MajecSTIC 2009
Avignon, France, 16—18 novembre 2009

Localisation interne et en contexte des logiciels commerciaux et libres

Amel Fraisse[1, 2], Christian Boitet[1], Hervé Blanchon[1], Valérie bellynck[1]

1 : GETALP, LIG-campus, BP 53, 385 rue de la Bibliothèque, 38041 Grenoble Cedex 9, France.
2 : Entreprise WinSoft, 24 rue Louis Gagnière, 38950 Saint Martin Le Vinoux, France

Contact : Amel.Zairi@imag.fr, Christian.Boitet@imag.fr,
Herve.Blanchon@imag.fr, Valerie.Bellynck@imag.fr

Résumé
Nous proposons une méthode permettant la localisation *en contexte* de la majorité des logiciels commerciaux et libres, programmés en Java et C++/C#. Pendant qu'ils utilisent l'application, les utilisateurs connaissant l'anglais peuvent traduire certains éléments textuels de l'interface comme les boutons, les menus, les étiquettes, les onglets…, ou améliorer la traduction proposée par des systèmes de traduction automatique (TA) ou de mémoire de traductions (MT): la localisation se fait donc *en contexte*. Pour cela, nous modifions le code minimalement, très localement, et de la même manière pour toutes les applications. Il s'agit donc d'un paradigme de localisation *interne.* Nous présentons une expérimentation complète sur NotePad++.

Abstract
We propose a method that allows *in context* localization of most commercial and free software programmed in Java and C++/C#. While using an application, users knowing English can translate strings of the interface such as buttons, menus, labels, tabs…, or improve translations proposed by MT or translation memory systems: localization happens *in context.* For that, we modify the code minimally, very locally and in the same way for all software. Hence this localization paradigm is *internal*. We present a complete experimentation on NotePad++.

Mots-clés : Localisation des logiciels, localisation en contexte, localisation interne.

Keywords: Software localization, in context localization, internal localization.

---

1. Introduction

Actuellement, la traduction des documents techniques ainsi que celle des éléments d'interface des logiciels commerciaux est confiée uniquement à des professionnels, ce qui allonge le processus de traduction, le rend coûteux, et quelquefois abouti à une mauvaise qualité. Le donneur d'ordre envoie la version source de documents constitués des « ressources » textuelles de l'interface et de la documentation à plusieurs traducteurs professionnels. Chaque traducteur traduit la ou les parties qui lui sont confiées et renvoie la version traduite au donneur d'ordre, qui se charge d'intégrer les différentes parties et de produire le logiciel dans les langues cibles.
C'est ce processus qui est utilisé par exemple pour la localisation des logiciels d'Adobe comme Photoshop, InDesign, AcrobatReader,…pour environ 17 langues, parmi lesquelles, le tchèque, le russe, l'arabe, l'hébreu, le turc, le polonais,… qui sont localisés par la société WinSoft.
Dès qu'on sort du petit ensemble des quelques langues les mieux dotées, et qu'on veut localiser un logiciel pour des« langues peu dotées », ce processus n'est plus viable pour des raisons de coût et surtout de rareté, de cherté, ou d'absence de traducteurs professionnels.
D'un autre côté, on observe qu'un logiciel libre comme Mozilla [14] est disponible dans beaucoup plus de langues (70) que les logiciels commerciaux. Le processus de localisation s'appuie

sur une organisation de la contribution bénévole des « codéveloppeurs » [4, 5, 6, 7, 8, 9]. Une autre situation (différente de la traduction de documentations techniques) est celle où les traducteurs sont des contributeurs bénévoles occasionnels, sans lien organique avec le projet [13, 15]. On obtient alors des traductions de qualité dans plus d'une centaine de langues. C'est le cas de l'encyclopédie Wikipedia, de textes d'Amnesty International ou de Pax Humana.

