A Solution for in Context and Collaborative Localization of most Commercial and Free Software

Amel FRAISSE [1, 2], Christian BOITET[1],
Hervé BLANCHON[1], Valérie BELLYNCK[1]

(1)GETALP, LIG
Joseph Fourier University,
385 rue de la Bibliothèque,
38041 Grenoble cedex 9, France
{Amel.Zairi, Christian.Boitet, Herve.Blanchon}@imag.fr
(2)Entreprise WINSOFT
24 rue Louis Gagnière
38950 Saint Martin Le Vinoux, France
afraisse@winsoft.fr

Abstract

We propose a novel approach that allows *in context* localization of most commercial and open source software. Currently, the translation of textual resources of software (technical documents, online help, strings of the user interface, etc.) is entrusted only to professional translators. This makes the localization process long, expensive and of poor quality because professional translators have no knowledge about the context of use of the software. This current workflow seems impossible to apply for most under-resourced languages for reasons of cost, and quite often scarcity or even lack of professional translators. Our proposal aims at involving end users in the localization process in an efficient and dynamic way: while using an application (in context), users knowing English could right-clicks on strings of the user interface to translate or improve translations proposed by machine translation (MT) or translation memory (TM) systems. To implement this new paradigm, we modify the code as little as possible, very locally and in the same way for all software. Hence our localization method is *internal*. We have experimented our approach on Notepad++, an open source software. This has allowed us to localize, in context, 95% of the strings of the user interface. The rest are strings that are hard coded in the source code.

Keywords: Software localization, in context localization, internal localization

1. Introduction

Currently, the translation of technical documents as well as strings of the user interfaces is entrusted only to professional translators. In practice, software editors send original versions of documents to several professional translators. Each translator translates and sends the translated versions to the editors. But, it seems impossible to continue in this way for most "new languages", for reasons of cost, and quite often scarcity or even lack of professional translators (costs increase while quality and market size decrease).

On the other hand, free software like Mozilla (Mozilla, 2005) is translated by volunteer co-developers in many languages (70), more than commercial software. The software localization is based on the contribution of volunteers (Vo-Trung, 2004), (Tong, 1987), (Lafourcade, 1991, 1996). Another situation (different from the translation of technical documentation) is that of occasional volunteer translators, who contribute without an organic connection with the project (Linux, 2005). Hence, it is possible to obtain high quality translations of documents over a hundred pages long (such articles of the Wikipedia encyclopedia, texts of Amnesty International and Pax Humana). Another problem of the classical process of translation is that strings of the interface are translated out of their context. In fact, the relationships of proximity of text fragments in time and space are inaccessible to the translator, while the context in which a text is read contains much information to which the text refers. Hence, the choice of the appropriate translation is not always possible out of context, and even a professional translator cannot produce a perfect translation. This is one of the major problems identified at the L4Trans-III workshop of LREC-06 by the person responsible (Kudo-san) of the localization of CATIA at IBM-Japan.

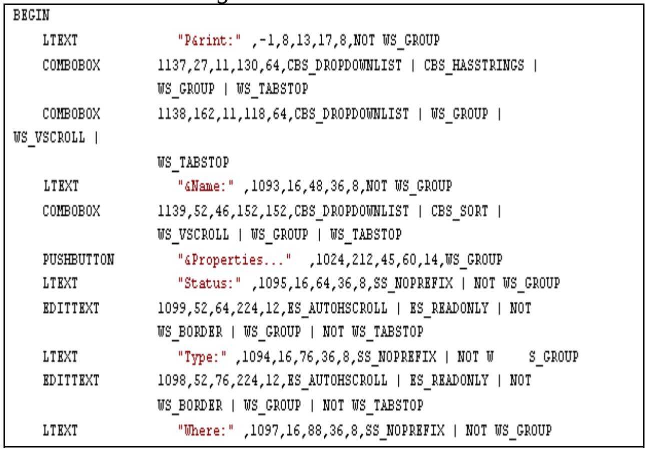
As proposed in (Boitet, 2001), one solution to this problem is to involve the end users knowing English and who, during the use of software products, translate or improve some translations proposed by machine translation systems (TA) or translation memory (MT) systems.

