Design of A Spell Corrector For Hausa Language

Lawaly Salifou salifoumma@yahoo.fr
Département de Mathématiques et Informatique
Faculté des Sciences et Techniques
Université Abdou Moumouni
Niamey, BP 10662, NIGER

Harouna Naroua hnaroua@yahoo.com
Département de Mathématiques et Informatique
Faculté des Sciences et Techniques
Université Abdou Moumouni
Niamey, BP 10662, NIGER

Abstract

In this article, a spell corrector has been designed for the Hausa language which is the second most spoken language in Africa and do not yet have processing tools. This study is a contribution to the automatic processing of the Hausa language. We used existing techniques for other languages and adapted them to the special case of the Hausa language. The corrector designed operates essentially on Mijinguini's dictionary and characteristics of the Hausa alphabet. After a brief review on spell checking and spell correcting techniques and the state of art in the Hausa language processing, we opted for the data structures trie and hash table to represent the dictionary. The edit distance and the specificities of the Hausa alphabet have been used to detect and correct spelling errors. The implementation of the spell corrector has been made on a special editor developed for that purpose (LyTexEditor) but also as an extension (add-on) for OpenOffice.org. A comparison was made on the performance of the two data structures used.

Keywords: Natural Language Processing, Spell Checker, Spell Corrector, Computerization of Hausa, African Languages.

1. INTRODUCTION

Automatic natural language processing (NLP) has many industrial applications including, among others, verification and correction of spelling and grammar, text indexing and retrieval of information from the Internet, voice recognition and synthesis, vocal control of domestic robots, automated response systems and machine translation [1 - 2]. The most commonly used application is of course spell checking. Indeed, it is integrated into computer tools used every day by millions of people worldwide. Computer programs in this area are of two kinds: spell checkers and spell correctors. A spell checker detects spelling errors in a given text whereas a spell corrector both detects spelling errors and seeks for the most likely correct words [3]. The correction can be automatic (in the case of a speech synthesizer for example) or interactive allowing the user to select the desired word from several suggestions [1]. This second approach is the one used by most of word processing softwares. Such programs are generally designed for a given language. Spell checking is nowadays present in almost all computer applications where text is expected to be entered by the user. Wrong words are generally underlined in red to alert the user. Examples of such applications are word processors, email clients, source code editors, programming environments and search engines. The causes of errors are of several orders and there is more than one way to classify them [4]. The most important causes are ignorance of the author, typographical errors and errors in transmission and storage [3]. A spell corrector performs two main functions, one after another: first detecting and then correcting spelling errors. In one of his articles, Kukich [1] said that the methods of detection and correction use three approaches:

- non-word error detection, eg. 'grafe' written instead of 'giraffe';

- isolated-word error correction;
- Context error detection and correction: each word is considered taking into account the context, thereby correcting spelling errors even when they consist of real words.

2. TECHNIQUES AND ALGORITHMS FOR ERROR DETECTION

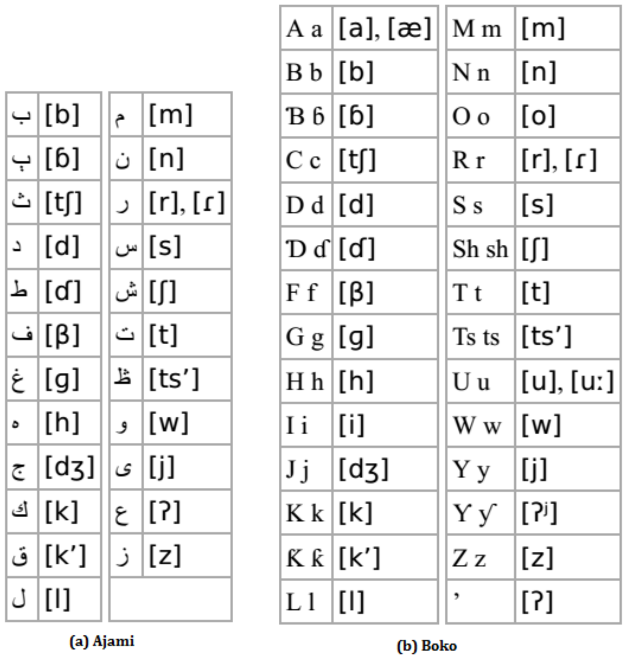
The search for solutions to spelling errors correction problem has been a challenge for many years. Efforts in that area led to the emergence of various techniques and algorithms. Error detection is to find incorrect words in a text. A wrong word is then marked by the application in charge of spell checking. If the word is really wrong - because it is not always the case - an error is said to be detected. Research in this area has been done by many authors [2 - 10]. The main techniques used for non-word error detection in a text are either based on analysis of n-grams, or dictionary lookup [1]. The techniques based on n-grams are to analyze each n-gram of a given input word and check its validity in a precompiled table. These techniques usually require a dictionary or corpus that's large enough to determine the statistics table of n-grams [1]. A dictionary is a collection of correct or acceptable words. Some authors use the term lexicon or the phrase "word list" instead of dictionary. The techniques based on the use of a dictionary or lexicon involve taking a word as input and verifying its existence in the dictionary. Any word that is not in the dictionary is then considered wrong [1]. A detection algorithm based on dictionary lookup is given by Peterson [3]. The hash table is one of the most commonly used data structures to reduce response time when searching in a dictionary [1]. The idea of hash table was introduced for the first time in 1953 [11]. With the hash code, a hash allows a selective access to the searched word and, therefore, significantly reduces the response time. But the major drawback is finding a hash function which admits very few collisions and provides uniformly distributed indices in the considered interval. UNIX spellchecker *Spell* illustrates the use of a hash table for fast search in a dictionary. Binary search trees are especially useful to check whether a given word belongs to a larger set of words. Several variations of binary search trees were used to accelerate dictionary search. Among them is the Median Split Tree, a modified frequency-ordered binary search tree that allows faster access to most frequently used words. Finite automata were also used in some search algorithms in a dictionary or a text. One of the famous algorithms in this area is that of Aho - Corasick [12]. The algorithm is to move through an abstract data structure called dictionary that contains the words to search by reading the text characters one by one. The data structure is implemented efficiently, which ensures that each character of the text is read only once. Generally, the dictionary is represented using a trie. A trie may be seen as a representation of the transition function of a deterministic finite automaton. The algorithm has a linear complexity in the size of the text and search strings. Comparatively, techniques using n-grams derived from a dictionary provides less accuracy than those using all the information in the dictionary. But, the latter ones are time consuming depending on the data structure used to represent the dictionary. A comparative study showed that the hash table provides better performance than the AVL tree, the Red-Black tree and Skip list [9]. A comparison of five data structures was performed for the Punjabi dictionary [13]. It concerned binary search tree, trie, ternary search tree, multi-way tree and reduced memory method tree. As a result, the binary search tree was found to be the most suitable data structure in terms of memory usage and time. But it is limited when it comes to suggest a list of candidates for the correction or find all words that differ by one or two characters. This limitation may be avoided by the use of a trie which offers almost the same time complexity with a binary search tree. Hash table and trie are shown to be the most suitable data structures for dictionary representation.

