d'Analyses et de Mathématiques Sociales), 15 p.

El-Sadany T. A., and Hashish M. A. 1989. "An Arabic Morphological System", In *IBM Systems Journal*, Vol. 28, No. 4, 600-612.

Jouis C. 1995. "SEEK, un logiciel d'acquisition des connaissances utilisant un savoir linguistique sans employer de connaissances sur le monde externe", In *Proceedings of 6eme Journées Acquisition,*

Validation, (JAVA), INRIA and AFIA, Grenoble, 159-172.

Kiraz G. A. 1998. "Arabic Computational Morphology in the West.", In *Proceedings of the 6th International Conference and Exhibition on Multi-lingual Computing*, Cambridge.

Mädche A., Neumann G., Staab S. 1999. "A Generic Architectural Framework for Text Knowledge Acquisition", *Unpublished Technical Report*, Kalsruhe University, 18p. Available at http://www.aifb.uni-kalsruhe.de/WBS

Miller G. A., Beckwith R., Fellbaum C., Gross D., and Miller K. J.. 1990. "Introduction to wordnet: an on-line lexical database." *International Journal of lexicography*, 3(4):235-244.

Mollá Aliod D., Berri J., Hess M. 1998. "A Real World Implementation of Answer Extraction", In *Proceedings of The 9th Conference and Workshop on Database and Expert Systems", Workshop "Natural Language and Information Systems" (NLIS'98)*, Vienna.

Qabawah F. 1994. *Morphology of nouns and verbs*, Al Maaref Edition, 2nd edition, Beyruth. (in Arabic)

Saliba B. and Al Dannan A. 1989. "Automatic Morphological Analysis of Arabic: A study of Content Word Analysis", In *Proceedings of the Kuwait Computer Conference*, Kuwait, March 3-5.

Smets M. 1998. "Paradigmatic Treatment of Arabic Morphology", In *Workshop on Computational Approaches to Semitic Languages COLING-ACL98*, August 16, Montreal.