The Impact of Visual Contextualization on UI Localization

Luis A. Leiva and Vicent Alabau
PRHLT Research Center, Universitat Politècnica de València
{luileito,valabau}@dsic.upv.es

ABSTRACT

Translating the text in an interface is a challenging task. Besides the jargon and technical terms, many of the strings are often very short, such as those shown in buttons and pull-down menus. Then, as a result of the lack of *visual* context in the traditional localization process, an important ambiguity problem arises. We study three approaches to solve this problem: using plain gettext (baseline condition), using gettext plus being able to operate the UI, and translating the UI in-place. We found that translators are substantially faster with plain gettext but commit a significantly higher number of errors in comparison to the other approaches. Unexpectedly, the mixed condition was slower and more error-prone than in-place translation. The latter was found to be comparable to plain gettext in terms of time, although some strings passed unnoticed as the UI was operated. Based on our results, we arrive at a set of recommendations to augment localization tools to improve translator's productivity.

Author Keywords

Localization; L10n; Internationalization; i18n; Translation

ACM Classification Keywords

H.5.2 User Interfaces: Prototyping, Interaction styles; D.2.2 Design Tools and Techniques: User interfaces

INTRODUCTION

Designing products and services that are intended to support more than one language is a challenging task. Translation is just one of the activities of software localization, yet the most important overall [1, 2]. So much so that when a product is translated into a new culture it becomes a *new* product [7].

On the UI, some texts may be confusing or offensive for a particular cultural target. Even more, UIs often introduce additional subtleties such as jargon, specialized words, and technical terms. This can be an obstacle for speakers of lesser known tongues because of a non-familiar vocabulary [4], and may prevent the access to cultural resources that are otherwise available in other languages.

What is more important, many of the strings that have to be translated are often very short, such as those shown in buttons and pull-down menus. Therefore, an important ambiguity problem arises as a result of the lack of visual context in the traditional localization process. A string may appear in more than one situation on the UI with completely different translations, e.g., the same word may behave as a verb when placed in a button, but also as a noun if attached to a label. Misinterpretation of these differences could result in major issues for the users of the application [5, 9].

It has been shown that context brings systematic variation in cognitive functioning and performance [6]. Hence, coupling translations with the UI from which all messages were drawn should help translators to better disambiguate, and today there is software to achieve this effect. However, we are not aware of any empirical evaluation of this capability among the research literature. As such, the impact of visually contextualized localization approaches still remains largely unexplored. We therefore contribute to HCI with a rigurous and comprehensive study on the influence of the visual context on translation quality and translator's productivity.

BACKGROUND AND RELATED WORK

UI localization is the process of translating and adapting (source) texts in UI buttons, menus, or headings, in order to support different (target) languages or "locales". To achieve this goal, these source texts must be decoupled first from the source code of the UI, basically by wrapping each string with a translation-capable function. A very popular tool to do so is GNU gettext, although other utilities such as Qt Linguist follow similar principles. Then, strings are exported to a file so that translators can use either a regular text editor or specialized applications; just to name a few: Pootle, Crowdin, Verbatim, and Launchpad. However, the main problem with these applications is that translators often do not have access to the UI from where the source texts were extracted.

Some general-purpose translation tools such as TRADOS or Google Translator Toolkit support visually contextualized translation, although they are oriented to translate *content* like text documents or static web pages. This means that most UI elements cannot be localized, e.g., buttons, placeholders, drop-down lists, etc. For that reason, a number of more sophisticated approaches to support UI localization have risen recently. For instance, dedicated localization tools for the desktop include Passolo, Catalyst, or RCWintrans; which seldom support more than one programming language but allow to perform contextualized translations. A more interesting approach is ScreenMatch [3], a platform-independent system to assist software translators by showing UI screenshots alongside each translatable string.

In addition, it is worth mentioning that what all these tools display as UI is not the actual UI end users will operate, just a static visualization, and may therefore be outdated over

CHI 2014, April 26–May 1, 2014, Toronto, Ontario, Canada.

