Localisation interne et en contexte des logiciels commerciaux et libres

Amel Fraisse

THÈSE
Présentée par

Amel FRAISSE

Pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE
(ARRÊTÉS MINISTÉRIELS DU 5 JUILLET 1984 ET DU 30 MARS 1992)

Discipline
INFORMATIQUE

École doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

Titre de la thèse :

Localisation interne et en contexte des logiciels commerciaux et libres

Présentée et soutenue publiquement le 10 juin 2010

Composition du jury :

Président :	Christophe Roche
Rapporteurs :	Jacques Chauché
	Michel Divay
Examinateurs :	Kamel Gaddas
	Christophe Chenon
Directeur :	Christian Boitet
Co-directeurs :	Hervé Blanchon
	Valérie Bellynck

Thèse préparée au sein de l'équipe GETALP du Laboratoire d'Informatique de Grenoble et de la société WinSoft (Saint Martin le Vinoux)

Remerciements

À l'heure où cette étape de mon travail s'achève, je tiens à remercier profondément mon directeur de thèse Christian Boitet, qui m'a toujours poussé jusqu'au bout, et m'a toujours soutenu aux moments les plus difficiles. C'est lui qui m'a appris la persévérance et la précision indispensables pour un être un chercheur.

Je remercie aussi mes co-directeurs Hervé Blanchon et Valérie Bellynck qui ont su m'accompagner, me conseiller et m'encourager et me soutenir durant ces trois années de thèse.

Je remercie bien sûr très vivement, monsieur Jacques Chauché et monsieur Michel Divay d'avoir accepté d'être rapporteurs à ma thèse.

Mes remerciements s'adressent également à ma famille pour leur encouragement et leur soutien moral. J'espère qu'à travers ce travail, j'apporte le témoignage de ma grande affection et mon amour.

Enfin et surtout, mon mari qui m'a soutenu tout au long de cette thèse avec beaucoup d'amour et de patience.

TABLE DES MATIERES

LISTE DES FIGURES

Introduction

Aux premiers temps de l'informatique, les utilisateurs se sont adaptés eux-mêmes aux exigences de l'ordinateur. Ils ont, par exemple, été obligés d'apprendre le langage de la machine pour s'en servir. Au contraire, les systèmes d'aujourd'hui doivent être adaptés aux besoins et cultures de leurs utilisateurs (Lustig, 1993). En communiquant l'information, le respect du client interagissant avec la machine est plus important qu'auparavant, et la conviction qu'il faut concevoir le multiculturalisme dans les produits dès le début est de plus en plus largement répandue.

Depuis quelques années, les impacts de la mondialisation, du village planétaire, atteignant les éditeurs de logiciels. De nouveaux marchés, de nouvelles langues, de nouvelles cultures doivent être considérés par les éditeurs de logiciels qui visent une distribution mondiale. L'informatique multilingue est alors face à une nécessité impérieuse de répondre à ces besoins croissants. Cela entraîne plusieurs problèmes concernant la réalisation et la production des logiciels et des applications pour permettre d'une part le traitement de ces langues et d'autre part leur mise à la disposition de cultures différentes. Une solution de ces problèmes est connue aujourd'hui sous le nom de *localisation*.

*« La **localisation ou la régionalisation de logiciel** concerne le processus de traduction de l'interface utilisateur d'un logiciel d'une langue vers une autre et en l'adaptant à la culture locale ». (Wikipedia, 2010).*

Le terme localisation est une transposition du mot anglais "localization" (faux ami). On écrit parfois **L10N** car le mot "localization" est composé de dix lettres encadrées par un *L* et un *N*.

L'approche mise en œuvre de façon majoritaire est l'approche américaine qui consiste à produire une première version pour les Etats-Unis en anglais « américain » ; on produit ensuite les versions localisées dans différentes langues cible. La localisation est d'autant facile que l'internationalisation a été bien préparée[1] (Microsoft, 2002), (Symmonds 2005).

Dans le cadre de ce travail, nous examinerons cette approche de la localisation en identifiant ses limites et en proposant des solutions adaptées. Ce mémoire est divisé en trois grandes parties.

Dans une première partie introductive, nous présentons la situation actuelle de la localisation des logiciels commerciaux et libres ainsi que les problèmes associés anciens et nouveaux. Nous esquissons des solutions possibles.

[1] Lors des premières localisations de la suite Mozilla (avant qu'elle ne devienne Firefox et Thunderbird), les fenêtres de messages de type *warning : ....* ne pouvaient être complètement traduites en français, car la taille des fenêtres ne variait pas en fonction de la longueur du texte. Or, le français ayant en général des mots plus longs que l'anglais, le texte traduit sortait du cadre de la fenêtre de message. Par la suite, le programme a été mieux internationalisé, puisqu'il a permis d'adapter la taille d'une fenêtre ou d'un menu en fonction de la longueur du message.

La deuxième partie présente la solution que nous avons adoptée pour résoudre certains problèmes du processus de localisation actuel, ainsi qu'une première expérimentation de notre solution sur un logiciel libre à code source ouvert : Notepad-plus-plus.

Enfin la troisième partie, présente notre solution sous forme d'un processus de localisation tripartite avec trois entités.

Partie 1 : Situation actuelle de la localisation des logiciels et problèmes associés anciens et nouveaux

Dans cette première partie nous étudions et analysons la situation actuelle en localisation des logiciels commerciaux et libres dans le but d'identifier les problèmes associés anciens et nouveaux afin de proposer d'éventuelles solutions.

Le premier chapitre présente les caractéristiques du processus de localisation actuel. Il nous permet aussi de présenter plus en détail le processus de localisation des ressources textuelles des logiciels, de la documentation technique, ainsi que de l'aide en ligne.

Le deuxième chapitre est consacré à l'identification et la classification des problèmes liés au processus de localisation actuel.

Enfin, dans le troisième chapitre, nous recherchons des solutions adaptées pour résoudre les problèmes que nous avons identifiés dans le chapitre précédent.

Chapitre 1 : Caractéristiques du processus de localisation actuel

Introduction

La localisation d'un logiciel inclut la localisation des ressources du programme (boîtes de dialogues, menus, messages d'erreur, commandes....), de la documentation technique fournie avec le logiciel (guides d'installation, guides de prise en main et d'introduction, manuels de l'utilisateur, guides de formation, ...) et de l'aide en ligne (Uren, 1993), (Schmitz, 2002).

En raison de besoins croissants liés à l'apparition de « nouvelles langues » et de nouveaux marchés, la plupart des éditeurs de logiciels font de plus en plus appel à des sociétés spécialisées pour localiser leurs logiciels. Par exemple, WinSoft[2], société spécialisée dans la localisation et la publication de logiciels, localise les produits d'Adobe et de FileMaker sur les marchés émergents d'Europe de l'Est, d'Europe Centrale, du Moyen-Orient et d'Asie du Sud-Est. Ayant effectué ma thèse CIFRE au sein des équipes de localisation de WinSoft, j'ai pu examiner de près le processus de localisation. Ainsi, dans ce chapitre, nous présentons les caractéristiques du processus de localisation actuel.

1.1 Localisation des éléments textuels des interfaces utilisateur de façon externe

Les outils de localisation de logiciels sont relativement récents. Avant 1990, les outils tels que nous les connaissons aujourd'hui n'existaient pas. En règle générale, les interfaces utilisateur « codées en dur » étaient directement traduites dans le code source et étaient adaptées par la suite. Selon la complexité des textes contenus dans le programme, l'adaptation de la traduction devait être effectuée soit par les développeurs, soit par des traducteurs possédant des connaissances en programmation, ou alors ces derniers devaient travailler avec des développeurs pour réaliser cette tâche. La contrainte est que le traducteur n'est pas un programmeur et que d'autre part, il est rare de disposer d'un traducteur en interne. De plus, il est difficilement envisageable de laisser le code source aux traducteurs, à cause de problèmes de confidentialité et du risque important d'introduction de bogues et d'accès au code source.

La situation a évolué dans les années 90. Les sociétés de développement de logiciels ont commencé à externaliser la localisation de leurs logiciels, soulignant ainsi la nécessité de créer des outils et processus spéciaux pour permettre aux traducteurs de traiter le texte contenu dans les interfaces utilisateur sans devoir posséder de connaissances particulières en programmation ni avoir constamment besoin de l'assistance des développeurs (Reineke, 2005).

Le concept de ressources est issu du monde Macintosh (M. Everson et T. Trosterud, 2000) et désigne des éléments structurels et constants d'un programme ou d'un fichier, auxquels une référence peut être faite à tout moment. Ce mode de développement a pour avantages d'alléger le programme (une seule ressource peut être réutilisée à de nombreux endroits) et de permettre de traduire directement un logiciel ou un fichier, sans avoir à le recompiler.

[2] WinSoft : www.winsoft-international.fr

Les ressources sont stockées dans des fichiers à part appelés « Fichiers de ressources » qui sont utilisés par l'application au moment de l'exécution. Par conséquent, une nouvelle approche de la localisation a été adoptée : traduire plusieurs fichiers de ressources (souvent des centaines) et modifier la taille des boîtes de dialogue à l'aide d'outils tels que *Microsoft Developer Studio* ou des éditeurs de texte. Il s'agit donc d'une localisation *externe* par rapport au code source de l'application, puisque la localisation est faite directement sur les ressources. Selon les logiciels, les fichiers de ressources peuvent avoir différents formats, extensions, structures, contenus, etc.

Avant d'aller plus loin nous donnons quelques définitions de certains termes métiers que nous allons utiliser.

Localiseur : transposition du mot anglais "localizer", désigne la personne qui localise un logiciel d'une langue source vers une ou plusieurs langues cible. Dans certains cas, le développeur de l'application est lui même le localiseur. Son travail consiste à : adapter le code source du logiciel afin que ce dernier puisse tourner correctement dans différentes langues cibles, extraire les éléments textuels à traduire, réinjecter les chaînes traduites, compiler les ressources, etc.

Traducteur et réviseur professionnel : personne maîtrisant la langue cible du logiciel avec ou sans connaissance du contexte d'utilisation du logiciel.

1.1.1 Format de ressources logicielles

On distingue deux grandes catégories de format de ressources : les ressources binaires et les ressources textuelles.

1.1.1.1 Ressources binaires

Certains développeurs utilisent des formats de ressources qui nécessitent une compilation pour qu'elles puissent être utilisées par l'application. Ces ressources sont installées avec l'application en format binaire (.Dll, .Exe, etc.). Ce type de ressources est utilisé généralement par les développeurs qui souhaitent découpler le code de l'application du code des interfaces utilisateur. En effet, en plus des éléments textuels des interfaces, ces fichiers de ressources contiennent aussi le code qui permet de fabriquer les interfaces. Avant d'être compilées par un compilateur de ressources, ces ressources binaires peuvent avoir différents formats d'origine (RC, R, etc.). Le format le plus utilisé par les développeurs des logiciels commerciaux et libres est le format RC.

1.1.1.1.1 Le format de ressources RC

Créé par Microsoft (Microsoft, 2008), ce format de ressources est utilisé par les applications Win323 développées sous Visual Studio (Microsoft, 2008). Comme le montre la Figure 1, le script utilisé dans le fichier de ressources RC du logiciel FileMaker contient le code ainsi que les chaînes de caractères de la boîte de dialogue de la Figure 2. Ce fichier de ressources est compilé par la suite par un compilateur de ressources pour générer un fichier binaire .RES qui est utilisé par l'application lors de son exécution.

[3] API Win32 (Win32 API) : Interface de programmation d'applications (API) pour les applications Windows 9x et Windows NT. Elle ajoute aux versions précédentes de l'API Windows les services sophistiqués d'un système d'exploitation, la sécurité, et des fonctions d'API pour l'affichage d'applications en mode texte dans des fenêtres.

```
/* Print95Dlg add DISCARDABLE style and style | 0x4 */
PRINT95DLG DIALOG DISCARDABLE  32, 32, 289, 240
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x4
CAPTION "Print"
FONT 8, "MS Shell Dlg 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 WS_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
    EDITTEXT        1101,52,88,224,12,ES_AUTOHSCROLL | ES_READONLY | NOT
                    WS_BORDER | WS_GROUP | NOT WS_TABSTOP
    LTEXT           "Comment:",1096,16,100,36,8,SS_NOPREFIX | NOT WS_GROUP
    EDITTEXT        1100,52,100,152,12,ES_AUTOHSCROLL | ES_READONLY | NOT
                    WS_BORDER | WS_GROUP | NOT WS_TABSTOP
    CONTROL         "Print to fi&le",1040,"Button",BS_AUTOCHECKBOX |
                    WS_GROUP | WS_TABSTOP,212,100,64,11
    GROUPBOX        "Print range",1072,8,120,144,50
    CONTROL         "&All",1056,"Button",BS_AUTORADIOBUTTON | WS_GROUP |
                    WS_TABSTOP,16,134,20,12
    CONTROL         "Pa&ges",1058,"Button",BS_AUTORADIOBUTTON,16,150,36,12
    RTEXT           "&from:",1089,52,152,20,8
    EDITTEXT        1152,74,150,26,12,WS_GROUP
    RTEXT           "&to:",1090,100,152,16,8
    EDITTEXT        1153,118,150,26,12,WS_GROUP
    GROUPBOX        "Copies",1073,160,120,120,64
    LTEXT           "Number of &copies:",1092,168,136,68,8,NOT WS_GROUP
    EDITTEXT        1154,240,134,32,12,WS_GROUP
    ICON            "",1086,168,152,68,24,WS_GROUP
    CONTROL         "C&ollate",1041,"Button",BS_AUTOCHECKBOX | WS_GROUP |
                    WS_TABSTOP,240,158,36,12
    LTEXT           "Nu&mber pages from:",1102,8,173,76,10
    EDITTEXT        1155,82,172,20,12,ES_RIGHT | WS_GROUP
    GROUPBOX        "OLE",1074,10,193,124,29
    CONTROL         "&Update all Links before printing",1042,"Button",
```

Figure 1 : *Exemple d'un fichier de ressources RC du logiciel FileMaker*

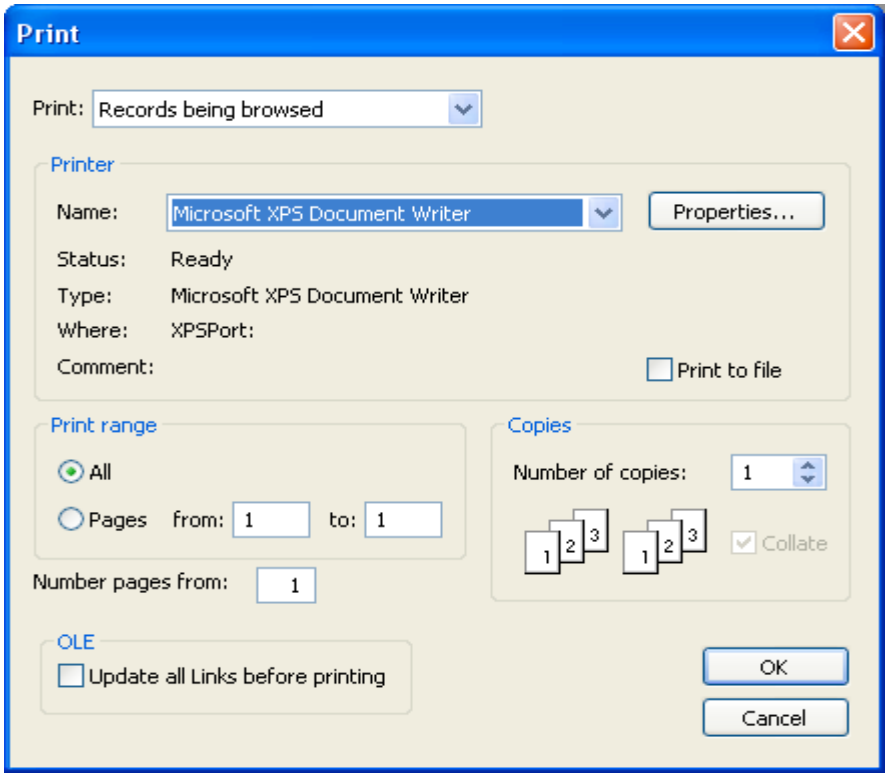


Figure 2 : *Boîte de dialogue « Print » du logiciel FileMaker fabriquée à partir du script de ressources RC de la Figure 1*

