The Annotation System

Ronan Martin
SAS Institute
ronan.martin@sas.com

Abstract

Two of the main challenges of translation are comprehension and terminology: understanding the text, and knowing what to call things in the target language. The focus of this topic is the comprehension component, not so much "how to help translators in understanding what is meant by a text string", but more, "how to deploy the solution to translation queries so that all translators have access to the solutions when they highlight a string in the translation (CAT) tool". The CAT tools we use do have partial solutions, but we did not find these viable for different reasons. We needed a way of annotating source files once and for all. However, it was vital that we did not leave a footprint in our source files. Any footprint would lead to a breakdown at build (compilation) time.

The challenge: How do you create an external annotation that will always find its target string in the source files. We opted for a methodology borrowed from the terminology paradigm. Our source string was like a term, and the annotation like a term comment. The termbank became a stringbank, and the term dictionary an annotation dictionary.

1 Introduction

An easily understood text can be translated quite quickly. However, the more technical the content, the more challenging the task becomes. For an experienced translator, the main challenge is one of comprehension, and this can be due to terminology, lack of context, ambiguity, difficult syntax, or a host of other reasons.

Terminology is a domain that is widely recognized as being central to any translation process and the preferred goal in most cases is to try and deal with difficult terminology in a proactive way. All the other things that make a text difficult to comprehend, however, tend to be dealt with on an ad hoc basis. For example, you arrive at a sentence or a text string and discover that it doesn't readily render a translatable meaning and it is then that you set in motion queries which move upstream back to the author of the text, or to somebody who is likely to know what the text is trying to express.

So terminology is dealt with proactively and centrally and resolved terminology moves downstream to translators via interactive dictionaries. Other translation queries tend to be dealt with retroactively, with queries moving upstream from each translator back to the source. Resolutions to queries can be published or distributed, but generally there is no way to integrate them with CAT tools in the same interactive way that you can with terminology.

The Annotation System is a method we are developing to try and address translation challenges that are not specifically due to terminology. We are hoping that it will be possible to make the task of solving comprehension issues much more streamlined and proactive than it is at present.

2 Old Problem - New Idea - Design - Implementation

2.1 Background

At ELC (the European Localization Center at SAS) we localize SAS software solutions. SAS is a large software company (the largest privately owned software company in the world). Most SAS products are extensive software suites aimed at corporate customers and relate to

Proceedings of the 38th Conference Translating and the Computer, pages 113–121,
London, UK, November 17-18, 2016.

the application of analytics to, or statistical analysis on large volumes of data. Many products are tailored to a specific domain and can encompass advanced technical concepts. There are two principal components of the typical localization task: localizing the user interface (UI) and localizing support documentation. By far the most challenging task for translators is localizing UI files.

When UI text strings appear in source files they are *de facto* decontextualized. Their natural context is the interface in which they will appear, but for now they reside in properties files as string lists and are only retrieved at runtime. Translators have to translate these lists which run up to volumes of 10, 50, sometimes 150 thousand words.

By contrast the user documentation is more verbose and coherent. When authoring documentation, technical writers provide a rich linguistic context for the reader to follow, and translators benefit from this.

UI strings are written by developers. Developers are not renowned for being great linguists. In addition, some development may take place overseas by developers whose mother tongue is not the same as the source language they are writing strings in. A comprehensive review of linguistic quality is difficult, given the typical kind of development environment. Groups of strings can be altered or removed one day, only to be added back the next day. Components are moved around, re-written, revised and recoded right up until the point of code lockdown, when the release phase starts. Time to market is pressed by competitors releasing similar products and trying to get these out there first. Linguistic reviewers tend to get in the way of rapid development and it isn't here where a great number of resources are generally employed. Thus translators inherit lists of strings that are of mixed quality. There are cryptic acronyms and abbreviations, developer speak, camel-case words, typos, sequences of words written in telegraph style. In addition, many strings hold just one or two words, the reason for the brevity being that the text is destined for a field in a dialog pane or screen with limited space.

