Allin Qillqay! A Free On-Line Web spell checking Service for Quechua

Richard A. Castro Mamani[1], Annette Rios Gonzales[2]

rcastro@hinantin.com, arios@ifi.uzh.ch

[1] Computer Science Department, Universidad Nacional de San Antonio Abad del Cuzco
[2] Institute of Computational Linguistics, University of Zurich

Abstract: *In this paper we analyze the advantages and disadvantages of porting the current available spell checking technologies in its primary form (meaning without speed and efficiency improvements) to the Internet in the form of Web services, taking the existing Quechua spell checkers as a case of study. For this purpose we used the CKEditor, a well-known HTML text processor and its spell-check-as-you-type (SCAYT) add-on on the client side. Furthermore, we built our own compatible server side application called "Allin Qillqay!" 'Correct Writing/Spelling!'.*

Key words: spellchecker traffic, spell checking parameters, HTML Editor, Quechua.

1 Introduction

This is a paper about the current spell-checking technologies and is based on two premises. "The first is that the Internet is becoming an increasingly important part of our lives" (The Mozilla Manifesto[1]). During the past few years, several new JavaScript applications have appeared that provide the user with functionalities on the web comparable to desktop programs. One of the main reasons behind this development is that the slow page requests every time a user interacts with a web application are gone; as the JavaScript engines are now sufficiently powerful to keep part of the processing on the client side [MacCaw2011].

Among the most well-known rich JavaScript *productivity applications*[2] figure iWork for iCloud[3] and, more recently, Microsoft Office 365[4], Google Docs[5], GMail[6] and also the CKEditor[7], a free open source HTML text editor which brings common word processor features directly to web pages.

Most of the web applications listed above are constantly being enhanced with new features, yet some important features, such as spell checking, have been neglected or are not integrated as web services but instead depend heavily on the web browser language configuration, or the spell checking plug-ins installed.

In 2012, we decided to implement our own productivity application using HTML, JavaScript and the state-of-the-art spell checking technology available for Quechua. The goal was to create an application with a user friendly interface similar to what users can expect from desktop applications. The integration of the spell checkers into the web application provides a comfortable and easy way to test the quality of the Quechua spelling correction.

[1] http://www.mozilla.org/en-US/about/manifesto/
[2] The term productivity software or productivity application refers to programs used to create or modify a document, image, audio or video clip.
[3] https://www.apple.com/iwork-for-icloud/
[4] http://office.microsoft.com/
[5] http://docs.google.com
[6] http://mail.google.com
[7] http://ckeditor.com/

The outline of this paper is as follows: Section 2 presents the basic concepts in spell checking. In section 3 we describe related work regarding the advancements in the field of on-line spell checking. Section 4 gives a general overview of the Quechua language family. Section 5 lists all the publicly available spell checkers for Quechua. The overall description of the system is given in section 6, and in section 7 we describe the some of the recent experiments and improvements.

2 Spell Checking

Liang [Liang2009] describes the overall spell checking task in computer science as follows: "Given some text, encoded by some encoding (such as ASCII, UNICODE, etc.), identify the words that are valid in some language, as well as the words that are invalid in the language (i.e. misspelled words) and, in that case, suggest one or more alternative words as the correct spelling".

The spell checking process can generally be divided into three steps (See *Figure 1*):

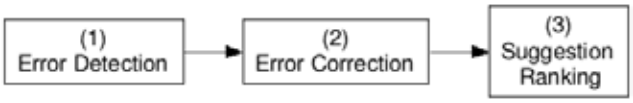

Figure 1. Spell checking process.

2.1. Error Detection

Error detection is a crucial task in spelling correction. In order to detect invalid words, the spell checker usually performs some kind of a dictionary lookup. There are three main formats for machine readable dictionaries used in spelling correction:

1. a list of fully-fledged word forms
2. a separate word (.dic) file and an affix (.aff) file
3. a data structure called 'finite state transducer' that comprehends the morphology of the language (i.e. the rules of word formation). This approach is generally used in spelling correction for languages with complex morphology, where one word (or root) may appear in thousands of different word forms, such as Quechua. As an illustration of Quechua word formation, see *Example 1* with the parts (i.e. morphemes)

contained in the Quechua word *ñaqch'aykuchkarqaykiñachum*[8]:

(1) *ñaqch'a -yku -chka -rqa -yki*

comb +Aff +Prog +Pst +1.Sg.Subj_2.Sg.Obj

-ña -chu -m

+Disc +Intr +DirE

Further details about finite state transducers applied to spell checking are not here presented for space reasons and can be consulted in Beesley & Karttunen [BeesleyKarttunen03].