2. Current localization process

2.1. Description

The software localization process includes the localization of (1) user interfaces (dialog boxes, menus, buttons, error messages, commands…), (2) technical documentation provided with the software (installation guides, instruction manuals, training guides…) and (3) online help. In this article, we are interested only in the localization of the user interfaces.

To facilitate software localization, and sometimes also for reasons of confidentiality, most developers of commercial and free software separate strings of the user interfaces or/and the user interface code from the source code of the application. In fact, most developers use resource scripts to create user interfaces. The most known and used resource script is the RC resource script. An example of RC resource script is given in *Fig. 1* which describes the user interface of *Fig. 2*.



```
BEGIN
    LTEXT              "P&rint:" ,-1,8,13,17,8,NOT WS_GROUP
    COMBOBOX        1137,27,11,130,64,CBS_DROPDOWNLIST | CBS_HASSTRINGS |
                    WS_GROUP | WS_TABSTOP
    COMBOBOX        1138,162,11,118,64,CBS_DROPDOWNLIST | WS_GROUP |
WS_VSCROLL |
                    WS_TABSTOP
    LTEXT              "&Name:" ,1093,16,48,36,8,NOT WS_GROUP
    COMBOBOX        1139,52,46,152,152,CBS_DROPDOWNLIST | CBS_SORT |
                    WS_VSCROLL | WS_GROUP | WS_TABSTOP
    PUSHBUTTON         "&Properties..."  ,1024,212,45,60,14,WS_GROUP
    LTEXT              "Status:" ,1095,16,64,36,8,SS_NOPREFIX | NOT WS_GROUP
    EDITTEXT        1099,52,64,224,12,ES_AUTOHSCROLL | ES_READONLY | NOT
                    WS_BORDER | WS_GROUP | NOT WS_TABSTOP
    LTEXT              "Type:" ,1094,16,76,36,8,SS_NOPREFIX | NOT W     S_GROUP
    EDITTEXT        1098,52,76,224,12,ES_AUTOHSCROLL | ES_READONLY | NOT
                    WS_BORDER | WS_GROUP | NOT WS_TABSTOP
    LTEXT              "Where:" ,1097,16,88,36,8,SS_NOPREFIX | NOT WS_GROUP
```

Fig. 1: Resource file of an American commercial software

Hence, to localize software interfaces, we have to localize the resource file(s) of the software. The current localization process consists of five main stages:

***Extraction and analysis of user interface strings*:** localizers extract all user interface strings from the resource files and store them in an Excel file called *glossary*. To extract user interface strings from resource files, localizers can use automatic extraction tools such as RCWINTRANS, POWERGLOT, etc. In order to reduce translation costs, localizers use translation memories such as TRADOS to analyze the contents of the glossaries. This allows them to find strings that have already been translated (strings with 100% match) in earlier versions of the software, and compute the matching rate of the rest. Indeed, the cost of translation depends on the matching rate: for example, it is cheaper to translate segments with a 90% matching rate then segments with 10%.

Translation: once analyzed, localizers send the glossary to professional translators. Some localization services give to translators the terminology database of the software in order to ensure consistency of translation between different versions of the same software.

Revision: localizers send the translated glossary to reviewers to review and check translations done by translators. Generally, reviewers have a better knowledge of the context of use of the software, so that they are able to correct translations that seem wrong.

Compilation: once all strings are translated, localizers re-inject them in the resource files and compile the software to generate a β-version that will be tested before being published.

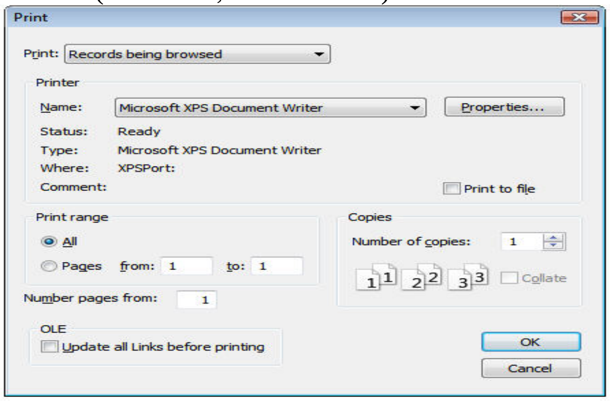
Test: there are two types of test:

Internal test: performed by localizers, it consists in verifying the proper functioning of the software.