2.2 Current Querying Practices

As far as possible localization project managers try and provide background information to translators about the strings like existing technical papers and screen shots, and the documentation itself if this has been completed. Dictionaries are also provided that target terminology and chunk-sized phrases. But there are still many strings which need to be queried by the translator. These queries are routed back to the developer responsible for authoring the strings. A query is often a longer thread than just a question-answer pair. The developer doesn't always understand what it is about the string that is a challenge, or they may need to re-route the query to another expert.

At some point the query is resolved. Quite often the string is removed or re-written, but there are many resolutions where the developer provides important contextual information, or explains the terminology or acronym or strange wording. Once the solution is received, the translator can translate the string appropriately and continue with the work.

Now, consider that we translate to 30 languages. Each translator needs to be aware of queries that are currently in progress, pending answers, or that have been resolved. Otherwise they will end up sending the same query again to developers.

2.3 Possible Solutions

One possible solution is to somehow annotate the files. The developer or someone else could enter comments into the properties files. This is doable, and developers sometimes do this, but there are drawbacks. One is that the developer cannot know up-front, which strings are going to cause translators problems. And by the time translators get to the file, developers are usually close to signing off on them. Secondly, we need to imagine that when translators are

working quickly, comments are not always displayed in a timely manner. They can fall outside the small display window as translators jump from a string to the next translatable string. Sometimes translators use subset views and the comments become detached from the string they are referring to.

As regards translators annotating the files, this is too risky. Any editing of source files bears a risk of introducing code errors and only a developer or someone working close to the developer should do this.

How about using mechanisms available in the CAT tool for storing annotations, for example in the TM? Here there are also a few drawbacks which reduce the viability of this approach. It is possible to enter notes for individual segments in the CAT tool. These notes are generally saved together with the segment. However, the TM is language-specific. Therefore the note does not reach other translators. Also, when TM is applied, the translation from a previous version may be suggested, but the note is not automatically displayed. Translators need to actively open the TM segment in order to view most metadata relating to it.

These considerations led us to wonder about whether it was possible to associate the query-solution text with a given segment in some other way. The problem had existed for many years and we seemed no closer to a solution apart from advising translators to track all queries and to try and implement them if they are relevant for the product they are currently sitting translating.

2.4 Analogies with Terminology

In the following it is important that you understand the dictionary mechanism of CAT tools. When you give focus to a translation segment (usually a sentence or text string), any relevant terminology for that string is displayed in a terminology window. The terminology content changes as you jump from segment to segment.

Quite a few of the queries had been routed to me, the terminology manager, over the years as terminology queries. A few were bone fide source-terminology queries and I could place the solution in the dictionary so that it was displayed whenever translators came to it. Others were clearly not terminology queries but were similar in some ways. A problem with terminology is a special case of a general comprehension problem. Perhaps we could place the solution to these non-terminology queries in a dictionary as well. But then it occurred to us that these kinds of queries related to a specific instance of a string, not every occurrence as is the case for terminology.

However, if you extend the length of the "term" beyond the boundaries of the word or phrase being asked about, what were the chances that your sequence of characters was unique within the corpus for a given project. Investigations showed me that you didn't have to include a very long string before you had a unique identifier for that string - and in essence, the identifier could be the string itself. If you include this string in a dictionary, it will display an entry for that segment only. Instead of a target term you could just write a text that explained to translators how to interpret the string in order to translate it.

This meant that we could annotate a document, and keep the annotations separately. The link would be a string reference which was the string, or a subset of the string itself. On the basis of a few tests we decided to pursue this option.

Taking the analogy of terminology a bit further, it seemed what we needed was a stringbank which would be the storage repository for annotations. The dictionary would be the delivery mechanism. Currently we already generate two kinds of xml-based term dictionaries for translators to use with a project. The annotation dictionary would just be a third one of these.