Un autre défaut du processus classique de traduction est que les éléments d'interface à traduire par les traducteurs sont sortis de leur contexte, c'est-à-dire que les relations de proximité dans le temps et l'espace sont inaccessibles pour le traducteur, alors que le contexte dans lequel un texte est lu contient de nombreuses informations auxquelles le texte peut avoir à faire référence. Il s'ensuit que le choix de la traduction la mieux adaptée n'est pas toujours possible hors-contexte, et qu'un traducteur, même professionnel, ne peut pas produire une traduction parfaite. C'est un des problèmes majeurs mentionnés à l'atelier L4Trans-III de LREC-06 par le responsable chez IBM-Japon (Kudo) de la localisation de CATIA en japonais. Comme le propose [1], une solution au problème posé serait de faire intervenir les utilisateurs finals connaissant l'anglais et dont la langue maternelle est la langue visée, de sorte que, pendant l'utilisation des produits, ils traduisent certains passages, ou puissent améliorer la traduction proposée par des systèmes de traduction automatique (TA) ou de mémoire de traductions (MT).

2. Processus de localisation actuel

2.1. Description

La tâche de localisation d'un logiciel comprend : la localisation des ressources du programme (boîtes de dialogues, menus, messages d'erreur, commandes...), la localisation de la documentation technique fournie avec le logiciel (guide d'installation, guide de prise en main et d'introduction, manuel de l'utilisateur, guide de formation...), et la localisation de l'aide en ligne. Nous nous intéressons ici uniquement à la localisation des éléments d'interface.

Afin de faciliter la localisation de leurs logiciels, la plupart des éditeurs séparent les ressources du code source du programme, et les stockent dans des « Fichiers ressources ».

Le processus de localisation de ces ressources comprend quatre étapes principales :

1. *Extraction et analyse des éléments textuels des interfaces utilisateur :* il s'agit d'extraire toutes les chaînes des interfaces utilisateur et de les stocker dans un fichier Excel appelé « glossaire »[1].

Afin de réduire les coûts de traduction, les éditeurs de logiciels utilisent des mémoires de traduction pour analyser le contenu des glossaires. Cela leur permet de retrouver les chaînes des interfaces qui ont déjà été traduites (chaînes ayant un taux de coïncidence de 100%) dans des versions antérieures du logiciel, et de calculer le taux de coïncidence du reste. En effet, le coût de la traduction dépend du taux de coïncidence : par exemple, il est beaucoup moins cher de traduire des segments qui ont un taux de coïncidence de 90% que des segments qui n'ont que 10% de taux de coïncidence.

Source File Name	Main Resname	Resname	Restype	English (US) [S; en-us]	Czech (CZ)
FMPSysDlg.rc	PRINT95DLG # -1	-1	caption (label)	P&rint:	
FMPSysDlg.rc	PRINT95DLG # 1093	1093	caption (label)	&Name:	
FMPSysDlg.rc	PRINT95DLG # 1024	1024	caption (button)	&Properties...	

Figure 1 : Extrait du glossaire du logiciel propriétaire d'un éditeur américain

2. *Traduction :* une fois analysé, le glossaire est envoyé aux traducteurs. Certains éditeurs fournissent en plus aux traducteurs une mémoire terminologique afin d'assurer une certaine cohérence de traduction entre les différentes versions d'un même logiciel.

[1] Terme métier utilisé dans le domaine de localisation de logiciels et différent de celui utilisé en terminologie.

3. *Révision* : elle consiste à vérifier les traductions qui ont été faites par les traducteurs. Elle est faite par des sociétés qui ont généralement une meilleure connaissance du contexte du logiciel. Les réviseurs reçoivent le glossaire qui a été traduit par les traducteurs et procèdent à la révision en utilisant la base terminologique si elle existe. Ils corrigent les traductions en ajoutant un commentaire explicitant les raisons pour lesquelles telle ou telle traduction proposée par les traducteurs leur paraît erronée.
4. *Test :* une fois les éléments textuels traduits et révisés, une version β est générée et fait l'objet de deux tests :
   a. *Test interne : il* est réalisé chez l'éditeur du logiciel, par des testeurs qui ne connaissent pas la langue cible. Il s'agit donc juste de vérifier le bon fonctionnement du logiciel.
   b. *Test externe :* il est assuré par des sociétés de LQA (**L**inguistic **Q**uality **A**ssurance) ainsi que par des β-testeurs qui sont des simples utilisateurs souhaitant tester le produit bénévolement, et connaissant la langue cible.

