

proofing-library

Table of contents

1 A new platform for proofing tools.....	3
2 Functionality areas.....	3
2.1 Observation.....	4
2.2 Assumption.....	4
3 Functionality areas grouped.....	4
3.1 Post-error correction.....	4
3.2 Editing aids (error-prevention tools).....	4
3.3 Infrastructure tools.....	4
3.4 Translation aids	5
4 Speller details.....	5
4.1 User configurable speller.....	5
4.2 Suggestions sorted according to lexeme frequency.....	5
4.3 Suggestions sorted according to inflection frequency.....	6
4.4 Arbitrarily complex replace rules for suggestions.....	6
4.5 Rejection nets.....	6
4.6 User dictionary as transducer.....	6
4.7 Short context disambiguation for suggestion filtering.....	6
4.8 Best N matches.....	7
4.9 Compound border heuristics.....	7
4.10 Compound analysis.....	7
4.11 Abbreviation compounds.....	7
5 OCR correction.....	7
6 Proofing library design.....	7
6.1 Design goal.....	9
7 Links.....	9

8 Other notes..... 10
8.1 Clarin..... 10

1 A new platform for proofing tools

The starting point is the transducer work done at Helsinki university, especially weighted transducers. The short-term plan is to make a speller engine using a transducer lexicon composed with a number of weighted replacement rules to give suggestions.

Expanding on this idea, there are a number of different possibilities. The purpose of this document is to lay out these options, and try to categorise them to guide the future work on proofing tools.

The work on [HFST](#) and [Foma](#) creates a unique opportunity to start over from scratch, and set a new standard for what proofing and linguistically founded editing tools could do. We need to ensure that we design a library that can grow with future development and technical innovations.

2 Functionality areas

This is a random list of areas of functionality that would be possible based on the linguistic processing library & API we're presenting:

- non-word detection (real typos)
- real-word spelling errors (typos that happen to be a real word, undetectable by standard spelling checkers)
- statistical spelling correction (an unknown word occurring several times is likely to be correct, and a similar but not identical unknown word is probably a misspelling of that unknown word)
- intelligent user dictionaries (user dictionaries that are directly compilable into an automaton, with proper inflection and morphophonology; using heuristics and disambiguation to guess POS and inflection class, suggested to the user when added)
- intelligent suggestions (remove impossible suggestions based on context) - target: only one suggestion, making it realistic to run (semi)automatic proofing
- grammar checking (syntactic errors like agreement conflicts, compound splitting, wrong argument case, etc)
- stylistic checking (use of abbreviations, punctuation, other formal elements in a text)
- dialectal/variation control (use only a subset of the available forms, based on user preferences - control the usage, detect errors)
- "linguistic editing" - manipulate linguistic entities in the editing process (swap coordination arguments, change person/number of subject and make the whole sentence automatically update, change the dialect from one variant to another coherently throughout the text, etc.)
- text prediction
- automatic word completion (up until the first splitting point, or returning a list of candidates matching the preceding context)
- terminology -- checking against a parallel document, and assure consistent translation of the terms in the originating language

- simultaneous MT - the document is translated as you write it, with the translated text showing up in another document in a separate window which is still synchronised with the original text

2.1 Observation

The present division of functionality and access to the text, which is governed by arbitrary (and old) API decisions, hinder development of new functionality. We now have a unique chance to start over, and we should take it.

2.2 Assumption

We assume that we can have access to the text of the whole document, and also information about basic text structure. In environments with less capabilities, we should downgrade gracefully (i.e. provide less functionality in a controlled manner), but still try to infer as much as possible about the text structure based on the available information.

Access to the whole document provides opportunities for new services to the user. Style and word-form alternatives can be deduced based on the whole document, and provide the basis for checks in a local context, unknown words can be spell-checked, topic identification can be used to filter suggestions, etc.

3 Functionality areas grouped

3.1 Post-error correction

Functionality designed to detect and correct errors after they have been made:

- non-word detection and correction
- real-word spelling error detection and correction
- grammar-error detection and correction
- statistical spelling correction of unknown words
- word-form variation check and correction

3.2 Editing aids (error-prevention tools)

- text prediction
- automatic word completion
- linguistic editing
- word-form changes (am. english -> brit. english)
- style-level changes
- terminology selection (present appropriate alternatives based on earlier text)

3.3 Infrastructure tools

- intelligent user dictionary

3.4 Translation aids

- terminology -- checking against a parallel document, and assure consistent translation of the same terms in the originating language

4 Speller details

Starting with the basics - the speller - here's a list of things to consider when remaking the proofing experience. The list is based on the assumption that we are using weighted transducers to implement spellers.