2.5 Components

We had envisaged a storage method and a delivery mechanism. Now we needed to set up an environment that supported a workflow where:

- queries were processed and resolved
- resolutions were entered in an annotation system (repository)
- an editing interface was provided
- an annotation-dictionary build process was made available
- there was an annotation selection mechanism that only returned relevant annotations for a given project

We already had a good query system which was just being set up and overhauled, and this provided a good opportunity for incorporating interaction between the query system and the new annotation system.

We wanted the whole process to be devolved rather than centrally steered, with self-service features wherever possible.

2.6 Processing and Resolution of Queries

The purpose here is not to describe the query system, which in many ways is a standard type application. It is mail based. When a translator runs into problems, they can enter a question into the query system. Some values need to be set indicating which project the query relates to, where the string comes from etc. After this the query is submitted. A "ticket" is set up, and mails are sent to interested parties. In this case we predefine interested parties to include the developers who have written the code.

A mail thread ensues between the translator and the developer. This may consist of two mails, question/answer, or can extend to as many as a dozen exchanges. Eventually the translator feels that they have received a satisfactory answer and the ticket is closed.

Previously we requested that all in-house translators follow all mail threads and monitor them for relevance. In the case of outsourced projects, the localization project manager for a project usually had to meticulously compile a list of resolved queries that related to a project, and send this list when they handed off a project to a language vendor.

2.7 Entering Query Solutions into the Annotation System

As we were in the process of implementing a homegrown query system, this gave us some freedom in designing some interfaces. For example, when a query is created it is possible to take some basic data and metadata and bundle this into a set of properties/values that can later be passed to the annotation system. The vehicle we chose was to add the property/value list to a URL that opened the Add View of the Annotation System. Something like this:

http://koelcterm.sdk.sas.com/add_string_info.htm?string=Remediation%%2ALead%%2A&product=\vertical\Products\RiskAndCompliance\Monitor\Source\mrm\Config\Deployment\Content\Preload\Config::d4grc62&properties_key=linkType.MRM.finding_remediationLead.name1.txt%3D&scope=MRM&author=spnmxc

It looks cumbersome, but this hyperlink was neatly embedded into the query ticket. When the query is resolved, the person who is in charge of the query (generally the person who

opened it), clicks on a pre-agreed word and this opens up the Annotation System on the "Add View" page.

Below is an example of the top of a query (which grows downwards as comments are added). I have highlighted the word String. This is the pre-agreed word that translators can click on to open the Annotation System web page and enter a new annotation. The hyperlink behind this word is the one above.