ACM 978-1-4503-2473-1/14/04..$15.00.
http://dx.doi.org/10.1145/2556288.2556982

time. Consequently, we believe that translators should be able to access the actual UI being translated. In this regard, applications like SmartLing or Drupal translations, featuring a gettext-like interface alongside the current web-based UI, may help to overcome such limitation. However, this capability has not yet been formally assessed.

EVALUATION

Intuitively, being able to access the actual UI should provide translators with better disambiguation capabilities and thus improve the quality of the produced translations. This should come at the expense of an increase in the time spent localizing, since localizable elements must be located on the UI and sometimes operated. Also, some elements may not be obvious (or even possible) to locate. Such is the case of error messages or dynamic content that disappear after a short amount of time. However, if translations are properly contextualized the final outcome would result in fewer errors to amend, resulting hopefully in higher productivity. Therefore, we formulated the following hypotheses:

H1. Visual context improves localization quality.
H2. When the UI is available, translations are finished later.
H3. In-place localization leaves strings untranslated.
H4. Visual context improves translator's productivity.

To test these hypotheses, we studied three UI localization possibilities, ranging from no visual context at all to a pure visually contextualized translation approach (see Design section). We developed a small airline website (Figure 1), by drawing inspiration from actual airline websites. The source language was English, and we decided to localize the website into Spanish. We noticed that some lexical ambiguities were evident. For instance, 'delayed' is a neutral word in English but can be both masculine and feminine in Spanish, depending on the context of the string. However, in the absence of context, it is usually translated using its masculine form.

Participants

Thirty participants aged 24–37 ($M = 30.3$, $SD = 4.1$) were recruited via email advertising. All were native Spanish speakers with an advanced English level according to CEFR[1] and had had previous experience as translators. Participants performed the assigned tasks at a dedicated laboratory room, and were rewarded with a gift voucher at the end of the evaluation.

Design

Participants were assigned to exactly one condition (between-subjects design). People were randomly split into 3 groups of 10 users each: one group would use a gettext-based application (GT condition), other group would use gettext plus being able to operate the UI when desired (GTVIS condition), and the other group would use in-place translation (VIS condition). An alpha level of .05 was used for all statistical tests. There were no departures from normality (verified by the Shapiro-Wilk test, n.s.) or homoscedasticity (verified by Levene's test, n.s.). Hence, we used the one-way ANOVA test to compare the differences between the 3 conditions, followed by post-hoc pairwise comparisons where applicable.

[1] `http://www.coe.int/lang-CEFR`

Apparatus

The website was coded in PHP and was manually internationalized using PHP's `gettext()` function. This way, all translatable messages could be automatically extracted using the `xgettext` command. Overall, the website comprised 92 strings of 1.99 words each on average ($SD = 1.87$).

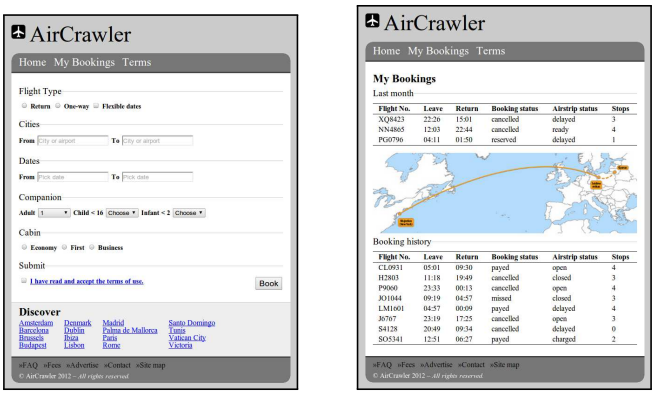

Figure 1: Screenshots of the website used in the study. The website comprised 5 sections in total.

For the baseline approach, GT, we used the traditional way of localizing interfaces at present, which is based on the GNU gettext model; i.e., the text is decoupled from the UI and is delivered as a PO file to the translator, who then uses a dedicated localization tool. Typically, such localization tools feature two sections: one that displays the resource strings (the texts in the PO file) and other that allows the translator to enter the target strings (the translated text); see Figure 2.

