Translation Quality Checking in LanguageTool

Marcin Miłkowski

Abstract: In large computer-aided translation (CAT) projects, especially in software localization, one of the main problems is to maintain the consistent style of the translated text. To tackle this problem, translators have to follow different guidelines defined in style guides for different translation jobs. Yet, in the case of conflicting guidelines (for example, terminological) for various projects it is very easy to make mistakes, and quite hard to find them because they are neither obvious nor glaring errors. Automated translation quality assessment (QA), on the other hand, are usually quite costly compared to other CAT tools and/or do not have any comprehensive natural-language processing features. Usually, their use is not really beneficial for languages other than English. Because of that, the proofreading process is costly and time-consuming, or the translation quality is negatively impacted.
In this paper, I will present the translation QA features available in LanguageTool, an open-source proofreading tool (Miłkowski 2010). LanguageTool currently (as of version 1.4 released on June, 26, 2011) supports 22 languages and is able to use the standard target language rules to check for the mistakes in the translated text, including false friends in translation, as well as specially designed translation QA rules. These rules may be specially crafted to conform to informal style guides and include the most frequent mistakes found by human proofreaders (using the method specified in Miłkowski, forthcoming). I will show some examples of such bilingual rules. It is hoped that thanks to XLIFF standard support the proofreading tool will be easily introduced to the standard translation QA workflow of translation agencies and individual translators.

Keywords: computer-aided translation, quality assessment, grammar checking.

1. Quality Assessments in Translation Jobs

In the real translation world, especially in the field of technical, business and legal translation, translators have to follow different style guides.[1] While they

[1] For example, see the style guide for Polish EU translators at http://ec.europa.eu/translation/polish/guidelines/documents/styleguide_polish_dgt_pl.pdf. In software localization, Microsoft style guides are the state-of-the-art documents for most languages: http://www.microsoft.com/Language/en-us/StyleGuides.aspx.

might be of great help to maintain the terminological and stylistic consistency of the resulting translation, they also contribute to problems for individual translators. This is especially the case when the advice offered by a style guide differs from the widely accepted language use. For example, Microsoft recommends "strona sieci web" as the translation for "web page" (thereby making "web" a proper name, left in the English form in the Polish text), whereas the more popular version is "strona internetowa". Such specific and often conflicting requirements make quality assessment harder, as the proof editors may simply not see that there is a non-standard translation of a term.

Another problem stems from translating technical text in a tagged form, which is usual for technical applications, such as software localization, web pages, help pages, DTP-ready text. Translators have to be careful to preserve the format (and other) tags from the original in the translation. For example, formatting tags abound in software strings ("%s" etc.), and they need to be preserved. Similarly, numbers in general should remain the same, yet there are exceptions such as unit conversions, different date formats or currency conversions. In other words, some of the numerical content should remain as is in the translated text, whereas some of it should be translated. As numerical content is rarely read with much attention, simple errors may creep in.

2. Automated Quality Assessment Tools

For these reasons, it is a good idea to automate the chores of proofreading. It is especially easy when the text to proofread is already in a bilingual form, which is what all current computer-aided translation (CAT) tools provide. Yet, the standard translation quality assessment (QA) tools that support bilingual formats are usually limited. For example, QA Distiller[2] has the following features:

- detect untranslated segments,
- detect inconsistency,
- detect formatting problems (spaces, commas...),
- detect language-dependent formatting problems (date format etc.),
- detect terminology errors,
- use regular expressions for customized rules.

Such features help find many obvious, yet hard-to-find problems in the translation. However, for morphologically rich languages, regular expressions are usually not enough to specify the inflected form, especially for consistency checking. In other words, QA Distiller, though arguably one of the leading QA tools, is mostly blind to the grammar of the natural language. It can easily

[2] http://www.qa-distiller.com/

support only languages that are similar to English with relatively stable word order and limited morphological variation.