4.1 User configurable speller

The speller should be user configurable, with reasonable built-in defaults. Since many or most of the relevant settings are language specific, the settable options have to be tied to the lexicon, ie transducer. Here are a couple of use cases:

4.1.1 Morphologically rare forms as optional

In north Sámi, possessive endings are used rather infrequently. Since they resembles other forms of the nouns, but with a different consonant gradation, possessive endings tend to mask real spelling errors of regular case forms. It is therefore desirable for a subset of the users to be able to turn the possessive endings off. There would likely be similar cases in other languages.

4.1.2 Free compounding could be limited

Even though compounding is free, for some groups of writers it can be problematic (dyslectics, language learners, ...), and it would be beneficial for such users to be able to turn off compounding, or limit it to a certain number of words.

4.1.3 Configuration summary

If the speller transducer is a regular, weighted transducer giving analyses on the upper side, then one possibility could be to dynamically change the weight of some arcs (tags) based on a settings file, i.e. change the weight of +Px tags and +Compound tags from 0 to 1, or apply a filter restricting the number of +Compound tags (you get one such tag each time you pass the dynamic compound point, i.e. dynamic word boundary, as opposed to lexicalised word boundaries).

4.2 Suggestions sorted according to lexeme frequency

The ordering of correction suggestions is of course governed by many different factors, but given otherwise equally conditions, the suggestions should be ordered according to corpus frequency. This can be implemented by giving each word a small weight based on said frequency.

4.3 Suggestions sorted according to inflection frequency

Independently of lexeme frequency, specific word forms or inflections also have different frequency patterns. Given two otherwise equal suggestions, the one with the most frequent inflectional ending should be suggested first.

4.4 Arbitrarily complex replace rules for suggestions

- weighted two-level rules
- weighted regular expressions
- typo -> correction lists

4.5 Rejection nets

Subtracting unwanted patterns or paths from a transducer is simple, but sometimes such subtractions are very expensive, either in terms of consumed memory, processing, or size of the resulting net. An example would be to subtract a specific compound from an otherwise productive pattern - since the pattern is productive for all other combinations of first and second parts, the only possibility to subtract it would be to expand the net to spell out all combinations except the one being subtracted, resulting in a substantially bigger file size.

To avoid such situations, a rejection net could be used instead - any string in that net should be rejected. By consulting this net before the regular accepting net, the effect would be the same as subtracting the rejection net from the accepting net.

4.6 User dictionary as transducer

For morphologically rich languages, the regular word list approach to user dictionaries is less satisfying. The correct thing would of course be a transducer - or rather a text file that can (and will) be compiled into a transducer at runtime. A user rejection dictionary should be made according to the same principle.

In both cases inflection templates should be included, and heuristics could be used to guide the user to the correct inflection (and possibly compounding) class. This could even give the extra bonus of proper analysis in a grammar checking / parsing context, since the inflection class would automatically provide the corresponding tags upon analysis.

4.7 Short context disambiguation for suggestion filtering

Several proofing API's provide access to the nearby context of the word being corrected. This could be used to disambiguate the suggestions such that only those matching the context would survive. What technology to use for this disambiguation should be open, and it could even be possible to use a combination of technologies, such as Constraint Grammar + trigram patterns.

4.8 Best N matches

With many and/or complex error correction rules, the search space can easily grow out of control. To avoid this, the weighted transducer should be searched in a best-first approach, with a set-able cut-off point (number and time limited, as in: the best 10 suggestions within 1 second).

4.9 Compound border heuristics

Another problem area is correcting compounds with multiple misspellings in each part of the compound. To reduce the search space and response time, one could use a language-specific model of word boundary patterns, to identify possible word boundaries in the input string. Then each compound candidate could be corrected alone, and the correction candidates for each part recompounded (and checked) before being returned to the user.

4.10 Compound analysis

Only accept two-part compounds, but register or lexicalise all such compounds, such that new compounds formed by known compounds would be accepted. This would reduce over-generation tremendously, and still allow free compounding.

4.11 Abbreviation compounds

Use compound heuristics to find the non-abbr part, and suggest lexicalisation of the abbr.

