Apertium Advanced Web Interface
A first step towards interactivity and language tools convergence

Arnaud Vié
Informations Systems Engineering
Grenoble INP - Ensimag
arnaud.vie@ensimag.fr

Luis Villarejo Muñoz, Mireia Farrús Cabeceran
Learning Technologies Office
Universitat Oberta de Catalunya
lvillarejo@uoc.edu, mfarrusc@uoc.edu

Jimmy O'Regan
Eolaistriu Technologies
Thurles
Ireland
joregan@gmail.com

Abstract

This document describes a project aimed at building a new web interface to the Apertium machine translation platform, including pre-editing and post-editing environments. It contains a description of the accomplished work on this project, as well as an overview of possible future work.

1 Introduction

One of the classic, and still open, tasks of Natural Language Processing (NLP) is Machine Translation (MT). Since its first steps, back in the 1950s (Hutchins and Somers, 1992), MT has increased its presence in several scenarios providing access to multilingual content. The number of MT initiatives has risen greatly in recent years, mainly in statistical MT, as a result of the availability of vast multilingual parallel texts, but also in rule-based MT, example-based MT or hybrid systems.

One such example is the Apertium machine translation engine[1]. Apertium is a transfer-based MT platform which provides the engine, tools and data to perform translations between a large number of language pairs, available under the terms of the GNU General Public License (GNU GPL)[2], and is being developed by a community of users worldwide. It has been integrated in a wide variety of translation workflows both at public institutions like The Open University of Catalonia (Villarejo et al., 2009) and private institutions such as Autodesk (Masselot et al., 2010).

Apertium's basic design is based on the earlier Spanish–Catalan MT system interNOSTRUM[3] (Canals-Marote et al., 2001), and the Spanish–Portuguese translator Traductor Universia[4] (Garrido-Alenda et al., 2004). developed by the Transducens group at the University of Alicante.

Apertium is designed to operate as a UNIX pipeline (McIlroy et al., 1978): each component is implemented as a separate program, which reads and writes a simple text stream. Components can be added or removed as required: typical components in an Apertium pipeline include:

- A *deformatter* which encapsulates the format information in the input as *superblanks* that will then be seen as blanks between words by the rest of the modules.
- A *morphological analyser* which segments the text in surface forms (*words*) and delivers, for each of them, one or more *lexical forms* consisting of *lemma*, *lexical category* and morphological inflection information.
- A *PoS tagger* which chooses the most likely lexical form corresponding to an ambiguous surface form.
- A *lexical transfer* module which reads each SL lexical form and delivers the correspond-

[1] http://www.apertium.org
[2] http://www.gnu.org/copyleft/gpl.html
[3] http://www.internostrum.com
[4] http://traductor.universia.net

F. Sánchez-Martínez, J.A. Pérez-Ortiz (eds.)
Proceedings of the Second International Workshop on Free/Open-Source Rule-Based Machine Translation, p. 45–51
Barcelona, Spain, January 2011. http://hdl.handle.net/10609/5641

ing target-language (TL) lexical form by looking it up in a bilingual dictionary.

- A *morphological generator* which delivers a TL surface form for each TL lexical form, by suitably inflecting it.
- A *post-generator* which performs orthographic operations such as contractions (e.g. Spanish *del* = *de* + *el*) and apostrophations (e.g. Catalan *l'institut* = *el* + *institut*).
- A *reformatter* which de-encapsulates any format information.

A complete description of the platform can be found in Forcada et al. (2009).

Despite the advances in MT, there has yet to be a system that can produce a perfect translation. For the purpose of *assimilation*, or the understanding of text, MT can be sufficient; however, for the purpose of *dissemination*, or the publication of translated material, correction (post-editing) by a human editor is typically required. Doyon et al. (2008) divides these edits into two main categories: *Full Edits* (larger changes, mainly stylistic), and *Brief Edits* (small changes, mainly syntactic).

Automating the process of post-editing is itself an active area of research, both in SMT (Chin and Rosart, 2008), and in rule-based MT. Llitjós et al. (2004) describes a method of automatically refining rules in a transfer-based system based on user feedback; however, the types of edits permitted are restricted to Brief Edits, while for this work, we aimed at allowing the editor full freedom to edit as they saw fit.