1.1.1.1.2 Le format de ressources EVE

Le format de ressources EVE a été créé par Adobe (Adobe, 2007). Ce format permet de créer des interfaces graphiques (Figure 3). L'utilisation de ce type de ressources nécessite la mise en place d'un analyseur (Figure 4) pour lire des fichiers de ressources ainsi qu'un moteur EVE. Ce dernier permet de charger les ressources EVE durant l'exécution.

```
layout spell_checking
{
    view dialog(name: "Spell Checking", placement: place_row)
    {
        column()
        {
            row(guide_mask: [ ])
            {
                static_text(name: 'Words not found:');
            }
            edit_text(size: @size_small, characters: 60, lines: 3, scrollable: false,
            monospaced: true, vertical: align_fill);
            row(guide_mask: [ ])
            {
                static_text(name: 'Suggestions:');
            }
            edit_text(size: @size_small, characters: 60, lines: 3, scrollable: false,
            monospaced: true, vertical: align_fill);
        }
        column(child_horizontal: align_fill, vertical: align_fill, spacing: 20)
        {
            column(child_horizontal: align_fill)
            {
                button(name: 'Start', default: true);
            }
            column(child_horizontal: align_fill)
            {
                button(name: 'Change');
                button(name: 'Change All');
            }
            column(child_horizontal: align_fill)
            {
                button(name: "Ignore");
                button(name: "Ignore All");
            }
            column(child_horizontal: align_fill)
            {
                button(name: 'Done', cancel: true, action: @cancel);
            }
        }
    }
}
```

Figure 3 : *Exemple d'un fichier de ressources dans le format EVE d'Adobe*

```
void parse_my_eve_file(const std::string& path_to_file)
{
    std::ifstream           stream(path_to_file.c_str());
    adobe::line_position_t  result_line(adobe::eve::parse(
                                stream,
                                adobe::line_position_t(path_to_file.c_str()),
                                adobe::eve::position_t(),
                                boost::bind(&client_assemble, _1, _3,
                                    boost::bind(adobe::eve::evaluate_arguments(), _4))));
}
```

Figure 4 : *Exemple d'un analyseur EVE*

1.1.1.2 Ressources textuelles

Les ressources textuelles sont celles qui sont utilisées par l'application dans leurs formats d'origine (TXT, XML, HTML, etc.). Comme le montre la Figure 5, les fichiers de ressources textuelles sont généralement stockés dans des sous-répertoires de l'application. Ces ressources contiennent principalement les chaînes des interfaces utilisateur. Ainsi, les interfaces sont fabriquées soit dans le code de l'application, soit depuis des fichiers de ressources séparés.

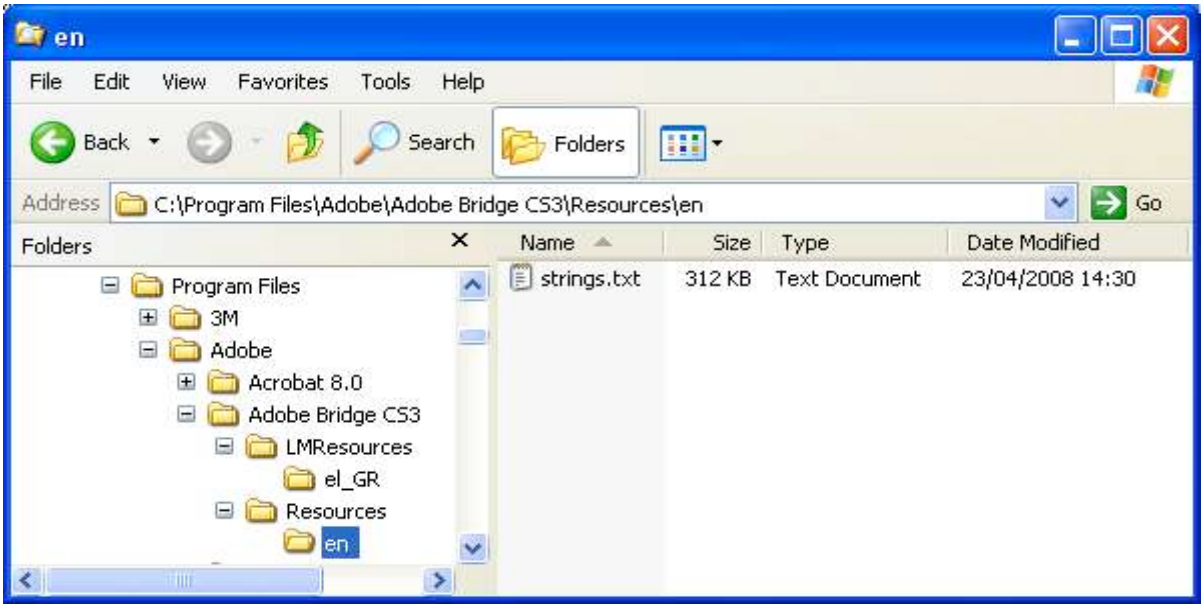


Figure 5 : *Répertoire « Ressources » du logiciel « Adobe Bridge CS3 »*

1.1.1.2.1 Ressources au format TXT

Pour certains de ses produits, Adobe utilise des fichiers de ressources textuels strings.txt pour stocker les chaînes de caractères des interfaces utilisateur. Ainsi, tous les éléments textuels de l'application sont rassemblés dans un seul fichier de ressources strings.txt. La Figure 6 montre le contenu du fichier strings.txt du logiciel Adobe Bridge CS3. Chaque chaîne de caractères appartient à une seule interface utilisateur. Ainsi, une même chaîne peut avoir plusieurs identifiants selon l'interface utilisateur à laquelle elle appartient. Cela s'explique par le fait qu'une chaîne de caractères peut être traduite différemment en fonction du contexte dans lequel elle apparaît.

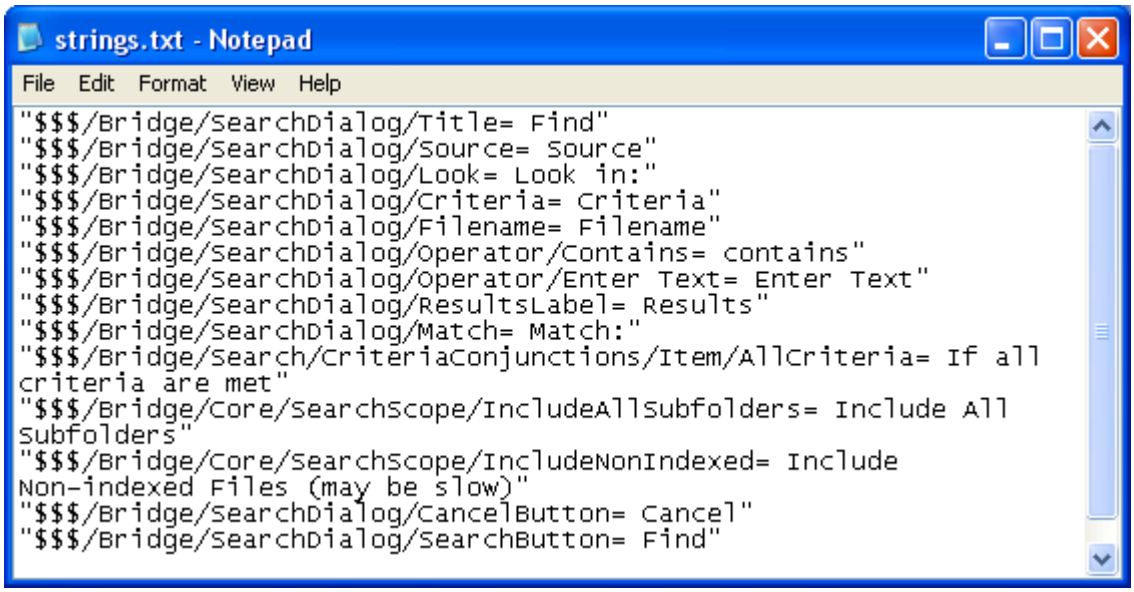


Figure 6 : *Un extrait du fichier de ressources strings.txt du logiciel Adobe Bridge CS3*

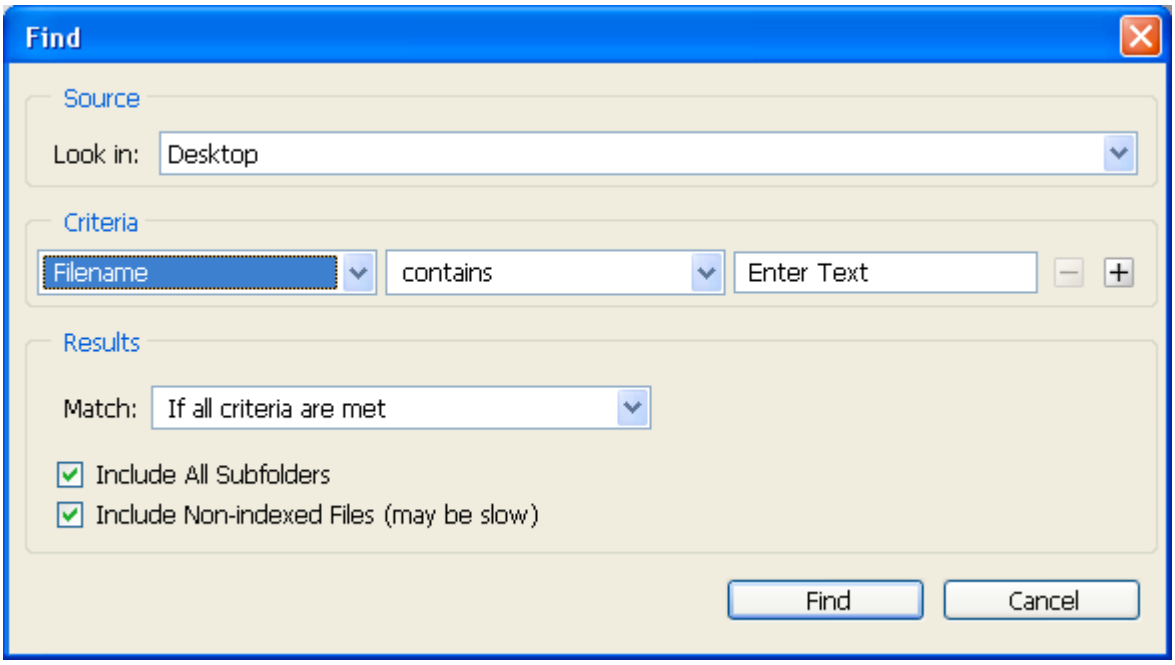


Figure 7 : *Boîte de dialogue "Find" du logiciel Adobe Bridge CS3*
Contenant les chaînes de caractères de la figure 6

1.1.1.2.2 Ressources au format XML

Adobe utilise aussi pour certains de ses logiciels le format XML pour créer les interfaces utilisateur. La Figure 8 est un exemple d'un fichier de ressources XML qui est utilisé pour créer la boîte de dialogue de la Figure 9 du logiciel Adobe Illustrator.

```
<OBJECT className="LDialogBox" layer="0" position="12298"
    title="$$$/SliceOptions/DialogTitle=Options de tranche" >
    <bounds w="467" h="260" />
    <children>
        <!-- OK and Cancel Buttons -->
        <OBJECT className="LADMItem" admItemType="ADM Text Push Button Type" cmd="900"
                text="$$$/Dialog/OK=OK" >
                <bounds x="375" y="15" w="85" h="20" />
        </OBJECT>
        <OBJECT className="LADMItem" admItemType="ADM Text Push Button Type" cmd="901"
                text="$$$/Dialog/Cancel=Annuler" >
                <bounds x="375" y="45" w="85" h="20" />
        </OBJECT>
```

Figure 8 : *Exemple du fichier de ressources XML du logiciel Adobe Illustrator.*

Figure 9 : *Boîte de dialogue « Options de tranche » du logiciel Adobe Illustrator créé à partir du fichier de la Figure 8*

1.1.2 Intervention des localiseurs sur les ressources

Depuis les années 90 (Reineke, 2005), l'approche de la localisation a changé. En effet, les localiseurs ainsi que les traducteurs n'interviennent plus sur le code source des applications, ils localisent et adaptent directement les fichiers de ressources. Nous détaillons ici les trois fonctions de base que proposent les outils de localisation et qui sont utilisées par les localiseurs pour localiser les fichiers de ressources, car cela est indispensable pour la suite (Vo-Trung, 2004). Afin d'illustrer cela, nous avons inséré quelques captures d'écran de l'outil de localisation de logiciels RC WINTRANS (Figure 10, Figure 11, Figure 12, Figure 14) qui a été créé par la société allemande *Schaudin.com*. Cet outil de localisation était à l'origine destiné à la traduction des fichiers RC. Aujourd'hui, il peut être utilisé pour Windows Win32, Microsoft.NET et les plate-formes de développement de logiciels Java. La dernière version de RC WINTRANS est compatible avec plusieurs formats de ressources tels que les fichiers de propriétés de Java, les fichiers Microsoft et Borland .NET, les fichiers binaires exécutables (EXE/DLL) Win32, etc.

1.1.2.1 Extraction des chaînes d'interface à partir des fichiers de ressources

L'extraction des éléments localisables est une fonction classique des outils de localisation de logiciels (Esselink, 2000). Elle englobe :

- la sélection de différents analyseurs syntaxiques en fonction des formats de localisation,
- l'extraction, le traitement et la préparation des éléments localisables.

Le processus d'extraction des éléments localisables est sensiblement le même pour tous les outils de localisation. Si nous créons un nouveau projet de localisation, avec un outil de localisation (Figure 10), ou si nous ajoutons de nouveaux fichiers de ressources ou des langues cible à un projet existant, l'outil génère des listes de segments contenant des éléments à localiser (Figure 11). Les éléments localisables sont extraits des fichiers de ressources et classés en fonction de leurs « ressources » (celles dans lesquelles ils sont contenus), par exemple : menu, boîte de dialogue, tableau de chaînes de caractères, raccourcis clavier, etc.

