Context-Aware Machine Translation for Software Localization

Victor Muntés-Mulero
Patricia Paladini Adell
CA Technologies
WTC Almeda Park
08940 Cornellà de Llobregat, Barcelona
Victor.Muntes@ca.com
Patricia.PaladiniAdell@ca.com

Cristina España-Bonet
Lluís Màrquez
TALP Research Center
Universitat Politècnica de Catalunya
cristinae@lsi.upc.edu,
lluism@lsi.upc.edu

Abstract

Software localization requires translating short text strings appearing in user interfaces (UI) into several languages. These strings are usually unrelated to the other strings in the UI. Due to the lack of semantic context, many ambiguity problems cannot be solved during translation. However, UI are composed of several visual components to which text strings are associated. Although this association might be very valuable for word disambiguation, it has not been exploited. In this paper, we present the problem of lack of context awareness for UI localization, providing real examples and identifying the main research challenges.

1 Introduction

Due to the rapid and worldwide development of Internet and IT applications, fast software localization is becoming essential, requiring *user interfaces* (UI) to be translated to different languages. One of the main obstacles when translating UI is the *word sense disambiguation* problem since strings are usually independent from other strings in the UI and, therefore, it is not possible to infer semantic information from other parts of the text.

In this paper, we want to show that the meaning of a string in this environment varies depending on its position in the UI. For instance, a word associated to a *menu* may be interpreted as a name, but it may be an action if the same word appears on a *button*. Although enriching the translation process with this alternative contextual information would benefit quality, previous software localization techniques ignore, in general, this approach.

It is commonly accepted that the number of words processed per day by a human translator is significantly increased when an efficient machine translation (MT) engine is used and human translators intervene in the post-editing phase. Specifically, it is becoming popular to use MT engines for software localization. Unfortunately, even if contextual information about the UI components associated to strings was gathered, current localization procedures using MT engines are not devised to absorb and exploit it to improve MT quality. Visually aided translation tools, like Passolo[1] or Catalyst[2], leverage contextual information and show it graphically to human translators. However, they depend on specific file formats which are not always available. Improving the quality of the output of MT allows both (i) to reduce the cost of translation by increasing translator's throughput by up to 50%, based on CA Technologies[3] experience, and (ii) to reduce the delay to market of the software products. The objective is simultaneous shipment.

The main contributions of this paper are as follows. Section 2 describes the most relevant state of the art. In Section 3, we define the problem of the lack of contextual information in UI localization providing examples extracted from real products of CA Technologies. In Section 4, we enumerate the main research challenges in terms of improving the quality of the output of MT by increasing the context awareness for UI localization. Finally, Section 5 concludes this paper.

[1] www.passolo.com
[2] www.alchemysoftware.ie
[3] CA Technologies is a worldwide software and solutions provider that helps customers to make ICT management more agile, secure and flexible. The company localizes many of its applications to several languages, using MT techniques and human post-editing.

2 Previous Work

Incorporating MT in the software localization process has been the focus of recent projects. For instance, Ruopp (2010) adapts well-known open source translation engines. Also, Hudik and Ruopp (2011) integrate them into computer-aided translation tools. However, to our knowledge, none of these previous works make use of context extracted from UI.

In general, most MT systems, translate text sentence by sentence independently, ignoring broader contextual information. Even at sentence level, a statistical system based on segments or phrases (phrase-based SMT, (Koehn et al., 2003)) uses the source lexical context of phrases only locally, considering a limited number of words next to the phrase being translated. Because of this, the discourse at document level is not considered.

Syntax-based SMT (Chiang (2005) among many others) tries to alleviate the lack of connection between long distance phrases by considering syntactic dependencies, still within a sentence. Also, factored models (Koehn and Hoang, 2007) include linguistic information in phrase-based models as extra factors associated to words. This information can be anything that can be codified, although the most extended use is to employ morphology to generate translations from the lemmatized text. An alternative way to consider context is by using word sense disambiguation techniques to choose between possible translations of a word or a phrase. In general, these approaches use machine learning methods to learn an adequate word selection model (see for example Giménez and Màrquez (2008)). None of these advances in standard phrase-based SMT tackles the context-aware problem in UI translation.

3 Lack of Context Awareness

In this section, we describe the overall process of UI software localization in an industrial environment and describe the problem of the lack of context in UI translation.

3.1 UI Software Localization Process

Figure 1 depicts a high level overview of the process for UI localization used at CA Technologies. A first common aspect that is important to remark is that, especially in large enterprises, programming and localizing are not only performed by separate human teams, but this work is usually done in different departments in very large and complex development organizations. As a consequence, in many cases direct collaboration between them is not straightforward due to different time zones or due to the fact that they might be using different and complex tools, highly specialized for their day-to-day tasks. Even worse, it is common that some of the UI to be translated might be coming from recently acquired software or part of the localization might be outsourced to third-parties. In addition, the skills and expertise of developers and translators are usually completely different. While developers are not expected to have comprehensive English language skills, translators are not supposed to interpret the source code of applications. As a result, development and localization are usually decoupled, their interaction is in general very complex and, in addition, translators rarely have access to the source code.