External test: it is provided by LQA companies (Linguistic Quality Assurance) or β-testers that are simply users who volunteer to test the product. They are not paid by the editors, but are often given a free product license.

Any anomaly detected by the β-testers is reported to the publisher. Editors generally have an online platform that allows testers to capture the following information:

- o Name of the tester who detected the bug
- o Date
- o Nature of the bug (display problem, inconsistency of translations for MacOS and Windows…).
- o How to find the interface that contains the bug (screenshot, instructions…).



Print

Print: Records being browsed

Printer
Name: Microsoft XPS Document Writer
Status: Ready
Type: Microsoft XPS Document Writer
Where: XPSPort:
Comment:
☐ Print to file

Print range
● All
○ Pages from: 1 to: 1

Number pages from: 1

Copies
Number of copies: 1
☐ Collate

OLE
☐ Update all Links before printing

Fig. 2: “Print” dialog box created by the script resource file of Fig. 1.

2.2. Problems of the current localization process

It is out of context: professional translators and reviewers have no knowledge about the context of use of the software.

It is not incremental: the software is published when it is totally localized.

It causes long delays between updates: all modifications are done by release. Hence, once the software is published, translations cannot be modified.

It does not involve a variety of contributors: currently, the different contributors on the localization process are professional translators, reviewers and testers.

End users are totally excluded from the localization process, although they have the capacity to participate effectively, since they have a better knowledge of the context of use of the software.

3. New paradigm of internal and in context localization

We propose a new paradigm that will permit the end users to take part in the localization process in an efficient and dynamic way: while using the software, the end users who know English can translate or improve the already existing translations. However, the editor may ask professional translators and reviewers to translate the crucial parts of the software.

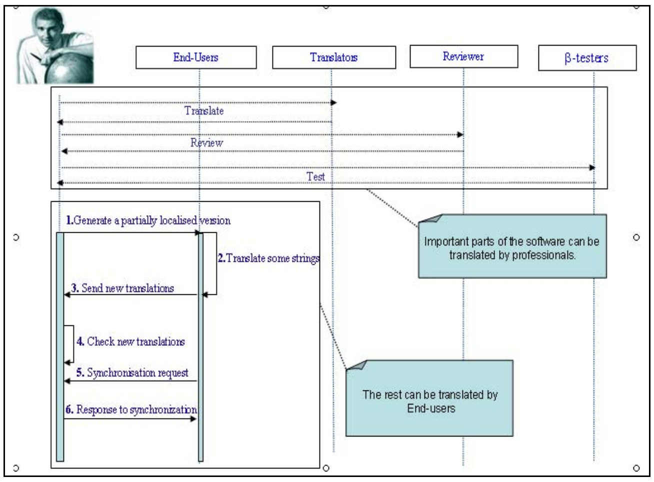

End-Users
Translators
Reviewer
β-testers
Translate
Review
Test
1.Generate a partially localised version
2.Translate some strings
3. Send new translations
4. Check new translations
5. Synchronisation request
6. Response to synchronization
Important parts of the software can be translated by professionals.
The rest can be translated by End-users

Fig. 3: Sequence diagram describing the in context localization process

The distinct characteristics of the new localization paradigm (*Fig. 3*) are as follows.

Contextual mode: the translation of the strings of the user interfaces is done from the application. This allows a better translation quality and that is mandatory to get end users to contribute.

Multiple participants: As well as the professional translators, reviewers and β-testers, the end users may also participate in the localization process.

Quick update: for each new translation proposed by the user, the interface is updated in real time. The synchronization with other contributors is made when the periodic updates of the application are complete.

Incrementality: the new process permits the incremental augmentation of both quality and quantity. The software editor can publish a partially localized version that will progressively improve with the contributions of the end users.

Minimal intervention on the code: in order to put the new paradigm in place, we need to modify the source code of the software. Hence, our process is one of internal localization. To be as generic as possible and to modify the software source code as little as possible, our modifications are made uniquely on the basic classes that produce interfaces. These modifications consist to add to strings of the user interfaces behavior adapted to the *in context* translation: by a simple right-click on a string of the interface, we can choose from a list of possible translations and we can also add our own. The end user can also translate any string of the interface, which is then updated in real time.