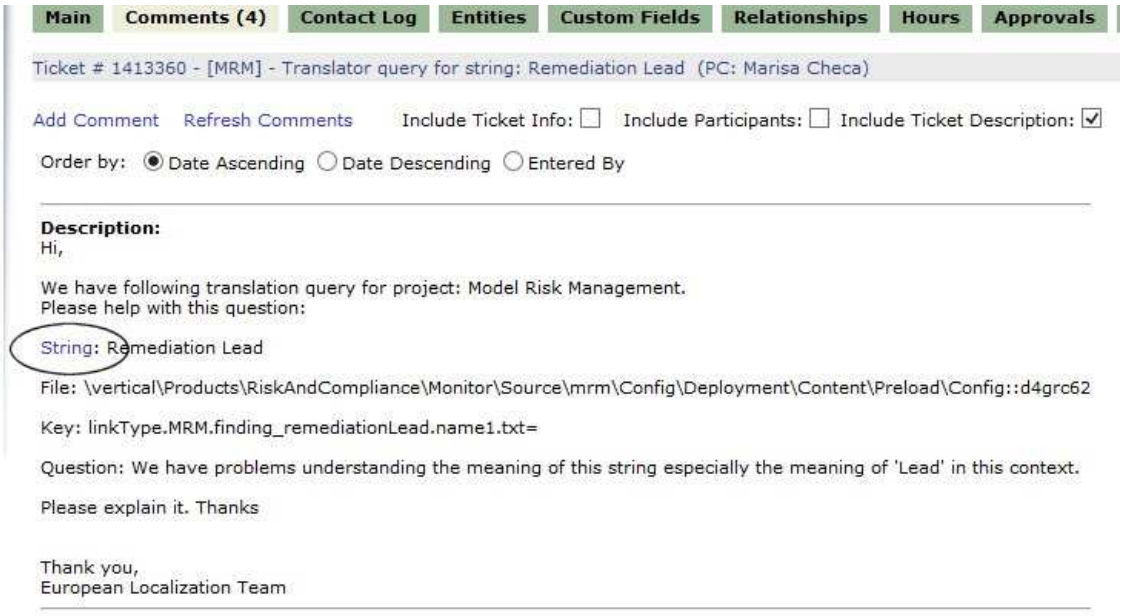

Figure 1: Ticket in the Query System

The Add View of the Annotation System is a simple web page containing a form. Thus, from the query it is possible to click on the URL, have the form displayed, and have many of the fields pre-populated. Javascript in the receiving page unwraps the parameters that are passed with the URL and inserts these in the relevant fields. Fields will display the troublesome string itself, the project it relates to, the key associated with the string in the source file (for UI properties files most strings appear as key/value pairs, where the value is the string), the sender etc.

Here is an example:

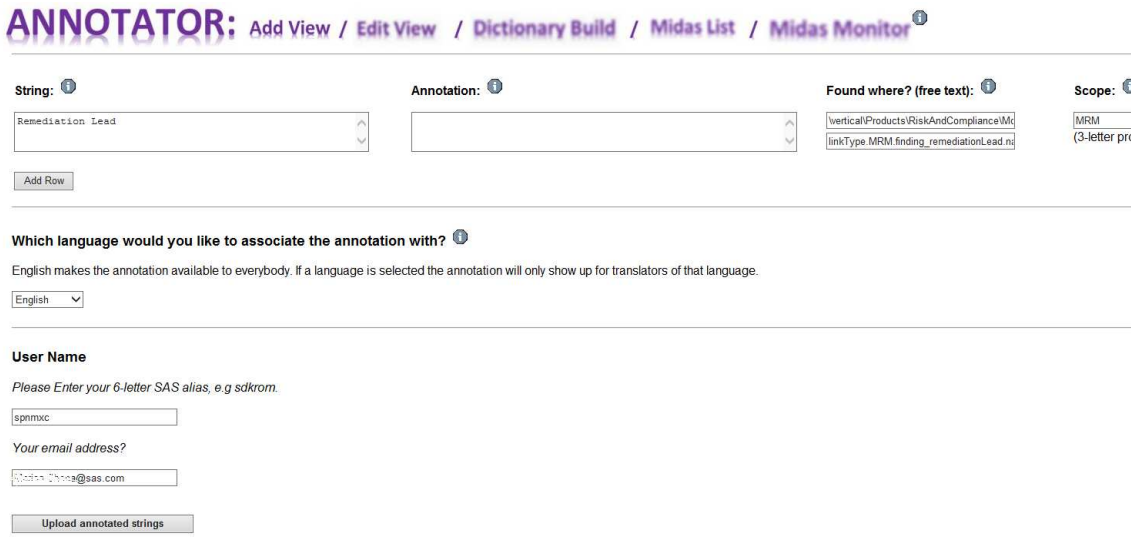

Figure 2: Example of the Add View

The only thing that isn't included is the resolution itself which must be placed in the Annotation field manually. Usually this can be copy/pasted from the mail thread for the relevant query. All of these things can be edited before submitting the form. When submitted, a new string/annotation entry is set up in the Annotation System repository.

2.8 Editing Interface

When a repository of this kind is being built up, it is important to give contributors the opportunity to edit their entries. The Edit View displays a list of existing annotations, reflecting the contents of the annotation repository. The string and the annotation are editable, and can be uploaded once more to overwrite the existing previous entries.

The interface contains various types of filter and subsetting so translators can locate the precise query they are looking for.

2.9 Dictionary Build

The whole point of the Annotation System is that a list of annotations relating to the source files for a localization project can be attached to the project, and thus warn or advise translators about certain segments when they bring these into focus in the CAT tool.