3. TECHNIQUES AND ALGORITHMS FOR ERROR CORRECTION

Error correction refers to the fact of equipping spell checkers with the ability to correct detected errors. This is to find words in the dictionary (or lexicon) that are similar in some ways to the misspelled word. The task of a spell corrector is thus composed of three sub-tasks: detecting errors, generating possible corrections, and ranking suggested corrections. To achieve this, various techniques were invented. Each is related either to non-word error correction, real-word error correction, or both. Spelling errors may be typographical, cognitive or phonetic. Typographical errors occur when the keys are pressed in the wrong order (eg ahnd instead of hand). Cognitive errors arise from ignorance of the correct spelling of the word (eg sicretary instead of secretary). Phonetic errors are special cases of cognitive errors. A phonetic error refers to a wrong word that is pronounced the same way as the correct word (eg speack / speak). In typed texts, 1% to 3% of the errors are spelling errors [14]. Damerau [6] stated that 80% of these errors are related to insertion, deletion, substitution, or transposition. The minimum edit distance or simply edit distance is until now the most widely used technique in the spelling errors correction. It has been applied in almost all spell checking functions in text editors and command language interfaces. The first spelling correction algorithm based on this technique was proposed by Damerau [6]. Almost at the same time, Levenshtein also developed a similar algorithm. Several other algorithms on edit distance were born thereafter. The edit distance is defined as the minimum number of edit operations required to transform a word to another [1]. These operations are insertion, deletion, substitution and transposition. In most cases, correcting a spelling error requires the insertion, deletion or substitution of a single character, or the transposition of two characters. When a wrong word can be transformed into a dictionary word by inverting one of these operations, the dictionary word is considered a plausible correction. Damerau's algorithm [6] for edit distance detects spelling errors by comparing words of four to six characters with a list of most frequently used words. When there are multiple candidate words for a given edit distance on a detected word, the first word in the dictionary appearing in alphabetical order is chosen. Levenshtein's algorithm is in the field of dynamic programming and seems to be the most widely used in edit distance computing. Each edit operation is assigned a cost, usually 1 for deletion and insertion and 2 for substitution and transposition. Given a dictionary of n words, the correction algorithms based on edit distance generally require n comparisons for each wrong word. To reduce the search time, reversed edit distance technique is used. Another approach used to reduce the number of comparisons involves sorting or partitioning the dictionary according to certain criteria (alphabetical order, word length, words occurrences). Many other techniques are also used in spelling errors correction like: similarity keys, rules system, n-grams, probabilistic techniques and neural networks. However, the most widely used technique in errors correction remains edit distance [7]. It has a time complexity of O(nm), with n and m the respective sizes of the two compared words. A technique developed by Horst [15] combining automata and edit distance was used to quickly find the closest correct word to a wrong word. It has a linear complexity in time relative to the length of the wrong word, regardless of the dictionary size. But the space complexity of the method is exponential ($O\left(3.\exp\left(\sum_{i=1}^{N}|A_i|\right)\right)$, where A_i are the words of the dictionary).