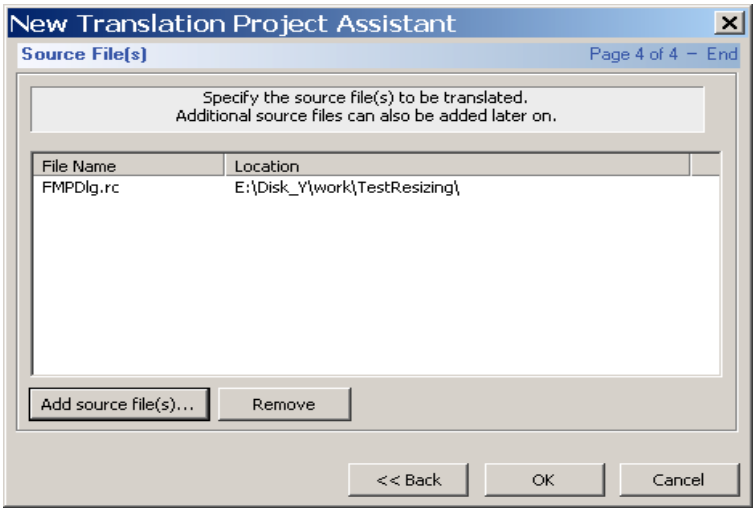


Figure 10 : *Ajout d'un fichier de ressources FMDlg.rc à un nouveau projet de localisation*

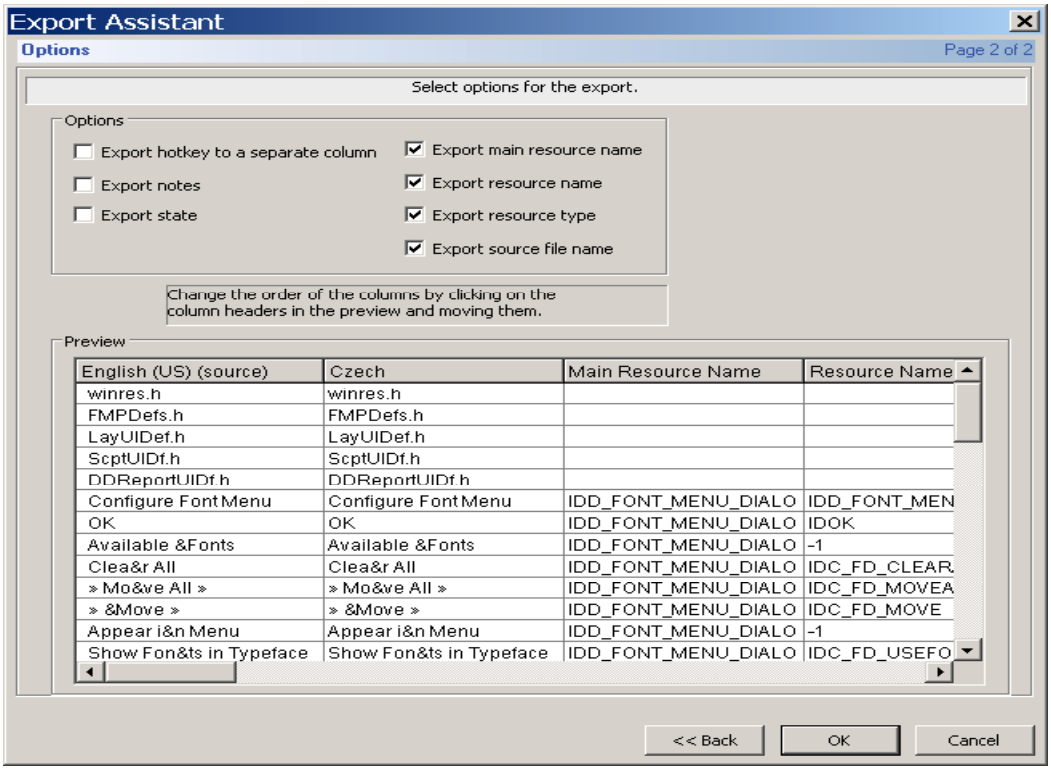


Figure 11 : *Segments (éléments localisables) extraits des fichiers de ressources.*

La première colonne de Figure 11 contient les segments source, la deuxième les segments cible (par défaut initialisé avec les segments source).

1.1.2.2 Construction du glossaire

Le terme « glossaire » est un terme métier qui est utilisé dans le domaine de la localisation des logiciels, où il a un sens tout à fait différent de son sens dans le domaine de la terminologie. Il désigne un fichier qui contient tous les éléments textuels des interfaces graphiques de l'application.

Les outils de localisation permettent aussi aux localiseurs de générer un glossaire contenant tous les éléments localisables (Figure 12). Le glossaire peut avoir différents formats : Excel, TMX, TXT, etc.

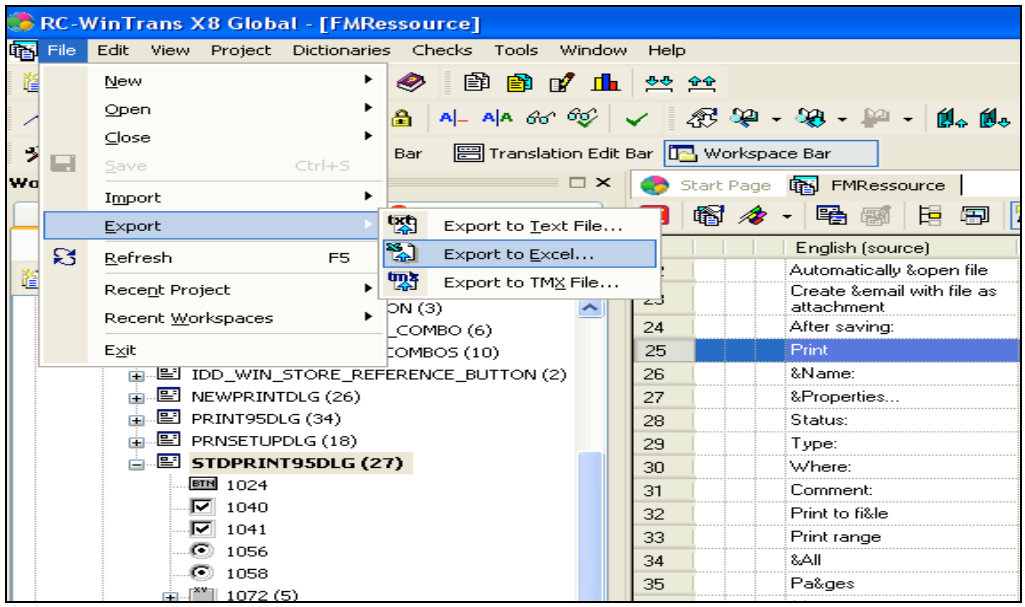


Figure 12 : *Export de la liste des éléments localisables dans un fichier Excel*

WinSoft utilise le format Excel pour ses glossaires, comme le montre la Figure 13 : la première colonne désigne le nom du fichier de ressources à partir duquel la chaîne a été extraite, la deuxième colonne contient l'identifiant de l'interface utilisateur qui contient la chaîne de caractères, la troisième colonne contient l'identifiant de la chaîne de caractères, la quatrième colonne contient le type de la chaîne (étiquette, bouton, etc.), la cinquième colonne contient la chaîne source et la dernière colonne contient la chaîne dans la langue cible.

Source File Name	Main Resname	Resname	Restype	English (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...	
FMPSysDlg.rc	PRINT95DLG # 1095	1095	caption (label)	Status:	
FMPSysDlg.rc	PRINT95DLG # 1094	1094	caption (label)	Type:	

Figure 13 : *Extrait d'un glossaire contenant les chaînes de caractères d'un logiciel propriétaire*

1.1.2.3 Réinjection des chaînes traduites dans les ressources

Les outils de localisation permettent aussi de réinjecter à partir du glossaire les chaînes traduites dans les fichiers de ressources (Figure 14). Une fois cela fait, les localiseurs génèrent des fichiers de ressources localisées.

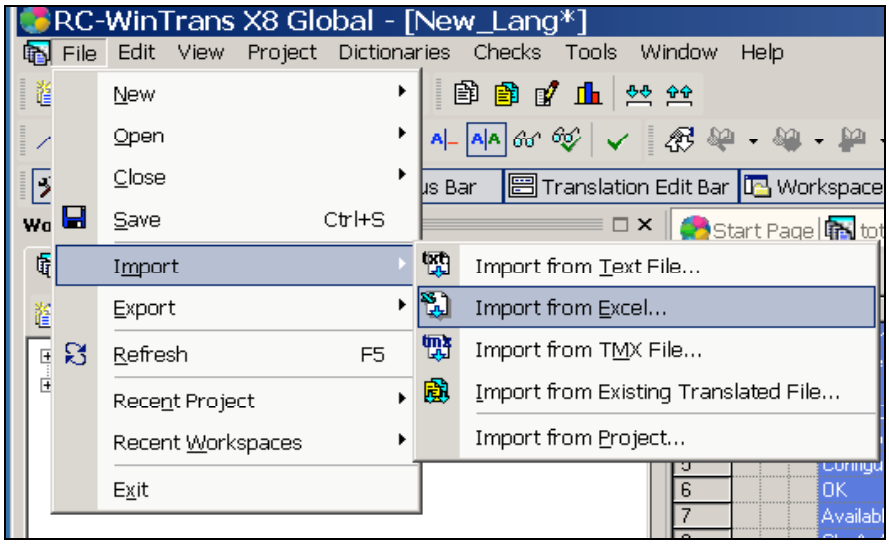


Figure 14 : *Import des traductions à partir du glossaire*

1.2 Traduction des ressources textuelles des logiciels confiée uniquement à des professionnels

Pour traduire les ressources textuelles (chaînes des interfaces, documentation technique du logiciel, aide en ligne, etc.), les localiseurs font appel à des traducteurs et réviseurs professionnels (Figure 15). Le donneur d'ordres envoie la version source des documents (glossaires, documentation technique, aide en ligne, etc.) à plusieurs traducteurs professionnels. Chaque traducteur traduit la ou les parties qui lui ont été confiées et envoie la version traduite au donneur d'ordres, qui se charge d'intégrer les différentes parties et de produire les documents dans les langues cible.

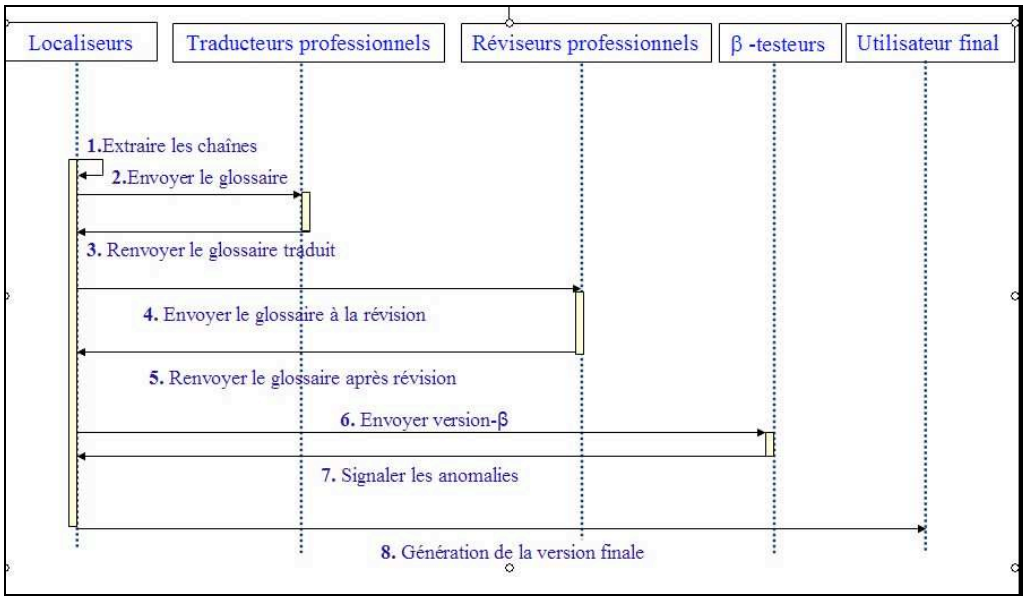


Figure 15 : *Diagramme de séquence du processus de localisation actuel*

1.2.1 Traduction des éléments textuels des interfaces utilisateur

Nous verrons ici comment les éléments textuels des interfaces utilisateur sont traduits.

1.2.1.1 Analyse du glossaire

Avant d'envoyer le glossaire aux traducteurs, WinSoft utilise la mémoire de traductions TRADOS pour calculer le taux de coïncidence des nouveaux segments avec les segments les plus proches d'eux dans la mémoire de traductions. Par exemple, si pour un segment donné on obtient un taux de coïncidence de 100%, cela signifie que le segment a déjà été traduit et que sa traduction est stockée dans la mémoire de traductions. On peut donc réutiliser cette traduction, ce qui permet de réduire le coût de traduction.

Par contre, un taux de coïncidence de 85% signifie qu'un segment ressemblant partiellement à celui que l'on souhaite traduire a déjà été traduit (Figure 16).

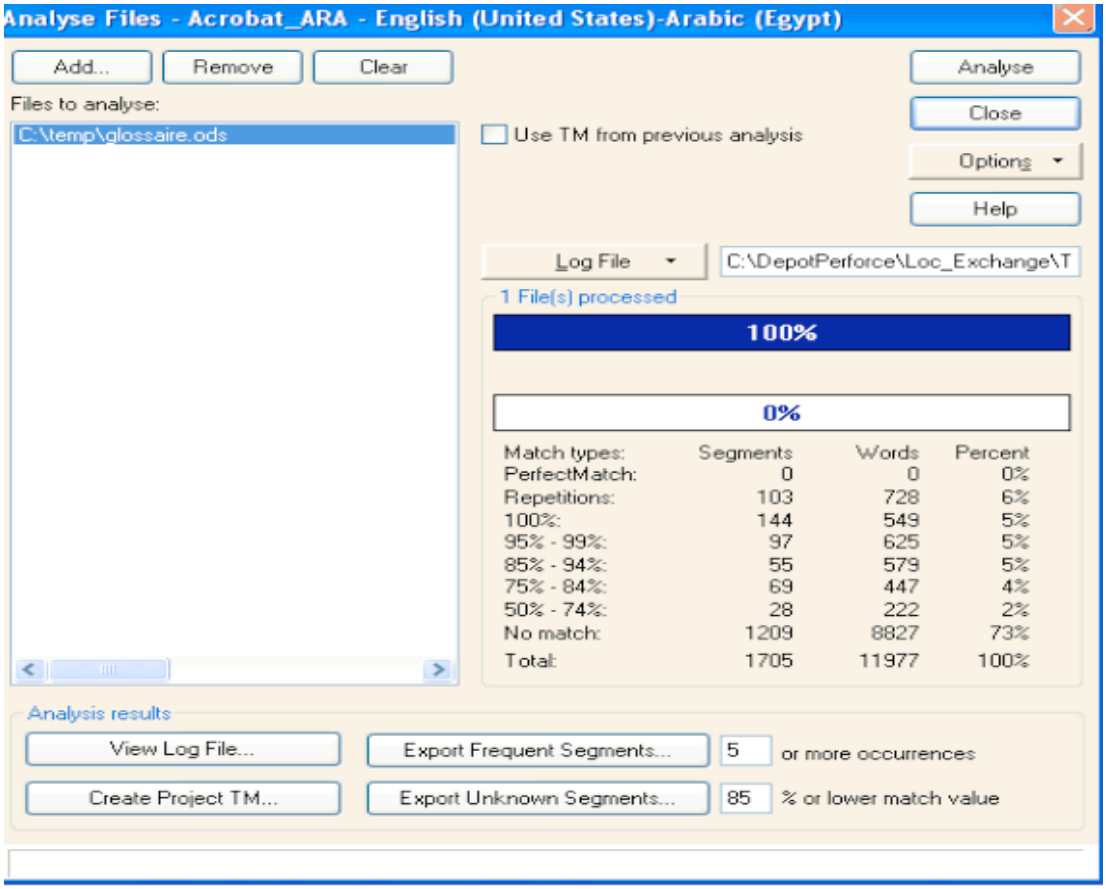


Figure 16 : *Analyse du contenu du glossaire « Acrobat » par la mémoire de traductions TRADOS*

La Figure 16 nous montre le résultat de l'analyse du glossaire « Acrobat_Arabe » : le glossaire contient en total 1705 segments dont 103 répétitions, 144 segments qui ont un taux de coïncidence de 100%, 97 segments qui ont un taux entre 95% et 99%, 55 segments qui ont un taux entre 85% et 94%, 69 segments qui ont un taux entre 75% et 84%, 28 segments qi ont un taux entre 50% et 74% et 1209 segments qui un taux de coïncidence entre 0% et 49%.