Les β-testeurs ne sont pas rémunérés par l'éditeur du logiciel, mais ils peuvent avoir une licence du produit en fonction de leur contribution au β-test. Les communautés de β-testeurs sont formées soit via des forums Internet, soit par l'intermédiaire des distributeurs des produits.

Toute anomalie détectée par la LQA ou par les β-testeurs est signalée à l'éditeur. Généralement, les éditeurs disposent d'une plate-forme en ligne qui permet aux testeurs de saisir :

- Le nom du β-testeur qui a rapporté l'anomalie, la date, etc.
- La nature de l'anomalie détecté (problème d'affichage de chaîne de caractère, problème d'inconsistance entre la traduction sur MacOs et celle qui est sur Windows…).
- Comment retrouver l'interface qui contient l'anomalie (copie d'écran, instructions…).
- etc.

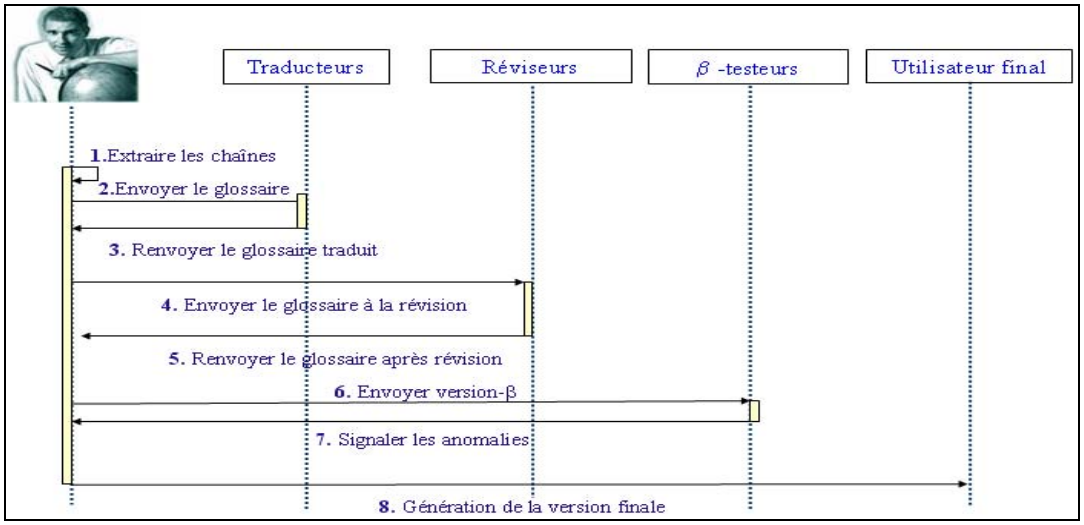

Figure 2 : Diagramme séquentiel décrivant le processus de localisation actuel

2.2. Problèmes

Le processus de localisation actuel présente des caractéristiques qui ont des conséquences négatives. Il est

- *hors-contexte* : les traducteurs et les réviseurs professionnels n'ont aucune connaissance du contexte d'utilisation du logiciel.
- *non incrémental :* la publication d'un logiciel ne se fait qu'une fois le logiciel est localisé à 100%.
- *à long délai de prise en compte :* les modifications des traductions se font à chaque version. En effet, une fois que le logiciel localisé est publié, les traductions ne peuvent être modifiées.
- *à intervenants peu variés :* les intervenants actuels sont uniquement les traducteurs, les révi-

seurs et les β-testeurs professionnels. Les utilisateurs finals sont exclus du processus de localisation alors qu'ils pourraient y participer d'une manière efficace, étant donné qu'ils ont une meilleure connaissance du contexte d'utilisation du logiciel.