Usually, different tools are provided to help developers generating code which is compliant with internationalization requirements (step 1 in Figure 1). These tools are devised to ensure, for instance, that text appearing in the code adheres to the basic formating rules, required in the localization process to digest and translate the text properly. Once a new product or release is ready, the source code is parsed and the text in the UI is extracted for localization (step 2 in Figure 1). First, text is run against translation memories in order to leverage previous translations (step 3). Second, the remaining strings are run through MT engines to obtain a machine translated output in the target language (step 4) that will be post-edited by human translators (step 5). This is one of the most time-consuming steps in the localization process since it consists in manually (or semi-automatically) editing the MT engine outputs in order to produce publishable content. The output is then passed to an automatic tool that prepares the new translated text to be inserted back to the original source code (step 6). Because in many cases human translators do not have access to any view of either the final layout or the source code of large and complex application, and therefore the UI components where each string is associated, they cannot guarantee a correct translation. As a consequence, it becomes necessary to perform a critical iterative step that we call *Language Quality Assurance* (LQA). This process is usually highly resource-consuming and requires programmers to generate a sample of evidences, such as screenshots, to allow translators to validate the translation in context. If errors are reported, they have to be solved by developers in an iterative

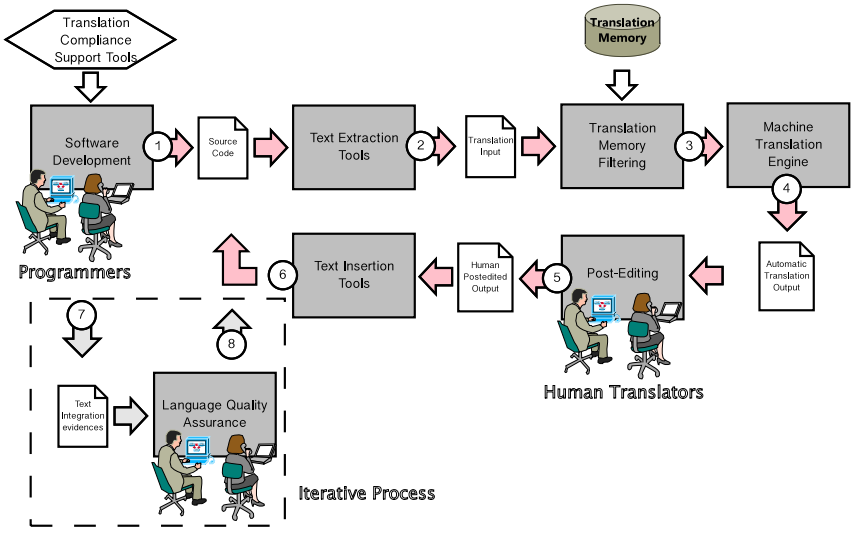
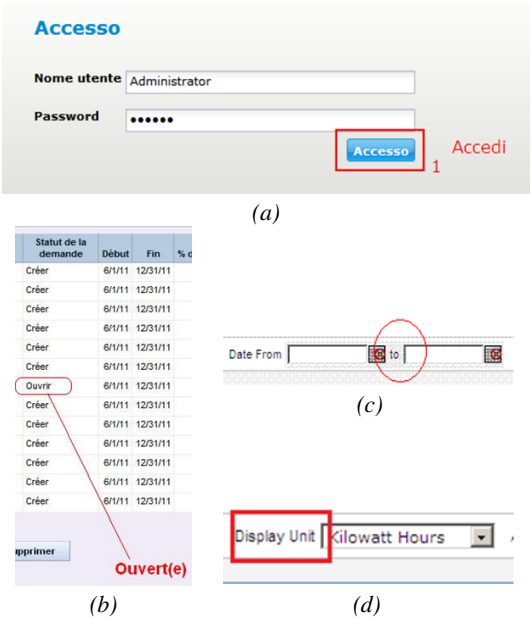


Figure 1: UI Software Internationalization and Localization Process at CA Technologies

procedure. This costly process is very inefficient and, therefore, expensive.