While QA Distiller is a commercial tool for translation agencies, there are also free applications that gain popularity among individual translators. The most notable of them are ApSIC Xbench,[3] featuring most of the checks offered by QA Distiller (with the exception of the date formats etc.), and CheckMate,[4] and open-source QA tool from the Okapi tool set, which offers even more. Figure 1 shows the dialog box of CheckMate. As can be seen, it displays several kinds of possible errors detected by the tool: untranslated segments, translated content with hidden content, special tags not found in the source, and suspicious ratio of the target text to the source text (too short or too lengthy). Some of the errors might be trivial for most uses (like adding or removing "http://" part in the URL addresses), but for financial data, any change of numbers should be highlighted.

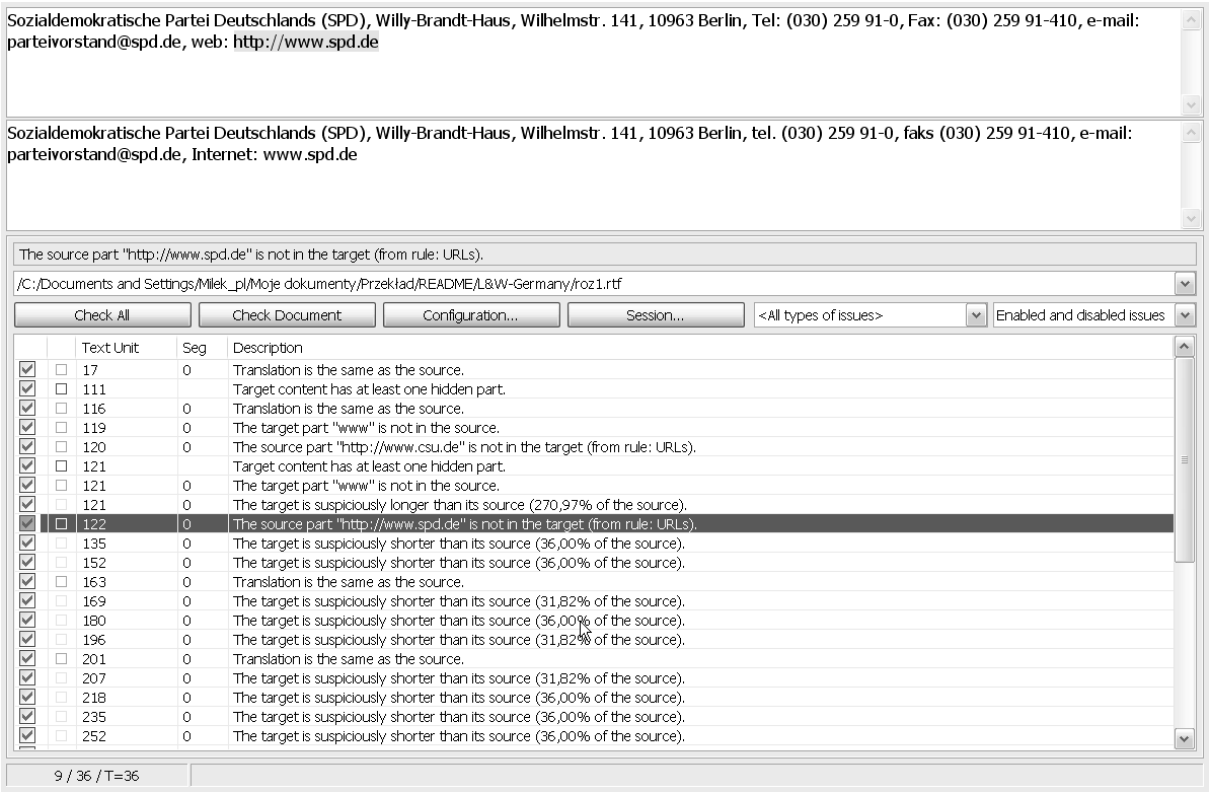

Figure 1. Screenshot of CheckMate.

Yet neither commercial nor free tools really support linguistic technology. None of them uses part-of-speech (POS) tagging of text to build rules that detect grammatical problems. They also do not support stemming, and custom rules

[3] http://www.apsic.com/en/products_xbench.html

[4] http://www.opentag.com/okapi/wiki/index.php?title=CheckMate

rely exclusively on regular expressions on strings. Moreover, words in the sentence are specified as linear sequences, and that is acceptable only for languages with strict word order.