For the mixed approach, GTVIS, each string was augmented with a link to the page where such string appeared. Links opened in a dedicated browser tab so that, if desired, the translator could browse each page and see the context of the task.

For the in-place translation approach, VIS, we instrumented the calls to the PHP `gettext` function to mark the localizable elements. Then, using JavaScript, these elements would be edited on the very same UI. By CTRL+clicking on an element, a pop-up dialog asked the user to enter the translation, which was then stored in a plain text log file.

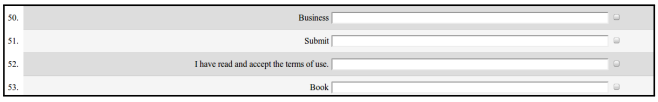

Figure 2: A typical localization interface. This is in turn the tool we used for GT and GTVIS conditions.

It was ensured that the airline website would be as similar as possible to that of an actual airline company, i.e., there were city names, dates, or loan words such as 'premium' or 'business'. As such, some messages were not required to be translated. For these cases, it was assumed that a VIS user would not submit a translation. Then, to align the capabilities of the other groups to that of this group, the gettext editor featured a checkbox near each target input field. This way, by clicking on a checkbox, GT and GTVIS users would indicate that the source string should not be translated (Figure 2).

Procedure

Each participant was briefly introduced to the localization approach assigned to them for about 5 minutes, including a motivating discussion on the importance of software localization. No restrictions were set regarding time to complete the task, using auxiliary tools such as online dictionaries, and so on. All participants were given the following statement: *"You have been hired to localize the website of an airline company. You are expected to deliver a translation as professional as possible."* By giving this high-level articulation, we expected that our hypotheses would be clearly assessed: being able to navigate the pages of the site while localizing them would provide GTVIS and VIS users with enough context to disambiguate properly. Therefore, these users should produce better translations than GT users, who were not shown the site. VIS users were expected to perform slower than the other participants, since the UI had to be operated at any time in order to translate the strings. However, it was expected that their translations would be of the highest quality, since the visual context would be always available.

For **H1**, quality was measured with the Human-targeted Translation Edit Rate (HTER): the number of word post-edits required to change a submitted translation into a valid reference [8]. For **H2**, we measured the time to complete the task (in minutes); whereas for **H3**, we counted the number of strings edited. Finally, for **H4**, we evaluated translator's productivity as follows. In the translation industry, productivity is measured in words per day. This assumption is reasonable if the provided translations are of high quality, and thus an additional round for reviewing them is not required. However, in UI localization some translations might be irresolvable without a proper context, regardless of how professional the translator is. As a result, more than one reviewing round may be needed. Hence, we define *raw* productivity as the number of *correct* words per day. We should emphasize that, since all wrongly translated strings would require to be re-submitted in a real-world localization workflow, *actual* productivity would be much affected by the turnaround time, which typically spans from 2 to 7 business days [5].

RESULTS

While inspecting the data, we found a GT user that took an unusual amount of time to complete the task. This participant took exactly 46 minutes to complete the task, while the rest of GT users spent 13 minutes on average, and therefore was removed from the analysis. He was asked about this unusual delay, and stated that *"translations can change completely according to the context of the sentence on the website. In addition, I found an important problem while determining the gender of some words, since in English adjectives are neutral but in Spanish they can be masculine, feminine, or both. [...] I had to browse other airline websites to give a more accurate picture of how to translate these sentences properly."*

Overall, GT users translated 81.11 strings ($SD = 1.76$) in 12.73 minutes ($SD = 5.03$), with an HTER of 34.60% ($SD = 6.06$). The remaining strings were accepted by clicking on the checkboxes, revealing that all strings were reviewed by all GT users. As shown in Figure 3a, it becomes clear that gettext-based approaches allow to entirely localize the website, since all strings are shown to the translator. Meanwhile, GTVIS users localized 79.88 strings ($SD = 3.95$) in 33.44 minutes ($SD = 12.99$). On average, they consulted the website 56.55 times ($SD = 42.64$), which accounted for 20% of the total task time. (Time spent on the translatable website was measured by looking at the focus/blur events on that website.) Then, the remaining time was spent either on the gettext editor or at auxiliary websites such as dictionary or thesaurus sites. This time HTER was more competent, 19.14% ($SD = 8.22$). On the other hand, VIS users localized 71.6 strings ($SD = 4.92$) in 19.84 minutes ($SD = 8.26$), with a very competent HTER of 10.79% ($SD = 3.64$).