3. Nouveau paradigme de localisation interne et en contexte

Nous proposons un nouveau paradigme permettant de faire participer l'utilisateur final d'une manière dynamique et efficace au processus de localisation (Figure 3). Pendant l'utilisation des produits, un utilisateur connaissant l'anglais pourra traduire ou améliorer les traductions qui ont déjà été proposées. Toutefois, l'éditeur pourra faire appel à des traducteurs et réviseurs professionnels pour traduire les parties du logiciel jugées cruciales.

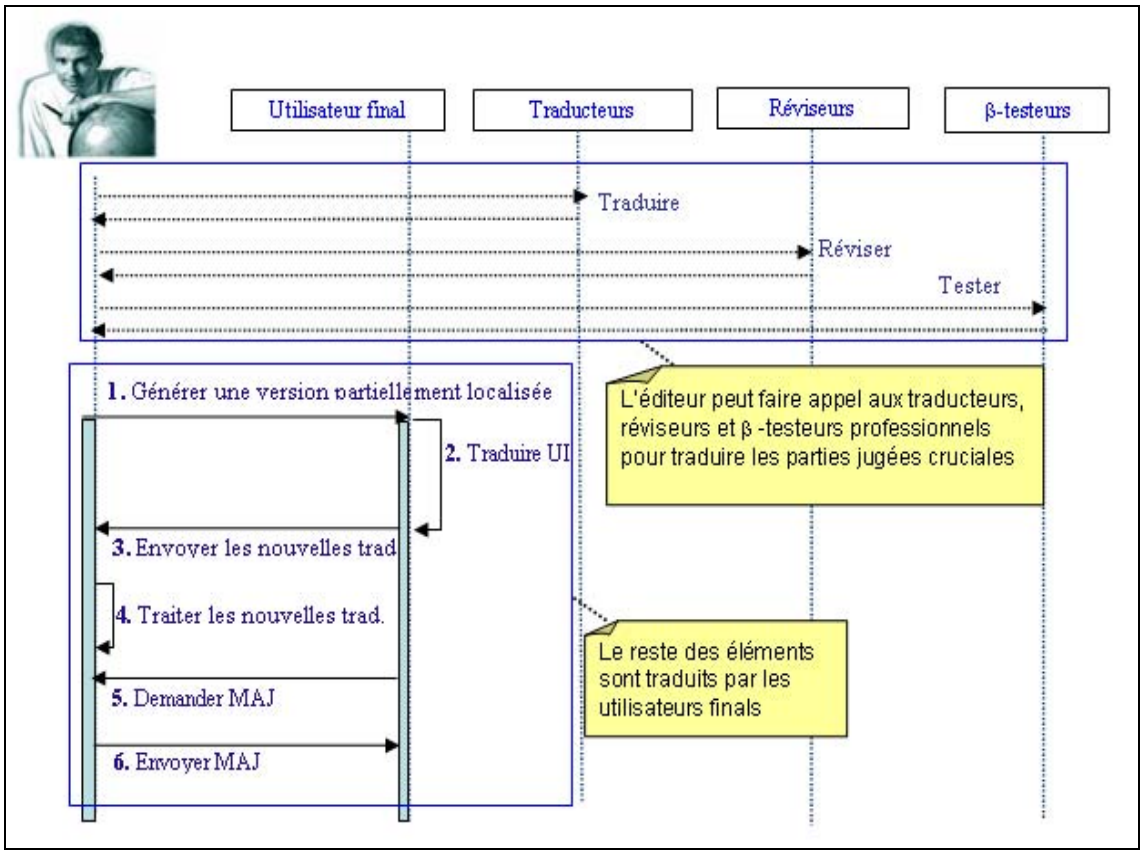

Figure 3 : Diagramme séquentiel décrivant le processus de localisation en contexte