2 Overview

The Apertium platform lacks one important aspect: the ability to get user feedback on the quality of the translation, in order to improve the translation process. Therefore, the initial aim of the project was to build a post-editing environment in which users may correct the translation they obtained, and that the changes they made may be logged by the system to benefit from that user's feedback.

With that goal in mind, we realised that a great number of the typical mistakes found in an automatically translated document are due to mistakes in the source document. Thus, the inclusion of automatic tools that can highlight to the user possible errors or perform efficiently certain repetitive tasks regarding the source document which could improve the translation process.

However, Apertium's web interface[5] is fundamentally inadequate for such a task, as it takes care of the whole translation process at once. Here, it is necessary to pause before translation (for pre-editing) and just after the automatic translation (for post-editing) to enable the user to check for mistakes and correct them in both the source document and the translated output, with all tasks performed via the web interface to allow logging of the edits, hence it proved necessary to build a whole new interface, as well as a framework for future post-editing tools.

This gave birth to the Apertium Advanced Web Interface (AWI) project. As part of the Apertium project, it is available under the terms of the GNU GPL, and hosted on Apertium's SVN repository[6].

3 Benefiting from user edits

3.1 Translation Memory generators and their limits

A simple way, quite commonly used in the NLP community, to benefit from the translation's result is decomposing both source and generated texts in smaller elements, and then find the correspondance between them with an aligning algorithm. This Advanced Interface embeds such a tool, mALIGNa[7] (Jassem and Lipski, 2008), making it able to generate a Translation Memory (TM) of the result in the TMX format (LISA, 2005). A small example of such file is given below in Figure 2.

Such a TM can then be reused by translation engines if they encounter the same elements later. Many engines allow for approximative matches to be considered as well, so that these memories be used in more different cases. Ideally, when Apertium has better support for TMX input[8] we could store this file locally as well to keep a corpus of completed translations and reuse it.

[5] Available on `http://www.apertium.org`

[6] http://apertium.svn.sourceforge.net/

[7] http://align.sourceforge.net/

[8] Currently, only a subset of TMX 1.1 is supported.

Figure 1: Overall interface layout

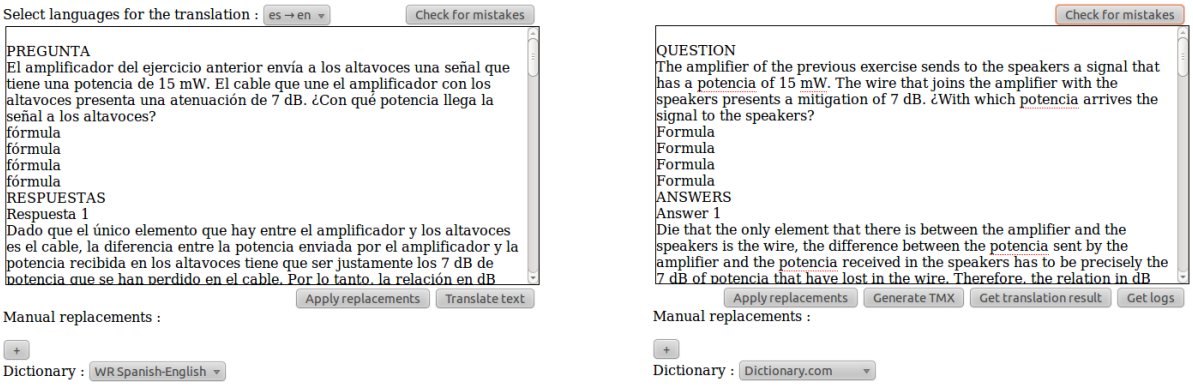


Figure 2: Example of TMX file

```
<?xml version="1.0"?>
<tmx version="version 1.1">
<header
  creationtool="Apertium TMX Builder">
</header>
<body>
<tu>
  <tuv xml:lang="en">
    <seg>This is a test</seg>
  </tuv>
  <tuv xml:lang="es">
    <seg>Esto es una prueba</seg>
  </tuv>
</tu>
<tu>
  <tuv xml:lang="en">
    <seg>This is test 3</seg>
  </tuv>
  <tuv xml:lang="es">
    <seg>Esto es prueba 3</seg>
  </tuv>
</tu>
</body>
</tmx>
```

However, while being widely supported and quite simple to use, this approach also has a major drawback: it is virtually impossible to have it working on a lower level than complete sentences. Indeed, while most sentences retain punctuation, relative order and length throughout the translation – making aligning sentences rather simple – words don't. The order of words is frequently changed, a single word can be translated into several, and so on.