4. REVIEW OF THE HAUSA LANGUAGE PROCESSING

Hausa is part of the family of Afro-Asiatic languages. It belongs to the group of Chadic languages (sub-group of West Chadic languages) [16]. Compared to other African languages, Hausa is remarkably unitary. Standard Hausa (Kano dialect) is to be distinguished from West dialect (Sokoto) and Nigerien dialects (Tibiri, Dogondoutchi, Filingué) [17]. From a vocal point of view, the Hausa words have high tones and low tones and one can observe a flexion of gender and number [18]. Geographically, Hausa is the second most spoken language in Africa. It is the most widely spoken language in sub-Saharan Africa with about a hundred million speakers worldwide. Hausa is now used by major radio stations of the world such as VOA (USA), BBC (UK), CRI (China), RFI (France), IRIB (Iran), Deutsche Welle (Germany) and Radio Moscow (Russia). In Niger, Hausa and other national languages are used by regional and local, public and private media [19]. Cinematographically, the Hausa language video industry has made a remarkable progress. Indeed, over 1000 Hausa films are produced each year, mainly from Nigeria. The presence of Hausa on Internet is very precarious. It is unfortunately the same case with all African languages even though they represent 30% of the languages of the world [20]. According to Van Der and Gilles-Maurice [20], Hausa texts can be divided into three main categories: popular culture (47%), newspapers (35%) and religion (17%). The percentage of texts produced on forums by Internet users is tiny (0.3% of total words). Today's famous search engines like Google and Mozilla Firefox as well as other softwares and electronic gadgets (including mobile phones) have a graphical user interface in Hausa. The ISO identifier for Hausa language is *ha* or *hau* (ISO 639-3 and ISO 639-1). On the academic side, the first poems written in Hausa with Arabic alphabet adapted to the notation of African languages (`Ajami), date from the early nineteenth century. To this tradition was added in the 1930s, as a result of British colonization, a literary production in Latin alphabet (dramas, tales, stories, poetry) [21]. Hausa language is now being taught in African and Western universities (Niger, Nigeria, Libya, Inalco (Paris) , Boston University, UCLA) . The written Hausa is essentially based on the dialect of Kano and there are two writing systems, one based on the Arabic script (Ajami) and the other using the Latin alphabet (Boko) as shown in Figure 1. We note, in the case of Boko, the presence of four additional special characters like consonants (ɓ , ɗ , ƙ and ƴ) and glottal stop (').



ب	[b]	م	[m]
ٻ	[ɓ]	ن	[n]
ث	[tʃ]	ر	[r], [ɾ]
د	[d]	س	[s]
ط	[ɗ]	ش	[ʃ]
ف	[β]	ت	[t]
غ	[g]	ڟ	[ts’]
ه	[h]	و	[w]
ج	[dʒ]	ى	[j]
ك	[k]	ع	[ʔ]
ق	[k’]	ز	[z]
ل	[l]		

(a) Ajami

A a	[a], [æ]	M m	[m]
B b	[b]	N n	[n]
Ɓ ɓ	[ɓ]	O o	[o]
C c	[tʃ]	R r	[r], [ɾ]
D d	[d]	S s	[s]
Ɗ ɗ	[ɗ]	Sh sh	[ʃ]
F f	[β]	T t	[t]
G g	[g]	Ts ts	[ts’]
H h	[h]	U u	[u], [uː]
I i	[i]	W w	[w]
J j	[dʒ]	Y y	[j]
K k	[k]	Ƴ ƴ	[ʔʲ]
Ƙ ƙ	[k’]	Z z	[z]
L l	[l]	’	[ʔ]

(b) Boko

FIGURE 1: Writing Systems for Hausa Language.

The Latin transcription, introduced by the British in Nigeria in the early 20th century, has emerged in 1930 as the official spelling [17]. In Niger, it was only in 1981 that the Latin alphabet was used as an official Hausa spelling. This alphabet was completed in 1999 by a decree [22]. This is the same alphabet as that of Figure 1 (b) to which are added digraphs fy, gw, kw, ky, ƙw, and ƙy representing specific and considered consonant sounds. The same decree defined the symbols in Table 1 as punctuation symbols.

Symbol name	Symbol
Full stop	.
Comma	,
Semi colon	;
Colon	:
Interrogation mark	?
Exclamation mark	!
Parentheses	()
Quotes	"
Union mark	-
Suspension marks	…
Dash, next line	-
Asterix	*
Dashes or parentheses	…-…-
Dash	-

TABLE 1: Hausa Official Punctuation Symbols In Niger.

The Boko became the dominant writing convention for scientific and educational materials, mass media, information and general communication since the second half of the 20th century [23]. The first step in the computerization of a language is the existence of language resources [24]. It is the only possible way to design computer tools (editors, spelling and grammar checkers, electronic dictionaries ...) adapted to the language and ensure its presence in the cyberspace. But these resources are scarce for African languages. Various projects and studies are carried out which aim is the constitution or usage of these linguistic resources for total computerization of African languages. For example, PAL is a project aimed at adapting information technology to African languages to make them more accessible to indigenous peoples [25]. The work on modern Hausa lexicography started in 1843 with Schön's "Hausa vocabulary". Schön compiled, in 1876, the first work that can be called Hausa-English bilingual dictionary with 3800 entries. A few years later appeared the first Hausa - French dictionary written by Le Roux in 1886. In the early 20th century, three other bilingual dictionaries were published, including Landeroin and Tilho's (1909) Hausa-French dictionary with 6000 entries. The dictionary of Bargery [26] seems to be the most important and the largest (with 39 000 words) Hausa dictionary. In their Hausa lexicography genesis, Roxana and Paul [27] mentioned several other dictionaries before discussing the Hausa - French bilingual dictionary written by Mijinguini [28], a Nigerien Hausa native linguist. This dictionary is, according to them, "the latest scientific reference in Hausa lexicography". It includes 10,000 well illustrated entries and is largely based on the standard Hausa of Niger, consisting essentially of the Damagaram dialect instead of the Kano dialect that dominated all previous lexicographical researches. It is important to recall that Paul [29] is the author of the most comprehensive work on modern Hausa grammar. The majority of well-resourced languages have well-formed corpuses. This is not the case for African languages. The current researches on these languages choose oral and written corpuses as a transitional alternative. Another alternative for African languages is to build a corpus from the Web [30]. For example, a search for four days on the Web ended up with a Hausa language corpus containing 858,734 words in total, including 30,996 different words [20].