2.2. Error Correction

There are two main approaches for the correction of misspelled words: isolated-word error correction or context-dependent error correction. With the former approach, each word is treated separately disregarding the context, whereas with the latter approach, the textual context of a word is taken into consideration as well.

Error Model produces the list of suggestions for a given misspelling, using different algorithms and strategies depending on the characteristics of the misspelled word.

A Typo is a small mistake in a typed or printed text.

A Real Word Error is an error which accidentally results in a valid word but it is not the intended word in sentence.

Only a context-dependent corrector can correct real-word errors, as the isolated-word approach will not detect this kind of mistake.

2.3. Suggestion Ranking

Ranking is the ordering of suggested corrections according to the likelihood that the suggestion is the originally intended word.

3 Related Work

In recent years there have been some advancements regarding online spell checking, mainly the incorporation of spell-check-as-you-type SCAYT technology, allowing users to have a much more responsive and natural experience. SCAYT is based purely on JavaScript and asynchronous requests to the server from its client applications.

It is not uncommon for a spell checker to start with a web application and then to get to the more traditional desktop version. Dembitz et al. [Dembitz2011] developed ***Hascheck***, an online spellchecker for Croatian, an under-resourced language with a relatively rich morphology which is spoken by approximately 4.5 million persons in Croatia. The dictionary used for this system is a list of fully-fledged word forms. What sets this spell checker apart from others is its ability to learn from the texts it spellchecks. With this approach they achieve a quality comparable to English spell checkers, as a consequence ***Hascheck*** was crucial during the development of other applications for NLP tasks.

Francom et al. [Hulden2013] developed ***jsft***, a free open-source JavaScript library which provides means to access finite-state machines. This API is used to build a spell checking dictionary on the client side of a web application obtaining good results. Although we did not use this API as part of the current version of our system, we believe that ***jsft*** is clearly a very important development in the evolution of spell-checking on the web.

WebSpellChecker[9] is a non-free spell checking service for a wide range of languages; it can be integrated in the form of a plug-in to the major open-source HTML text editors. WebSpellChecker was used as a model for *our* project, although ours is open-source and freely available.

4 Quechua

Quechua [Rios2011] is a language family spoken in the Andes by 8-10 million people in Peru, Bolivia, Ecuador, Southern Colombia and the North-West of Argentina. Although Quechua is often referred to as a language and its local varieties as dialects, Quechua is a language family, comparable in depth to the Romance or Slavic languages [AdelaarMuysken04, 168]. Mutual intelligibility, especially between speakers of distant 'dialects', is not always given. The spell checkers used in our experiments are designed for different Quechua varieties.

5 A case of study: Quechua Spell Checkers

When it comes to elaborating a spell checker, *Hunspell*[10] and *MySpell* are the most well-known technologies. Nevertheless, these formalisms have serious disadvantages concerning the suggestion quality for morphologically complex agglutinative languages such as Quechua. In order to overcome the problems of *HunSpell*, several spell checkers for agglutinative languages rely on finite-state methods, as these are better suited to capture complex word formation strategies. An example of such a finite-state spelling corrector is part of the *Voikko*[11] plugin for Finnish. The Quechua spell checkers used in our experiments make also use of this approach.

These are the spell checkers used in our web application:

- Cusco Quechua spell checker (3 vowels), implemented with the Foma Toolkit [Rios2011]. The orthography used as standard in this corrector adheres to the local Cusco dialect. We will use the abbreviation "**cuz_simple_foma**" to refer to this spell check engine.
- Normalized Southern Quechua spell checker, implemented in Foma as well. The orthography in this spell checker is the official writing standard in

[8] Abbreviations: +Aff: affective, +Prog: progressive, +Pst: past, Sg: singular, Obj: object, +Disc: discontinuative ('already'), +Intr: interrogative, DirE: direct evidentiality

[9] http://www.webspellchecker.net

[10] http://hunspell.sourceforge.net/

[11] http://voikko.puimula.org/

Peru and Bolivia[12], as proposed by the Peruvian linguist R. Cerrón Palomino [Cerrón-Palomino94]. We will use the abbreviation "**uni_simple_foma**" for this spell checker.

- Southern Unified Quechua, with an extended Spanish lexicon and a large set of correction rules. This spelling corrector is also implemented in Foma, and it uses the same orthography as **uni_simple_foma**. The Spanish lexicon permits the correction of loan words consisting of a Spanish root combined with Quechua suffixes. The additional set of rules, on the other hand, rewrites common spelling errors directly to the correct form. By this procedure, the quality of the suggestions improves considerably. We will use the abbreviation "**uni_extended_foma**" to refer to this spell checker.
- Bolivian Quechua spell checker (5 vowels) by Amos Batto, it was built using MySpell. In the following we will use the abbreviation "**bol_myspell**" for this spell checker.
- Ecuadorian Unified Kichwa (from Spanish, *Kichwa Ecuatoriano Unificado*) spell checker, implemented in Hunspell by Arno Teigseth. We will use the abbreviation "**ec_hunspell**" for this spell checker.

