The Integration of Moses into Localization Industry

Tomáš Hudík
Moravia Worldwide
Hilleho 4, Brno,
Czech Republic
thudik@moraviaworldwide.com

Achim Ruopp
Digital Silk Road
2800 Quebec St NW #816
Washington, DC 20008
achim@digitalsilkroad.net

Abstract

A majority of the localization industry is still unable to benefit from SMT, since it does not know how to integrate SMT system into its production workflow. This paper describes a workflow on how to integrate the open source SMT system Moses into computer-aided translation (CAT) tools. It introduces a software package which is able to export XLIFF data into a special *InlineText* format that is further processed and is sent as plain text to Moses as input. Then, the translated text is inserted back to the original file.

This transformation is needed, otherwise the use of MT in the localization industry would hardly be advantageous, since formatting information would have to be manually added by localization professionals. A few attempts to bridge the localization industry and Moses were already done in the past, however, the method described here represents a new approach.

1 Credits

This work was done with the support of Moravia Worldwide and in cooperation with Digital Silk Road and the Let'sMT! consortium.

2 Introduction

An integration of *Statistical Machine Translation* (SMT) into *Computer Aided Tools* (CAT) has been needed for a long time (Ruopp, 2006).

The localization industry started to use so-called *Translation Memory* (TM) (Reinke, 2006) some 30 years ago. A lot of valuable information is stored in these datasets. TMs, if available, are used for each translation project, since they contain things like a translated previous version of a product, or translation of some other product from the same client. With the deployment of TMs, translation has become much easier, often pre-translated text just need to be post edited. Therefore, TMs are very useful for human translators and should be heavily used also by SMT systems.

Mainly universities and research centers are creating various SMT systems. Each of them provides two phases – training and translation. For both phases, it is necessary to have input. Due to completely different environments, the issues involved and broad variability of research questions, universities do not have a uniform format similar to TM. For this reason, SMT systems usually require general plain text as an input. The localization industry is different, its TMs contain a lot of additional information, for example XML, RTF, HTML tags, or even binary data. Some of it is related to formatting – e.g. end of line, italics. Other kind of the information is related to content placeholders – e.g. untranslatable company names; specific translations of dates, numbers, names etc. Such additional information is not a big problem in the training process, since additional formatting tags can be easily removed and special tags like placeholders can be substituted. A much bigger problem is to use TMs in real translations due to a necessity to have all the described information included in a result (target language)[1].

Moravia in collaboration with *Digital Silk*

[1]Simplification – TM is exported to TMX, the vendor-neutral open XML standard for the exchange of TM data created by CAT tools, which is used in translation

Road[2] (project called *m4loc* – Moses for Localisation), and the consortium *Let'sMT!*[3] created a software toolkit that is capable to use TMs in SMT system called Moses. The developed toolkit is released under LGPL license and is freely downloadable from `code.google.com/p/m4loc`.

A few attempts to achieve this goal has been done in the past already, e.g. (Simard, 2009; Du, 2010), although, none of them were released as an open source[4], or covered the whole cycle: using TM in Moses and putting the results back to TM as some alternative (suggestion) translation which can be accepted, rejected, or modified by a post editor.

The most similar approach is followed by (Du, 2010). However, methods 2 and 3 would bring a lot of noise into SMT systems. Consequently, the noisy systems would give worse results and require additional post-editing. Our work can be considered as an improvement of method 1 (Markup transformation).

3 Technology Overview

3.1 XLIFF and TM

TMs can be stored in a variety, often proprietary, formats (e.g. *tmx, po, btx, XLIFF*). The formats are often based on XML. The data is formatted in two layers. The first layer is format, like the type of font (bold, or italic) and the second one is a content layer, like a reference to full product name which should not be translated at all, or numbers (dates) which are translated in a special way. A deeper knowledge can be gained in (Reinke, 2006).

Recently, TMX became one of the standards in the localization industry for TM exchanges. *XLIFF* – XML Localisation Interchange File Format, overseen by *Oasis*[5] is widely used as well. All the major CAT tools providers support TMX and their tools are capable to convert TMX to/from XLIFF. Unfortunately, these conversions are often far from perfect. OASIS is working hard to improve XLIFF in the next version (2.0) that should solve many issues, some of them related to MT. XLIFF allows to store translation candidates with an associated quality score. Due to these features, XLIFF was recognized as the best alternative for our effort to bridge TM/CAT and Moses.