1.2.1.2 Traduction du glossaire

Une fois analysé, le glossaire est envoyé à la traduction. En plus du glossaire, certains éditeurs de logiciel ou sociétés de service de localisation envoient aux traducteurs la base terminologique du logiciel, si elle existe. WinSoft par exemple, a construit sa propre base

terminologique “MultiTerm” qui se présente sous la forme d’un fichier Excel et qui contient la terminologie de base de chaque logiciel localisé par WinSoft, c’est-à-dire l’ensemble des termes qui ont été traduits et validés auparavant. L’intérêt de cette base terminologique est d’assurer une certaine cohérence de traduction entre les différentes versions d’un même produit. Le coût de traduction dépend du taux de coïncidence trouvé entre les segments : par exemple, il est beaucoup moins coûteux de traduire des segments qui ont un taux de coïncidence de 90% que des segments qui n’ont que 10% de taux de coïncidence (Figure 17).

		Arabe	Hébreu	Grec	Turc	Ukrainien	Bulgare	Hongrois
Traduction	**Unité de mesure**	**€**	**€**	**€**	**€**	**€**	**€**	**€**
Éléments d'interface								
Taux de coïncidence (0-49%)	mot	0,13	0,120	0,135	0,135	0,077	0,099	0,195
Taux de coïncidence (50-74%)	mot	0,13	0,120	0,135	0,135	0,077	0,099	0,195
Taux de coïncidence (75-84%)	mot	0,1	0,120	0,081	0,081	0,0513	0,066	0,195
Taux de coïncidence (85-94%)	mot	0,075	0,120	0,081	0,081	0,0513	0,066	0,195
Taux de coïncidence (95-99%)	mot	0,045	0,030	0,081	0,081	0,0513	0,066	0,049
Taux de coïncidence (100%)	mot	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Répétitions	mot	0,038	0,030	0,081	0,081	0,0257	0,033	0,049

Figure 17 : *Coût de traduction des chaînes d’interfaces en fonction du taux de coïncidence*

Étant donné que tous les éléments textuels de l'interface utilisateur sont extraits et enregistrés dans des fichiers de traduction spéciaux (glossaires), leur contexte n'est alors plus visible. Par conséquent, il se peut qu'une coïncidence à 100 % avec une chaîne contenue dans une base de données de mémoire de traductions ne corresponde pas au segment source en raison du contexte.

1.2.1.3 Révision du glossaire

La révision consiste à vérifier les traductions qui ont été faites par les traducteurs. Dans le cas de WinSoft, 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 se basant sur la base terminologique du logiciel si elle existe.

Les réviseurs vérifient les traductions qui ont été faites par les traducteurs et corrigent celles qui leur paraissent erronées en ajoutant un commentaire explicitant les raisons de leur correction (Figure 18).

Baseline string	**Localised string**	**Reviewer's Updated string**	**Reviewer's Comments**
Text Variables	Переменные текста	Текстовые переменные	Terminology
Table Styles Panel Menu	Меню панели “Стили таблиц”	Меню панели \"Стили таблиц\"	Syntax
Placeholder &Frames	Шаблонные&Фреймы-заполнители	Фреймы-местозаполнители	Consistency

Balance Ragged Lines	Сбалансировать невыключенные строки	Сбалансировать не выключенные строки	Grammar
No table selection!	Таблицы не выделены!	Таблицы не выделены.	Punctuation
Small Panel Rows	Узкие строки	Строки малой панели	Wrong translation
Package for InCopy and Email	Пакет для InCopy и Email	Пакет для InCopy и электронной почты	Style

Figure 18 : *Extrait d'un glossaire contenant des corrections et des commentaires des réviseurs*

		Arabe	Hébreu	Grec	Turc	Ukrainien	Bulgare	Polonais	Hongrois
	Unité de mesure	**€**	**€**	**€**	**€**	**€**	**€**	**€**	**€**
Révision	**mot**	**0,07**	**0,06**	**0,0528**	**0,055**	**0,0412**	**0,046**	**0,0475**	**0,0528**
		Croate	**Russe**	**Roumain**	**Slovaque**	**Slovène**	**Kazakh**	**Azéri**	**Estonien**
Révision	**mot**	**0,0528**	**0,05**	**0,0528**	**0,048**	**0,05**	**0,07**	**0,04**	**0,0528**

Figure 19 : *Coût de révision par mot pour quelques langues*

1.2.2 Traduction de la documentation technique et de l'aide en ligne

En plus de l'aide en ligne, plusieurs documents techniques peuvent accompagner le logiciel (guide utilisateur, guide d'installation, etc.). Pour traduire des documents de ce type, les localiseurs envoient la version source des documents dans leurs formats d'origine (HTML, DOC, RTF, XML, etc.) aux traducteurs, qui les traduisent en se basant sur la base terminologique associée au logiciel. Chaque traducteur traduit la ou les parties qui lui ont été confiées et envoie la version traduite au donneur d'ordre, qui se charge d'intégrer les différentes parties et de produire les documents dans les langues cible. Comme pour les chaînes d'interface, le coût de la traduction des documents technique ainsi que de l'aide en ligne dépend du taux de coïncidence trouvé entre les segments (Figure 20).

		Arabe	Hébreu	Grec	Turc	Ukrainien	Bulgare	Hongrois
Traduction	**Unité de mesure**	**€**	**€**	**€**	**€**	**€**	**€**	**€**
Documentation								
Taux de coïncidence (0-49%)	mot	0,11	0,100	0,120	0,120	0,066	0,088	0,165
Taux de coïncidence (50-74%)	mot	0,11	0,100	0,120	0,120	0,066	0,088	0,165
Taux de coïncidence (75-84%)	mot	0,085	0,100	0,072	0,072	0,044	0,0587	0,165
Taux de coïncidence (85-94%)	mot	0,06	0,100	0,072	0,072	0,044	0,0587	0,165
Taux de coïncidence (95-99%)	mot	0,04	0,025	0,072	0,072	0,044	0,0587	0,042
Taux de coïncidence (100%)	mot	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Répétitions	mot	0,033	0,025	0,072	0,072	0,220	0,0293	0,042

Figure 20 : *Coût de traduction des documents en fonctions du taux de coïncidence*

La cohérence entre l'interface et la documentation est un problème difficile et particulièrement important pour les logiciels : par exemple, si l'on change le nom d'un bouton dans le document, son nom doit aussi changer dans l'interface.

Conclusion

Nous avons étudié et présenté dans ce premier chapitre les caractéristiques du processus de localisation actuel. Il s'agit d'un processus de localisation externe dans la mesure où les localiseurs n'interviennent pas sur le code source des applications. La traduction des ressources textuelles (chaîne d'interfaces, documentation technique, aide en ligne, etc.) est confiée uniquement à des professionnels.

Cette étude nous sera utile pour identifier et classifier les problèmes associés à la localisation que nous présenterons dans le chapitre suivant.

Chapitre 2 : Problèmes liés au processus de localisation actuel

Introduction

La situation actuelle de la localisation de logiciel présente certains problèmes récurrents qui sont signalés partout dans le monde de la localisation et pour lesquels il n'existe, jusqu'à maintenant, aucune solution. Avec, l'explosion de « nouveaux marchés » ainsi que de « nouvelles langues », des nouveaux problèmes se sont apparus. Le but de ce chapitre est d'identifier ainsi que de classifier ces problèmes anciens et nouveaux liés à la localisation.

2.1 Problème de qualité

Le problème de qualité est un problème majeur du processus classique de localisation. Le vice président de *Translation.com* (Hans Fenstermacher) dit de la qualité: *« we can't define quality, but we know it when we see it »*[4]. C'est un problème récurrent qui est signalé partout dans le monde. Il a été mentionné à l'atelier L4Trans-III de LREC-06 par le responsable de la localisation de CATIA en japonais chez IBM-Japon (Kudo). Il a été aussi mentionné lors de la 14éme conférence annuelle LRC XIV de localisation et internationalisation[5] par plusieurs responsables de localisation. Nous analysons dans la suite les sources de ce problème ainsi que les approches actuelles utilisées pour tenter de résoudre.

2.1.1 Analyse des sources de problème

Nous analysons ici les principales sources de problème de la qualité.

2.1.1.1 Traduction hors contexte

La première raison de ce problème de qualité est que la traduction des éléments d'interface est faite hors contexte. En effet, le contexte des éléments d'interface à traduire par les traducteurs n'est pas fourni aux traducteurs, ce qui fait que les relations de proximité dans le temps et l'espace leur sont inaccessibles, alors que le contexte dans lequel un texte est lu contient de nombreuses informations auxquelles le texte fait souvent référence (identification d'un bouton par sa couleur, d'une icône par sa position). 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.

Prenons l'exemple du mot "layer" en anglais. Selon le contexte dans lequel il apparaît, il peut être traduit en français par « couche » ou par « calque ». De même, le mot anglais "Type" est traduit différemment en grec dans le même logiciel "Adobe InDesign" par

[4] Définition de la qualité donnée par Hans Fenstermercher lors la 14^{éme} conférence annuelle LRC XIV de la localisation et de l'internationalisation à Limerick en Septembre 2009.

[5] J'ai participé à cette conférence et j'ai assisté aux présentations de Dereck McCann, Francis Tsang, Jason Rickard et d'autres personnes travaillant dans le domaine de localisation.

« Τύπος » ou par « Κείμενο » en fonction de la boîte de dialogue dans laquelle la chaîne apparaît (Figure 21, Figure 22, Figure 23, Figure 24).

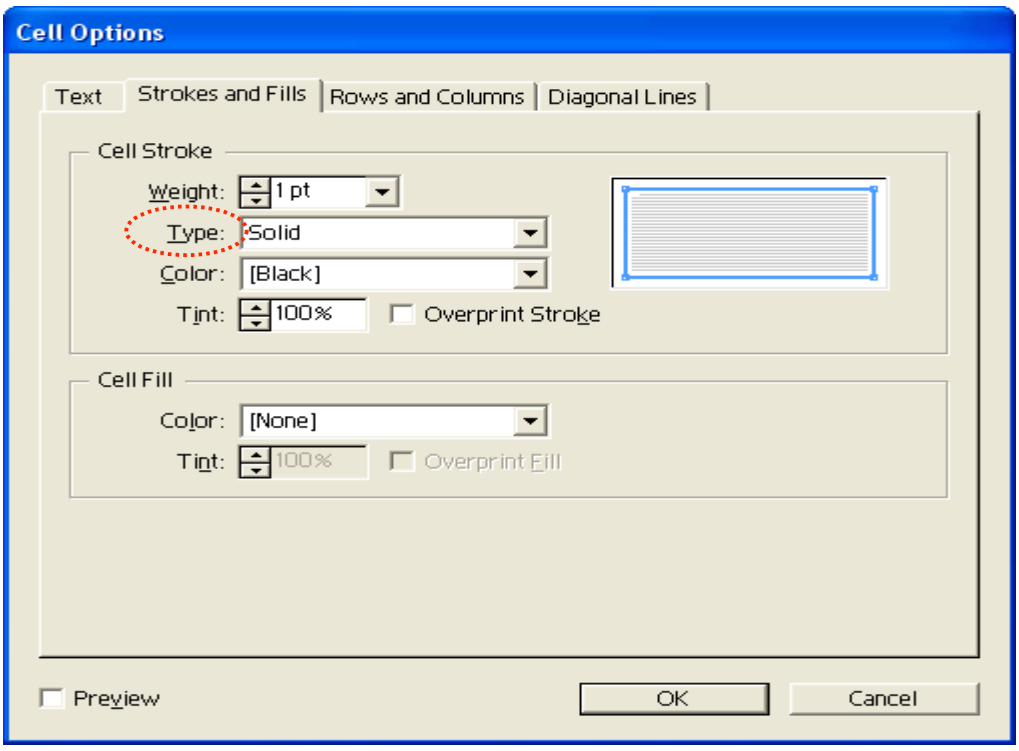


Figure 21 : *Boîte de dialogue du logiciel InDesign 6 d'Adobe en anglais*

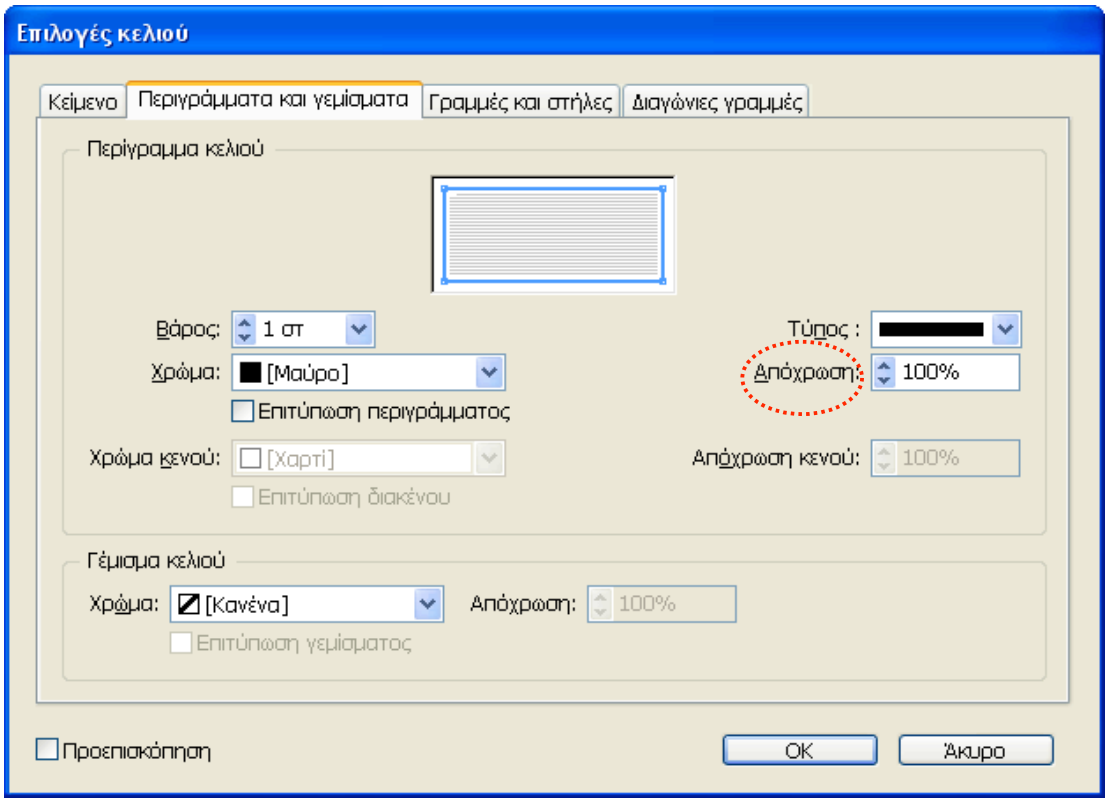


Figure 22 : *Boîte de dialogue de la figure 1, localisée en grec, contenant la chaîne « Τύπος » comme traduction du mot anglais « type »*

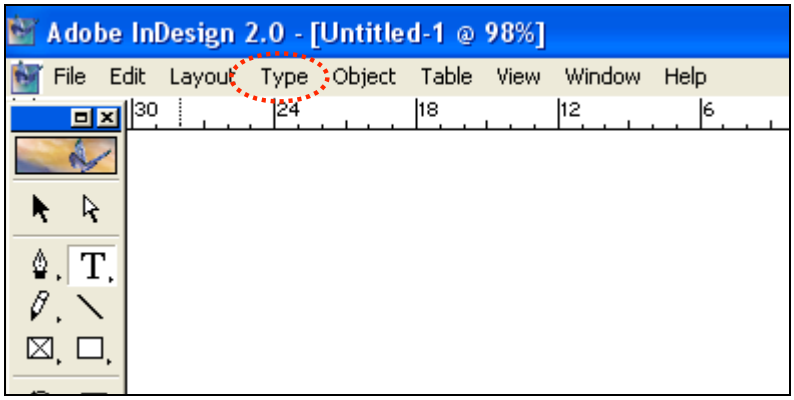


Figure 23 : *Fenêtre principale du logiciel InDesign 6 en anglais, contenant le menu « Type »*

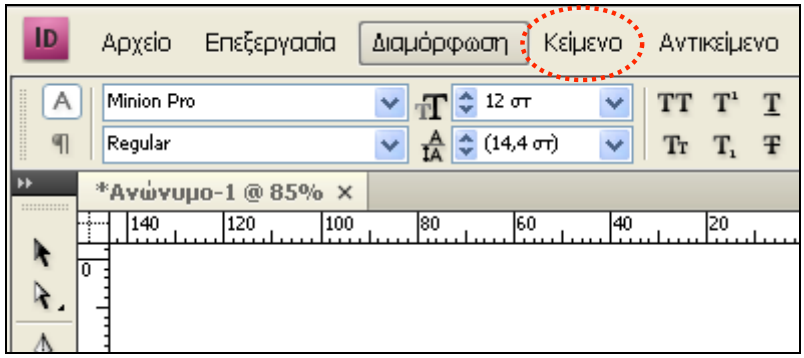


Figure 24 : *Fenêtre principale de InDesign 6 localisée en grec et contenant le menu « Κείμενο » comme traduction du menu « Type »*

2.1.1.2 Traduction arbitraire des termes métier

La maîtrise de la langue cible est une condition insuffisante pour produire des traductions de qualité. Les traducteurs professionnels ne sont pas des spécialistes du contexte métier dans lequel le logiciel est utilisé. En effet, comme l'explique (Lepouras&Weir, 2001), pour traduire un logiciel de l'anglais vers une autre langue cible, les traducteurs essaient de trouver les termes correspondant aux termes anglais dans la langue cible. Inévitablement, les critères de choix des termes dans la langue cible sont arbitraires. Par conséquent, certains aspects des logiciels localisés semblent étranges à leurs utilisateurs, ce qui explique le fait que certains préfèrent la version anglaise à la version localisée (Lepouras & Weir, 2000).