Text entry is another difficulty to overcome in the computerization of Hausa and other African languages. Indeed, computer keyboards are not compatible with these languages. Thus entering some Hausa characters on a keyboard now requires an acrobatic work. The solution for this problem is the use of keyboard layouts to write African languages specific characters. An evaluation of such keyboards for 5 Nigerien languages (Fulfulde, Hausa, Kanuri, Songhai - Zarma, Tamasheq) recommended the LLACAN keyboard [31]. In fact, LLACAN covers all the symbols of the alphabets of those languages, produces valid Unicode code and requires less buttons to press.

Word processing softwares such as MS Word and OpenOffice.org Writer can be used for the correction of written Hausa text through the establishment of a user dictionary. However, all existing methods are limited and inadequate in the case of African languages, hence the need to develop spell correctors adapted to these languages [32]. Despite the scarcity of linguistic resources, it is possible to develop such spell correctors and improve them over the time. With the possibility to create extensions for some popular softwares (MS Word, OpenOffice.org Writer, Firefox, etc.), it would be advantageous to develop spell correctors that can be easily integrated to them.

5. DESIGN OF A SPELL CORRECTOR FOR HAUSA LANGUAGE

After synthesizing spelling correction techniques and presenting the Hausa language, we can now design a spell corrector for that language. We expose the approaches and techniques chosen and the implementation details of the proposed solution.

5.1. Chosen Techniques

In this section, we present the data structures used for the design of the corrector and the procedures necessary for the detection and correction of errors in Hausa. We decided to approach the subject with an algorithmic design inspired from Java [33]. We will not dwell on the theory of the underlying concepts such as class, object, method, attribute, instance, etc. [33] and [34] are good references on this subject. Taking into account the linguistic resources available to us, a technique based on a dictionary seems to be more suitable for the design of the corrector. Regarding the dictionary, we opted for that of Mijinguini [28] for its assets and accessibility to us. The dictionary contains all the words (including inflections and derivations). It is stored in secondary memory as a text file using UTF-8 character encoding. Error detection is independent of the context. An erroneous word is identified by a simple dictionary lookup. To represent the dictionary in primary memory, we use either a hash table or a trie. The implementation must allow at least the following operations:

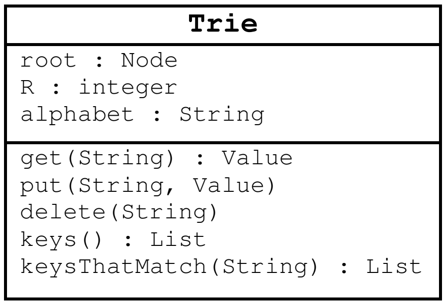
• Add a word to the dictionary (add method)

• Check if a word is in the dictionary (contains method)

• Delete a word from the dictionary (remove method)

Each node in the trie has as many links as there are characters in the alphabet and the latter ones are stored implicitly in the data structure. Each valid character string is assigned a value. This may be of any type. It can be used here to store information on every word in the dictionary (definition, grammatical class, translation into another language, etc.).

In the object notation, a trie is as shown in Figure 2. Each node of the trie is represented by the Node data structure of Figure 3.



```
Trie
root : Node
R : integer
alphabet : String
get(String) : Value
put(String, Value)
delete(String)
keys() : List
keysThatMatch(String) : List
```

FIGURE 2: Representation of the Trie Data Structure.

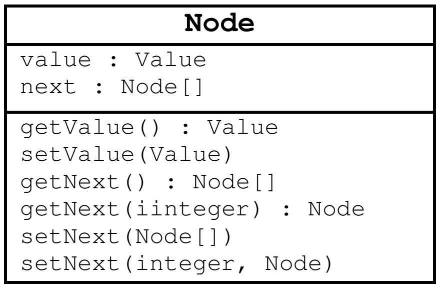


FIGURE 3: Representation of The Node Data Structure.

The R attribute of the class Trie is the number of symbols or letters of the alphabet. Since the digraphs of the Niger Hausa alphabet are not coded as single characters, we shall consider only the monographs which make a total of 28 letters. To these letters, we add the dash ('-') (Unicode code \ u002D) in order to store compound words. If the language were supported by ASCII code, it would not be necessary to have an alphabet attribute for Trie class, as the characters are represented by consecutive numbers from 0 to 127 therefore by indices of *next* array (Node [ ]) . This is not the case for the Hausa language where letters have code points in the following ranges:

• Uppercase letters: 39, 65-80, 82-85, 87-90, 385, 394, 408, 435.

• Lowercase letters: 97-112, 114-117, 119-122, 595, 599, 409, 436.