3.2 Text edit logging: different possibilities

A different approach, allowing for more precise analysis, is to log the changes made by the user in the target text after the translation. This isn't directly reusable in other translation processes, but it can help language pair maintainers get an idea of the changes to make. There are two main ways to envision this analysis.

The first is to compare the initial and final forms of the document. A simple diff utility can provide the shortest list of transformations leading from the source text to the result. If the user has only made small changes in the process, this should give out good results. However, important changes may result in it being difficult for the diff tool to located changed and unchanged portions of the text properly.

The second is to log events in real time. In other words, getting information on the user edits at the precise moment they are made. The main advantage of this approach is that some specific events can then be handled in a specific way. For example, it enables logging a global replacement as such, while a comparison between input and output texts would point out one difference for each occurrence of the replacement. This is the approach implemented in the Apertium AWI project.

The central point of this approach is therefore the need to be able to interpret a small event at the moment it occurs, in the web browser, in terms of an operation on words and sentences. Indeed, the web browser gives technical information about events in terms of affected document node and

position inside of the node, that have to be converted to the linguistic information we wish to obtain. This is all the more complex as the text can be contained in a lot of different document nodes, due to the original document's formatting information.

3.3 An interface between browser information and linguistic data

To deal with that problem, this project relies on a text data structure, basically a sequence of sentences that all are a sequence of words, with information on the position of each word in the document. A word in that structure is represented by the object containing the following properties:

- The current text data of the word, that will change as edits occur.
- The original text data of the word when it was loaded. This won't change during editing.
- A reference to the sentence object containing this word
- A reference to the document node containing this word
- The position of the word inside of that node
- References to the previous and next words of the text

All the same, a sentence is represented as:

- References to its first and last words
- References to the previous and next sentences

And the text object contains references to its first and last sentences and words.

This way, locating the word affected by an event is quite easy. It is also easy to get the context around a word - the neighboring words, or event the complete sentence it belongs to. The hard part is keeping that structure synchronised with the real text as the edition goes on, and that is what the logging_lowlevel module of the project does. It is done by handling elementary operations, ie inserting or deleting a character, and determine the structural change to operate considering the properties of the said event.

3.4 The log structure

Now that we can get the information to log in a useful form, it is time to generate a proper log. The `logging_lowlevel` module will call functions from the logging module whenever a basic operation is being performed. These operations include adding, deleting or editing a word, as well as splitting and merging sentences (by inserting or deleting punctuation). This logging module can then be written to handle these basic operations in any desired way; for now, its main job is to group events regarding the same entity. For example, if the user deletes the word "the", three events would be sent by the lowlevel module: edited the word twice, and then deleted it. That is, the elementary events involved are three character deletions, where the first two leave a part of the word (hence a word edition report), whereas the last event deletes the only character of the word (hence a word deletion report). The logging module takes care of grouping all these elementary logs into one word deletion log, and so on. Of course, that module could quite simply be changed to group the events in different ways, to output more of the context of an event, and so on, without changing the lowlevel module that does most of the work.

4 Improving the user experience with the translation

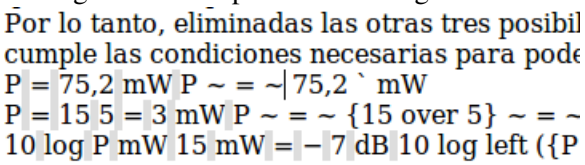
The second objective of this project was to provide a simple interface as well as to integrate convenient tools into it to help the user get an accurate translation. The following tools were included:

4.1 Formatted document handling

The text editor is able to handle formatted documents through the `apertium-unformat` and `apertium-reformat` modules, which replace the usual Apertium format handling tools. Currently, it supports the ODF (ISO, 2006) format, notably used in the OpenOffice suite, but this can be extended to other formats. Translating formatted documents changes only one aspect of the user interface: non-deletable characters (displayed as grey spaces at the moment, see Figure 3) appear in the text to represent the superblanks containing format information, so the user can make

more accurate decisions on where words should be placed in relation to their formatting.