4. In context localization of applications

4.1. Illustration of the new scenario on Notepad++

We have conducted a complete experiment with Notepad++ (free software programmed in C++). It is a code of reasonable size (60000 sources lines) compared to other software such as FileMaker and Photoshop that we have studied previously. This allowed us to try his internal localization ourselves, without relying on external collaborators.

Take the example of the dialog box ‘ColumnEditor ‘of Notepad++:

- The user right-clicks on the string "Text to insert", which allows him to edit the string.
- She/he enters his new translation, or chooses one of the proposed translations.
- She/he clicks on the “Localize” button.
- The interface is updated in real time.

Our approach does not handle messages with variables (Boitet, 2005), but that is part of our planned evolutions.

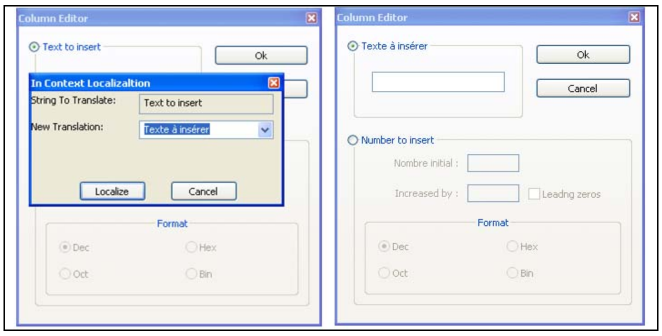


Fig. 4: In context localization of the ‘ColumnEditor’ dialog box

4.2. Internal localization of Notepad++

As mentioned above, to enable the *in context* localization of applications, we need to perform an internal intervention on the source code. To be as generic as possible and modify the source code as little as possible, our modifications are done only on the base classes that generate all GUI of the application.

4.2.1. Architecture

In the case of Notepad++, there are two main generic classes that produce all user interfaces of the application: ‘StaticDialog’ and ‘StaticMenu’. Thus, we integrated our module in these two generic classes.

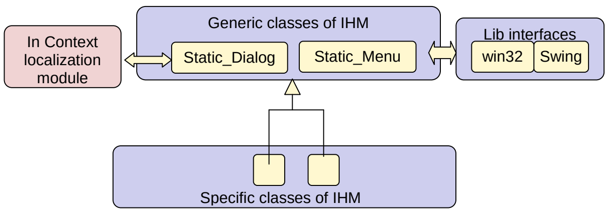
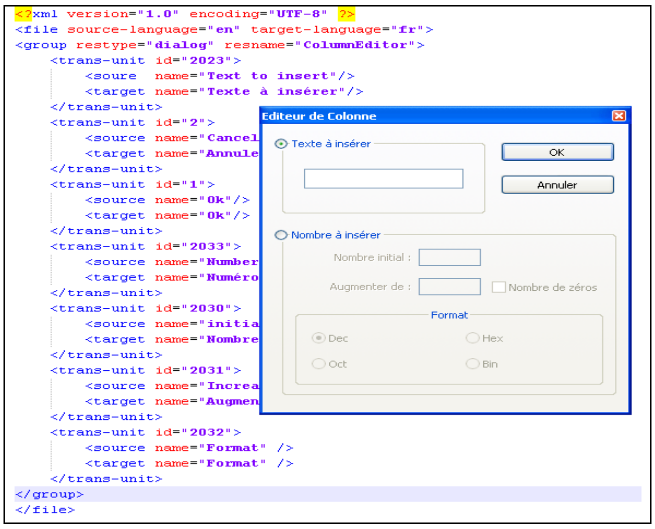


Fig. 5: Integration of the “in context localization module” on the Notepad++ architecture