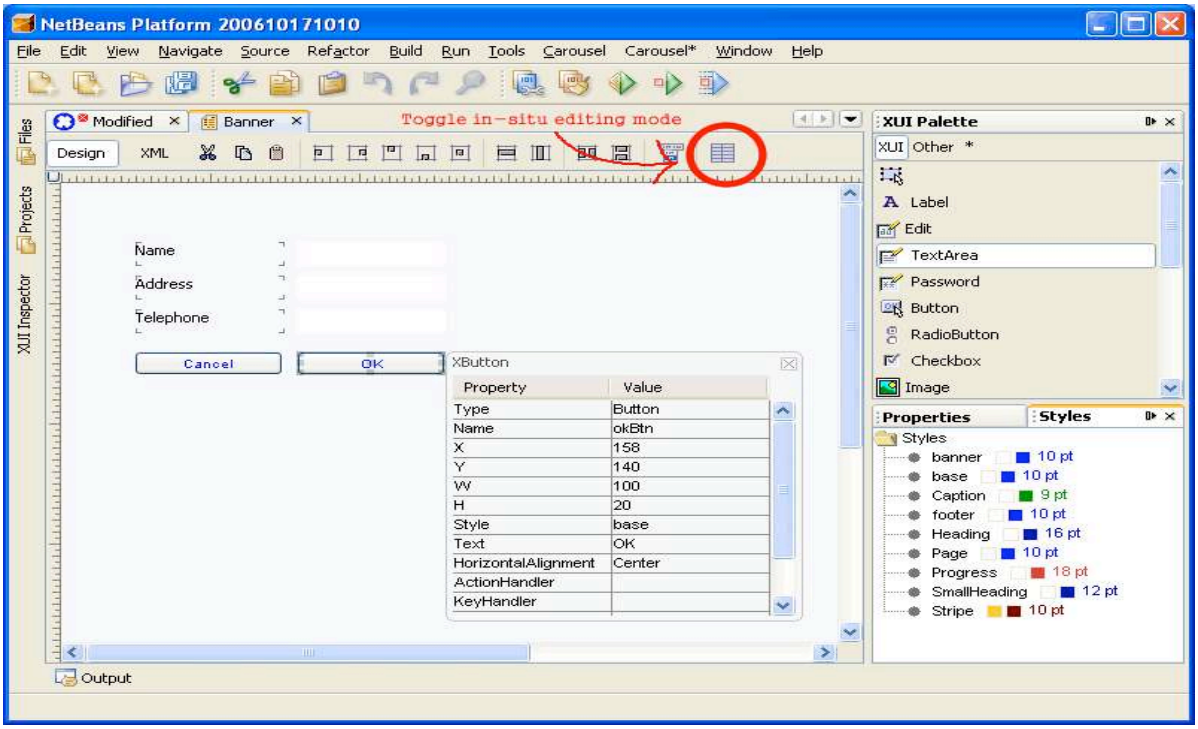
L'incohérence des traductions entre applications est évidente, ce qui valide le fait que le choix des termes est arbitraire (Figure 25). Par exemple, le terme "Header" en anglais est traduit an grec moderne par « κεφαλίδα », dans une des versions de Microsoft Word. Le même terme est traduit par « υπέρτιτλος » dans une version de Lotus Ami Pro (ce qui veut dire « titre principal »). Enfin, dans une version en grec de WordPerfect, "Header" est traduit par « τίτλος σελίδας » (littéralement « titre de page »).

Terme en anglais	Word 6.0	Lotus Ami Pro	WordPerfect
Footer	υποσέλιδο	υπότιτλος	υποσέλιδο
Bullet	κουκκίδα	σύμβολο	σφαίρα
Undo	αναίρεση	ακύρωση	ακύρωση
View	προβολή	όψη	θέα

Figure 25 : *Exemples de variation de terminologie dans les logiciels localisés*

2.1.2 Approches actuelles pour combattre le problème de la qualité

Pour l'instant, à notre connaissance, aucune approche ne permet de résoudre le problème de la qualité de traduction, même partiellement, en ce qui concerne des logiciels existants. Une solution en génie logiciel "In Situ translation" a été proposée par *Xeotrope*[6] (Xeotrope, 2008) pour résoudre le problème de la traduction hors contexte, mais il s'agit d'une méthodologie de construction d'un logiciel. Cette solution est destinée aux développeurs des applications XUI[7]. Le plugin KalIDEoscope 3.0[8] de *Xeotrope* fournit un éditeur visuel pour les applications XUI sous Eclipse ou NetBeans. Cet éditeur visuel fonctionne suivant le principe du glisser-déposer et il peut être utilisé en mode "*in situ*" (Figure 26) pour éditer les composants graphiques (bouton, étiquette, etc.).



[6] http://www.xoetrope.com/

[7] Plate-forme JAVA et XML permettant de construire des applications internet riches (RIA). Swing, AWT et d'autres ensembles de Widgets peuvent être utilisés.
http://sourceforge.net/projects/xui

[8] KalIDEoscope : plugin libre à code source ouvert.
http://xui.sourceforge.net/wikka/wikka.php?wakka=KalIDEoscope

Figure 26 : *Éditeur visuel des composants graphiques en mode "in-situ editing"*

Pour traduire le texte d'un composant graphique, il suffit de faire un double clic sur le composant graphique et de choisir la traduction correspondante parmi une liste de traductions disponibles ou de rajouter une nouvelle traduction.

Figure 27 : *Choix du texte parmi la liste de traductions disponibles*

Les inconvénients de cette approche sont (1) que les développeurs ne sont pas des traducteurs et ne maîtrisent pas toutes les langues cibles, et (2) qu'elle est inapplicable à la très grande majorité des applications, car elles ne sont pas construites avec cet outil.

2.2 Problèmes de gestion des données à localiser

Le problème de gestion est aussi un problème majeur du processus de localisation actuel. Pour localiser leurs produits, la majorité des éditeurs de logiciels font appel à des sociétés de service de localisation qui font appel à leur tour à des sociétés de traduction, des sociétés de révision, des sociétés de LQA (Linguistic Quality Assurance), des β-testeurs et des distributeurs. Les bêta-testeurs et les distributeurs participent uniquement au test du produit. Le processus est le suivant : le donneur d'ordres (éditeur de logiciel ou société de service de localisation) envoie les documents en langue source (documents techniques, aide en ligne, glossaire, version source du logiciel, etc.) aux différents intervenants. Chaque intervenant contribue de son côté et envoie ses contributions au donneur d'ordres qui se charge d'intégrer les contributions des différents intervenants. Ainsi, le donneur d'ordres se retrouve face à des flux de données importants qu'il doit gérer simultanément tout au long du processus de localisation. La gestion de ces flux est une tâche difficile, surtout quand il y a un nombre important de langues cibles. À cause du problème de gestion, il est presque impossible de publier le même jour un logiciel dans toutes les langues cible. Localiser N logiciels vers n langues revient à gérer $N \times n$ glossaires, $N \times n$ documents techniques, $N \times n$ aides en ligne, etc. Par exemple, WinSoft localise 12 produits d'Adobe vers 17 langues, et doit donc gérer 204 glossaires, 204 documents techniques et 204 aides en ligne, soit au total 612 documents.

Cela représente un coût important en personnel s'occupant uniquement de la gestion de ces flux de données. L'importance de ce problème a été soulignée à plusieurs reprises par plusieurs responsables de la localisation lors de la conférence LRC XIV tels que le directeur de localisation chez Adobe (Francis Tsang), le directeur de la localisation chez Microsoft (Derek McCann) ainsi que la directrice des opérations internationales chez EMC "Entreprise Content Management System" (Jessica Rolland).

2.2.1 Analyse des sources du problème de gestion

La principale source du problème de gestion est le manque d'outils de travail collaboratif, de centralisation et de partage des données entre les différents intervenants. En effet, les éditeurs de logiciels, les sociétés de localisation, les traducteurs, les réviseurs, les

sociétés de LQA, les testeurs et les distributeurs produisent des résultats hétérogènes, et donc de mauvaise qualité, car ils travaillent tous chacun de leur côté, sur des parties différentes du contenu textuel, sans vision d'ensemble (Figure 28). Ce modèle est tout à fait à l'opposé du modèle de localisation des logiciels libres (Figure 29), qui se base sur la collaboration et le partage des données entre les différents contributeurs à localisation.

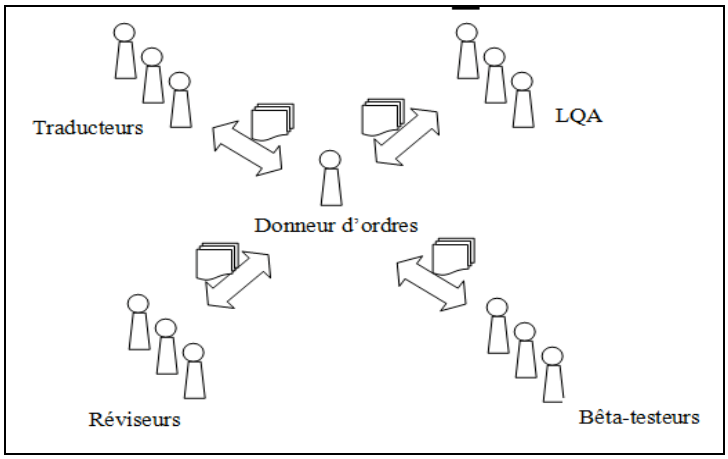


Figure 28 : *Modèle de localisation des logiciels commerciaux*

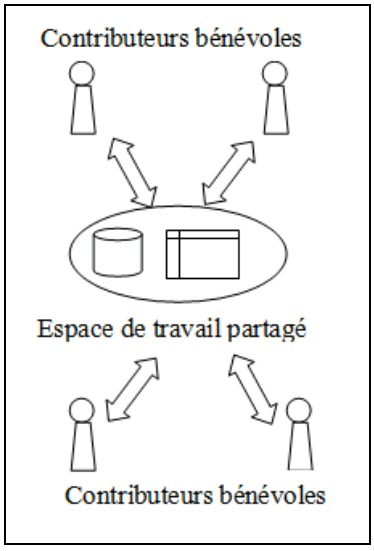


Figure 29 : *Modèle de localisation des logiciels libres*

2.2.2 Débuts de solutions

Certains éditeurs de logiciels ont essayé de mettre en place des outils permettant la centralisation et le partage des données entre les différents intervenants dans le processus de localisation. EMC[9] (Entreprise Content Management) (EMC, 2009) par exemple a mis en place un environnement nommé "Business To Business" qui permet à différents distributeurs, qui sont installés dans 80 pays, de participer à la localisation. Il s'agit d'une plate-forme en ligne qui permet à ses employés et plus particulièrement à ses distributeurs de visualiser, éditer et traduire vers leurs langues maternelles les documents en langue source (glossaires, documents techniques, aide en ligne, etc.).

[9] http://www.emc.com/products/category/content-management.htm

Les contributions des distributeurs sont centralisées dans une base de données contrôlée par l'éditeur de logiciels, qui peut ainsi visualiser toutes les contributions des distributeurs et les valider. Il s'agit donc d'une solution partielle qui permet de gérer uniquement les contributions des distributeurs, et pas celles des traducteurs, réviseurs et testeurs.

2.3 Problèmes de coût et de délais

Les problèmes de coût et de délai de localisation sont des problèmes majeurs du processus de localisation actuel, auxquels font face tous les éditeurs de logiciels.

2.3.1 Problème de coût

Le coût de localisation est trop élevé, surtout quand il s'agit de localiser vers des langues peu dotées. WinSoft adapte et traduit les produits d'Adobe et de FileMaker pour le monde arabe et l'Europe Centrale, et le budget explose avec le nombre de langues (17 langues). La Figure 30 montre le coût de traduction et de LQA de l'anglais vers l'arabe des éléments d'interface du logiciel Photoshop 11.

Comme on l'a vu, le coût de traduction dépend du taux de coïncidence trouvé. Par contre le coût de LQA est fixe et dépend du nombre de jours passés pour assurer la qualité linguistique du logiciel. Ainsi, plus le logiciel est volumineux, plus le coût de LQA est élevé. Par exemple, le logiciel Adobe Photoshop 11 nécessite environ 22 jours de LQA contre 3 jours pour FileMaker 10 (Figure 31).

Traduction éléments d'interface	Coût /mot	Nombre de mots	Coût de traduction (€)
No match	0,13	12725 mots	1654,25
Match 50-74%	0,13	196 mots	21,97
Match 75-84%	0,1	984 mots	98,4
Match 85-94%	0,075	1072 mots	80,4
Match 99-86%	0,045	756 mots	34,02
Match 100%	0,000	1383 mots	0,000
Répétition	0,038	3586 mots	136,26
Total traduction		20702 mots	2025,30
LQA	Coût /Jour	Nombre de jours	Coût LQA (€)
	250	~ 22 jours	5590,432
Coût total			7615,74

Figure 30 : *Coût de traduction et de LQA des éléments textuels des interfaces utilisateur de Photoshop 11*

Traduction éléments d'interface	Coût /mot	Nombre de mots	Coût de traduction (€)
No match	0,135	5136 mots	699,36
Match 50-74%	0,135	25 mots	3,375
Match 75-84%	0,081	46 mots	3,726
Match 85-94%	0,081	69 mots	5,589
Match 99-86%	0,081	6 mots	0,486
Match 100%	0,000	19 mots	0,000
Répétition	0,081	1451 mots	117,53
Total traduction		6752 mots	830,06
LQA	Prix/Jour	Nombre de jours	Coût LQA (€)
	352	~ 3 jours	1106,93
Coût total			1937

Figure 31 : *Coût de traduction et de LQA des éléments textuels des interfaces utilisateur de FileMaker 10*

En plus du coût de la traduction, de la révision et de la LQA, le coût de la localisation inclut aussi le coût des personnels qui gèrent la localisation en interne : les localiseurs, les testeurs internes, etc. Par exemple, le coût total du projet de localisation d'Acrobat Reader vers les 3 langues arabe, hébreu et grec, est d'environ 200.000 euros. Le coût total du projet de localisation de FileMaker est de 248.000 euros. Il comporte la localisation de l'interface utilisateur vers 4 langues (turc, russe, tchèque, polonais) et la localisation de la documentation technique vers 4 langues (arabe, hébreu, grec, hongrois).