Figure 3: Example of formatting artefacts



4.2 Spell checking and Grammar checking

The interface integrates the ability to check both input and output texts for mistakes. The interface provides a button "Check for mistakes" on top of each text editing field. When pressed, it runs spell checking and grammar checking on the text, in the language specified by the language pair selected for the translation, and underlines mistakes in different colours (red for spelling, blue for grammar). The user can then click on a mistake to see suggestions if they are available (Figure 5) and select one; the description of a grammar error can also be seen when hovering over it, as shown in Figure 4.

Figure 4: Grammar mistake description

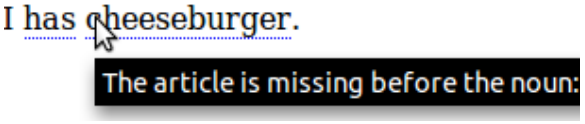
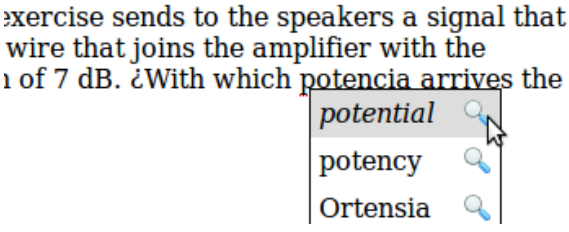


Figure 5: Mistake correction suggestions

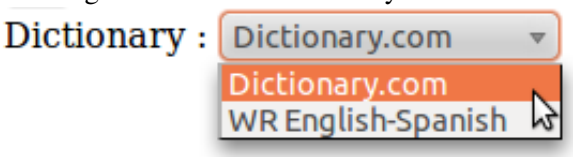


All checking is done on the server, using AJAX to update the text in place. The server uses two main tools to provide the translation: Aspell (Atkinson, 2004) for spell checking, and LanguageTool (Naber, 2003) (in its server configuration) for grammar checking. The `language.php` module written for this project handles reading and applying the output from both programs onto the text.

4.3 Link to external dictionaries

The user interface includes links to dictionaries next to all suggestions on mistakes, so that the user may easily find which one corresponds to the expected meaning. Whenever a dictionary for the language is available, a dictionary selection list is displayed under the text editing field, as shown in Figure 6, and links to the right page on this dictionary included next to all suggestions. These links are visible in Figure 5 (page 5). The interface selects available dictionaries by reading the OpenSearch (Clinton et al., 2005) XML files organised in a folder on the server, making it easy to add new dictionaries.

Figure 6: External dictionary selection



4.4 Search and replace

The interface contains search and replace forms to instantly search for a word throughout the whole text and replace it, according to a specific pattern. Available patterns are "Case sensitive", "Case insensitive" and "Apply source case". This last mode runs a case insensitive search, executes a short case analysis on each match and tries to imitate the case on the replacement. For example, when an "Apply source case" replacement is set for "herr" to "sir", any occurrence of "Herr" will be replaced by "Sir", and so on. This should allow speeding up the correction of long documents.

5 Project conclusion and possible evolutions

The main goals of this project have been reached, as it provides three major modules.

First, a basic interface for asynchronous translation, with document deformatting and reformatting and PHP Apertium interface on the server side, as well as a quick AJAX interface to update the text in place on client side.

Second, a set of tools to make translation more comfortable; mainly interfaces between the PHP script and other tools like LanguageTool and As-

pell, and the ability to display their result in a clean way in the translation environment.

Third, tools to log the edits made by the user during the translation process, including TMX generation through mALIGNa and a real time edition logging engine.

However, there are still a few bugs happening during edition, depending on the browser used. The next step for the project obviously is to correct them and, if possible, make it more cross-browser compatible, as it has been mostly designed and tested on Firefox so far.

Another goal for the future is to make it smoother, to feel more like a usual translation environment, by integrating common user interaction events and removing the limitations of the current interface, such as the undeletable characters generated by the formatted document handling module and so on. Copy/Pasting have just been implemented, but there are still other limitations that could be frustrating for the user.

A quite important progress with benefiting from user input would be being able to reuse TMXs of successful translations when making a new one. Unfortunately, the Apertium TMX compiler isn't quite usable yet, but it should be possible to integrate an external tool like OmegaT[9] to process part of the translation using the provided memories, before feeding the rest into Apertium.