Looking ahead, I was concerned about the number of annotation entries that may build up over time. I didn't want to distribute the whole list every time a project was started. In fact, I was faced with a number of questions. How often should annotation dictionaries be built, how could I limit their size.

The Annotation System and the query system change from day to day. Queries only start to roll in once a localization project has been launched. Thus, it wasn't enough to be content with creating an annotation dictionary when you sat down to start localizing a project. After a few days many resolutions may have resulted in new annotations. The dictionary had to be small, and of a throw-away variety, because translators may have to apply the dictionary several times during the localization, and probably also at the end.

The dictionary build interface is simple enough. It is possible to build an annotation dictionary for any project which has annotations associated with it.

2.10 Ensuring only Relevant Strings are Returned

What goes on behind the scenes is a little more complex. When a dictionary is requested I need to ensure that any annotation associated with it, does in fact relate to a string that is still part of the project. Strings become removed from projects, and I didn't want the size of the dictionary to be swollen with obsolete strings. For this reason the build script needs to retrieve the very latest version of the source files and perform a concordance search between the source strings and the annotated strings in the repository.

Once created the dictionary is zipped and made available through an ftp link, and can be downloaded and attached to a project.

3 What we discovered along the way...

3.1 Embedded Dictionary Entries Hidden ☹☹

We are to an extent restrained by our CAT tool. We have a clear idea about what we would like to achieve, but this has to be squeezed through the functionality that is available in our CAT tools. As we are using a terminology tool to surface the annotations, certain challenges arise.

The first major challenge is that our current tool gives precedence to the longest match, when comparing dictionary entries and the source segment. If there is a match for both a single-word term, and a multi-word term, and the one-word term is embedded in the longer term, then only the multi-word term entry is displayed in the terminology pane. The other entry is hidden away. This means that an annotation entry, which is like a long term, always hides any term entries that may occur in the same string.

To date we have no fix for this, but we are in the process of changing CAT tools and this phenomenon does not occur in our new CAT tool. As a workaround we have advised translators to have dedicated sessions where they work with the annotated entries, and otherwise only attach term dictionaries.

Interestingly, I discovered straight away that the dictionary only displayed entries if there was a target term. Initially I placed the annotation in the comments field for the dictionary entry and left the target term field blank. This didn't work and instead I entered a "-" as a kind of dummy target for all languages. This was enough to get the record displayed when you gave focus to a segment.

Because of this it is possible to run a Terminology Check, which quickly runs through all the segments and checks to see whether the target term has been applied to segments where the annotation matches all or part of the source string. If it finds a target hasn't been applied, it stops and prompts the translator. Thus it stops at all segments that have an annotation entry. Here translators can just check whether they have heeded the information, then jump to the next segment where there is a hit.

At any rate, our collection and storage mechanism is tool-independent. It is only at the delivery level we have this reliance. This leaves us with possibilities of overcoming many of these problems in creative ways as we move forward.

3.2 Language-Specific Entries ☺

Originally the system was only intended for general annotations. However, once we started up, we noticed that some query/annotation workflows were language-specific. The query system allows translators at external vendors to open queries which are sent to our in-house translators, or in-house experts. For example, which of two translation targets is preferable. These types of query relate to one language only and the resolution is very important, but only relevant for that language.

Luckily the possibility of dealing with language-specific annotations was available to us. The dictionary format we use when creating dictionaries is MARTIF (a predecessor of TBX). What I decided to do was to place the source string where the source term would have been, have the annotation as an entry-level comment, and any language-specific annotations as the target term.

Dictionaries are still multi-lingual so one size fits all for one particular project. However, when translators work with files, they will have their target language selected, and they will only see target entries for this language. Everyone will see the general annotations, but translators will only see the language-specific annotations for their language.

This mechanism opened up the possibility of storing language specific notes and annotations that weren't necessarily the resolutions to queries, but just the result of a decision made while translating.