Representing characters by indices of *next* array will set the value of R to 599 instead of 56 (28x2). This will inevitably lead to a waste of memory space and additional checks to prevent foreign words from being added to the trie. To avoid this problem, a trick [35] is to find a mapping function between indices of the next array and letters of the alphabet. That is why the *alphabet* attribute is present in the class *Trie*. It is here of type String but it may also be an array of characters. Two additional methods, toChar and toIndex, assure the conversion from indices to characters and vice versa. The charAt and the indexOf methods of the String class can be effectively used. And to make the trick more flexible, we can totally delegate this task to an interface *Alphabet* that defines *toChar* and *toIndex*. The *KeysThatMatch* is another interesting method. Indeed, it allows to search the trie for words that match a given pattern. The patterns used here are those with a wildcard, for example a dot ('.'). For instance, given the pattern '.ada', this method will return the dictionary words which consist of a letter (any) followed by the suffix 'ada' : dada, fada, kada, lada, tada, wada. It is this possibility that we use to implement the reverse editing distance. The *KeysThatMatch* method uses a data structure *List* (a linked list of Strings) to keep the search results. The *List* class has methods to add an item, to verify the existence of an item and to delete an item.

To abstract the implantation of the real dictionary, add flexibility, simplify maintenance and facilitate scalability of the spell corrector, an abstract dictionary is represented by a class (*TrieBasedDico* or *HashBasedDico*) that implements *Dico* interface (or abstract class). It defines the methods (add, remove, contains) needed to operate on a dictionary. *TrieBasedDico* and *HashBasedDico* classes are designed by composition from Trie and HashSet (hash) classes respectively as shown in Figure 4.

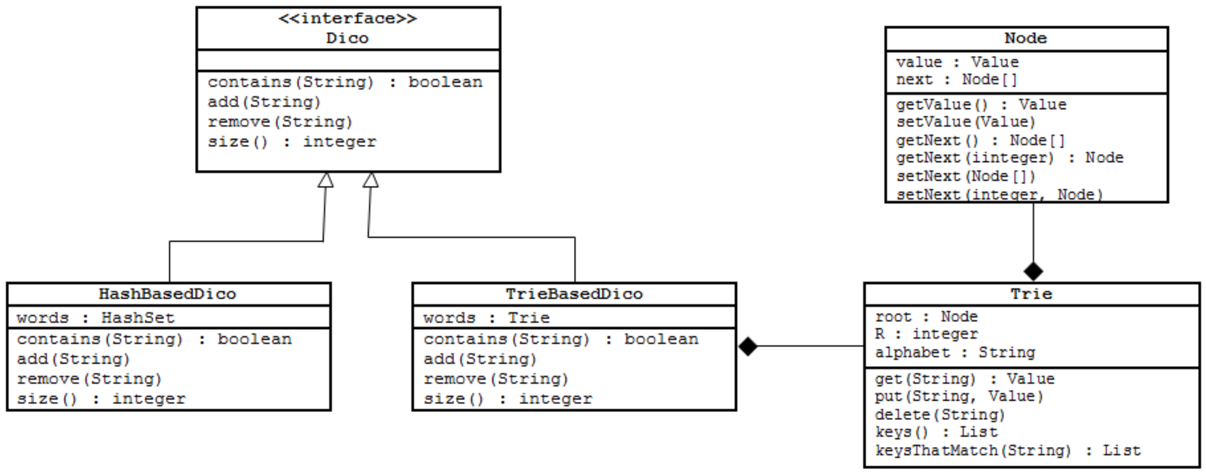


FIGURE 4: Class Diagram For The Implantation of The Dictionary.

The list of candidate words for the correction of an erroneous word is determined in several steps that we describe here. Once a word is identified as being erroneous, the procedure for determining the type of the error follows. We defined three types of errors (inspired by our research on OpenOffice.org):

- IS_NEGATIVE_WORD: Error caused by the presence of a number or a character not belonging to the alphabet (eg x , v, q, etc.) in the word. The word is called negative.
- CAPTION_ERROR: Case Error. This is when a word that should be written with the first letter capitalized is written entirely in lowercase.
- SPELLING_ERROR: represents all other types of spelling errors.

The types of errors are short integers encapsulated as static fields in the *LySpellFailure* class. The corrector has two methods for the determination of errors. First, the *getSpellFailure* method which analyzes a given word and returns -1 if the word is correct or one of the three types of errors mentioned above otherwise. Then *isValid* method that checks whether a given word is valid according to the result returned by *getSpellFailure* and spellchecking settings. If *getSpellFailure* returns a value:

- equal to -1, the word is valid and isValid returns true
- other than -1, the correction parameters are taken into account to determine the validity of the word. For example when you choose not to correct words with numbers and the erroneous word contains digits, *isValid* returns true. This method can be exploited to correct spelling as you type.

currentLanguage represents the language being supported by the spell corrector. It is an instance of *Language* class. Searching suggestions is performed by *propose* which is an instance of a class that implements the interface *Proposer*.

The method *getProposals* provides correction suggestions for an invalidated word by *isValid* depending on the type of error detected by *getSpellFailure*.

a) Use of the characteristics of the alphabet

The processed language is represented by the Language class given in figure 5:

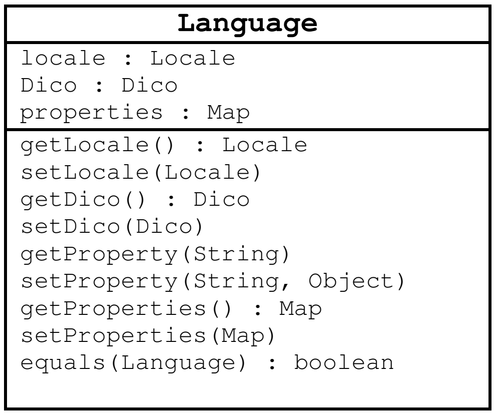


FIGURE 5: Language Class Diagram.