4.2.2. Exchange Format

XLIFF is the XML Localization Interchange File Format designed by a group of software providers, localization service providers, and localization tools providers. It has been created to standardize localization. XLIFF itself has been standardized by OASIS in 2002 (OASIS, 2002). This format can be applied to documents that can have complex structure (in fact, documents with any DTD), and is perfect for simple cases of textual elements of the programs (strings of the user interfaces), and other documents accompanying software. Hence, we use the XLIFF format as exchange format to communicate with the application. In fact, all new translations proposed by users are stored in the XLIFF file. Hence, when a user clicks on the “Localize” button, the application loads the new translation from the XLIFF file and updates the interface. At runtime, before displaying any interface element, the program checks whether there is a new translation in the XLIFF file.



```
<?xml version="1.0" encoding="UTF-8" ?>
<file source-language="en" target-language="fr">
<group restype="dialog" resname="ColumnEditor">
    <trans-unit id="2023">
        <soure  name="Text to insert"/>
        <target name="Texte à insérer"/>
    </trans-unit>
    <trans-unit id="2">
        <source name="Cancel
        <target name="Annule
    </trans-unit>
    <trans-unit id="1">
        <source name="Ok"/>
        <target name="Ok"/>
    </trans-unit>
    <trans-unit id="2033">
        <source name="Number
        <target name="Numéro
    </trans-unit>
    <trans-unit id="2030">
        <source name="initia
        <target name="Nombre
    </trans-unit>
    <trans-unit id="2031">
        <source name="Increa
        <target name="Augmen
    </trans-unit>
    <trans-unit id="2032">
        <source name="Format" />
        <target name="Format" />
    </trans-unit>
</group>
</file>
```

Fig. 6: Structure of the XLIFF file

4.3. Interactions

The application interacts with our *in context localization module* during the edition of the user interface strings by end users and during the update of the user interfaces.

4.3.1. Edition of user interface strings

When the end user right-clicks on a string of the interface, the application retrieves and sends the string identifier to the *in context localization module*.

This displays the *in context localization dialog box* containing by default the source string and the different propositions of translations available in the XLIFF file. Once editing is finished, the XLIFF file is updated and the application is notified to update the user interface.

4.3.2. Update of the user interface

The graphical libraries that are widely used with Java are: Swing, AWT and SWT. With C++, it is Win32 (Microsoft, 2008) (on Windows).

The operating principle of these libraries (Java, 2008), (Sun, 2000), (Microsoft, 2008) is the same: each interface is represented in memory by a data structure. At runtime, the application loads all interfaces and especially strings of the interface from these data structures. Thus, updating the user interface causes updating of the corresponding data structure. Once editing is complete, the *in context localization module* updates the XLIFF file and requests the application to refresh the user interface containing the edited string. Then, the application replaces in the data structure the value of the string by the new translation proposed by the user.

5. Collaborative localization

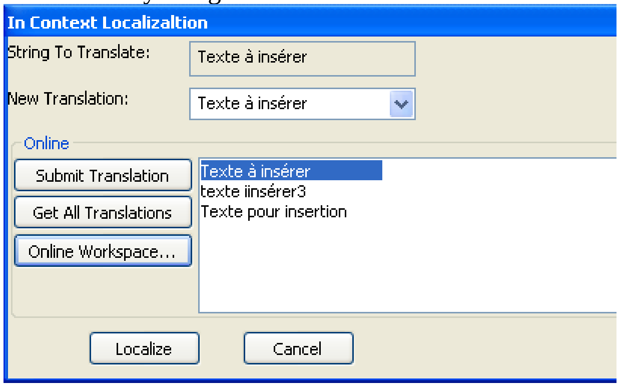
5.1. Triangular localization

The establishment of such a localization process requires the intervention of 3 entities: the software editor, the collaborative online platform (Eneko, 2000), (Kageura, 2007), (Bey, 2006, 2008) and the end user.

The software editor manages the global quality of the localization of the software. He can check and validate all strings of the software. He can also promote a new localized (complete or not) version of the software for the end user without having to wait for the next release.

The collaborative platform manages the registration of the various end users. It collects their submissions and stores all the information concerning users, submitted translations and software editor validations.

The end user directly localizes, in context, the software he is currently using.



In Context Localizaltion

String To Translate: Texte à insérer

New Translation: Texte à insérer

Online

Texte à insérer
texte iinsérer3
Texte pour insertion

Fig. 7: “In context localization” dialog box connected to collaborative platform

5.2. Interactions between the entities

The software editor validates the submitted translations directly on the collaborative platform. He is assisted by the comments of the users, their profiles, and their scores. The promotion of a totally or partially localized version is also realized directly on the collaborative platform.

The end user submits the localized strings to the collaborative platform. S/he can see all the contributions stored in the collaborative workspace, and can get access to translation memories and specialized dictionaries. S/he can also get a specified validated string, or get a localized version promoted by the software editor.