Les caractéristiques distinctives du nouveau paradigme de localisation sont les suivantes :

- *Mode « en contexte »* : la traduction des éléments textuels se fait à partir de l'application, ce qui permet d'avoir une meilleure qualité de traduction, et est indispensable pour que des utilisateurs finals contribuent.
- *Intervenants multiples* : en plus des traducteurs, réviseurs et β-testeurs professionnels, les utilisateurs finals peuvent participer au processus de localisation.
- *Prise en compte rapide* : à chaque nouvelle proposition de traduction par l'utilisateur, l'interface est mise à jour en temps réel sur sa machine. La synchronisation avec les autres « contributeurs » se fera lors des mises à jour périodiques de l'application, et/ou via un site Web dédié à la communauté des « contributeurs de traductions » de l'application.
- *Incrément* : le nouveau processus permet l'augmentation incrémentale de la qualité ainsi que de la quantité. En effet, l'éditeur de logiciels peut publier une version partiellement localisée qui sera améliorée au fur et à mesure par les utilisateurs finals.
- *Intervention sur le code* : afin de mettre en place ce nouveau paradigme, nous avons besoin

d'intervenir sur le code source du logiciel, donc de procéder à une localisation *interne*.
Pour être le plus générique possible et toucher le moins possible au code source des logiciels, nos modifications sont faites uniquement sur les classes de base produisant les IHM. Ces modifications consistent à rajouter un comportement adapté à la traduction aux éléments textuels de l'interface : par un simple clic droit sur un élément textuel, on peut choisir dans une liste de traductions possibles, et on peut aussi ajouter la sienne. L'utilisateur final peut ainsi traduire n'importe quel élément textuel de l'interface, qui est mise à jour en temps réel.

4. Localisation en contexte des applications

4.1. Scénario de la localisation en contexte sur le logiciel Notepad++

Nous avons mené une expérience complète avec Notepad++ (logiciel libre programmé en C++). Il s'agit d'un code de taille raisonnable (60.000 lignes de code), comparé à des logiciels de taille bien supérieure comme FileMaker et PhotoShop que nous avions étudiés auparavant. Cela nous a permis de tenter sa *localisation interne* nous-mêmes, sans faire appel à des collaborateurs extérieurs. Prenons l'exemple de la boîte de dialogue « ColumnEditor » de Notepad++ (Figure 4) :

- L'utilisateur fait un clic droit sur l'élément textuel « Text to insert », ce qui lui permet de passer dans un contexte d'édition.
- Il saisit sa nouvelle proposition de traduction, ou en choisit une parmi celles disponibles.
- Il enregistre sa proposition.
- Il voit apparaître sa nouvelle traduction dans l'interface.