After several attempts, we decided that the dictionary is an attribute of the language and not the reverse. The Local attribute of the Language class stores information about the processed language. It is of type Locale (representation of a language in Java) and provides among others : a 2-letter ISO 639-1 code of the language, a 2-letter ISO 3166 code of the country as well as the complete names of the language and the country. This corresponds to ha, NE, Hausa (Niger) respectively for Hausa of Niger. We use this data for naming resources and for user display. The *properties* attribute is of type Map (mapping key / value) and stores other properties of the language that we use to design the spell corrector and which are not provided by *Locale*. They are currently the alphabet of the language (value of the key "alphabet"), the special characters in the alphabet (value of the key "specialChars"), the characters that look like special characters (value of the key "specialCharsLike") and the punctuation symbols that we divided into two parts: word separators (value of the key "punctuation") and end of sentence signs (value of key "endOfSentence"). All the characters of the alphabet are coded in Unicode. The class in charge of finding suggestions implements *Proposer* interface which defines two methods: *isNegativeWord* and *propose* as shown in the class diagram of Figure 6. The *TrieBasedDicoProposer* and *HashBasedDicoProposer* classes use some features of the alphabet to find candidate words.

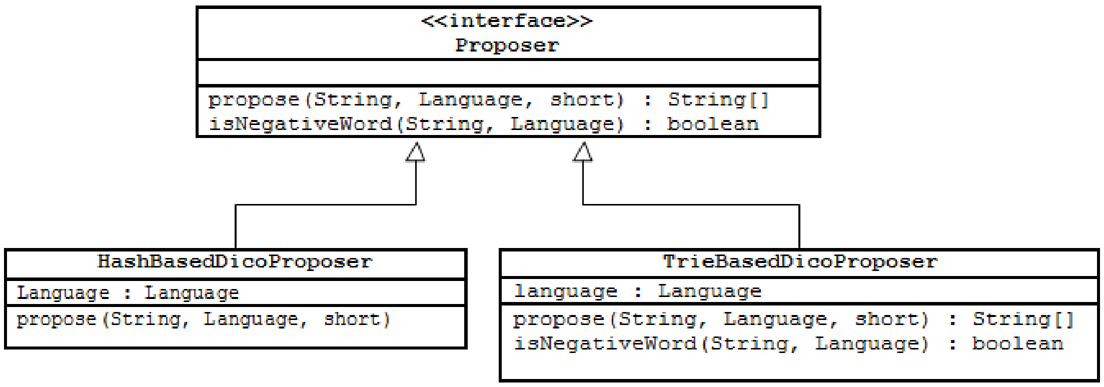

<<interface>>
Proposer
propose(String, Language, short) : String[]
isNegativeWord(String, Language) : boolean
HashBasedDicoProposer
Language : Language
propose(String, Language, short)
TrieBasedDicoProposer
language : Language
propose(String, Language, short) : String[]
isNegativeWord(String, Language) : boolean

FIGURE 6: Class Diagram For Correction Suggestions.

b) Use of the reverse edit distance to find candidate words

Candidate words are found using the reverse edit distance as follows:

- All words having an edit distance equal to 1 with the wrong word are generated by applying edit operations such as insertion, deletion, substitution and transposition. A total of 60n+28 words are generated for a wrong word of length n.
- Each previously generated word is searched in the trie or the hash table (which represents the dictionary). If it is there, then it is retained as a possible correction of the erroneous word.

The research is conducted by a private method called *proposeByReverseEditDistance*. This method is actually based on *keysThatMatch*. It takes an argument of type *TrieBasedDico* and a word or a pattern and returns the result as an array of Strings. A similar method is designed in the case of hash table. Methods that perform editing operations on a given word are provided by the *StringTools* class which consists of tools shared by different classes.

c) Use of edit distance to rank candidate words

The minimum edit distance is used to rank the suggested words. Those who are closest to the wrong word are placed at the top of the list. To implement that, a comparator was designed.

The entire class diagram of the designed spellchecker is given in Figure 7 below:

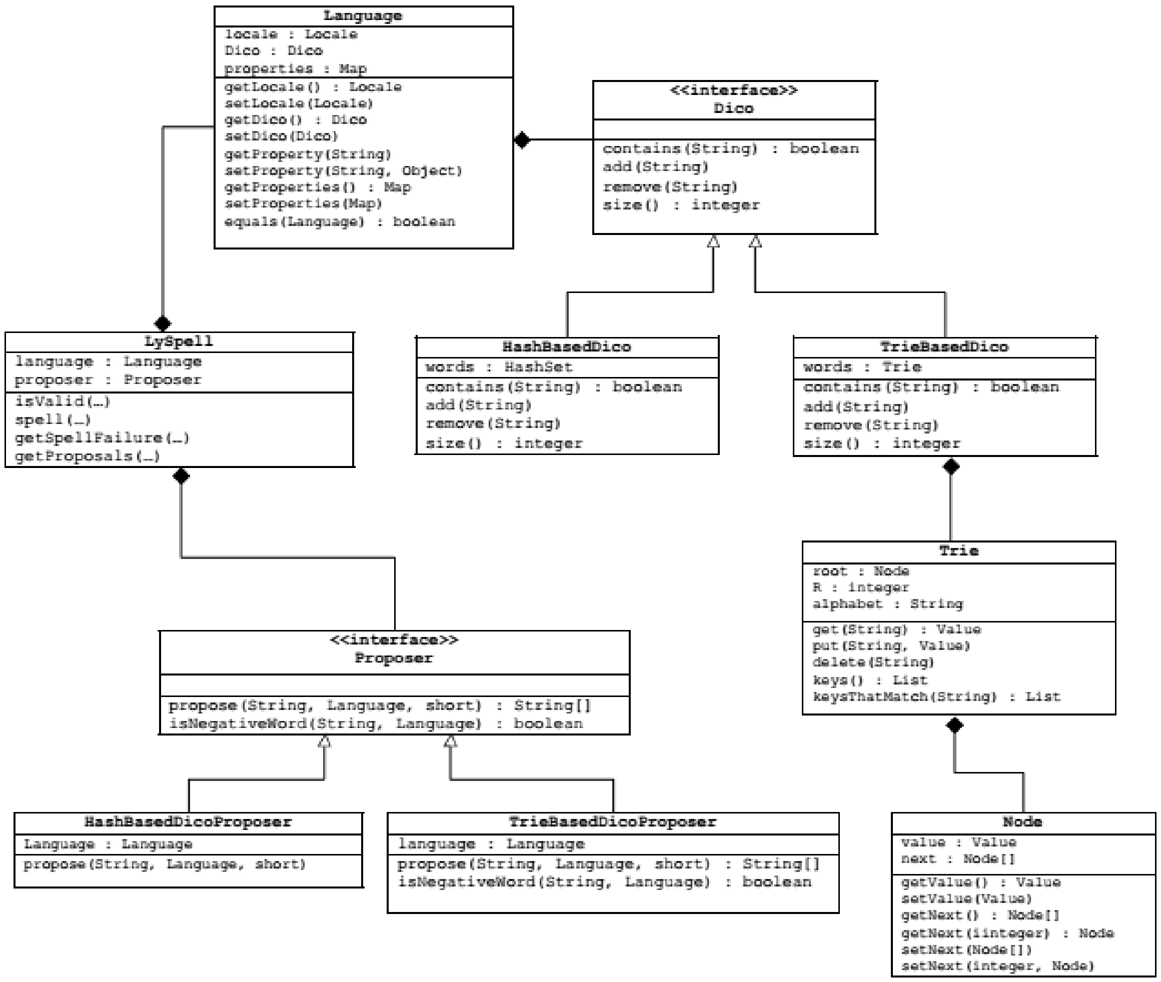


FIGURE 7: Global Class Diagram.

5.2. Implementation of The Spell Corrector

To implement the solution, we opted for Java and NetBeans IDE. Two versions have been developed.

a) Standalone LyTexEditor and LySpell

This solution includes a text editor "LyTextEditor" that integrates a spell corrector "LySpell". LyTextEditor gives the following possibilities:

• type a text;
• open an existing text file;
• correct a text with LySpell;
• save a text.

Spell checking and correction is accessed via the Tools menu or by pressing F7.

b) Add-on for OpenOffice.org

With the help of the OpenOffice.org Developer's Guide [36], we were able to develop the add-on.

With the portability of Java, LyTextEditor and LySpell can normally be used on all platforms. LySpell also offers the opportunity to correct spelling errors for other languages without any need to modify the code. One can do so by simply providing a dictionary file and the alphabet of the intended language.

6. RESULTS

In this work, we developed a spell corrector for Hausa language. The final product was tested as a standalone program through a text editor designed for this purpose and as an extension for the OpenOffice.org office suite. These results show that it is possible, from proven techniques and language resources, to develop automatic processing tools for African languages in general and for Hausa in particular. They also confirm that the data structures trie and hash table offer better performance for storing a dictionary and compare very well with [9] and [13]. A trie is actually a DAWG (Directed Acyclic Word Graph), a way to represent an acyclic deterministic finite automaton. However, the possibilities and the results provided by the data structure trie are significantly better than those of the hash table. Note that when the number of wildcards is greater than 1, only the trie gives easily a satisfactory result. For example, for the incorrect word "zurmakakke", the correct word "zurmaƙaƙƙe" is suggested when the dictionary is implanted using a trie while no suggestion is obtained in the case of the hash table.

The corrector LySpell resulting from this study uses only a dictionary as language resource and the alphabet of Hausa language. However, it was designed and implemented so that it can also be used for other languages. The specificities of the Hausa and other African languages are efficiently handled.

Although we were not able to perform all necessary tests on the performance of LySpell, we believe that the results we have obtained will add value to the computerization of the Hausa language and contribute to its effective use in institutions of education and on media.

To improve the performance of the designed corrector, it may be considered in future works the possibility to:

- use the morphologic rules of the Hausa language. This will have a triple advantage. First the size of the dictionary in memory will be significantly reduced. Then, the suggestions for correction could be more precise. Finally, it is possible to create a Hunspell oriented spell corrector that can be integrated easily and appropriately to a wide range of programs.

- Strengthen the spell checking and correction by adding grammar checking.

7. REFERENCES

[1] K. Kukich. "Techniques for automatically correcting words in text", ACM Computing Surveys, vol. 24, No. 4, 1992.

[2] M.N. Pierre. "An introduction to language processing with Perl and Prolog", Springer-Verlag Berlin Heidelberg, p.2-3, 2006.

[3] J.L. Peterson . "Computer Programs for Detecting and Correcting Spelling Errors", Comm. ACM, vol. 23, No. 12, December 1980.

[4] V. Suzan. "Context-sensitive spell checking based on word trigram probabilities", Master thesis, February - August 2002.