At last, it could be useful to make this project more modular and adaptable. First, by reorganising the project's source code as thematic modules: some people might want a simple and light translation interface without many tools, and some people may not care about translation logging. For this project to reach a wider use, reorganising it as more clearly separated and more configurable modules would be useful. With the same idea, it would also be a good thing to leave more choice on what tools are to be used for all server-side tasks. For example, integration of After The Deadline[10] as an alternative to LanguageTool could prove useful.

[9] http://www.omegat.org/

[10] http://afterthedeadline.com/

Acknowledgements

Development for this project was funded by Google as part of the Google Summer of Code 2010 event; many thanks to Carol Smith and the GSoC team for organising it.

Thanks to the LanguageTool developers, in particular Marcin Miłkowski for his help with solving a few bugs.

Thanks also to the anonymous reviewers for their comments.

References

Atkinson, K. (2004). GNU Aspell manual.

Canals-Marote, R., Esteve-Guillén, A., Garrido-Alenda, A., Guardiola-Savall, M., Iturraspe-Bellver, A., Montserrat-Buendia, S., Ortiz-Rojas, S., Pastor-Pina, H., Pérez-Antón, P. M., and Forcada, M. L. (2001). The Spanish-Catalan machine translation system interNOSTRUM. *Proceedings of MT Summit VIII: Machine Translation in the Information Age, Santiago de Compostela, Spain*.

Chin, J. and Rosart, D. (2008). US Patent Application No. 20080195372: Machine Translation Feedback.

Clinton, D., Tesler, J., Fagan, M., Gregorio, J., Sauve, A., and Snell, J. (2005). OpenSearch 1.1 Draft 4. Technical report.

Doyon, J., Doran, C., Means, C., and Parr, D. (2008). Automated machine translation improvement through post-editing techniques: Analyst and translator experiments. In *Proceedings of the Eighth Conference of the AMTA*.

Forcada, M. L., Tyers, F. M., and Ramírez-Sánchez, G. (2009). The free/open-source machine translation platform Apertium: Five years on. *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation FreeRBMT'09, Alacant, Spain*.

Garrido-Alenda, A., Gilabert-Zarco, P., Pérez-Ortiz, J. A., Pertusa-Ibáñez, A., Ramírez-Sánchez, G., Sánchez-Martínez, F., Scalco, M. A., and Forcada, M. L. (2004). Shallow parsing for Portuguese-Spanish machine translation. *Language technology for Portuguese:*

shallow processing tools and resources, pages 135–144.

Hutchins, W. J. and Somers, H. L. (1992). *An Introduction to Machine Translation*. Academic Press, London, UK.

ISO (2006). Open Document Format for Office Applications (OpenDocument) v1.0. Technical Report ISO/IEC 26300:2006.

Jassem, K. and Lipski, J. (2008). A new tool for the bilingual text aligning at the sentence level. *Intelligent Information Systems*, pages 279–286.

LISA (2005). TMX 1.4b Specification. Technical report.

Llitjós, A. F., Probst, K., and Carbonell, J. (2004). Error analysis of two types of grammar for the purpose of automatic rule refinement. In Frederking, R. E. and Taylor, K. B., editors, *Machine Translation: From Real Users to Research*, volume 3265 of *Lecture Notes in Computer Science*, pages 187–196. Springer Berlin / Heidelberg. 10.1007/978-3-540-30194-3_21.

Masselot, F., Ribiczey, P., and Ramírez-Sánchez, G. (2010). Using the Apertium Spanish-Brazilian Portuguese machine translation system for localization. *Proceedings of the EAMT Conference, Sain-Raphaël, France*.

McIlroy, M., Pinson, E., and Tague, B. (1978). Unix time-sharing system forward. *The Bell System Technical J.*, 57(6 part 2):1902.

Naber, D. (2003). A Rule-Based Style and Grammar Checker. Diplomarbeit, Technische Fakultät, Universität Bielefeld, Germany.

Villarejo, L., Cullen, D., and Corral, A. (2009). La integració de les tecnologies de la llengua en el flux de treball del Servei Lingüístic de la UOC. *Llengua i ús, revista tècnica de política lingüística*, (46).