Out of the three standard QA tools, CheckMate is special in one regard: not only does it support custom regular-expression-based rules, but it also uses external proofreading engines, such as LanguageTool, and that greatly enhances its abilities.

3. LanguageTool

LanguageTool is an open-source proofreading tool, used for example in OpenOffice.org and LibreOffice (Miłkowski 2010). As of June 2011, it supports 22 languages to a different degree, described in Table 1, and more are in preparation (during the current Google Summer of Code project, Tao Lin is preparing support for Chinese, and there are also preliminary prototypes for Filipino).

Table 1: Languages supported by LanguageTool.

Language	**XML rules**	**Java rules**	**Rule maintainers**
Belarusian	7	0	Alex Buloichik
Catalan	213	1	Ricard Roca
Danish	22	0	Esben Aaberg
Dutch	311	0	Ruud Baars
English	490	3	Marcin Miłkowski, Daniel Naber
Esperanto	177	0	Dominique Pellé
French	1891	1	Agnes Souque, Hugo Voisard (2006/2007), Dominique Pellé
Galician	166	0	Susana Sotelo Docío
German	150	8	Daniel Naber
Icelandic	39	0	Anton Karl Ingason
Italian	86	0	Paolo Bianchini
Khmer	22	2	Nathan Wells
Lithuanian	4	0	Mantas Kriaučiūnas
Malayalam	18	0	Jithesh.V.S
Polish	1028	4	Marcin Miłkowski
Romanian	418	1	Ionuț Păduraru
Russian	129	3	Yakov Reztsov
Slovak	55	2	Zdenko Podobný
Slovenian	58	0	Martin Srebotnjak
Spanish	70	1	Juan Martorell
Swedish	26	1	Niklas Johansson

Figure 2 shows the dialog box of OpenOffice that uses LanguageTool as its backend to check text.

Figure 2. LanguageTool checking an English sentence in OpenOffice.org.

LanguageTool (LT) is based on surface text processing, without deep parsing, not to mention semantic analysis, yet, it manages to get significantly better results (for some languages) than commercially available products (see Miłkowski 2010 for comparison for Polish). Some rules in LT are built semi-automatically from corpora (Miłkowski forthcoming). Most interestingly for the translation QA, LT is used as the grammar-checking engine in several tools, such as the aforementioned CheckMate or the open-source CAT tool OmegaT. Figure 3 shows how LT is supported in OmegaT.

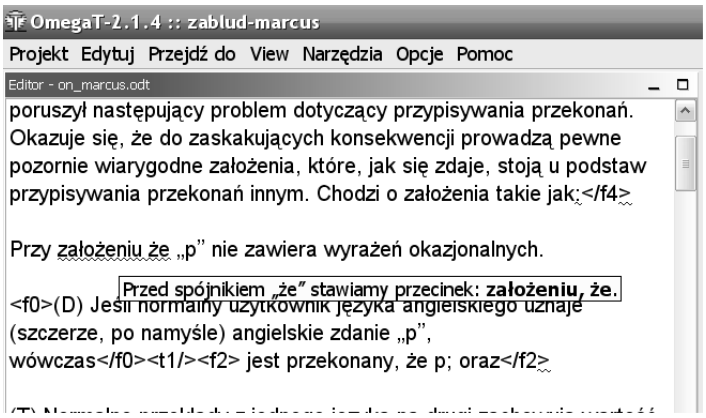

Figure 3. LanguageTool in OmegaT displays an error in a Polish sentence.

The API of LT allows its integration via simple HTTP requests in different software packages, as well as direct use of its methods in Java. LT can also natively support bilingual text files in tabbed format (where the first field is the

source segment, and the second field – the target one), which makes it useful for inclusion in batch processing workflows.
In CheckMate, the errors detected by LT might fall into two categories: target-text errors, which are supported in a monolingual mode, and bilingual-mode errors. CheckMate itself supports numerous bilingual input formats, such as XLIFF, TMX, PO, TTX, tagged RTF (Trados-style) *etc*., which means that LT does not have to duplicate this functionality. Figure 4 displays CheckMate dialog box including the errors found in the target-text error mode.