[5] N.A. Cyril. "String similarity and misspellings", Communications of the A.C.M., vol. 10, no. 5, pp. 302-313, May 1967.

[6] F.J. Damerau. "A technique for computer detection and correction of spelling errors", Comm. ACM 7, vol. 3 , pp. 171-176, March 1964.

[7] L.L. Hsuan. "Spell Checkers and Correctors: a unified treatment", Master dissertation, November 2008

[8] B. Laurent. "Production de logiciels et d'utilitaires pour le traitement informatique de langues africaines dans un contexte de NTIC multilingues", 2nd World Congress of Community Networks, Buenos Aires, Argentine, du 5 au 7 décembre 2001.

[9] P.N. Mark. "A Comparison of Dictionary Implementations", April 10, 2009.

[10] E.M. Zamora. Pollock J. J. and Antonio Z., "The use of trigram analysis for spelling error detection", Information Processing & Management, vol. 17. No. 6, pp. 305-316, 1981.

[11] K. Donald. "The Art of Computer Programming", Addison-Wesley Publishing Co., Philippines, vol. 3, 1973.

[12] A.V. Aho and M.J. Corasick. "Efficient String Matching: An Aid to Bibliographic Search", Communications of the ACM, vol. 18, No. 6, pp. 333-340, June 1975.

[13] G.S. Lehal and K. Singh. "A Comparative Study of Data Structures for Punjabi Dictionary", 5th International Conference on Cognitive Systems, reviews & previews, ICCS'99, pp. 489-497, 2000.

[14] J. Daniel and H.M. James, "Speech and Language Processing", Prentice Hall, Englewood Cliffs, Inc., 2000.

[15] B. Horst. "A Fast Algorithm for Finding the Nearest Neighbor of a Word in a Dictionary", IAM-93-025, November 1993.

[16] The Online Encyclopedia of Writing Systems & Language, 16/12/2013 09:31, http://www.omniglot.com/writing/definition.htm.

[17] http://www.humnet.ucla.edu/humnet/aflang/Hausa/hausa.html.

[18] A. Mijiguin and H. Naroua. "Règles de formation des noms en haoussa", Actes de la conférence conjointe JEP-TALN-RECITAL 2012, Atelier TALAf 2012: Traitement Automatique des Langues Africaines, pp. 63-74, 2012.

[19] M.G. Maman and H.H. Seydou. "Les Langues de scolarisation dans l’enseignement fondamental en Afrique subsaharienne francophone : cas du Niger", Rapport d’étude pays, 2010.

[20] A.V. Van Der and D.S. Gilles-Maurice. " The African Languages on the Internet: Case Studies for Hausa, Somali, Lingala and isiXhosa", Cahiers Du Rifal, vol. 23, pp. 33–45, 2003.

[21] C. Bernard. "Les langues au Nigeria", Notre Librairie, Revue des littératures du Sud, Littératures du Nigéria et du Ghana, vol. 2, no. 141, pp. 8-15, 2000

[22] Arrêté N°0212 MEN/SP-CNRE du 19 oct. 1999 modifiant et complétant l'arrêté n°01/MEN/SCNRE/MJSC/MESR/M.INF/MDR/MI du 15 mars 1981 relatif à l'orthographe de la langue hausa.

[23] N. Ahmed. "Adaptation des écritures et de la lecture des langues étrangères au pays Haoussa de l’Afrique de l’Ouest", Synergies Algérie n°6 – 2009, pp. 61-69, 2009.

[24] C. Chanard and A. Popescu-Belis. "Encodage informatique multilingue : application au contexte du Niger", Les Cahiers du Rifal, No. 22, pp. 33-45, 2001.[33]Christophe D., "Apprendre à programmer, algorithmes et conception objet", 2e ed., Eyrolles, 2008.

[25] O. Don. "Les langues africaines a l’ère du numerique, défis et opportunités de l’informatisation des langues autochtones", Les Presses de l’Université Laval, CRDI 2011.

[26] G.P. Bargery. "A Hausa-English Dictionary and English-Hausa Vocabulary", Oxford University Press, London, 1934.

[27] M.N. Roxana and N. Paul. "The Hausa Lexicographic Tradition", Lexikos11, AFRILEX-reeks, series 11, pp. 263-286, 2001

[28] A. Minjinguini. "Dictionnaire élémentaire hausa-français", les éditions GG, 2003.

[29] N. Paul. "The Hausa Language An Encyclopedic Reference Grammar", Yale University Press, New Haven, 2000.

[30] D.S. Gilles-Maurice. "Web for/as Corpus: A Perspective for the African Languages", Nordic Journal of African Studies, vol. 11, No. 2, pp. 266-282, 2002.

[31] C. Enguehard and H. Naroua. "Evaluation of Virtual Keyboards for West-African Languages", Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08), Marrakech, Morocco, 28-30 May 2008 .

[32] C. Enguehard and C. Mbodj. "Des correcteurs orthographiques pour les langues africaines", Bulletin de Linguistique Appliquée et Générale, 2004.

[33] D. Christophe. "Apprendre à programmer, algorithmes et conception objet", 2e ed., Eyrolles, 2008.

[34] M. Brett, P. Gary and W. David. "Head First Object-Oriented Analysis and Design", O'Reilly, Nov. 2006.

[35] S. Robert and W. Kevin. "Algorithms", 4e ed., Addison Wisley, 2011

[36] OpenOffice.org 3.1 Developer's Guide, https://wiki.openoffice.org.