

# Proofing - what it should have been

## Table of contents

1 Background - the problem.....	2
2 Terms.....	2
3 A better proofing framework.....	3

## 1 Background - the problem

What is proofing all about, and how is it done in different systems today?

The essence of proofing is to find errors in a text, and be able to correct them – as many errors as possible, irrespective of type. With the help of computers we should be able to do it faster, easier and better than before. But in most cases, what we are given is a very simplistic tool that looks at one word at a time, and evaluates it without any knowledge of the context. And for most such systems, they are even without any knowledge about the grammar of the words themselves, and thus unable to recognise, analyze and correct new words formed by regular grammatical patterns such as compounding and derivation.

If one would like to have a proofing system that also looks at the context, maybe even the whole document, the options are few and limited, and certainly expensive – so-called grammar-checkers aside. Most, if not all systems that proofread the whole document and use the whole document when checking smaller units, are targeted at professional, large-scale users. There are historical reasons for this, but they are gone years ago. What is surprising, is that the open-source community hasn't been able to come up with even something remotely near a decent, cross-platform, linguistically advanced proofing solution yet. The dominance of English, and the simple requirements for an acceptable, word-level speller is one obvious explanation, but far from the only one.

This document is an attempt at setting a goal for what should be, and will be accompanied with sketches for how to achieve those goals.

## 2 Terms

The following terms are used throughout this document with the meaning defined below:

### **speller**

*traditionally:* a ...

*here:* a ...

### **proofing levels**

this document operates with the following three proofing levels:

#### **word**

the word level proofing is close to (and can be identical to) the traditional speller, correcting one word at a time (interactively or as a batch process). A *word* is here a linguistic unit, and can span whitespace characters. Whether a given proofing engine supports whitespace in words or not is set by the plugin for that engine. A proofing engine that does *not* support whitespace characters in words, the behaviour will be identical to (most) traditional spellers. — For proofing engines that *do* support whitespace characters in words, there will be a window of negotiable size containing strings to be considered for the engine. For practical (speed) reasons, there needs to be a limit to how big this window can be. The upper limit of this window will define the upper limit of the number of single string elements that can be part of a *word*. Setting the window size to "1" will

effectively turn the word-level proofing into a standard speller checking only single strings, identical to the engines that do *not* support words containing whitespace.

**paragraph**

a section of text delimited by carriage returns or similar characters (the class of linebrake characters).

**document**

the whole document, or, in the case of applications supporting multiple, independent text flows, the whole text flow including linked headers etc.

### 3 A better proofing framework

The following list defines the basic goals of a redesigned proofing framework:

**scalability**

it should scale well from traditional, word-based (single-string-based) spellers to document proofers with document repository backends

**level negotiation**

starting from the document level, the framework should negotiate between the application frontend and the proofing backend to find the maximum common level of proofing - the highest level that both support. It should be possible for an end user to override the negotiation result, to set a lower level proofing when desired.

**open source**

the framework itself needs to be open source, and all APIs should be open and well defined

**pluggable backends**

the framework itself does not provide any proofing facilities, it only intermediates between hosting applications and systems on one side, and proofing engines (installed by the user) on the other; for this to work, the framework needs to have a plug-in architecture for proofing engines

**pluggable frontends**

it is as well a goal to provide a uniform interface to the text to be proofed and the application/system hosting it; hopefully applications and systems will be written to directly support the API of the framework in the future — in the meantime, the framework should provide a front-end plug-in architecture that allows developers to write interfaces to specific applications and systems (without such interfaces, the framework would be useless for all applications not supporting its native front-end API)

**multilingual**

the world is multilingual, and so should all decent proofing APIs be; at best, each language/proofing engine pair should be running in a thread/process of their own, both for security and efficiency reasons. One problem is that overall, document analysis in a multilingual document might need interaction between completely different engines. One can just define the problem away, and state that all documents

have one main language, for which document analysis could be made available, and the other languages are just lower-level fragments (maximally paragraphs?). Or one could define a minimal interaction/inter-communication language to be used between engines of different providers/languages.

**Unicode-based**

this is merely a technical requirement to support the previous goal

**linguistically motivated levels**

why is it that spellers almost invariantly stick to a definition of "word" that assumes it is (basically) the same as a space-delimited string? From a user point of view, what is interesting is linguistic units roughly like lexemes, that is: *på folkemunne* (ADV), *United Nations* (N) - linguistic units that can't be meaningfully further broken up for syntactic analysis.