We observed that problematic sentences are often those with few words, as there is an important ambiguity problem. In general, shorter sentences (up to 2 words) are more error-prone. In this regard, we found a moderate correlation: the longer the sentence, the more likely it will be successfully translated with GT [$r(292) = -0.34, p < .001$] or GTVIS [$r(300) = -0.29, p < .001$]. With VIS, the correlation is smaller [$r(269) = -0.21, p < .001$] because ambiguities can be properly solved with contextual information. In the next section we provide some examples of sentences that can be easily translated with and without visual context.

ANOVA revealed that differences between groups were statistically significant for all measures, i.e., HTER [$F_{2,25} = 35.84, p < .001, \eta_p^2 = 2.87$], time [$F_{2,25} = 11.52, p < .001, \eta_p^2 = 0.92$], number of localized strings [$F_{2,25} = 17.52, p < .001, \eta_p^2 = 1.40$], and raw productivity [$F_{2,25} = 8.77, p < .001, \eta_p^2 = 0.70$]. Effect sizes suggested a high practical significance. Post-hoc pairwise comparisons using the t-test with Bonferroni correction revealed that GT translations were of significantly worse quality than GTVIS, and VIS produced the highest quality results. Unexpectedly, no time differences between VIS and GT were found. GTVIS users were significantly slower than the other approaches, and VIS localized significantly less strings than GT and GTVIS. As expected, no differences between GTVIS and GT were found in this regard, since both groups saw all localizable strings. Productivity results are discussed next.

DISCUSSION

It can be observed that GT allowed users to complete the task sooner than the other approaches (Figure 3b). This was unsurprising, as gettext aims for a completely sequential workflow while the other approaches required to interact with the UI. However, we must remark that less time does not necessarily mean higher productivity. Actually, if translation errors are found then more iterations between translators and developers are likely to happen, which would lead to higher turnaround times for companies [5]. For instance, assuming that all translation errors would be fixed after a second pass over the GT data, participants would have needed on average 10 additional minutes, which is comparable to the time GTVIS users spent browsing the UI. This is nevertheless an upper bound, since not being able to access the UI would probably result in more than one review cycle for GT users.

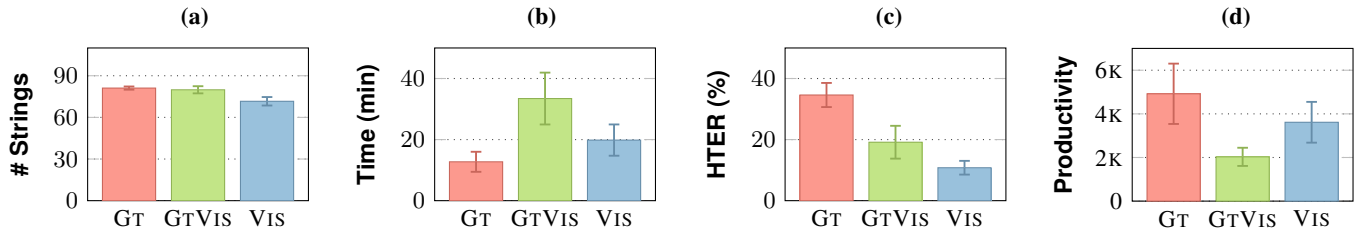

Figure 3: Evaluation results. Error bars denote 95% CIs. Productivity (3d) is estimated as the number of correctly translated words per day.