3.2 Moses

As it is well-known, many different SMT and RBMT systems emerged in the past ten years. Probably the most developed ones are: *Moses* (Koehn, 2007) and *Apertium* (Armentano, 2006). At the beginning, Apertium was devoted to the languages of Spain. Since Moses is well-documented and is better- known in the localization industry, it was chosen for our effort.

Moses is still a hot spot of research, it was developed mainly by Edinburgh University and is heavily funded by many organizations, for example European Commission (translation and interpretation consumes 1% of whole annual EU budget). One of major Moses disadvantages is how it is releasing. Only builds (Subversion repository), it means almost no quality assurance, or usability testing are available (no versions). This is acceptable in university environment, however, it is causing some pain for industry partners.

4 XLIFF and Moses integration

For a better understanding of the overall process: people from the localization industry often face a problem similar to the one shown in Figure 1 on how the screen dialog should be correctly translated and *also* correctly placed (aligned) as shown in Figure 2.

Figure 1: Source sentence – English is the source language. The word "IBM" is italicised and the line break is placed after "created". The right part of the figure is some graphic

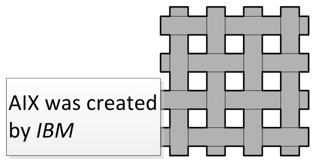
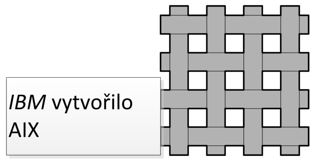


Figure 2: Solution. Target language – Czech. The sentence was translated and correctly placed



[2]`www.digitalsilkroad.net`
[3]`www.letsmt.eu`
[4]PangeaMT probably has such a system, however, all trainings and testing have to be done internally on theirs servers. No scripts, programs, or specification are possible to reach
[5]`www.oasis-open.org/committees/xliff`

Figure 3: XLIFF ⇔ Moses transformation workflow. Regular darker bars refer to scripts and lighter bars with curly bottom line refer to the state of data (e.g. whether data is tokenized). Start and end state are marked as it is common in UML – solid circle is the starting point, double circle indicates finite state. Dot-dashed lines show additional information needed by a script. For example, markup re-inserter needs tokenized target but also tokenized InlineText source