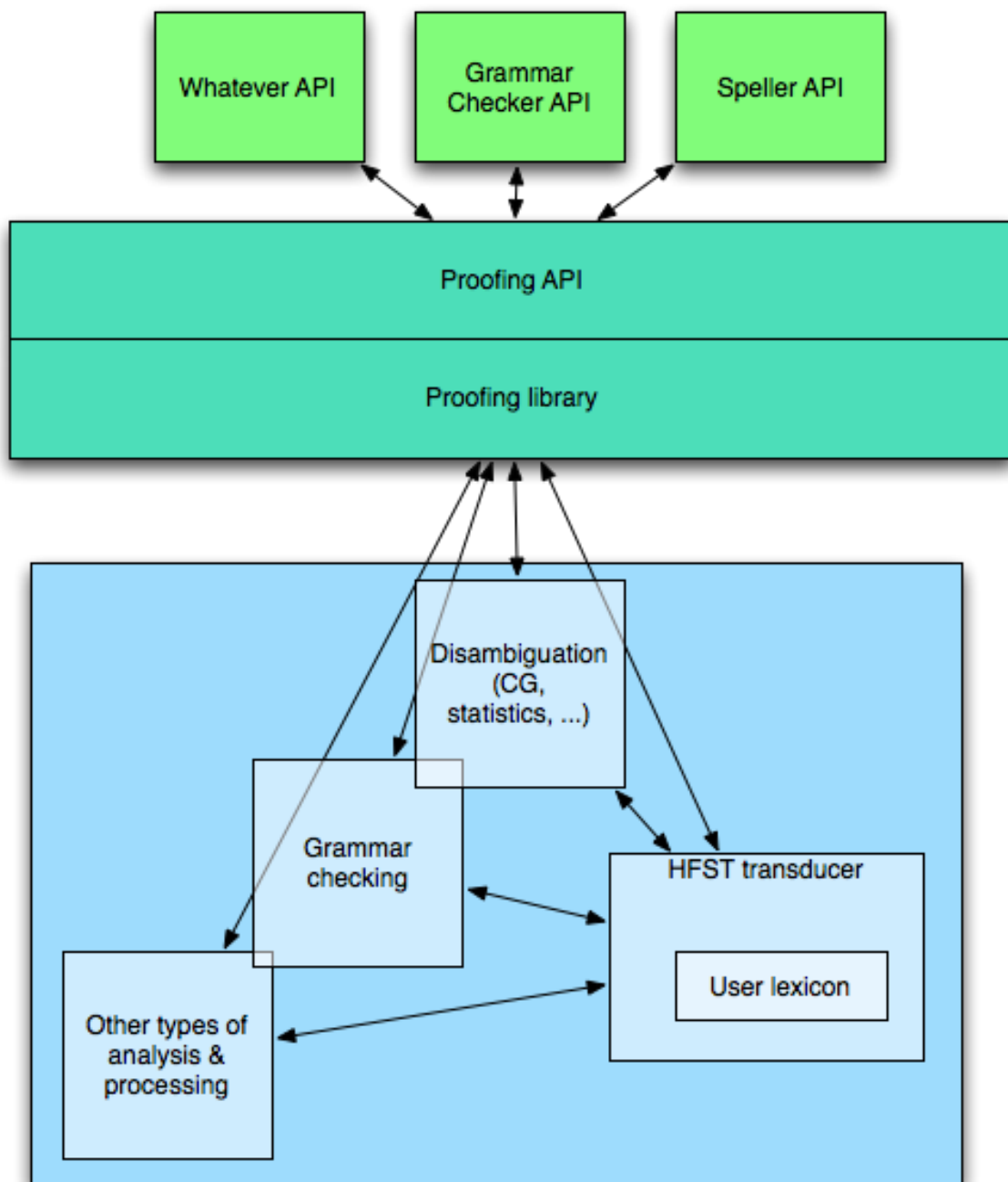
5 OCR correction

Automatised correction of OCR-scanned text (cf Google books, Finnish literature bank, Nasjonalbiblioteket, etc). St. Michels - there is always some OCR errors left, which resembles human spelling errors. Using the document-oriented approach, the quality of automatic correction could be high enough to be interesting.