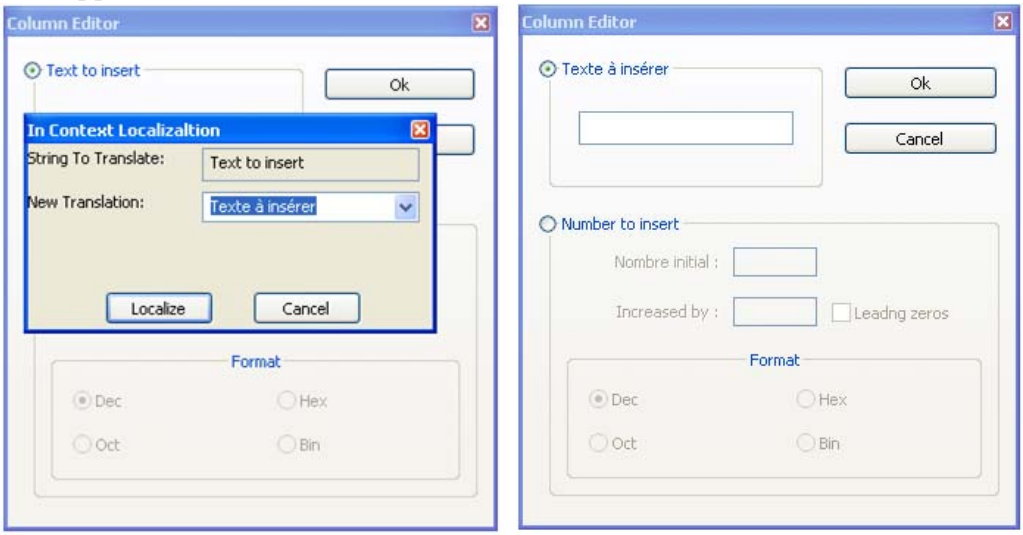

Figure 4 : Localisation en contexte de la boite de dialogue « ColumnEditor » de « Notepad++ »

Notre approche de localisation en contexte ne gère pas les messages avec des variables [2], mais cela fait partie des évolutions prévues.

4.2. Localisation interne de Notepad++

Comme nous l'avons mentionné ci-dessus, pour permettre la *localisation en contexte* des applications, nous avons besoin d'intervenir de façon *interne* au niveau du code source. Pour être le plus générique possible et toucher le moins possible au code source, nos modifications sont faites uniquement sur les classes de base produisant les IHM. Pour cela, il est important de comprendre l'architecture logicielle de l'application à localiser afin d'identifier les classes génériques d'IHM sur lesquelles nous devrons intervenir

4.2.1. Architecture

Dans Notepad++, il y a une classe générique d'IHM dont dérivent toutes les interfaces de l'application : la classe « Static_Dialog » est la classe de base de toutes les boîtes de dialogues. Nous avons intégré notre module de localisation au niveau de cette classe de base (Figure 5), ce

qui permet à tous les éléments textuels qui appartiennent aux différentes boîtes de dialogue d'hériter d'un comportement adapté à la traduction en contexte.

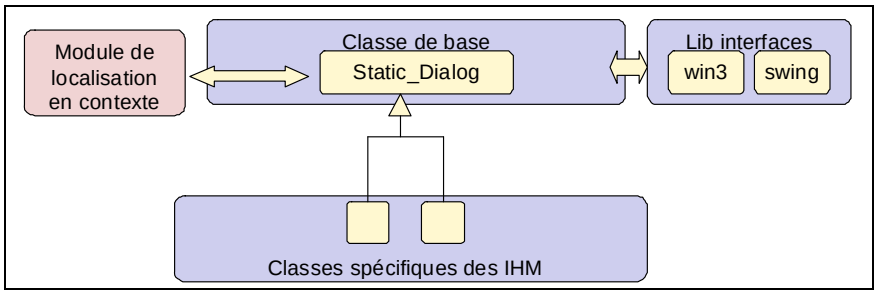

Figure 5 : Intégration du module « Localisation en contexte dans l'architecture générale de Notepad++ »

4.2.2. Format d'échange

XLIFF (**X**ML **L**ocalization **I**nterchange **F**ile **F**ormat), introduit en 2001 par OASIS, est un format standard d'échange de données textuelles multilingues, ou à traduire en plusieurs langues. Ce format s'applique à des documents qui peuvent être de structure complexe (en fait, à des documents dans n'importe quelle DTD), et convient parfaitement aux cas simples des éléments textuels des programmes (interfaces utilisateur, aides en ligne), et aux autres documents accompagnant un logiciel (manuels divers). Nous utilisons donc le format XLIFF comme format d'échange pour communiquer avec l'application. Pour construire le fichier XLIFF, nous faisons un import depuis les fichiers de ressources des applications. Par exemple, dans le cas de Notepad++, nous avons construit le fichier XLIFF à partir du fichier « NativeLang.Xml ». Le fichier XLIFF est installé avec l'application sur l'ordinateur de l'utilisateur. En effet, toutes les propositions de traduction de l'utilisateur sont stockées dans le fichier XLIFF. Ainsi, au moment de l'exécution de l'application, les chaînes qui ont été modifiées par l'utilisateur sont affichées à partir du fichier XLIFF et non pas à partir des fichiers de ressources de l'application.

```
<?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>
```