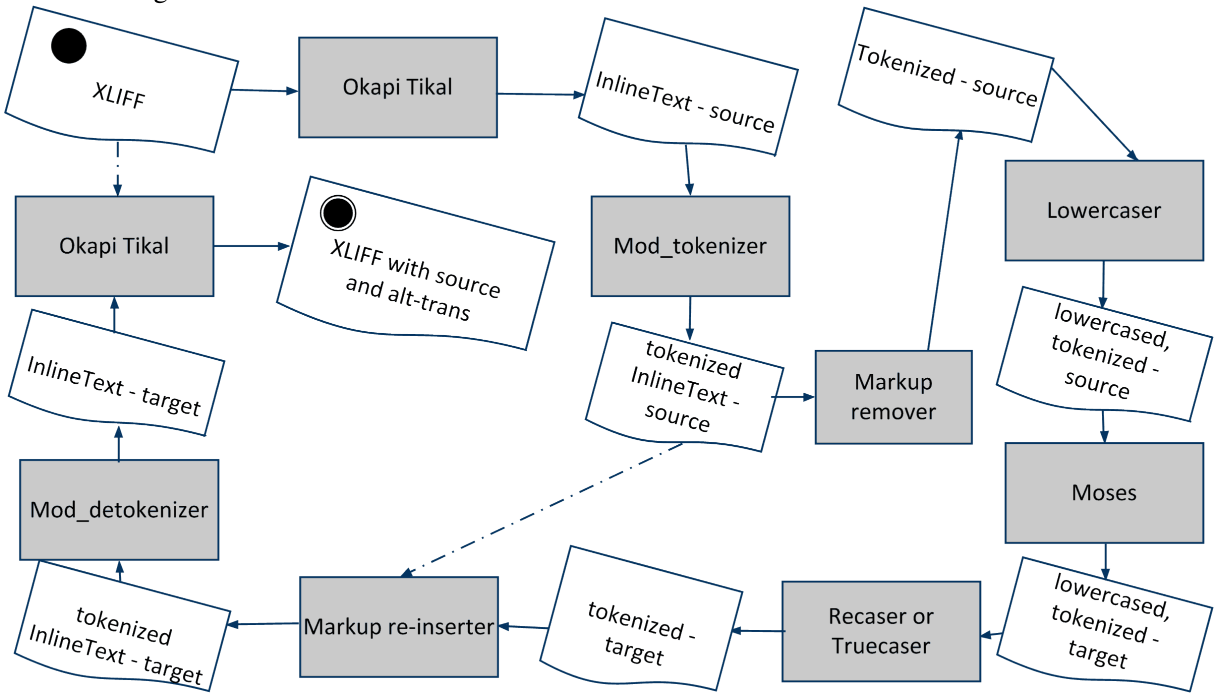


Figure 4: Extracted string in XML

```
AIX was created<x id="10A" ctype="lb"/>by <i>IBM</i>.
```

Figure 5: Example of InlineText source

```
AIX was created<x id="1"/>by <g id="2">IBM</g>.
```

Figure 6: Tokenized InlineText source (white spaces added)

```
AIX was created <x id="1"/> by <g id="2"> IBM </g> .
```

Figure 7: Incorrectly created InlineText

```
AIX was created &lt;x id="1"/> by &lt;i>IBM&lt;/i>.
```

Figure 8: Source sentence from Figure 1 inserted into XLIFF. The whole source element is on one line, "\" indicates no-end of line

```
<source>AIX was created by\
<ph id="1">&lt;x id="10A" ctype="lb"/>\
</ph><bpt id="2">&lt;i></bpt>IBM\
<ept id="3">&lt;/i></ept>.</source>
```

The focus, throughout the article, will trace this particular translation. A complete description of the workflow of how Figure 1 is translated into Figure 2 will be given.

At first, the source sentence needs to be exported into XLIFF format. This can be easily done by many available CAT tools. Let's assume some CAT tool extracted the sentence from Figure 1 to the XML string shown in Figure 4. A localizer needs to translate this XML string, which means to retrieve (from some external MT or SMT system) XML string in target language which would be easily inserted into Figure 2 by the CAT tool. The problem is that this XML string is different in every CAT tool, therefore export to XLIFF is a necessary next step.

After translation, the CAT tool reinserts the XML code into XLIFF, an example of which is shown in Figure 8 (for the specification of XLIFF tags see `docs.oasis-open.org/xliff/xliff-core/xliff-core.html`). Each CAT tool can produce slightly different XLIFF. Note, XLIFF is XML again,therefore one can see `&lt;/i>` instead of `</i>`. The description of the conversions into XML is out of scope of this article, however,can be easily found on the Internet. The tag `<x/>` is a placeholder and means the place where line is divided, the end of line is due to the image on the right.

The whole cycle XLIFF ⇔ Moses can be seen from Figure 3. Direction XLIFF ⇒ Moses, it means Okapi Tikal, Modified tokenizer, Markup remover and Lowercaser is covered in Section 4.1. And the rest, Moses ⇒ XLIFF, is explained in Section 4.2.

4.1 XLIFF to Moses

XLIFF is processed by *Tikal* which is a part of *Okapi Framework*[6]. Tikal is a cross-platform command-line tool that performs various localization-related tasks. It is especially good at transformation from one file format into another. In our workflow,Tikal converts XLIFF file into so-called *InlineText* format. This conversion is done by `-xm` option. The conversion to InlineText and InlineText back to original XLIFF file was developed by Yves Savourel.

In Figure 3, this is referred to as `InlineText source` because only the source language is needs to be extracted. For the extraction of source and target language, Tikal uses the parameter `-2`. While the source InlineText is created from the XLIFF `<source>` tag, the target is created from `<target>`. Throughout XLIFF ⇔ Moses transformations, the target file is not needed. However, it is good to have a possibility to check how some particular segment was translated, for instance by some other MT tool. Target and source files have the same number of lines, equivalent to the number of segments in the XLIFF file. In some cases the segment is not translated and the target file will contain an empty line instead of translation. Basically, the output of Tikal's `-xm` process is a XML document without the XML notation and some root element. It can have just four elements (tags): `x, g, bx` and `ex`. All of them have mandatory attribute *id*, nothing else is permitted – no other attributes, namespaces, etc. These tags are a specific subset of XLIFF inline elements.