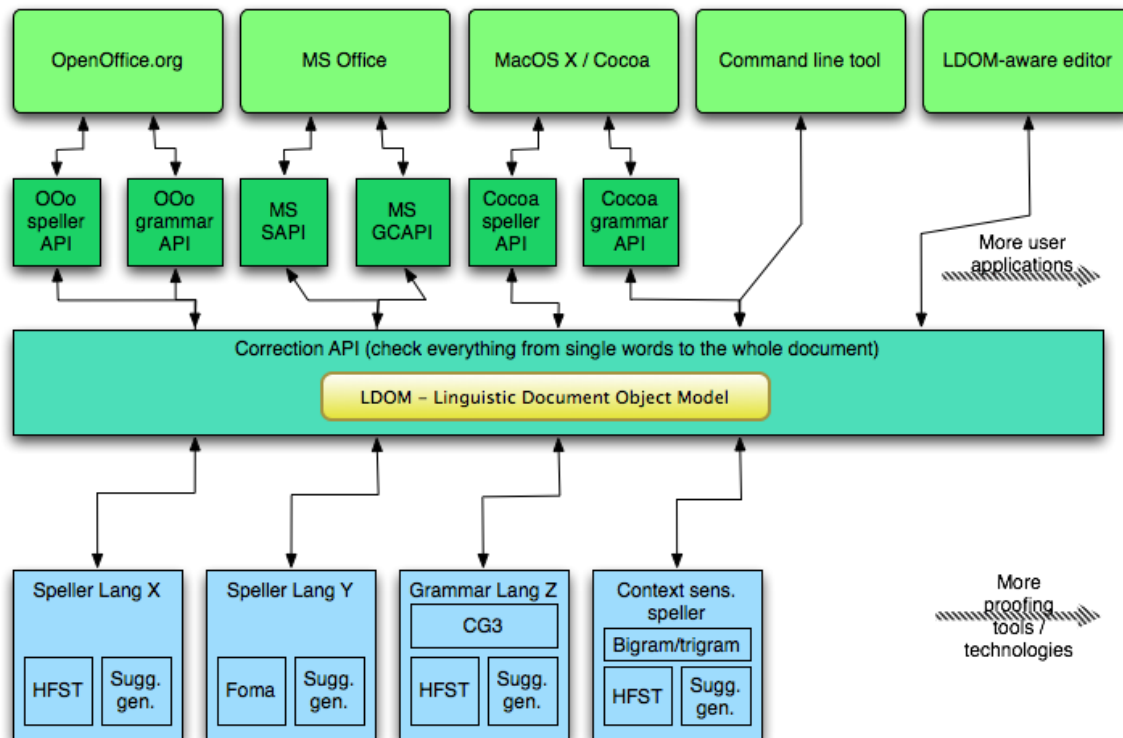
6 Proofing library design

- modular
- expandable
- somewhat technology-agnostic, although using the HFST transducers at the core / for lexically based processing
- should be able to process a whole document (cf intelligent treatment of unknown words)
- a defined way to communicate between the modules

Here's a graphical sketch of the design - the end user applications are at the top, the linguistic modules at the bottom:



A newer version of the same ideas:



The **LDOM** part is meant to be a full representation of the textual content of the document, automatically kept in sync with the "real" document. The LDOM automatically gives a very basic linguistic structure of the whole document, which can be expanded and enriched by the available linguistic tools for each language. Further details [on a separate page](#).

6.1 Design goal

The library should be multithreading. This is needed to allow slower, full-text analysis to take place at the same time as local processing around the editing point can happen "immediately" (as perceived by the user).

There could also be several documents or applications accessing the library at any single time. The library needs to be able to keep track of the different documents and the corresponding linguistic processing.

7 Links

Some links to papers on improving error detection and suggestion quality, as well as real-word error detection:

- Eckhard Bick: A Constraint Grammar Based Spellchecker for Danish with a Special Focus on Dyslexics - http://www.ling.helsinki.fi/sky/julkaisut/SKY2006_1/1.6.1.%20BICK.pdf

- Amber Wilcox-O’Hearn, Graeme Hirst, and Alexander Budanitsky: Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model - <http://ftp.cs.toronto.edu/pub/gh/WilcoxOHearn-etal-2008-poster.pdf>
- Eneko Agirre, Koldo Gojenola, Kepa Sarasola & Atro Voutilainen: Towards a single proposal in spelling correction - http://ixa.si.ehu.es/Ixa/Argitalpenak/Barne_txostenak/1001000786/publikoak/TR-8-98.ps

8 Other notes

8.1 Clarin

Samisk korpus lagra i eit Clarin-senter. Den lette vegen å gje korpuset til forskningssamfunnet. CSC er eit slikt datasenter i Finland.