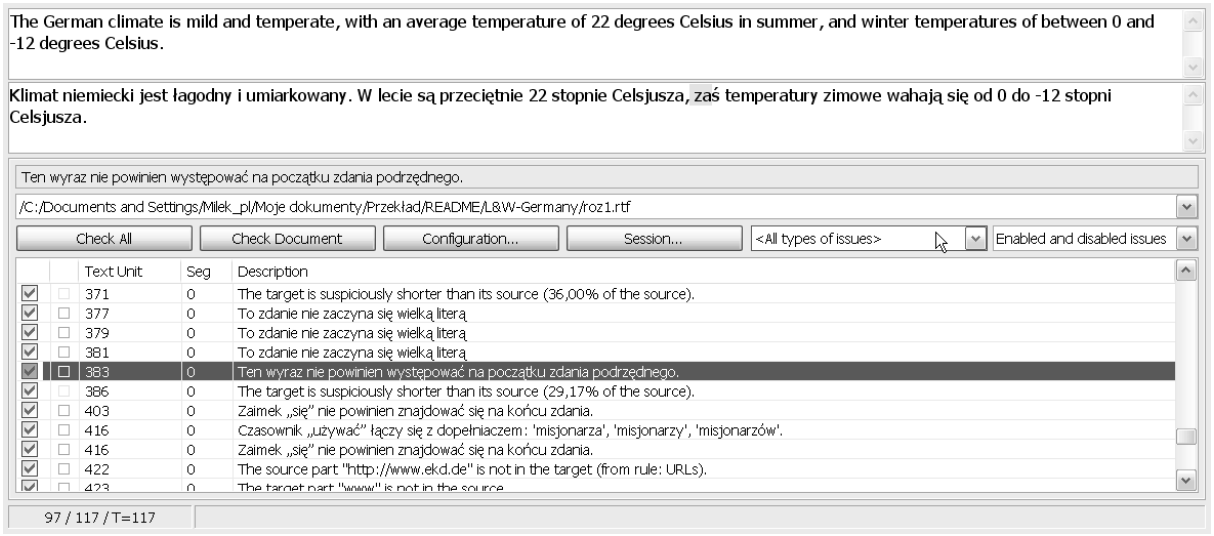

Figure 4. CheckMate uses LanguageTool to check target text.

As can be seen on the screenshot, the errors in this mode are much more related to linguistic structure of the text than previously: usage of capital letters, pronouns, valence of verbs etc. However, only the bilingual mode of LT shows its real strength in checking translations.
In bilingual (bitext) mode, LT uses:

- false friend rules: the rules are matched only when both source and target contain the false friend terms (to avoid false alarms);
- rules for target language;
- generic bitext rules (in Java);
- XML bilingual rules for target language (if any).

False friend rules. In LT, there are many rules that detect possible false friends in translation (in different language combinations, mostly for English, German, Polish, Italian, and French). They are useful for non-native speakers writing documents in a foreign language. In bilingual mode, these rules are triggered less frequently because the source text is also checked, so the number of false alarms is much lower. Table 2 shows the number of false friend rules per language (the languages not included do not have them defined at all). The number of rules per languages is the sum of all rules in which the language is

mentioned (in all combinations). In other words, if there are 4 rules for Catalan, it does not mean that they all support Polish-Catalan false friends. At the same time, English words have usually false friends defined for multiple languages ("actually" is a false friend in all languages in Table 2).

Table 2. False friend rules in LanguageTool.

Language	Number of false friend rules
Catalan	4
English	231
French	5
Galician	6
German	88
Italian	36
Polish	140
Spanish	27
Swedish	5

Rules for target language. These are standard target-text rules, which include declarative rules (saved in XML format), and procedural rules written in Java. Most of them are XML rules as they are much easier to write for people without programming experience, as well as they allow focusing on the linguistic content of rules (for the detailed description of the formalism, see Miłkowski 2010). The Table 1 under **XML rules** includes the number of such rules per language, and the column **Java rules** displays the number of generic Java rules per language. Note that some of the generic Java rules work for multiple languages (for example, rules to detect word repetition and unpaired brackets).