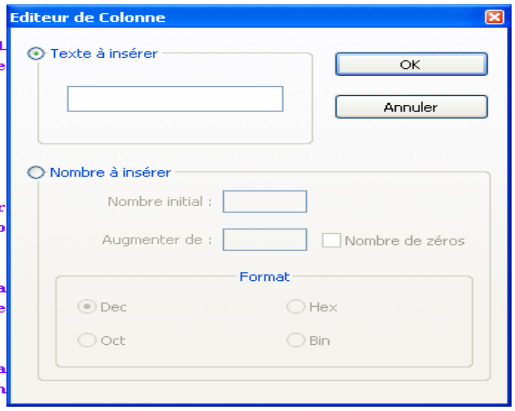

Figure 6 : Structure d'un fichier XLIFF

4.3. Interactions

L'application interagit avec notre module de *localisation en contexte* lors de l'édition d'un élément textuel de l'interface par l'utilisateur, ainsi qu'au moment de la mise à jour de l'interface.

4.3.1. Edition d'un élément textuel de l'interface

Lors du clic droit sur n'importe quel élément textuel de l'interface, l'application récupère et envoie l'identifiant de l'élément textuel au module *localisation en contexte*. Ce dernier affiche alors à l'utilisateur le contexte d'édition qui contient par défaut l'élément à éditer ainsi que les différentes propositions de traduction disponibles dans le fichier XLIFF associé à l'application. Une fois l'édition terminée, le fichier XLIFF est mis à jour et l'application est avertie pour qu'elle mette à jour l'interface.

4.3.2. Mise à jour de l'interface

Les librairies graphiques les plus connues et les plus utilisées en Java sont Swing, AWT, SWT [12,16]. En C++, il s'agit de Win32 (sous Windows) [17]. Le principe de fonctionnement de ces librairies est le même : chaque IHM est représentée en mémoire par une structure de données. Au moment de l'exécution, l'application charge toutes ses IHM et plus particulièrement les éléments textuels des IHM à partir de ces structures de données.
Ainsi, mettre à jour l'interface utilisateur revient à mettre jour la structure de données correspondante. Une fois l'édition terminée, le module *localisation en contexte* met à jour le fichier XLIFF et demande à l'application de rafraîchir l'IHM contenant l'élément textuel venant d'être édité. L'application accède alors à la structure de données et remplace la valeur de l'élément textuel par la nouvelle proposition de l'utilisateur, lue depuis le fichier XLIFF.

5. Expérimentation

Pour valider cet outil, nous avons procédé à la localisation du logiciel NotePad++ qui contient 500 chaînes d'interface. Etant donné la taille du corpus et le caractère très « prototype » de notre outil, l'expérimentation est minimale, on pourrait la qualifier de pré-expérimentation.
Les utilisateurs sollicités ont été au nombre de 3, mais ce sont des traducteurs spécialisé dans la localisation des logiciels. Ainsi sur ces 500 chaînes d'interface, 475 chaînes ont été traduites. L'effort de traduction a été pour chacun des bénévoles d'environ 3h. Les 25 chaînes restantes correspondent aux messages d'erreurs dont la localisation en contexte est à ce jour possible.

6. Perspectives

6.1. Gestion des contributions à intervenants multiples

Actuellement, les utilisateurs traduisent en local sur leur machine. Nous travaillons actuellement sur la prochaine version de notre prototype, qui permettra aux utilisateurs de collaborer via un gestionnaire de flot de traduction qui communique directement avec l'application, ainsi que d'interagir avec des outils d'aides à la traduction [3, 5, 9, 10, 11]. Ainsi, l'utilisateur pourra directement, depuis l'application :

- visualiser les propositions qui ont été faites par les autres utilisateurs ;
- choisir une des traductions disponibles ou saisir une nouvelle traduction ;
- commenter ou évaluer sa traduction ainsi que celles des autres utilisateurs ;
- envoyer sa traduction ;
- mettre à jour son interface.