6 Our spell checking web service: Allin Qillqay!

The system[13] is an on-line spell checking service which offers a demo version of all the different spell checkers for Quechua that have been built so far in a user friendly HTML text editor. The system operates interactively, preserving the original formatting of the document that the user is proofreading. The most important advantage of online spell checking lies in the community of users (See *Figure 2*). Unlike conventional spell checking in a desktop environment, where the user-application relation is one-to-one, in on-line spell checking, there is a many-to-one relation. This circumstance has been beneficial for the enhancement of the spell checker dictionary: Unlike the user-defined customized dictionary in a desktop program, which stores the false positives[14] of only one user, all of the false positives that occur in on-line spell checking are stored in a single dictionary and thus benefit the entire community. Hence, our on-line spell checking service is constantly improving its functionality through interaction with the community of users.

6.1. Client side application

This section describes the different resources we use for the client side of the web service and how they interact with each other.

6.1.1 CKEditor

The CKEditor[15] is an open source HTML text editor designed to simplify web content creation. This program is a WYSIWYG[16] editor that brings common word processor features to web pages.

6.1.2 Dojo Toolkit

The Dojo Toolkit[17] is an Open-Source JavaScript library used for rapid development of robust, scalable, rich web projects and fast applications, among diverse browsers. It is dual licensed under the BSD and AFL license.

6.1.3 SpellCheckAsYouType (SCAYT) Plug-in

This *Spell Check As You Type* (SCAYT) plug-in[18] for the CKEditor, is implemented using the Dojo Toolkit JavaScript libraries. By default it provides only access to the spell checking web-services of **WebSpellChecker.net**[19].

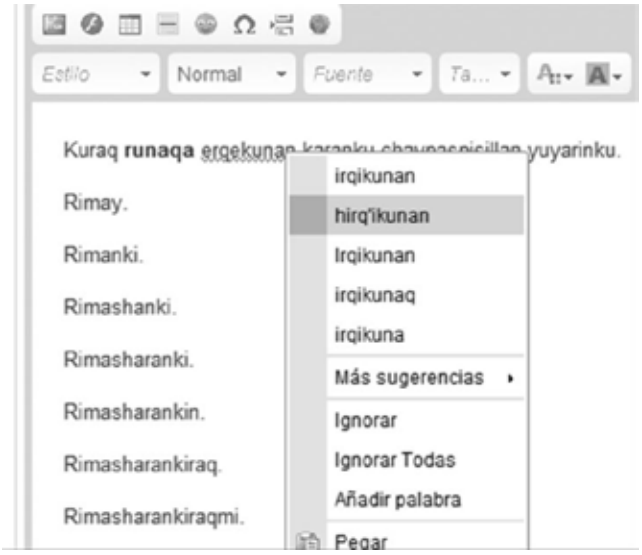

Figure 3. SCAYT working with our spell checking web service and the **cuz_simple_foma** spell-checking engine in the same manner as it works with WebSpellChecker.net service.

[12] There is one small difference: Bolivia uses the letter <j> to write /h/, whereas Peru uses <h>, e.g. Peru: *hatun* vs. Bolivia: *jatun* ('big').
[13] http://hinantin.com/spellchecker/
[14] A false positive refers to words that are correctly spelled, but unknown to the spell checker. In this case, the user can add those words to the dictionary.
[15] http://http://ckeditor.com
[16] What You See Is What You Get
[17] http://dojotoolkit.org
[18] http://ckeditor.com/addon/scayt
[19] http://www.webspellchecker.net/scayt.html