Generic bitext rules. As of writing, there are only two such rules: one checks if the translation length is roughly the same as the source, and another if the translation is the same as source. Both are written in Java. Obviously, using them in CheckMate makes no sense, as CheckMate already includes this functionality.

XML bilingual rules. The XML bilingual rule files specify errors found for many source languages and one target language, and their format is based on the formalism used already in monolingual rules. They declare language-aware specific checks and corrections, including, but not limited to:

- terminology, even in complex phrases (and with rich morphology);
- incorrect syntax patterns (copying original grammar structure to a target language with a different syntax);
- inconsistency in terminology;
- dates, currencies, number formats...

LT includes lemmatization for most languages it supports, and declaring that any form of the word should be detected is as easy as adding a single XML attribute *inflected* with a value "yes". Even though the XML rules are restricted to shallow parsing, their expressive value, is, as we believe, roughly comparable to that of the Constraint Grammar (Karlsson et. al 1995).[5] Shallow parsing, moreover, is successful in many applications in computational linguistics (Przepiórkowski 2008), and it is easier to develop it step-by-step, starting from very simple resources, which is important for the success of open-source projects. Our XML notation allows specifying most errors in a concise way, mostly without recourse to cryptic regular expressions (though they are also supported).

In the current release of LT (1.4), there are only a handful of rules for English included. As the XML notation is documented and there are already thousands of examples in LT code base, it is however possible to formalize at least some of the advice offered in the translation style guides. This way, LT might become part of the QA workflow for complex translation jobs, where multiple translators and proofreaders collaborate. Using LT, the proofreader will be able to focus on important mistakes, not just mechanical and stupid ones.

4. Anatomy of a Rule

Let us take a real-world mistake: a translator uses Google Translate to make his job quicker and mistranslates "similar in kind" literally as "podobny w naturze". This is an obvious style problem in Polish. The pattern in XML rule for Polish would be:

```
<pattern>
 <source lang="en">
     <token>similar</token>
     <token>in</token>
     <token>kind</token>
</source>
<target>
      <token inflected="yes">podobny</token>
      <token>w</token>
      <token>naturze</token>
</target>
</pattern>
```

[5] One of the projects for the Google Summer of Code 2011 in LanguageTool is preparing a (hopefully lossless) conversion tool from Constraint Grammar to LT formalism.

The rule is processed in the following way: if LT finds an English sentence (or segment) with words “similar in kind” and the Polish sentence with words “podobny w naturze” (whereas “podobny” might occur in any of its inflected forms), it will match the rule. After the rule is matched, a suggestion may be offered (if it is defined in the rule). For example:

```
<message>
Czy chodziło o
   <suggestion>podobnego rodzaju</suggestion>?
</message>
```

In this case, LT will offer “podobnego rodzaju” as a correction in the Polish text. (If the automatic mode is used, LT may automatically apply this correction. This is especially useful in automated post-editing (see section 5) and in applying terminological changes to multiple files automatically.)
The last part of the rule are examples of correct usage (which are not detected, though usually similar to the source pattern), and examples of incorrect sentences that are detected as incorrect. For example:

```
<example type="correct">
   <srcExample>These apples are similar in
kind.</srcExample>
   <trgExample>Te jabłka są podobnego
rodzaju.</trgExample>
</example>
<example type="incorrect" correction="podobnego
rodzaju">
   <srcExample>These apples are <marker>similar in
kind</marker>.</srcExample>
   <trgExample>Te jabłka są <marker>podobne w
naturze</marker>.</trgExample>
</example>
```

These examples serve several purposes. First, they check if the rule was declared properly and it actually matches the incorrect sentence in the place identified with the tag “marker”. They also check if the suggestion applied is actually the same as the developer of the rule intended (in this case, this is trivial, but suggestions may be created dynamically by inflecting words, changing them using regular expressions etc.). Second, they can be used as the internal documentation of the rule: if properly displayed on the screen, they can help to easily identify the purpose of the rule.