Cela oblige les éditeurs de logiciels, dont la majorité se trouvant aux USA et en Europe, à limiter le nombre de langues vers lesquelles ils localisent leurs produits. Par exemple, les produits Microsoft ne sont localisés qu'en 38 langues, ceux d'Adobe en 32 langues et ceux d'IBM en 38 langues (Figure 32).

			Adobe	Microsoft	IBM
	Langues	**Nombre de locuteurs (millions)**			
1	Allemand	90,3	×	×	×
2	Anglais (US)	341	×	×	×
3	Arabe	221	×	×	×
4	Bulgare	9.1	×	×	×
5	Catalan	11.5			×
6	Chinois simplifié	845	×	×	×
7	Chinois traditionnel		×	×	×
8	Chinois traditionnel HK			×	
9	Coréen	66.3	×	×	×
10	Croate	6	×	×	×
11	Danois	5.6	×	×	×
12	Espagnol (Espagne)	329	×	×	×
13	Estonien	1.1	×	×	×
14	Finnois	5	×	×	×
15	Français (France)	67.8	×	×	×
16	Grec	13.1	×	×	×
17	Hébreu	5.3	×	×	×
18	Hindi	182		×	×
19	Hongrois	12.5	×	×	×
20	Italien	61.7	×	×	×
21	Japonais	122	×	×	×
22	Kazakh	8.3		×	×
23	Letton	1.5	×	×	×
24	Lituanien	3.2	×	×	×
25	Néerlandais	21.7	×	×	×
26	Norvégien	5	×	×	×
27	Polonais	40	×	×	×
28	Portugais (Portugal)	178		×	×
29	Portugais-Brésil		×	×	×
30	Roumain	23.4	×	×	×
31	Russe	144	×	×	×
32	Serbe	7		×	×
33	Slovaque	5	×	×	×
34	Slovène	2.4	×	×	×
35	Suédois	8.3	×	×	×
36	Tchèque	9.5	×	×	×
37	Thaï	20.4		×	×
38	Turc	50.8	×	×	×
39	Ukrainien	37	×	×	×

Figure 32 : *Langues vers lesquelles sont localisés certains produits de Microsoft, Adobe et IBM*

2.3.2 Problèmes de délais

Le délai de localisation est aussi un problème du processus de localisation actuel. En effet, le décalage entre la commercialisation de la version source du logiciel et des versions localisées est souvent important. Dans leur livre (A. Souphavan et T. Karoonboonyanan, 2005) proposent une formule permettant d'estimer le délai de localisation d'un logiciel donné (Figure 34). La formule est pondérée par un cœfficient qui varie entre 1,5 et 0,75. Ce cœfficient indique le niveau d'expérience des traducteurs (0.75 pour un traducteur qui a une grande expérience et 1.5 pour un traducteur débutant). En effet, plus ce coefficient est petit, plus le délai de traduction est court.

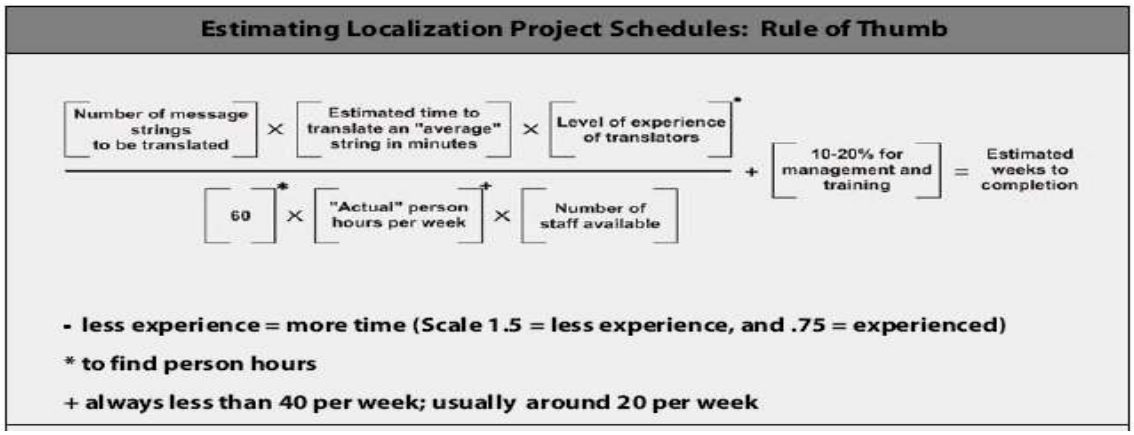


Figure 33 : *Estimation de nombre de semaine d'un projet de localisation*

Exemple 1 :

- 10.000 chaînes à traduire
- 10 minutes par chaîne
- Coefficient d'expérience en traduction : 0,75 (traducteurs expérimentés)
- 10 personnes

Ainsi, en choisissant 10 traducteurs expérimentés (coefficient 0,75) et en appliquant la formule de la Figure 33, le délai de localisation est d'environ 2 mois pour un logiciel qui contient 10.000 chaînes (entre l'interface utilisateur, la documentation technique et l'aide en ligne).

2.3.3 Analyse des sources des problèmes de coût et de délais

La principale raison des problèmes de coût et de délais est du au fait que les traductions des ressources textuelles sont faites uniquement par des professionnels. Cela rend le coût trop élevé et les délais de localisation trop longs, surtout quand il s'agit de localiser vers des langues peu dotées. Pour la plupart des « nouvelles langues », pour des raisons de coût et surtout de rareté, de cherté, ou d'absence de traducteurs professionnels, il faut trouver une autre façon de faire. La Figure 34 montre le coût de traduction par mot des éléments textuels des interfaces utilisateur ainsi que de la documentation technique et de l'aide en ligne en fonction du taux de coïncidence trouvé. Cela nous permet de constater la différence de prix de la traduction entre les différentes langues. Par exemple, le coût de traduction d'un segment

qui a un taux de coïncidence compris entre 0 et 49% en hongrois (0,195 €) est environ quatre fois plus cher que celui en croate (0,0451 €) si on traduit à partir de l'anglais.

Traduction	Unité de mesure	Croate €	Grec €	Turc €	Ukrainien €	Bulgare €	Hongrois €
Éléments d'interface							
Taux de coïncidence (0-49%)	mot	0,0451	0,135	0,135	0,077	0,099	0,195
Taux de coïncidence (50-74%)	mot	0,0451	0,135	0,135	0,077	0,099	0,195
Taux de coïncidence (75-84%)	mot	0,0451	0,081	0,081	0,0513	0,066	0,195
Taux de coïncidence (85-94%)	mot	0,120	0,081	0,081	0,0513	0,066	0,195
Taux de coïncidence (95-99%)	mot	0,030	0,081	0,081	0,0513	0,066	0,049
Taux de coïncidence (100%)	mot	0,000	0,000	0,000	0,000	0,000	0,000
Répétitions	mot	0,030	0,081	0,081	0,0257	0,033	0,049
Documentation technique et aide en ligne							
Taux de coïncidence (0-49%)	mot	0,100	0,120	0,120	0,066	0,088	0,165
Taux de coïncidence (50-74%)	mot	0,100	0,120	0,120	0,066	0,088	0,165
Taux de coïncidence (75-84%)	mot	0,100	0,072	0,072	0,044	0,0587	0,165
Taux de coïncidence (85-94%)	mot	0,100	0,072	0,072	0,044	0,0587	0,165
Taux de coïncidence (95-99%)	mot	0,025	0,072	0,072	0,044	0,0587	0,042
Taux de coïncidence (100%)	mot	0,000	0,000	0,000	0,000	0,000	0,000
Répétitions	mot	0,025	0,072	0,072	0,220	0,0293	0,042

Figure 34 : *Coût de traduction par mot des éléments d'interface, de la documentation technique et de l'aide en ligne en fonction du taux de coïncidence trouvé*

2.3.4 Idées de solutions

Afin de réduire le coût et les délais de localisation, certains éditeurs de logiciels essaient de s'approcher du modèle de localisation utilisé dans le monde des logiciels libres. Ce dernier repose sur la contribution collaborative en ligne des bénévoles. En effet, on note récemment qu'il y a un nombre important de traducteurs volontaires qui traduisent des centaines de documents dans différents domaines. La suite Mozilla par exemple est localisée maintenant en 76 langues par des centaines de bénévoles (paragraphe 3.1.1.1) qui sont partout dans le monde (soit le double des langues des produits Microsoft et Adobe). D'autre part, Symantec (Symantec, 2009), a mis en place l'outil "pctools" qui permet à des contributeurs volontaires en ligne de traduire les éléments textuels des interfaces utilisateur de certains produits Symantec (Figure 37). Pour pouvoir utiliser l'outil "pctools", il suffit à un utilisateur de se créer un compte (Figure 35), puis de choisir le projet ainsi que la langue cible vers laquelle celui-ci souhaite traduire les chaînes d'interface (Figure 36).

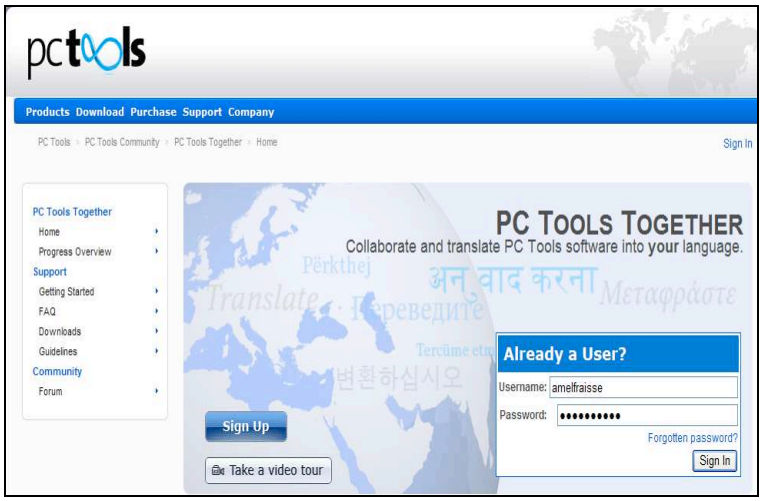


Figure 35 : *Page d'identification de l'outil "pctools".*

Figure 36 : *Choix de la langue cible dans l'outil "pctools".*

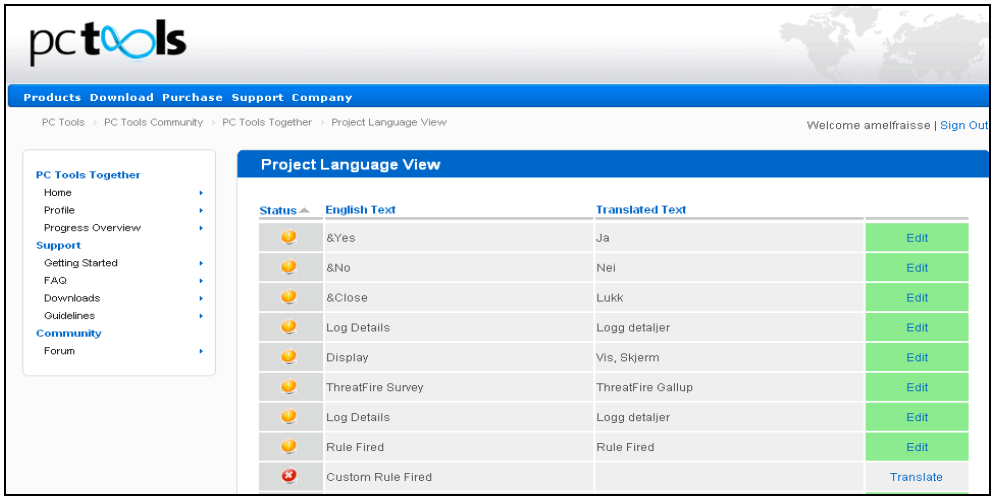


Figure 37 : *Interface permettant l'édition et la traduction des éléments textuels des interfaces.*

La première colonne de la Figure 37, indique le statut de la chaîne, la deuxième colonne contient les chaînes source, la troisième les traductions disponibles, et la dernière sert à éditer ou traduire les chaînes.

Cette solution permet donc à des contributeurs bénévoles de traduire certains éléments textuels des interfaces utilisateur, mais ne résout pas le problème de la qualité, étant donné

que le contexte dans lequel les chaînes sont utilisées n'est pas visible par les contributeurs. Par contre, pour résoudre le problème de délai, il n'existe pour l'instant aucune solution.

Chapitre 3 : Recherche de solutions adaptées

Introduction

Nous avons présenté dans le chapitre précédent les problèmes liés au processus de localisation actuel. Nous avons aussi analysé les sources de ces problèmes ainsi que les pistes de solution existantes. Ce chapitre est organisé autour de deux grandes parties qui représentent les deux solutions que nous proposons pour résoudre les différents problèmes liés à la localisation de logiciels commerciaux et libres.

3.1 Faire participer plus que les traducteurs professionnels

Un des problèmes du processus de localisation actuel, identifié dans le chapitre précédent, est que les traductions sont effectuées uniquement par des professionnels, et qu'il semble impossible de continuer de cette façon, pour la plupart des « nouvelles langues », pour des raisons de coût et surtout de rareté ou d'absence de traducteurs professionnels.

D'un autre côté, on observe que les logiciels libres comme Mozilla sont traduits par des contributeurs bénévoles dans beaucoup plus de langues que les logiciels commerciaux (76 langues). Une autre situation (différente de la traduction de documentation technique et des éléments d'interfaces) est celle où les traducteurs sont des contributeurs bénévoles occasionnels, sans lien organique avec le projet, et dans laquelle on obtient des traductions de qualité dans plus d'une centaine de langues (cas de l'encyclopédie Wikipedia ou de textes d'Amnesty International ou de Pax Humana[10]), les applications IToldU (Bellynck et al., 2005), Yakushite (Yakushite.Net, 2008), (Murata et al., 2003) ou Papillon (Papillon, 2009) sont aussi des exemples d'applications qui reposent sur la contribution des bénévoles en ligne.

Une solution au problème posé serait donc de s'inspirer du modèle de localisation utilisé dans le monde des logiciels libres en faisant intervenir des contributeurs bénévoles dans le processus de localisation.

3.1.1 Faire participer des contributeurs bénévoles

Dans cette section, nous présentons deux exemples d'importants projets de localisation qui se basent sur la contribution des contributeurs bénévoles : le projet de localisation de Mozilla et celui d'Ubuntu.

3.1.1.1 Projet de localisation de Mozilla

[10] http://paxhumana.info/fr.php3.

Mozilla est localisé maintenant vers 76 langues uniquement par des contributeurs bénévoles (Figure 38). Cela permet à un très grand nombre d'utilisateur, ne parlant pas l'anglais (qui est la langue source des applications Mozilla), de pouvoir utiliser les applications Mozilla dans leurs langues maternelles. Prenons l'exemple d'un pays comme l'Inde, qui compte 23 langues officielles et plus de 1600 dialectes. Selon les estimations (IndexMundi, 2008), sur les 1.147.995.898 habitants il y a que 5% qui parlent anglais de façon « opérationnelle ». Le hindi par exemple, qui est considérée, comme la première langue officielle de l'Inde est parlé par 420 millions de locuteurs dans le monde, le bengali par 215 millions de locuteurs et le punjâbi par 88 millions de locuteurs (Figure 38).