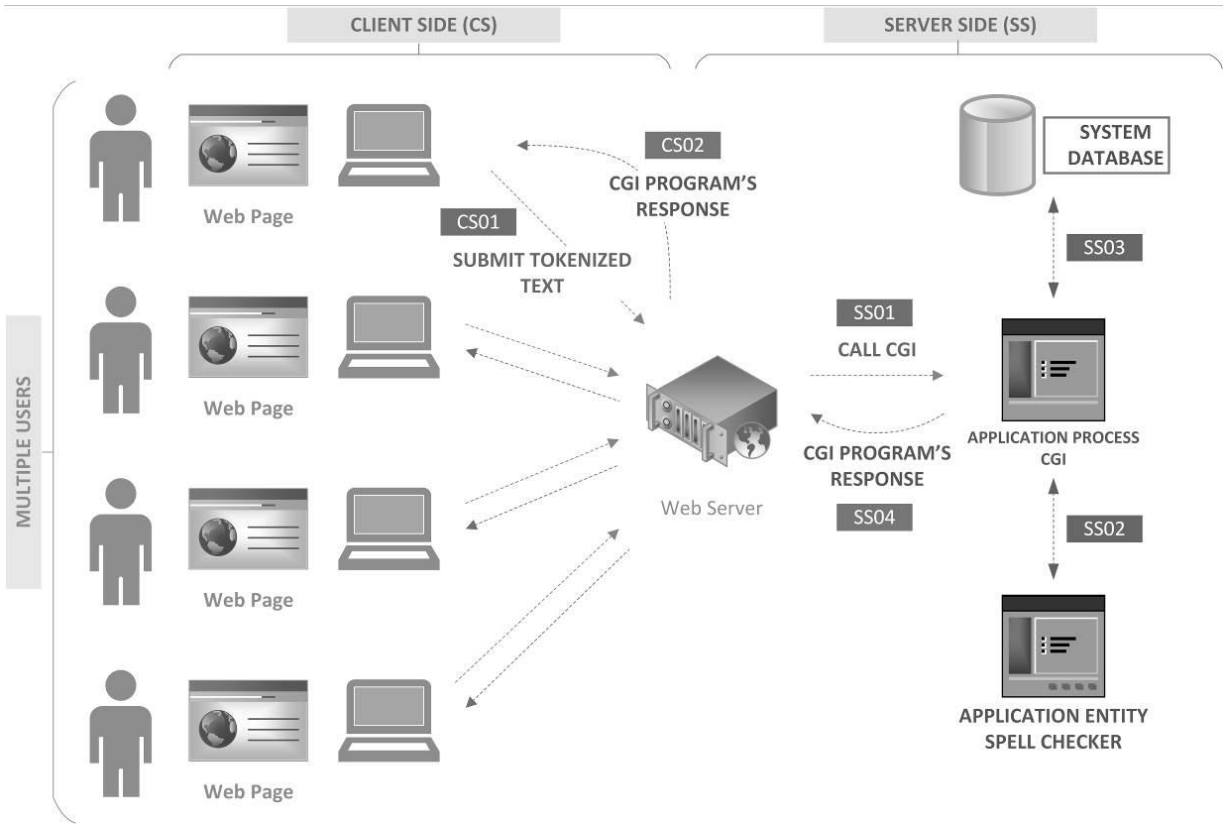

Figure 2. Online Web Spell checking Client/Server CGI System Diagram, every step is explained in section 6 (notice the codes **CS**** client side and **SS**** server side with their corresponding step number).

The SCAYT product allows users to see and correct misspellings while typing, the misspelled words are underlined. If a user right-clicks one of those underlined words he will be offered a list of suggestions to replace the word, see *Figure 3*. Furthermore, SCAYT allows the creation of custom user dictionaries. SCAYT is available as a plug-in for CKEditor, FCKEditor and TinyMCE. The plugin is compatible with the latest versions of Internet Explorer, Firefox, Chrome and Safari, but not with the Opera Browser.

6.1.4 Client Side Pipeline

The encoding used throughout the processing chain is UTF-8, since the Quechua alphabet contains non-ASCII characters.

CS01 Submitting tokenized text:

The tokenization process is done entirely by the SCAYT plug-in. For instance, if the original text written inside the CKEditor textbox is:

Kuraq runaqa erqekunan karanku chaypaspisillan yuyarinku.

The data submitted to our web server is the tokenized input text:

Kuraq, runaqa, erqekunan, karanku, chaypaspisillan, yuyarinku

Note that each word is separated by a comma and none of the format properties such as **Bold** or *Italic* are sent to the server. There are, however, other parameters that can be included in the data sent to the server, such as the language, the type of operation, or whether or not the word should be added to the user dictionary.

CS02 CGI program's response

JSON is the format of the response data from our CGI program:

{incorrect:[[“erqekunan”,[“irqikuna”]],

[“chaypaspisillan”,[“chaypaspasllam”,

“chaypaspasllas”, “chaypaspaslla”]]],

correct:[“Kuraq”,“runaqa”,“karanku”,“yuyarinku”]}

The response data is processed and rendered by the SCAYT plug-in.

6.2. Server side application

In summary, the server side implementation is an interface which interacts with the spell checkers for Quechua, as well as with the user dictionary and the error corpora, see Figure 4.

We developed a server side application that is compatible with the SCAYT add-on. The application makes it possible to use state-of-the-art spell checking software, such as finite-state transducers, in a web service. Our web server runs on a Linux Ubuntu Server 12.04 x64 operating system[20].