Note that the tokens in the source and target sentences are not aligned with each other: LT does not try to align words with each other, so it does not know which source word has "naturze" as its translation. If such alignment is required, it can be added in the generic Java rules. However, we found no need for it in our practice.

5. Other Uses and Future Work

The application of bilingual checking in LT is not limited to translation quality control. Another possible use is automatic post-editing of statistical machine translation.[6] For example, it may be used to make sure that financial data get translated in a consistent manner (i.e., that currency remains consistently translated). Such rule-based checks are useful when the translation itself was done statistically. It may also fix missing negation words, which is common in Google Translate in Polish.

Bilingual checks may also be used to validate syntactic correctness of complex multilingual structures. For example, custom rules for checking validity of abbreviations and notation in a large bilingual dictionary project were built by the present author.

In the future, there are several important rules that we plan to implement. The first is a smart number matching rule that uses textual way of translating figures ("one" gets translated into "1", or "1" gets translated into "jeden"). Standard QA tools cannot do that, but thanks to the NumberText library[7], it is relatively easy to convert numbers to text for multiple languages. The second rule will be partly based on the first: it will include smart currency and date conversion checks.

Another area of future development is inclusion of conversion tools to convert terminology bases in TBX and text format to the richer LT format (and, alternatively, direct support for glossaries in terminology checks but without special linguistic features). For this to work, it will be useful to add support for multiple custom rule files and rule file sets to LT user interface.

It is also possible to create some of the false-friend rule files semi-automatically by using terminology bases (especially for multiword expression) and machine

[6] Usually, it is the rule-based machine translation (RBMT) which is post-edited using statistical methods to enhance the fluency of the translation (see Simard et al. 2007, Dugast et al. 2007). However, post-editing in the case I mention serves another purpose: applying strict, rule-based checks on statistical input in order to preserve certain strict translations (like currency conversions). It may also be used to remove false friends and translate idiomatic phrases that are underrepresented in corpora (instead of including the lexicon as part of the training corpus, one may simply apply lexicon-based translation on garbled statistical translation).

[7] http://www.numbertext.org/

translation to see which multiword expressions are usually mistranslated by default, and then supply the correct translation as the proper one.[8] This can enhance the capabilities of the QA system far beyond the style guides and mechanic mistakes.

6. Conclusion

Translation QA should be done in language-aware fashion, not just like pure text search (even with regular expressions). Otherwise, many mistakes will not be detected, though they are arguably mechanic in nature and hard to spot for human proof readers.

There are free and open-source tools that enable checks that go beyond what most commercial packages can do. We hope that their development will help the proof readers and translators to focus on the content of the translation. Otherwise, the proofreading process is costly and time-consuming, and the translation quality is negatively impacted.

References

Miłkowski, M. (2010). Developing an open-source, rule-based proofreading tool. *Software: Practice and Experience* 40, no. 7: 543-566.

Miłkowski, M. (forthcoming). Automating rule generation for grammar checkers. In S. Góźdź-Roszkowski, *Proceedings of PALC 2009*.

Karlsson, F., Voutilainen, Heikkilä, J., Anttila, A. (ed). (1995), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, Mouton de Gruyter, Berlin.

Dugast, L., Senellart, J. & Koehn, P. (2007). "Statistical post-editing on SYSTRAN's rule-based translation system". In *Proceedings of the Second Workshop on Statistical Machine Translation - StatMT'07*. Morristown, NJ, USA: Association for Computational Linguistics, 220-223.

Przepiórkowski, A. (2008). *Powierzchniowe przetwarzanie języka polskiego.* Akademicka Warszawa: Oficyna Wydawnicza EXIT.

Simard, M., Ueffing, N., Isabelle, P. & R. Kuhn (2007). "Rule-based Translation With Statistical Phrase-based Post-editing." In: *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague: Association for Computational Linguistics, 203–206.

[8] This idea was suggested to me by Jimmy O'Regan (personal communication).