	Langues / Systèmes d'ecriture	Pays principal	Nombre de locuteurs natifs (millions) (Ethnologue estimation, 2005)	Nombre de locuteurs total dans le monde (millions)
1	Afrikaans	Afrique du Sud	6.45	13
2	Albanais	Albanie	6	~9
3	Allemand	Allemagne	101	128
4	Anglais	Royaume-Uni	341	510
5	Anglais	États-Unis		
6	Arabe	Arabie Saoudite	221	230
7	Assamais	Inde		16,8
8	Basque	Pays Basque		6
9	Bengali	Bengladesh	196	215
10	Bengali	Inde		
11	Biélorusse	Biélorussie		9
12	Bulgare	Bulgarie		9,1
13	Catalan	Espagne		11,5
14	Chinois simplifié	Chine		845
15	Chinois traditionnel	Chine		
16	Chinois traditionnel HK	Chine		
17	Coréen	Corée du Nord et Corée du Sud		71
18	Croate	Croatie		6
19	Danois	Danemark		5,6
20	Espagnol	Espagne	350	420
21	Espagnol	Argentine		
22	Espagnol	Chili		
23	Espagnol	Mexique		
24	Espéranto			2
25	Estonien	Estonie		1,1
26	Finnois	Finlande		5
27	Français	France	67	130
28	Frison	Nord des Pays-Bas		0,6
29	Galicien	Espagne		3,2
30	Gallois	Pays de Galles		7,5
31	Géorgien	Géorgie		4,3
32	Goudjarâtî	Inde		46
33	Grec	Grèce		13,1
34	Hébreu	Israël		5,3
35	Hindi	Inde	370	490
36	Hongrois	Hongrie		12,5
37	Indonésien	Indonésie		225
38	Irlandais	Irlande		3,5
39	Islandais	Islande		3,2
40	Italien	Italie		61,7
41	Japonais	Japon	126	127
42	Kannada	Inde	35	44

	Langues / Systèmes d'écriture	Pays principale	Nombre de locuteurs natives (millions) (Ethnologue estimation, 2005)	Nombre de locuteurs total dans le monde (millions)
43	Kazakh	Kazakhstan		8,3
44	Kurde	Iraque		16
45	Letton	Lettonie		1,5
46	Lituanien	Lituanie		3,2
47	Macédonien	Macédonie		2,3
48	Malayalam	Inde		37
49	Marathe	Inde	68	71
50	Mongol	Mongolie		5,7
51	Néerlandais	Pays-Bas		21,7
52	Bokmål	Norvège		5
53	Nynorsk	Norvège		
54	Occitan	France		1
55	Oriya	Inde		32
56	Pendjabi	Inde	60	88
57	Persan			54
58	Polonais	Pologne		46
59	Portugais	Portugal	203	213
60	Portugais-Brésilien	Brésil		
61	Romanche			
62	Roumain	Roumanie		23,4
63	Russe	Russie	145	255
64	Serbe	Serbie		7
65	Singhalais			15,6
66	Slovaque			5
67	Slovène	Slovénie		2,4
68	Suédois			8,3
69	Tamil		68	77
70	Tamil	Sri Lanka		
71	Tchèque	République Tchèque		9,5
72	Télougou		70	75
73	Thaï	Thaïlande	20	60
74	Turc	Turquie	60	75
75	Ukrainien	Ukraine		39
76	Vietnamien	Vietnam	70	86

Figure 38 : *Langues vers lesquelles les applications Mozilla sont localisées*

La participation aux projets de localisation de Mozilla est plus ou moins simple. Elle nécessite des connaissances particulières ainsi que la maîtrise de certains outils de localisation spécifiques aux applications Mozilla. En effet, dans le cas de ces applications, toutes les chaînes d'interface sont stockées dans le dossier chrome de l'application, et plus précisément dans le fichier binaire .jar de ce dossier, qui porte généralement le nom de la langue cible, par exemple fr.jar pour la langue française (Figure 39). Dans certains cas, les chaînes sont stockées dans plusieurs fichiers de ressources .jar.

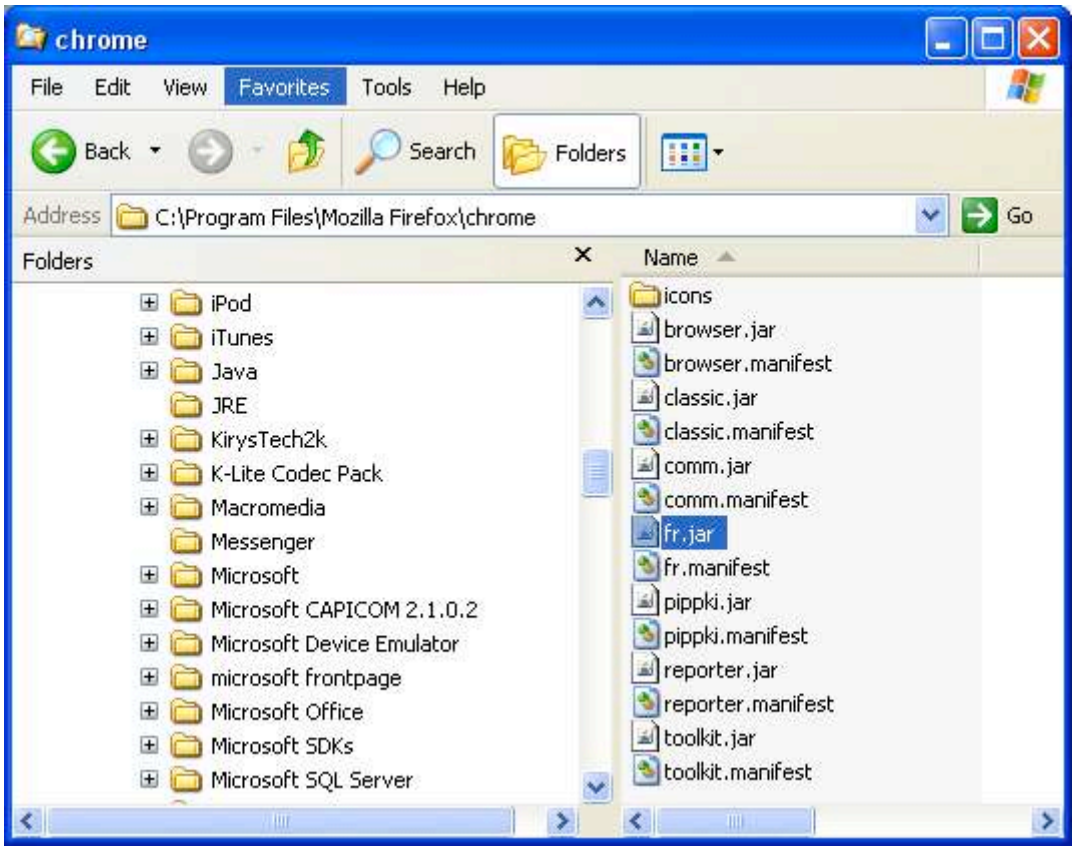


Figure 39 : *Le fichier fr.jar du logiciel Mozilla Firefox qui contient les éléments textuels des interfaces utilisateur*

Ainsi, pour localiser une application Mozilla (Firefox, Thunderbird, etc.), il faut éditer tous les fichiers de ressources du dossier chrome. L'édition de ces fichiers de ressources binaires se fait en utilisant un des outils de localisation de Mozilla (Mozilla, 2006) tels que *MozExpTool*, *MozillaTranslator*, *MozPOTools* et *L10NZilla.*

L'outil le plus utilisé est *MozillaTranslator*. Pour l'utiliser, il faut importer le dossier chrome et spécifier la langue cible vers laquelle on souhaite localiser l'application Mozilla. La Figure 40 montre la fenêtre principale de *MozillaTranslator*. À gauche de la fenêtre, il y a la liste des fichiers de ressources du dossier chrome et à droite les chaînes des interfaces utilisateur. Dans le cas de la Figure 40, il s'agit des chaînes d'interface qui se trouvent dans le fichier de ressources security.properties.

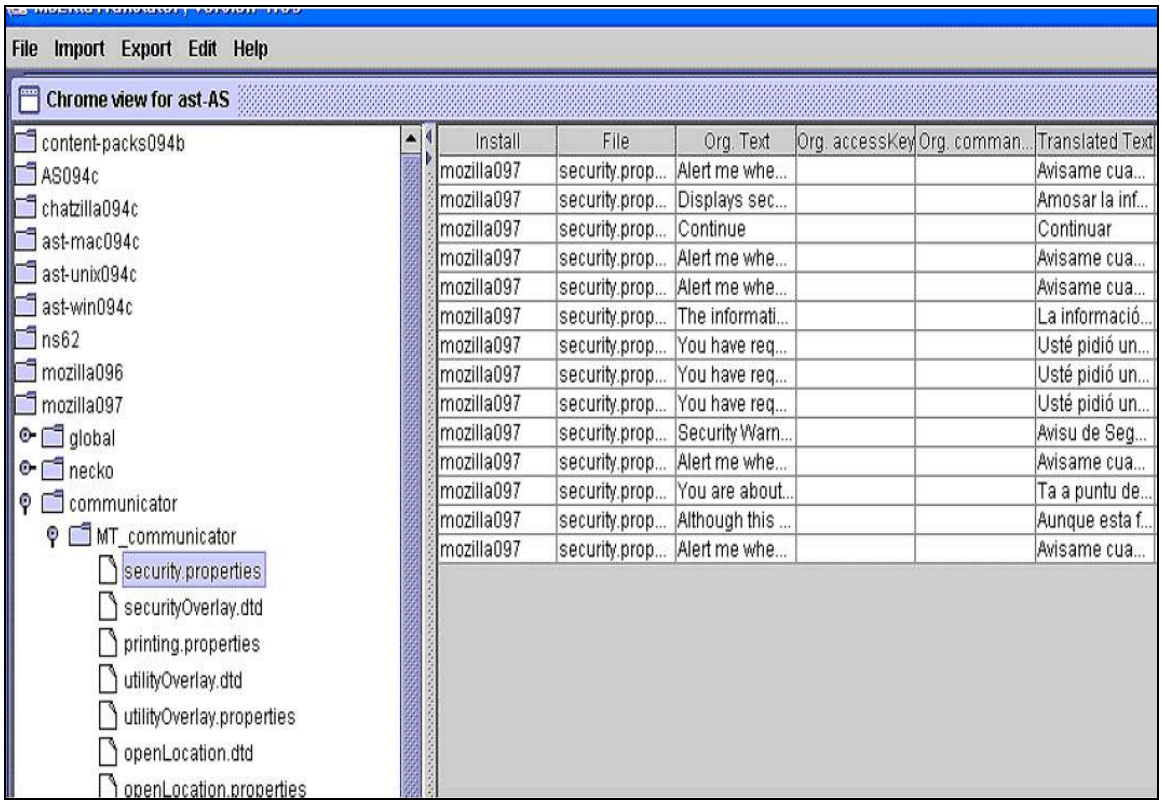


Figure 40 : *Fenêtre principale de l'outil MozillaTranslator qui permet la traduction des éléments d'interface qui se trouvent dans le dossier chrome de l'application*

Une fois le dossier chrome importé, le contributeur peut commencer à traduire les chaînes des interfaces (Figure 41).

Le contributeur peut à tout moment envoyer ses propositions de traduction en cliquant sur le menu Export (Figure 42). Comme le montre la Figure 42, pour exporter ses contributions, l'utilisateur doit spécifier la langue cible vers laquelle il a traduit, son identifiant ou son nom, la version de l'application, l'url vers laquelle il souhaite exporter ses contributions (par exemple dans le cas de la Figure 42 c'est l'url www.Proyectonave.es/productos/seamonkeys qui correspond au site du projet de localisation des applications Mozilla en espagnol), et finalement le chemin d'accès du dossier qui contient le fichier .jar.

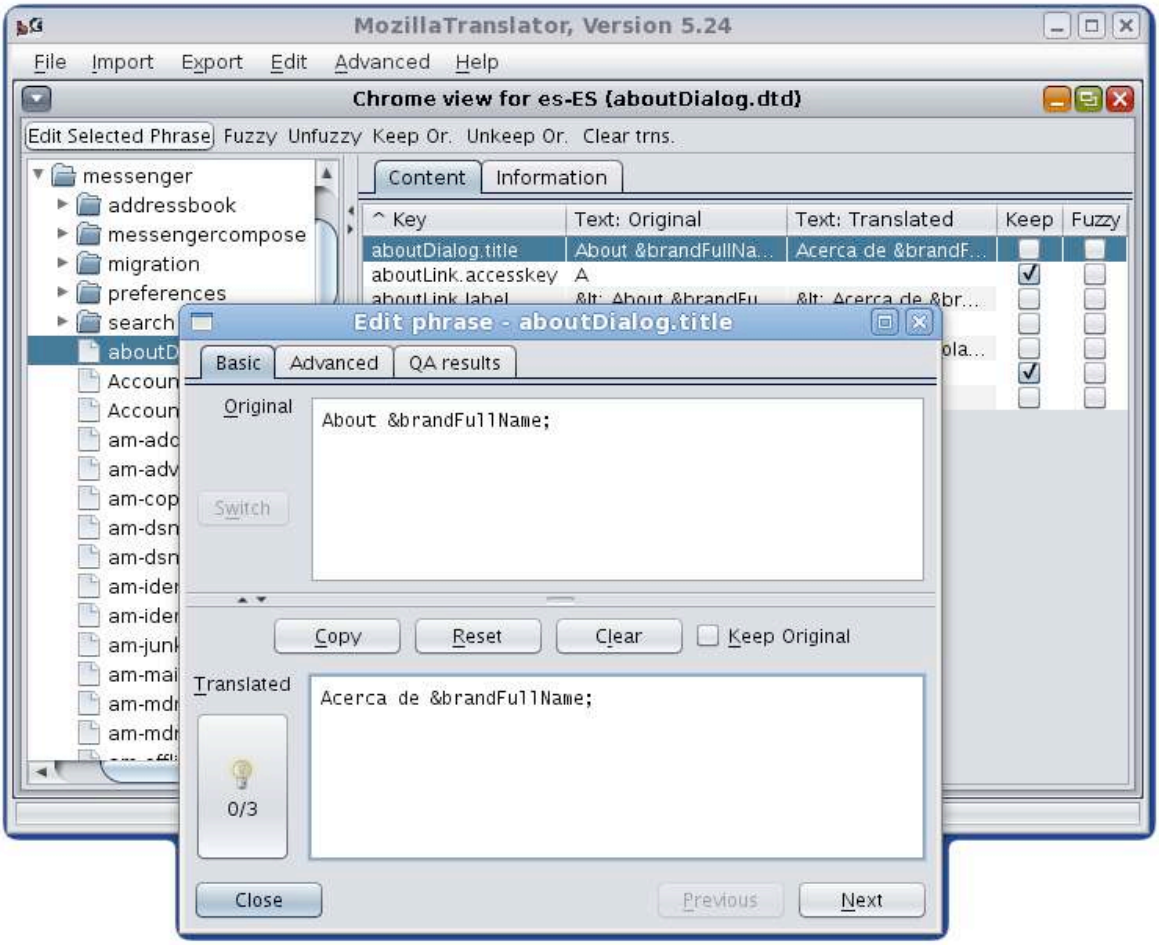


Figure 41 : *Traduction des chaînes d'interface avec l'outil MozillaTranslator*

Figure 42 : *Export du fichier .jar qui contient les contributions de l'utilisateur avec MozillaTranslator*

L'utilisateur peut aussi mettre à jour son application *Mozilla* en utilisant l'outil *MozillaTranslator* (Figure 43). Pour cela, il doit se connecter sur le site de *Mozilla*, choisir l'application ainsi que la langue correspondante, et télécharger le dernier fichier de ressources.