[20] We did not test our server-side application on a server running Windows Server.

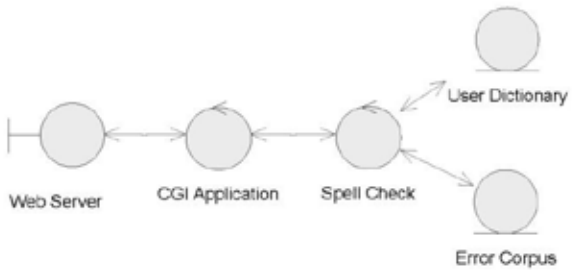

Figure 4. Server Side Application: This robustness diagram is a simplified version of the communication/collaboration between the entities of our system.

The user dictionary is stored in a *MySQL*[21] relational database. More specific information (classification, type, language, language variety, etc.) concerning misspellings and unknown words is stored in an object oriented database, in a XML format, using *Basex*[22].

6.2.1. Server side pipeline

SS01 Call CGI:

The web server calls and uses CGI as an interface to the programs that generate the spell-checking responses.

SS02 Spell check input terms:

The CGI program splits the comma separated words, and checks their correctness using the corresponding spell checker back end.

If a word is not recognized as correct, a request for suggestions is sent to the spell checking back end and the received suggestions are then included into the JSON response string.

We used two different approaches for the interaction with the spell checking back end:

The first approach consisted in a re-implementation of foma's *flookup* for the processing chain of the different finite state transducers used for spell checking in **uni_simple_foma**. This module can process text in batch mode, but it has to load the finite state transducers into the memory with every new call. As the finite state transducers, especially with the improved version **uni_extended_foma**, are quite large, loading those transducers takes a few seconds, which in turn makes the text editing through CKEditor noticeably slower.

For this reason, we implemented a TCP server-client back end for spell checking: the server loads the finite state transducers into memory at start up and can later be accessed through the client. As the transducers are already loaded, the response time is much quicker, see *Section 7.3*.

SS03 Save relevant data into the database:

The misspellings are saved in our MySQL database in the form of *custom user dictionaries* and a list of *incorrect terms* to be analyzed. More information about the misspellings is saved in our

XML Object Oriented Database in BaseX, since these misspellings will conform our *error corpus*.

Two linguists from the UNMSM[23] are currently analyzing and categorizing those misspellings according to the type of error, this information will be used as feedback to improve our spell-checking engines (lexicons, suggestion quality).

7 Experiments and Results

Evaluating Suggestion Accuracy from each Spell Checking Engine

In this section, we present a comparison between the different approaches used in the spell checking back end, and we hope to answer the following question: *Does finite-state spell checking with foma give more reliable suggestions than MySpell and HunSpell for an agglutinative language?*

Our online application makes it possible to group all the available spell checking engines in one place, which in turn allows for an easy comparison.

7.1.1 Minimum Edit Distance as a Metric for Spell Checking Suggestion Quality

We used the *Natural Language Toolkit*[24] (NLTK), publicly available software, to calculate the edit distance. *Suggestion Edit Rate* (SER) reports the ratio of the number of edits incurred to the total number of characters in the reference word; we used this toolkit for easy replicability of the tests we present here.

Misspelled term:

Rimasharankiraqchusina

Suggestions by **uni_simple_foma** (number of edits):

- Rimacharankiraqchusina (0.04)
- rimacharankiraqchusina (0.08)
- rimacharankiraqchusuna (0.13)
- Rimacharankiraqchusuna (0.08)
- rimacharankitaqchusina (0.13)
- Rimacharankitaqchusina (0.08)

Reference word:

Rimachkarqankiraqchusina

7.1.2 Evaluation of Spell Checkers using Minimum Edit Distance

Table 1 contains the suggestions produced by **uni_simple_foma**, **ec_hunspell** and **bol_myspell**.

[21] http://www.mysql.com/
[22] http://basex.org/
[23] Universidad Nacional Mayor de San Marcos
[24] http://www.nltk.org/

Table 1. Comparing the suggestions from each spell checker engine.