3.2 Context Description and Examples of Lack of Context

We define context as the minimum required information needed to solve an ambiguity. This contextual information should be added to the raw strings sent for translation. We classify the different types of ambiguities found in our scenario in four different categories: (i) *part of speech*: this is one of the ambiguities requiring solution and it is needed in order to provide an accurate translation (Figure 2.a). In this example, the source text was "*Access*", which in English can be a verb (to access) or a noun (an access). However, as the text is embedded into a button, it has to be translated as a verb, in this case "*Accedi*" (verb in Italian) instead of "*Accesso*" (noun in Italian). In most cases, this ambiguity can also be solved by providing the UI element in which the text will be showing up (a button, a menu, a dialog box header, a drop down list, etc.); (ii) *gender*: this is the most difficult ambiguity to solve as the gender will always depend on a different element. For example, in some languages, the gender of a word included in a table cell will depend on the gender of the column header (Figure 2.b). In the example, you can see two overlapped ambiguities: first, the original English word "*Open*" could be a verb or an adjective, and it has been translated automatically as a verb ("*Ouvrir*") while it should be translated as an adjective ("*Ouvert*") and, second, the gender of the adjective[4] will depend on the gender of the title in the column header (in English "*Request Status*" is translated into French as "*Statut de la demande*" which is feminine), in this case, "*Ouverte*"; (iii) *prepositions*: prepositions like "*to*" or "*from*" always need context information for disambiguation (Figure 2.c). For example, the word "*to*" has at least three possible interpretations: destination, recipient or date, and in Spanish this would be translated to "*a*", "*para*" and "*hasta*", respectively; (iv) *syntactic ambiguity*: caused by word order in English: "*Display Unit*" might be translated to "*Unità*



(a) (b) (c) (d)

Figure 2: Examples of lack of context effects

[4]Note that adjectives in roman languages are affected by number and gender.

di visualizzazione" (unit to be displayed) or "*Visualizza unità*" (to display a unit) (Figure 2.d).

4 Research Challenges

In this section we summarize the main challenges posed by the ambiguities identified.

Adapting MT engines to exploit contextual information: MT engines must be improved in order to handle UI contextual information and improve quality. As a first approximation, we envisage writing a set of rules. This way the system is informed so that it translates, for instance, an ambiguous word as a noun in a menu and as a verb in a button. The validity of this approach depends on the degree of ambiguity and the coverage of the rules. A more competitive method could be adding the context as factors in phrase-based SMT. This way, it is different to translate *(archive, noun, menu)* from *(archive, verb, button)* or *(archive, noun, button)*. Within this framework the translation is not selected by a rule, but each alternative translation has a probability estimated from frequencies in a corpus, and the translation of a word is also conditioned by the translation of the neighboring words.

Although the move towards a probabilistic approach ensures a high coverage, it might not be enough to solve some kinds of ambiguities. The information needed to properly translate the gender and number of a text might be encoded by several types of context at the same time, so it is necessary to deal with a high number of features. Factors are not appropriate for a large number of features, but machine learning techniques can be used to learn the best translation according to its context codified with those features. This methodology has been already successful as stated in Section 2.

Context extraction and internationalization-compliant programming standards: all of these approaches assume that the context can be extracted from the code. Besides, those which rely on statistical methods also need to gather an annotated corpus. A second line of research, thus, will involve establishing automatic methods to extract context information from the source code. Challenges range from parsing the information of complex UI components, such as tables, where content in the table header might affect the translation of the text in the cells for instance, to defining programming standards that make the code compliant with localization needs or creating new tools that aid developers to use writing style guidelines that make the localization process easier.

There may be several ways of including contextual information in the files sent for translation: (a) to include information of the UI components in which the ambiguous text will be embedded, next to ambiguous words; (b) for recurrent pre-defined ambiguities like "*To*" and "*From*", provide a predefined standard explanation of the context, like for example, "*To*" as in "*date*", or "*To*" as in "*e-mail*"; (c) in case none of the previous options works, to allow for a free text option to provide information necessary for disambiguation. Any of these possible solutions require establishing practical methods that do not overload developers with unnecessary extra work and, specially in the last case, sophisticated methods to extract information from free text.

5 Conclusions

Specializing MT engines used in software localization processes is vital for reducing costs both in terms of time and budget. However, to our knowledge, the problem has not been tackled yet. Reducing the mistranslations produced by the lack of context will have a direct impact on both the pots-editing phase and the LQA phase, which are the most costly phases in such a process. Therefore, it is necessary to include UI components information in the localization process. We strongly believe that near future research efforts should be pointed towards these types of solution.

References

Chiang, D. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 263–270, June.

Giménez, J. and L. Màrquez, 2008. *Discriminative Phrase Selection for SMT*, pages 205–236. NIPS Workshop Series. MIT Press.

Hudik, T. and A. Ruopp. 2011. The integration of moses into localization industry. In *15th Annual Conference of the EAMT*, pages 47–53.

Koehn, P. and H. Hoang. 2007. Factored Translation Models. In *Proceedings of the Conference on EMNLP*, pages 868–876.

Koehn, P., F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the HLT/NAACL*, Edmonton, Canada, May 27-June 1.

Ruopp, A. 2010. The moses for localization open source project. In *Conference of the AMTA*, October.