6.2. Synchronisation et mise à niveau de l'application

Les utilisateurs pourront se synchroniser sur le site de l'éditeur lors des mises à niveau périodiques de l'application par l'éditeur ou sur requête de l'utilisateur. Au moment de la synchronisation l'utilisateur pourra visualiser les nouvelles traductions qui ont été validées et évaluées par l'éditeur de logiciel. Chaque traduction est accompagnée d'un des labels suivants :

***** traduction certifiée par un traducteur agréé par l'éditeur,
**** traduction certifiée par un traducteur, non agréé par l'éditeur,
*** traduction proposée par un bêta-testeur ou un utilisateur, non certifiée,
** traduction automatique, ni post-éditée ni certifiée,
* traduction mot à mot (venant d'un dictionnaire), non post-éditée ni certifiée.

L'utilisateur pourra donc accepter toutes les traductions ou sélectionner seulement celles avec lesquelles il est d'accord.

Conclusion

La méthode présentée permet la localisation *en contexte* de la plupart des logiciels commerciaux et libres, programmés en Java et C++/C#, au prix d'une modification très locale du code source. L'expérimentation faite sur NotePad++ concerne la bibliothèque Win32, utilisée dans de nombreuses applications. Nous sommes en train d'expérimenter cette méthode à des logiciels utilisant Swing et QT. Il nous reste à mettre en place un environnement de gestion des modifications incrémentales des traductions (application en local, éditeur de l'application, et communauté des utilisateurs-contributeurs).

Bibliographie

1. C. Boitet (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.
2. C. Boitet (2005). Message Automata for Messages with Variants, and Methods for their Translation. Proc. CICLING-05, *LNCS 3406*, pp. 352—371, Mexico.
3. E. Eneko & al. (2000). A Methodology for building Translator-oriented Dictionary Systems. *Machine Translation 15*, pp. 295-310.
4. H. Vo-Trung (2004). *Méthodes et outils pour utilisateurs, développeurs et traducteurs de logiciels en contexte multilingue*, thèse d'informatique, Institut National Polytechnique de Grenoble.
5. K. Kageura et T. Abekawa (2007). QRedit: An Integrated Editor System to Support Online Volunteer Translators. *Digital Humanities*, pp. 3-5.
6. L. C. Tong (1987). The Engineering of a translator workstation, *Computers and Translation*, pp. 263-273.
7. M. Lafourcade (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.
8. M. Lafourcade et G. Sérasset (1996). Apple Technology Integration.
   A Web dictionary server as a practical example. *Mac Tech magazine*, 12/7, pp. 25-32.
9. Y. Bey, C. Boitet et K. Kageura (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.
10. Y. Bey, K. Kageura et C. Boitet (2008). BEYTrans: A Wiki-based Environment for Helping Online Volunteer Translators. *In Topics in Language Resources for Translation and Localisation*, Yuste Rodrigo, Elia (ed.), 135–150.
11. Y. Bey, K. Kageura and C. Boitet (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.

Netographie

12. *Java™ Platform, Standard Edition 6 API Specification*
    http://java.sun.com/javase/6/docs/api. 2008.
13. Linux documentation translation. *Traduc project*, http://traduc.org/. 2005
14. *Mozilla project & Mozilla French Localization project*.
    http://frenchmozilla.online.fr/. 2005.
15. *Specification translation.* W3C, http://www.w3.org/consortium/Translation. 2005.
16. Sun Micosystems Inc. Building International Applications, *documentation Sun*: http://docs.sun.com/db/doc/806-6663-01. 2000.
17. *Win32 and COM Development.* http://msdn.microsoft.com/en-us/library. 2008.