Misspelled Term (Cuzco Quechua)	Suggestions with its corresponding SER values		
	Southern Unified Quechua FOMA (**uni_simple_foma**)	Ecuadorian Kichwa Hunspell (**ec_hunspell**)	Bolivian Quechua MySpell (bol_myspell)
Rimay	.	.	.
Rimanki	.	.	.
Rimashanki	rimasqanki (0.18), Rimasqanki (0.09), rimachanki (0.18), Rimachanki (0.09), rimasqanku (0.27), rimasqanka (0.27)	Rimashkanki (0.09) 'expected', Rimashkani (0.18), Imashinashi (0.55)	.
Rimasharanki	rimacharanki (0.14), Rimacharanki (0.07), rimacharanku (0.21), rimacharankis (0.21), rimacharankim (0.21), Rimacharanku (0.14)	Imashashunchik (0.57)	Rimasharqanki (0.08) 'expected', Rimashawanki (0.08), Khashkarimunki (0.54), Kimsancharinki (0.38), Rimarichinki (0.54), Rankhayarimunki (0.69)
Rimasharankin	rimacharanki (0.2), rimacharankis (0.2), rimacharankim (0.2), Rimacharanki (0.13), Rimacharankis (0.13), Rimacharankim (0.13)	Imashashunchik (0.57)	Kimsancharin (0.47), Rankhayarin (0.47)
Rimasharankiraq	rimacharankiraq (0.12), Rimacharankiraq (0.06), rimacharankitaq (0.18), Rimacharankitaq (0.12), rimacharankuraq (0.18), Rimacharankuraq (0.12)	Imashashunchik (0.71)	Rankhayarimuy (0.63)
Rimasharankiraqmi	Rimacharankiraqmi (0.05), rimacharankiraqmi (0.11), rimacharankiraqmá (0.21), Rimacharankiraqmá (0.16), rimacharankiraqsi (0.16), rimacharankiraqri (0.16)	Imashashunchik (0.86)	Marankiru (0.56)
Rimasharankiraqchu	Rimacharankiraqchu (0.05), rimacharankiraqchu (0.1), rimacharankiraqchá (0.2), rimacharankiraqchus (0.15), rimacharankiraqcha (0.15), rimacharankiraqchum (0.15)	Imashashunchik (0.86)	Charancharimuychu (0.63)
Rimasharankiraqchusina	Rimacharankiraqchusina (0.04), rimacharankiraqchusina (0.08), rimacharankiraqchusuna (0.13), Rimacharankiraqchusuna (0.08), rimacharankitaqchusina (0.13), Rimacharankitaqchusina (0.08)	Imashashunchik (1)	Wariwiraqocharunasina (0.61)

The first column of *Table 1* contains the word forms for testing, taken from Paredes-Cusi [Paredes-Cusi2009]. All of these words have the same root (*rima-* 'to speak') which is a highly used word across dialects and is contained in the lexicons of each spell checking engines we presented in *Section 5*. The test words are written in the standard proposed by the AMLQ[25], and are spelled correctly according to the **cuz_simple_foma** spell checking engine.

The columns on the right contain the suggestions provided by the spell checking engines **uni_simple_foma**, **ec_hunspell**, **bol_myspell**. Note that the dot "." sign in a cell stands for a correct word, additionally we provided the SER value for each suggestion and we signal if the suggestion is correct by pointing it out with the '*expected*' flag and by highlighting it, otherwise, we do not signal anything.

A glance at the suggestions by **ec_hunspell** and **bol_myspell** reveals that the quality varies according to the complexity of the word: the more suffixes the misspelled word has, the less adequate and more distorted the suggestions become (see *Table 1*).

The suggestions offered by each one of the spell checker engines, especially by Ecuadorian Kichwa and Bolivian Quechua do not cope adequately with the rich morphology of this language, as some of their suggestions do not even share the same root as the misspellings in the first column of *Table 1*.

[25] Academia Mayor de la Lengua Quechua in Cusco.

Suggestion Edit Rate (SER) measures the amount of editing that a human would have to perform to change a system output (a spell checking suggestion) so it exactly matches a reference word. We calculated the value using *equation 2*.

$$SER = \frac{edit\_distance(original, suggestion)}{length(expected)} \quad (2)$$

Where *original* is the word to be spell checked, *suggestion* is the output from the spell checking engine and *expected* is the referenced word.

In *Figure 5* we present SER values for each misspelling (we calculated the average SER value when there are more than one suggestion), if the quality of the suggestion are good the SER value ought to be low, otherwise a high one. It becomes evident that the quality of the suggestions by **ec_hunspell** and **bol_myspell**, *Hunspell* and *MySpell* respectively are poor, because they do not cope well with complex words.

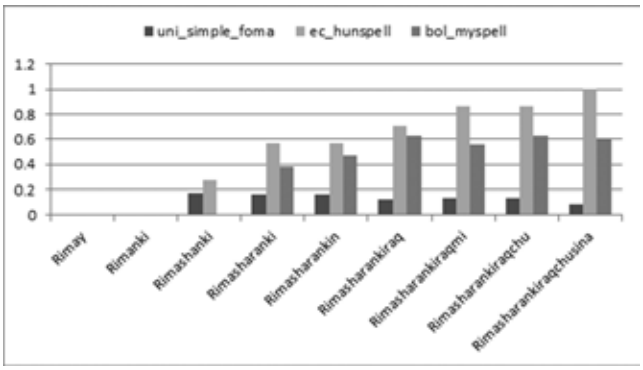

Figure 5. Suggestion Edit Rate.