- `<g>` – paired tags; e.g. `<i></i>`
- `<x>` – standalone tags; e.g. `<x/>`
- `<bx>`,`<ex>` – broken tags

Broken tags are the ones which were broken for some reason – e.g. during segmentation, some errors in CAT tools, etc.

InlineText source, for our example, is shown in Figure 5, where:

- `<x id="1">` points to XLIFF's linebreak
- `<g id="2">` points to `<i>` tag in XLIFF

Keep in mind that InlineText tags just refer to XLIFF tags. Therefore, in the first item, `<x id="1">` is not a linebreak but rather a reference to the original `<x id="1" ctype="lb"/>` which is the linebreak. It can be a bit misleading since both of them use the same element name.

Many different XLIFF files have been processed during testing phase. Tikal's advantage is an ability to work with XML's namespaces. IDIOM

[6]okapi.opentag.com

World Server[7] is particularly problematic, it creates its own, very complicated XML namespace that is incorporated into each XLIFF file produced by this tool. However, Tikal is able to handle it.

The application of *mod_tokenizer.pl*[8] is the next step. It is a modification of the original Moses' *tokenizer.perl*. It treats input as XML document and does not tokenize XML tags and URL addresses, all the other contents is tokenized via Moses' original tokenizer. The modified tokenizer needs the `nonbreaking_prefixes` directory, part of the original tokenizer, to be in the same directory. The original tokenizer is language-depended, it tokenizes a text string according to rules specific for some particular language. If the appropriate language is not found, rules for English are used instead. More info can be found in the Moses documentation.

At the moment, a number of interested parties are involved in the development of the XLIFF ⇔ Moses workflow and it is still in a testing phase. Disadvantages of such development are that some procedures could be programmed in a more efficient manner, fewer lines of code could do the same job and also could be faster.

The output of the modified tokenizer can be seen in Figure 6. A space was added before and after the `<x>` and `<g>` tags.

Of crucial importance is to have correctly created XLIFF. If this would not be the case, Tikal would produce incorrect results and the other scripts in workflow would increase problems. For example, if the sentence from Figure 4 would be treated as a text instead of XML in e.g. SDL Trados Studio, the XLIFF would look like one in Figure 9.

Figure 9: Wrongly encoded text can lead to incorrect XLIFF. The whole code is on one line, "\" indicates no-end of line

```
<source>AIX was created by\
&lt;x id="10A" ctype="lb"/>\
&lt;i>IBM&lt;/i>.</source>
```

Then, Tikal's output would be the text in Figure 7 instead of Figure 5. With such corrupted Inline-Text, all the information about formatting is lost and also the translation would be of low quality because of strange input.

Later on, the *markup remover* is invoked. It is easy and fast perl script which removes all markups (XML tags). Even the documentation states that Moses is able to handle XML tags, it is not always true as it was found during our research.

Finally, Moses' *Lowercaser* is deployed. It simply converts all text into small letters.

4.2 Moses to original XLIFF

The Moses decoder translates lowercased, tokenized source text into lowercased, tokenized target text.

In order to insert the translations back into the original XLIFF file as <*alt-trans*> or <*target*> elements, Moses needs to be run with `-t` option which reports the phrase segmentation for the best translation hypothesis. When translating the example sentence `aix was created by ibm .` the Moses output would be `ibm |4-4| vytvořilo |1-3| aix |0-0| . |5-5|`

The first step of the XLIFF ⇐ Moses chain is to use the Moses *Recaser* or *Truecaser* to correct text capitalization. After this step the example would look like this:

```
IBM |4-4| vytvořilo |1-3| AIX
|0-0| . |5-5|
```

Then, the recased, tokenized target text including the phrase segmentation is processed by the *markup reinserter*. The markup reinserter combines the inline element information from the source file with the target text including the phrase alignment information to output target text with inline elements. The algorithm used attempts to the place the inline elements in the target text under the following constraints:

- All inline elements that are present in the source text have to be placed in the target text
- For paired inline elements the closing tag always has to be placed after the opening tag
- Multiple paired inline elements can only enclose each other, they cannot overlap (this is required by XML)
- Opening tags of inline elements are to be placed as close as possible before the correct target word token

[7] `www.idiominc.com`
[8] documentation can be found in `code.google.com/p/m4loc` click on *Specification*