3.3 Short and Common-Word Entries ☹

It quickly became apparent that there were problems with a certain type of string. For example, things like "Current", "Home", "Issue", "none". Queries had been entered for strings like this. They are complete segments because in the source files, there is a discrete string with this single word. If I allowed these strings to go into the dictionary they would create false positive hits in very many segments, and these annotations nearly always related to one specific segment only.

My first thought was to investigate whether the dictionary match could be made dependent on the properties key. For example, this is what the string may look like in the source file:

```
rules.ruleSasTest.js.none.txt=None;
```

The key is on the left-hand side of the equals sign. In our translation windows, the key is protected and cannot be edited. The dictionary entry only matches the editable part of segments. So, the possibility of using the key was not open to us.

In the end I had to devise an alternative workaround. Instead of writing the strings/annotations to the MARTIF file only I also wrote them to a CSV file that was packaged together with the dictionary. Here I could include the problem string together with the key and the annotation, like this:

None	"None" means the used variables value is an empty string.	rules.ruleSasTest.js.none.txt	Only printed here - not in dictionary

Translators were asked to make sure and check these entries as well, which wouldn't automatically be displayed to them while translating.

3.4 Annotations Up-Front ☺☺

As an extra bonus we now have the possibility of being truly pro-active. In cases it is possible for a reviewer to work with developers and try and improve the quality of their strings. The reviewer has a very small window in which to operate. Changes are recommended for some strings, while there is not time enough to change others. The reviewer can also query the developer about what kind of context the string will be displayed in. Any explanatory information about strings is now added to the Annotation System. Thus when translators receive the files and view the string in question, they now have vital information that will help them translate it correctly and quickly. And this may be strings that are handed off to translators in up to 30 locations simultaneously.

4 How do things look at present...?

4.1 Savings

We feel pretty sure that this system is providing us with savings on several fronts. The savings are difficult to document - we are in the process of gathering data so that we can analyze some of the effects. The system has been in operation for about six months and we have been slowly rolling it in, becoming fully operational about 3-4 months ago. At this point we have 775 annotation entries and this grows at a linear pace.

The main saving lies in the fact that only one translator sends a query, and enters the annotation in the system if this is appropriate. The other translators reap the benefit of this, and because of the dictionary mechanism they can be sure that they will not inadvertently fail to implement the solution.

Translators have a safe mechanism for storing annotations about strings they translate. It may be important in future to remember why you chose a particular target string, or perhaps it will be another translator reviewing the string in future.

Overall quality probably improves. Tricky issues in the form of ambiguous source strings are less likely to result in a bad translation. It can be easy to misinterpret many common words like "issue" (issue shares, or is issue just a problem?), "apply" (for a license, or apply a strategy?) , "policy" (company policy or insurance policy?). If just one translator notices the ambiguity, queries it and gets a solution, all translators will be made aware of it.

In-house translators are more likely to query company-specific concepts, but external translators may not even know that the concept differs from the normal usage of the word(s). Our in-house translators always get the source files first. We can now be sure that their discoveries will be passed on to translators of languages we normally outsource.

And last but not least, often we "roll in" a new language which starts localizing the product from scratch. Here the benefits will be considerable as we will be able to hand off to language vendors a fully "annotated" set of files, saving many time-consuming queries for our PMs or developers.

4.2 Some Closing Thoughts

The system needs to be scalable - time will tell whether we will need to introduce changes when the volume increases. The repository will grow in size, and at some point we may need to start tracking whether certain strings have gone away for ever. However, the dictionaries should remain at a stable size, perhaps growing slowly into an optimum size for each product over time.

I would like to see translators making more use of the language-specific annotation possibilities. This entails being able to share with others various reasons for choosing certain types of translation targets over others. This is especially important when there is a collaboration between in-house and external translators, but could be important if localization projects change hands between translators within the same language.

I've been pleasantly surprised with the involvement that our translators have shown in this new system. I couldn't really say the same for our Terminology Management System in the early days, where it took some time to get people on board. I think the degree of involvement is a measure of the relevance of the system for the daily work of translators.