7.2. Improving (Error Model) spell checking quality

7.2.1. Improving Spell Checking Suggestion Quality

The misspelled morpheme in the test words (*rimasha-*) in *Table 1* is the suffix *-sha*, that should be spelled *-chka* in the unified standard. The Edit Distance between *sha* and *chka* is 2 (delete *k*, substitute *s* with *c*). As the spell checker **uni_simple_foma** relies on Minimum Edit Distance as the only error metric, it will first suggest Quechua words with a smaller edit distance, e.g. with the suffixes *-sqa* or *-cha* (edit distance to *-sha* is 1).

From the results in *Table 1* it becomes clear that using edit distance as the only algorithm to find the correct suggestions is not good enough. For this reason, we built the improved version of the spell checker **uni_extended_foma**: This back end uses several cascaded finite state transducers that employ a set of rewrite rules to produce more useful suggestions. For instance, the suffix *-sha* will be rewritten to the corresponding form in the standard, *-chka*. Furthermore, we included a Spanish lexicon of nouns/adjectives and verbs into the spell checker. This allows the correction of words with Spanish roots and Quechua suffixes (very frequent in Quechua texts)[26].

Table 2 illustrates the quality of the suggestions with this improved approach, the results are encouraging as SER values are low, see *Figure 6*, this results compared with its counterparts are much better, see *Figure 7*.

Moreover **uni_extended_foma** presents us with the correct alternatives for every test word (see *Table 2*).

Table 2. The suggestions and SER values provided by **uni_extended_foma**.

Term (Cuzco Quechua)	uni_extended_foma
Rimay	.
Rimanki	.
Rimashanki	rimachkanki (0.27) '*expected*'
Rimasharanki	rimachkarqanki (0.29) '*expected*'
Rimasharankin	rimachkarqankim (0.33) '*expected*'
Rimasharankiraq	rimachkarqankiraq (0.24) '*expected*'
Rimasharankiraqmi	rimachkarqankiraqmi (0.21) '*expected*'
Rimasharankiraqchu	rimachkarqankiraqchu (0.20) '*expected*'
Rimasharankiraqchusina	rimachkarqankiraqchusina (0.17) '*expected*'

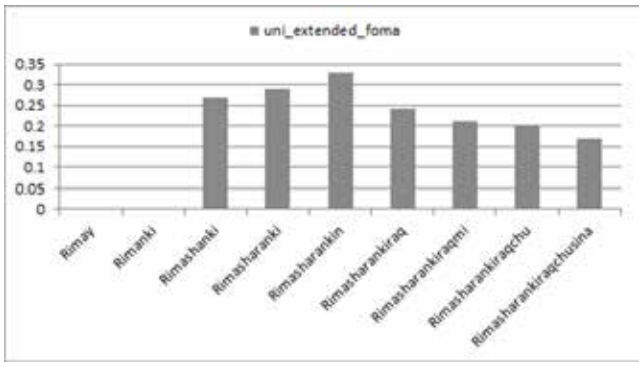

Figure 6. Graphical interpretation of the SER values for the suggestions provided by **uni_extended_foma**

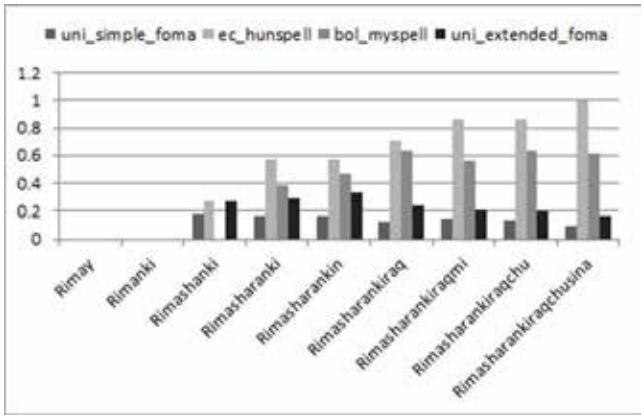

Figure 7. Suggestion Edit Rate for **uni_extended_foma** in contrast with the others.

[26]The Spanish lexicon has been built with part of FreeLing, an open source library for language processing, see http://nlp.lsi.upc.edu/freeling/

7.3. Improving CGI Program's Speed Response

The first implementation of our application (see Section 6.2.1, *SS02: Spell check input terms*) was fast enough for the web service, the lookup tool could load the spell checker consisting of only one transducer of approximately 2MB very quickly.