- Closing tags of inline elements are to be placed as close as possible after the correct target word token (unless this violates second constraint)

The algorithm implements these requirements in the following steps:

- Based on the source text with inline elements the algorithm determines the position of the inline elements relative to the source word tokens
- For each of the target phrases
  - It is determined which elements that are to be closed and opened based on the phrase alignment information (enclosed in the vertical bars)
  - All elements that need to be opened are added to the output
  - The target phrase is added to the output
  - All elements that need closing and are currently open are added to the output (remaining closing tags are stored in a hash for output in later phrases)
- If any inline elements remain, output them at the end of the sentence

With the phrase alignment information and the markup positions from the tokenized InlineText source, the markup reinserter is capable of placing the inline elements at appropriate positions in the target text. Due the phrase segmentation performed by the MT engine, the placement might not exactly match the placement a human editor would perform. The output is tokenized InlineText target.

The *Modified detokenizer* simply removes white spaces in markup tags and non-markup text is processed by the original Moses *detokenizer.perl* for final detokenization.

At the end of the XLIFF ⇐ Moses chain is Okapi Tikal which takes InlineText target as an input and inserts it as an alternative translation or target element into the original XLIFF. The option `-lm` needs to be used. The final text is shown in Figure 10.

Such results go to a post editor who, together with some CAT tool, creates the final results – correctly translated and aligned output (Figure 2).

Figure 10: Target (Czech) sentence inserted into XLIFF as <*alt-trans*> tag. The whole segment is on one line, “\” indicates no-end of line

```
<target><bpt id="2">&lt;i></bpt>IBM\
<ept id="3">&lt;/i></ept>\
<ph id="1">&lt;x id="10A" ctype="lb"/>\
</ph>vytvořilo AIX.</target>
```

5 Conclusions

The integration of SMT into CAT tools was described in this article. Due to easy exchange and broad support, the XLIFF format was chosen to be the connector on the side of CAT tools. As the SMT system, Moses was picked because of its good support, open source license and continued innovation.

During the Let'sMT! and m4loc (Moses for Localization) projects, the described system will be tested by volunteers[9] from both projects on various free and proprietary data.

In the future, we will be looking support for tree-based models (Chen, 2009), since our approach is currently only applicable to phrase-based models. Also of interest would be the development of a metric that measures the accuracy of the inline element placement and the required post-editing effort. This would also allow the comparison of various approaches for the placement of inline formatting.

References

Armentano-Oller, C., Carrasco, R. C., Corbí-Bellot, A. M., Forcada, M. L., Ginestí-Rosell, M., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Ramírez-Sánchez, G., Sánchez-Martínez, F., and Scalco, M. A. 2006. *Open-source Portuguese-Spanish machine translation*. In: Computational Processing of the Portuguese Language, Proceedings of the 7th International Workshop on Computational Processing of Written and Spoken Portuguese, PROPOR 2006, pp 50–59.

Du J., Roturier J. and Way A. 2010. *TMX markup: a challenge when adapting SMT to the localisation environment*. In: EAMT 2010 – 14th Annual Conference of the European Association for Machine Translation, Saint-Raphaël, France.

Chen Q., and Yao T. 2009. *Flattened Syntactical Phrase-Based Translation Model for SMT*. In: Proceedings of the 22nd International Conference on

[9] `demo.letsmt.eu`

Computer Processing of Oriental Languages. Language Technology for the Knowledge- based Economy (ICCPOL '09), pp 345–353.

Koehn, P., Hoang, H., Birch, A., Callison-Burch,C., Frederico, M., Bertoldi, N., et al. (2007). 2007. *Moses: Open Source Toolkit for Statistical Machine Translation*. MT Summit XII: proceedings of the twelfth Machine Translation Summit, Annual Meeting of the Association for Computational Linguistics (ACL). Prague, Czech Republic.

Reinke U. 2006. *Translation Memories*. In: Encyclopedia of Language & Linguistics, Elsevier

Ruopp A. 2010. *The Moses for Localization open source project*. AMTA 2010: the Ninth conference of the Association for Machine Translation in the Americas, Denver, Colorado.

Simard M, Isabelle P. 2009. *Phrase-based Machine Translation in a Computer-assisted Translation Environment*. MT Summit XII: proceedings of the twelfth Machine Translation Summit, pp.120-127, Canada, Ontario.