5.3. In Context and Collaborative Localization of Notepad++

We have used an online collaborative platform developed by Huynh Cong P. of our team.

As shown in *Fig. 7*, the user can submit a new translation, get the contribution of other end users, and directly access the collaborative platform.

6. Conclusion

We have proposed a novel approach that allows *in context* localization of most commercial and open source software. The current translation workflow, based on the exclusive recourse to professional translators and synchronized delivery of all localized versions at the same time, seems impossible to apply for most under-resourced languages for reasons of cost, and quite often scarcity or even lack of professional translators.

In our *in context* approach, end users are involved in the localization process in an efficient and dynamic way: while using an application (in context), users knowing the current language of the GUI can right-click on strings of the GUI to translate or improve translations proposed by machine translation (MT) or translation memory (TM) systems.

As we have to modify the source code to change the behavior of the GUI, our localization method is *internal*. However, our changes are minimal, very local, and can be done in the same way for all software.

We have experimented our approach on Notepad++, an open source software. The first author has been able to localize, in context, 95% of the strings of the user interface. In the near future, we plan to set up an experiment involving the same software, but one or more other target languages, with a group of volunteers for each, to evaluate the gains in terms of delay and quality, and get feedback. Another perspective is to adapt our technique to applications written in Java and using the: Swing, AWT or SWT libraries.

Acknowledgments

This work has been mainly funded by a CIFRE ANRT scholarship. Thanks also to WinSoft for allowing us to go deep into the current practice of professional localization.

References

Boitet, C. (2001). *Four technical and organizational keys for handling more languages and improving quality (on demand).* Proc. MTS2001, IAMT 8 p., Santiago de Compostela, September 2001.

Boitet, C. (2005). *Message Automata for Messages with Variants, and Methods for their Translation.* Proc. CICLING-05, LNCS 3406, pp. 352—371, Mexico.

Eneko, E. (2000). A Methodology for building Translator-oriented Dictionary Systems. *Machine Translation 15,* pp. 295

Vo-Trung, H. (2004). *Méthodes et outils pour utilisateurs, développeurs et traducteurs de logiciels en contexte multilingue,* thèse d'informatique, Institut National Polytechnique de Grenoble.

Kageura, K. and Abekawa, T. (2007). *QRedit: An Integrated Editor System to Support Online Volunteer Translators.* Digital Humanities, pp. 3-5.

Tong, L.C. (1987). *The Engineering of a translator workstation*, Computers and Translation, pp. 263-273.

Lafourcade, M. (1991). ODILE-2, un outil pour traducteurs occasionnels sur Macintosh. *Presses de l'université de Québec*, Université de Montréal, AUPELF-UREF ed. pp. 95-108.

Lafourcade, M. and Sérasset, G. (1996). Apple Technology Integration. A Web dictionary server as a practical example. *Mac Tech magazine*, 12/7, pp. 25.

Bey, Y., Boitet, C. and Kageura, K. (2006). *The TRANSBey Prototype: An Online Collaborative Wiki-Based CAT Environment for Volunteer Translators.* LREC 2006-Fifth International Conference on Language Resources and Evaluation. Genoa, June 2006.

Bey, Y., Kageura, K. and Boitet, C. (2008). *BEYTrans: A Wiki-based Environment for Helping Online Volunteer Translators.* In Topics in Language Resources for Translation and Localisation, Yuste Rodrigo, 135–150.

Bey, Y., Kageura, K. and Boitet, C. (2006). Data Management in QRLex, an Online Aid System for Volunteer Translators. *International Journal of Computational Linguistics and Chinese Language* Processing, 11/4, pp. 349-376

Web references

Java™ Platform, Standard Edition 6 API Specification retrieved from: http://java.sun.com/javase/6/docs/api. Access date: 2008.

Linux documentation translation. Traduc project retrieved from: http://traduc.org/. Access date: 2005

Mozilla project & Mozilla French Localization project retrieved from: http://frenchmozilla.online.fr/. Access date: 2005.

Sun Microsystems Inc. Building International Applications, documentation Sun. retrieved from: http://docs.sun.com/db/doc/806-6663-01. Access date: 2000.

Win32 and COM Development retrieved from: http://msdn.microsoft.com/en-us/library. Access date: 2008.

XLIFF 1.2 Specification retrieved from: http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html Access date: 2009