However, this is not the case for the cascaded transducers of the improved version **uni_extended_foma**, for which the same lookup takes 40 to 60 seconds for a group of 6 to 10 words. This results in a deficient and slow web application. In order to overcome the slow response with the extended spell checker, we re-implemented the lookup module as a TCP server-client application.

We measured the time with both approaches on our server for a single word. The standard lookup took 4.434 seconds, whereas with the TCP server-client, the lookup took only 0.021 seconds.

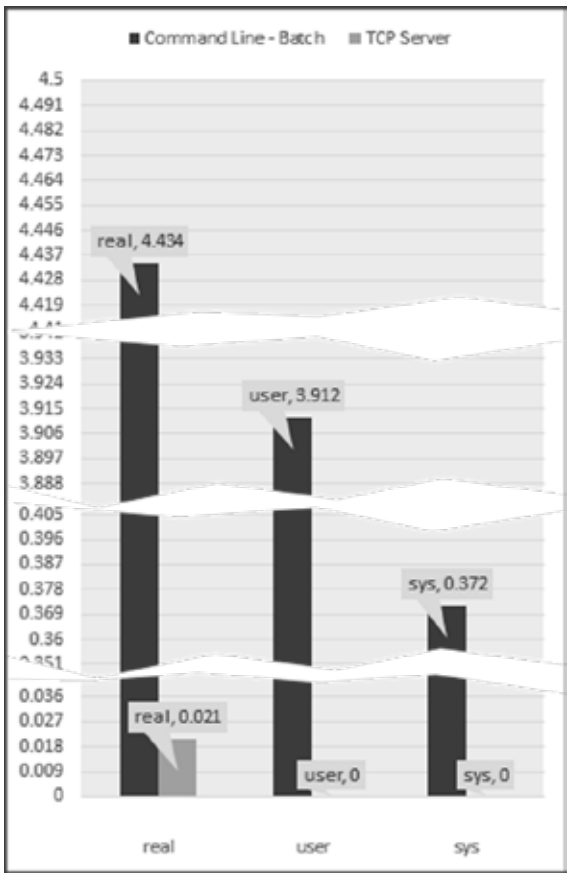

Figure 8. Speed response (measured in seconds) comparison between the two implementations *Command Line - Batch Mode* and *TCP Server*.

As illustrated in *Figure 8*, the response time of the TCP service is 0.021 seconds, as compared to 4.434 seconds with the regular lookup. Using the TCP sever-client thus solves the problem for the web service.

8 Conclusions and Future Work

We integrated existing spell checkers for Quechua into an easy to use web application with functionalities comparable to a desktop program. Furthermore, we improved the spell checker back end by using a more fine-grained set of rules to predict the correct suggestion for a given word form.

Additionally, we implemented a TCP server-client lookup for finite state transducers written in Foma, in order to mitigate the low response time for the enhanced spell checker.

In order to further improve our spell checker, we collect the unknown words from the web service in an error corpus, which gives us an indication for missing lexicon entries or missing morpheme combinations.

The Foma spell checkers described in this paper are already available as plug-ins to OpenOffice and LibreOffice, and we are currently working on a version for MS Office programs.

References

[AdelaarMuysken04] Adelaar, W. F. H. and Muysken, P. (2004). The Languages of the Andes. Cambridge Language Surveys. Cambridge University Press.

[BeesleyKarttunen03] Beesley, K. R. and Karttunen, L. (2003). Finite-state morphology: Xerox tools and techniques. CSLI, Stanford.

[Cerrón-Palomino94] Cerrón-Palomino, R. (1994). Quechua sureño, diccionario unificado quechua-castellano, castellano-quechua. Biblioteca Nacional del Perú, Lima.

[Dembitz2011] Dembitz, v., Randić, M., and Gledec, G. (2011). Advantages of online spellchecking: a Croatian example. Software: Practice and Experience, 41(11):1203–1231.

[Hulden2013] Hulden, M., Silfverberg, M., and Francom, J. (2013). Finite state applications with Javascript. In Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013); Linköping Electronic Conference Proceedings, volume 85, pages 441–446.

[Liang2009] Liang, H. L. (2009). Spell checkers and correctors: a unified treatment. Master's thesis, Universiteit van Pretoria.

[MacCaw2011] MacCaw, A. (2011). JavaScript Web Applications. O'Reilly Media, Inc.

[Paredes-Cusi2009] Paredes Cusi, B. (2009). Qheswa Simi, Lengua Quechua. Editorial Pantigozo, Cuzco, Perú.

[Rios2011] Rios, A. (2011). Spell checking an agglutinative language: Quechua. In Proceedings of the 5th Language and Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics, Poznań, Poland.