On the contrary, a visually contextualized localization tool facilitates the disambiguation of messages that depend on the UI context for a proper interpretation, leading to higher-quality translations (Figure 3c). As a result, the number of iteration cycles and refinements over the localization process would be dramatically shortened. This is interesting for software localizers, as well as for companies that decide to go global, since time is often a scarce resource and translation errors can be exceedingly costly. Unexpectedly, GTVIS quality was significantly worse than VIS. While the visual context is always available to VIS users, in GTVIS it is the translator's decision to check the context by inspecting the UI, which was often skipped to save time (29% of the strings in our data).

A deep analysis of the productivity reveals that VIS is statistically comparable to GT, and so both can produce the same number of correct words per day (Figure 3d). The reason is that HTER is much higher in GT but in VIS some strings were not localized. A particular example were error messages, which tend to have more words on average (e.g. 'Please fill in the required form fields'), and thus their actual meaning is easier to notice in the absence of visual context. In general, messages composed by one or two strings should be translated with visual context, as their meaning depend on the UI element; e.g. buttons ('Book'), labels ('One way', 'Display unit'), or date fields ('From', 'To'). More practical examples can be found in Muntés-Mulero *et al.* [5, p.3].

On the other hand, GTVIS is significantly less productive than the other approaches. In fact, we observed that VIS leads to significantly faster translation time if compared with GTVIS. As hinted by 3 participants, *"[gettext] strings are not sorted by page, so I had to switch often from page to page, which may slow down the overall task time."* Indeed, if we discount the time GTVIS users spent searching for the strings on the UI, then there would be no differences between GTVIS and VIS, as indicated by the Bonferroni-corrected pairwise t-test.

In consequence, we argue that pure visually contextualized localization tools should be complemented with a gettext-like interface to reach *all* resource strings. On the other hand, while screenshots are a very useful approach to allow translators to get an overview of the visual context of the task, we recommend gettext-like interfaces to be complemented with pointers to the actual UI elements to safely save lookup time. This could be implemented by e.g. dynamically generating screenshots were each resource string is highlighted whenever the translator is localizing it on the gettext editor. Currently, the UI context is provided via textual descriptions, or not at all [3]. Therefore, we hope researchers and practitioners will consider this work when developing new software localization tools.

CONCLUSION

This paper puts forward the fact that the impact of visual contextualization on UI localization can be considerable, leading to significant effects on translation quality and translator's productivity. In short, when the visual context is available to the translator, better translations are produced. Additionally, raw productivity is comparable to the traditional workflow if translations are edited in-place. Furthermore, if the translator needs to locate the UI elements without an additional aid, productivity may decrease significantly. We thus recommend current localization tools be augmented with pointers to the actual UI elements, clearly highlighting them, if possible.

ACKNOWLEDGMENTS

This work is supported by the 7th Framework Program of the European Commision (FP7/2007-13) under grant agreements 287576 (CASMACAT) and 600707 (tranScriptorium).

REFERENCES

1. Dunne, K. J., Ed. *Perspectives on Localization*. John Benjamins Publishing Company, 2006.
2. Keniston, K. Software localization: Notes on technology and culture. Working Paper #26, MIT press, 1997.
3. Kovacs, G. ScreenMatch: providing context to software translators by displaying screenshots. In *Proc. CHI EA* (2012).
4. Leiva, L. A., and Alabau, V. An automatically generated interlanguage tailored to speakers of minority but culturally influenced languages. In *Proc. CHI* (2012).
5. Muntés-Mulero, V., Adell, P. P., España-Bonet, C., and Màrquez, L. Context-aware machine translation for software localization. In *Proc. EAMT* (2012).
6. Nardi, B. A., Ed. *Context and consciousness: Activity Theory and Human Computer Interaction*. MIT Press, 1996.
7. Russo, P., and Boor, S. How fluent is your interface? designing for international users. In *Proc. CHI* (1993).
8. Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. A study of translation edit rate with targeted human annotation. In *Proc. AMTA* (2006).
9. Sun, H. Building a culturally-competent corporate web site: an exploratory study of cultural markers in multilingual web design. In *Proc. SIGDOC* (2001).