Figure 43 : *Mise à jour d'une application Mozilla*

La mise à jour se fait en remplaçant le fichier de ressources original par celui qui est choisi par le contributeur.

3.1.1.2 Projet de localisation du système d'exploitation Ubuntu

Le projet de localisation d'Ubuntu comporte soixante-sept communautés locales officielles, appelées équipes "LoCo" (pour "Local Community") et cent neuf autres communautés sur le point d'obtenir ce statut « officiel » (Canonical Ltd, 2008). Certaines de ces communautés sont liées à un pays (l'Indonésie ou l'Allemagne par exemple), d'autres sont soudées par la langue (le catalan ou le kurde par exemple), d'autres encore par un lieu géographique (Austin, Texas aux Etats-Unis, ou Bangalore en Inde). Les communautés locales impliquées dans la traduction des logiciels décident ce qui doit être traduit, comment et quand.

L'équipe de localisation d'Ubuntu en français est un exemple concret de communauté au service de la localisation des logiciels libres qui s'articule autour de la contribution bénévole. Plus précisément, on constate que cette équipe de traduction fonctionne comme une communauté de pratique, au sein de laquelle les usagers d'une langue la font vivre et évoluer au contact des nouvelles technologies qu'ils utilisent. *Une communauté de pratique* peut se définir, pour reprendre les termes de Lave et Wenger (Lave et Wenger, 1991), comme « *un ensemble de relations parmi des personnes, une activité et le monde, relations nouées au fil du temps et en interaction avec d'autres communautés de pratique dont les activités sont proches voire se recoupent en partie.* »

Le fonctionnement de cette équipe de traduction en français s'articule autour d'un site Web, qui est un sous-site général *ubuntu-fr.org* de l'équipe *LoCo* française reconnue officiellement par Ubuntu. Ce site sert aussi de lieu d'échange et d'information pour la communauté francophone des utilisateurs d'Ubuntu. La gestion des traductions se fait entièrement en ligne via une interface Web nommée *Launchpad Translations* (Canonical Ltd, 2009). Pour pouvoir participer à la localisation d'Ubuntu, les contributeurs bénévoles doivent s'enregistrer sur le site afin de savoir quels sont les projets qui sont en train d'être traduits, et dans le cadre de ces projets, quelles chaînes spécifiques sont à traduire.

Dés qu'il est enregistré, le contributeur a la possibilité de proposer une traduction française (Figure 44). De fait, très souvent, plusieurs traductions du même texte sont proposées par différents traducteurs. Mais seuls les traducteurs dits « expérimentés », c'est-à-dire les membres officiels du noyau dur des traducteurs français, ont le droit de « valider » les traductions qui seront ensuite incluses dans Ubuntu. Plus l'implication de l'individu est importante, plus sa participation à la communauté devient pleine et entière.

C'est ce que confirme la procédure officielle d'obtention du statut de membre officiel de cette communauté. La démarche décrite dans la foire aux questions des traducteurs d' Ubuntu en français (Patri, 2008) indique que le nouveau venu doit, dans un premier temps, se présenter au groupe et préciser ses intentions de traduction. Cela se fait, en règle générale, par

l'intermédiaire de la liste de diffusion. Après, on peut devenir membre officiel en faisant un certain nombre de suggestions.

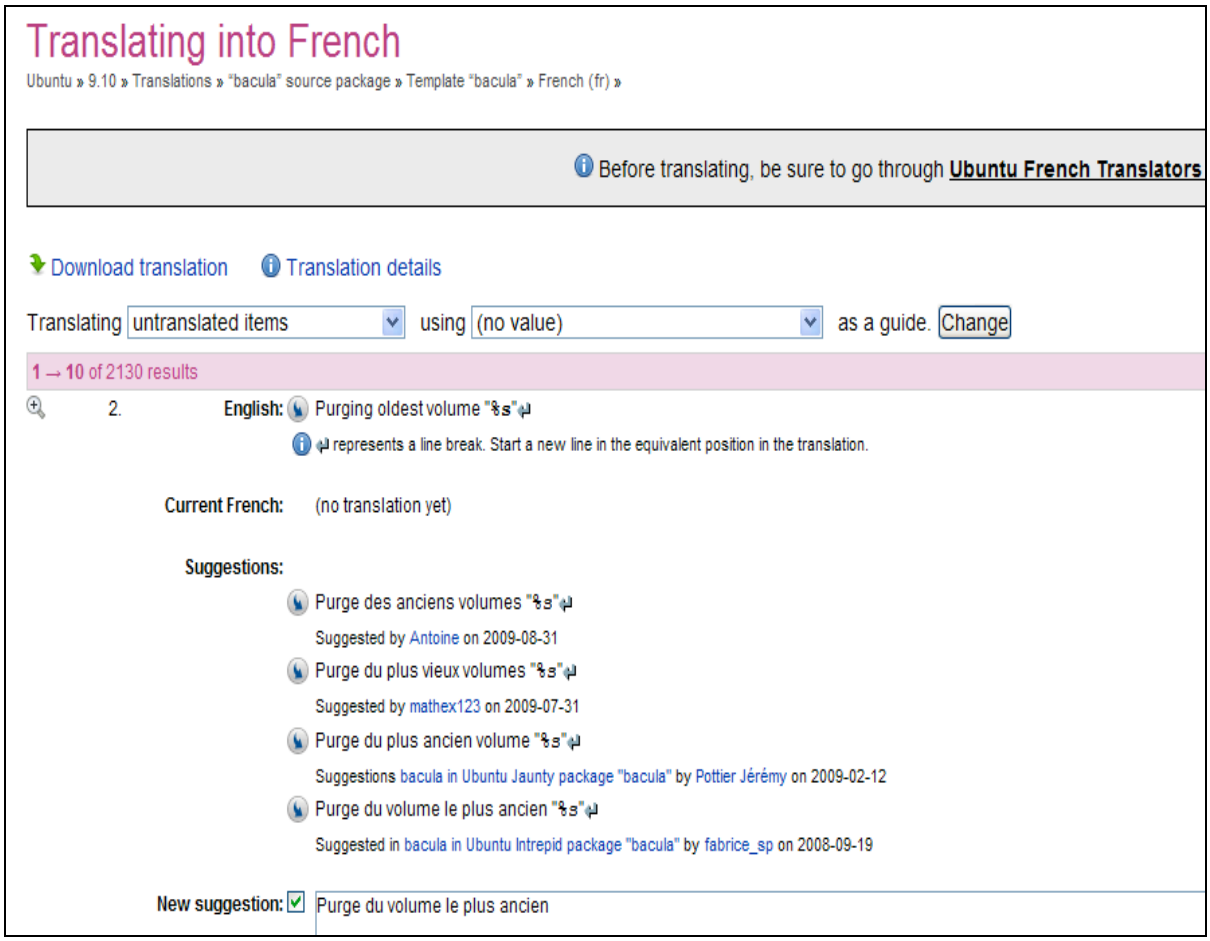


Figure 44 : *Site Launchpad Translators du projet de localisation Ubuntu Anglais-Français*

3.1.2 Faire participer l'utilisateur final

Certains projets de localisation font participer directement l'utilisateur final de l'application au processus de localisation.

3.1.2.1 Projet de localisation des applications Ubuntu

La localisation des applications Ubuntu peut être faite directement par les utilisateurs finals. En effet, depuis Ubuntu, une nouvelle option dans le menu Aide est désormais disponible dans de nombreux programmes : « Traduire cette application » (Figure 45). Ainsi, en cliquant sur cette entrée de menu, l'utilisateur est dirigé automatiquement sur la page Web contenant les traductions de cette application (Figure 46).

Si l'utilisateur ne fait pas partie de la communauté des traducteurs Ubuntu, ses traductions ne seront pas directement incluses, mais figureront comme « Suggestions » pour approbation par la communauté de traduction.

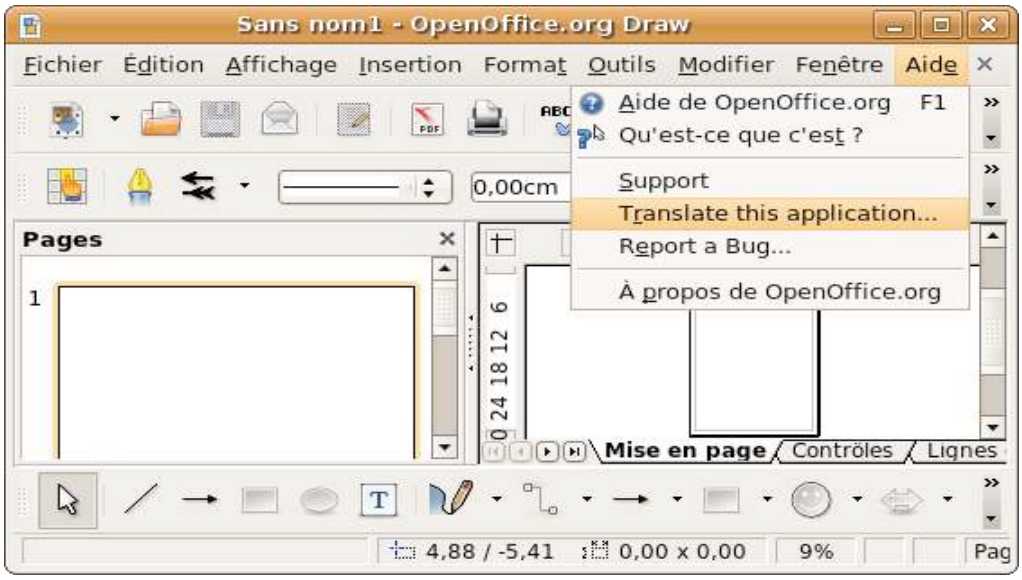


Figure 45 : *Interface principale de l'application OpenOffice sous Ubuntu dont le menu Aide contient l'option "Translate this application"*

Figure 46 : *Page Internet contenant la liste des fichiers à traduire pour OpenOffice sous Ubuntu*

3.1.2.2 iMAG (intercative Multilingual Access Gateway), passerelles d'accès multilingue dédiées

Une iMAG[11] (interactive Multilingual Access Gateway, ou en français Passerelle Interactive d'Accès Multilingue) est une « passerelle de traduction » à première vue très semblable à Systranet[12] ou à Google Translate[13]. La différence essentielle est qu'elle contient un éditeur de traductions, ainsi que des ressources « génériques » (dictionnaires bilingues d'usage en ligne, traductions trouvées sur le Web, corpus parallèles comme EuroParl[14], appel à des traducteurs automatiques gratuits, support au travail collaboratif de post-édition ou de traduction quand il n'y a pas encore de ressources automatisées pour un couple de langue donné).

Le concept d'iMAG a été proposé par Christian Boitet (Boitet et al., 2007). Le scénario type est le suivant : si l'on veut accéder à une page Web d'un site élu dans une langue étrangère pour laquelle son iMAG est « équipée », on copie l'URL de cette page dans un formulaire de l'iMAG, on y choisit aussi la langue, et l'on navigue dans le site élu dans la langue choisie (Figure 47). À ce niveau, tout se passe apparemment comme quand on utilise les services linguistiques de Google ou Systranet.

Il y a cependant deux différences majeures. D'abord, une page en langue *cible* n'est pas construite uniquement par traduction automatique. On la découpe en *segments* (phrases ou paragraphes), et on les traduit par les meilleurs résultats possibles trouvés dans la mémoire, et sinon par traduction automatique. La qualité s'améliorera donc avec le temps.

Ensuite, et c'est l'essentiel, on peut, lors de la lecture d'une page Web du site élu *accédée* en langue cible, sélectionner tout ou partie de la page, et passer en contexte d'édition des traductions et des contributions au dictionnaire (Figure 48). C'est l'aspect wiki de ce concept. Les lecteurs deviennent alors des contributeurs potentiels. Notons que Google Translate propose depuis 2008 aux internautes d'améliorer la qualité des traductions qu'il propose, mais phrase par phrase, et sans aucune spécialisation possible à tel ou tel site.

Deux *maquettes* d'iMAG ont été implémentées, par Mohammad Daoud dans son M2R (Daoud, 2007), et par Carlos Ramisch dans son stage ENSIMAG (Ramisch, 2008). Dans la seconde, Valérie Bellynck a intégré un relai de traduction (Redirection Gateway) qui sert d'interface entre l'iMAG et son site élu (en particulier pour la gestion des utilisateurs et des droits).

[11] Ce texte est extrait de la proposition du projet iMAG/LIG 2008 rédigé par C. Boitet
[12] http://www.systranet.com
[13] http://www.google.fr/langage_tools?hl=fr
[14] http://www.statmt.org/europarl/

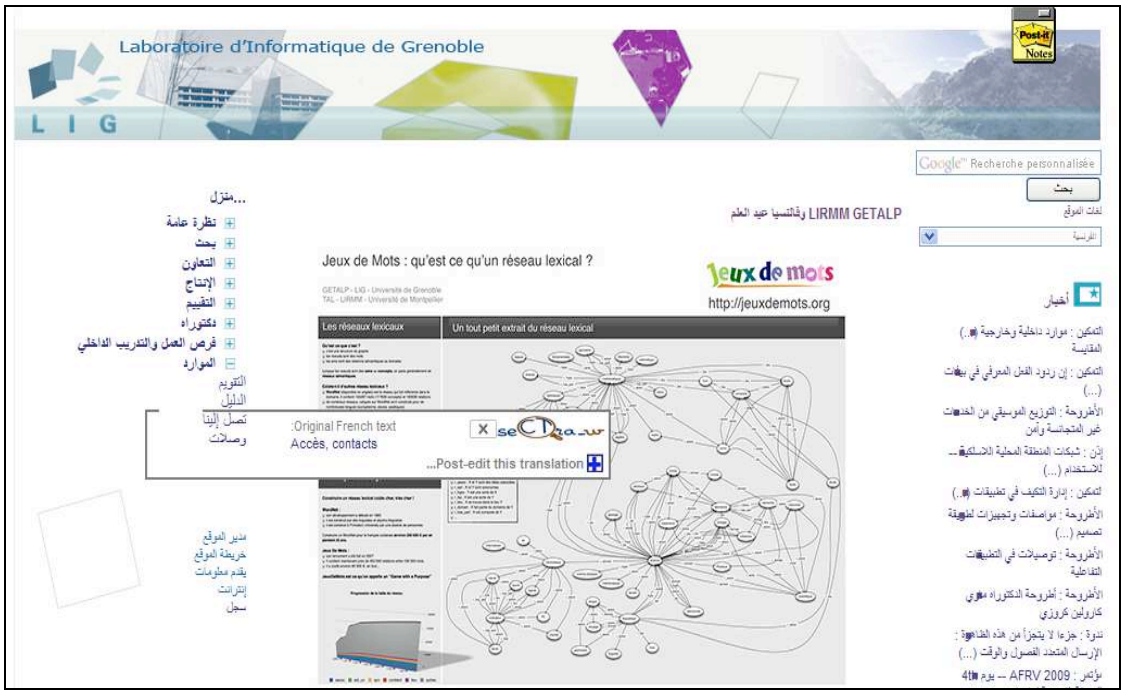


Figure 47 : *Instance d'iMAG sur le site du LIG (Laboratoire Informatique de Grenoble) en arabe et permettant à l'utilisateur d'améliorer les traductions disponibles.*

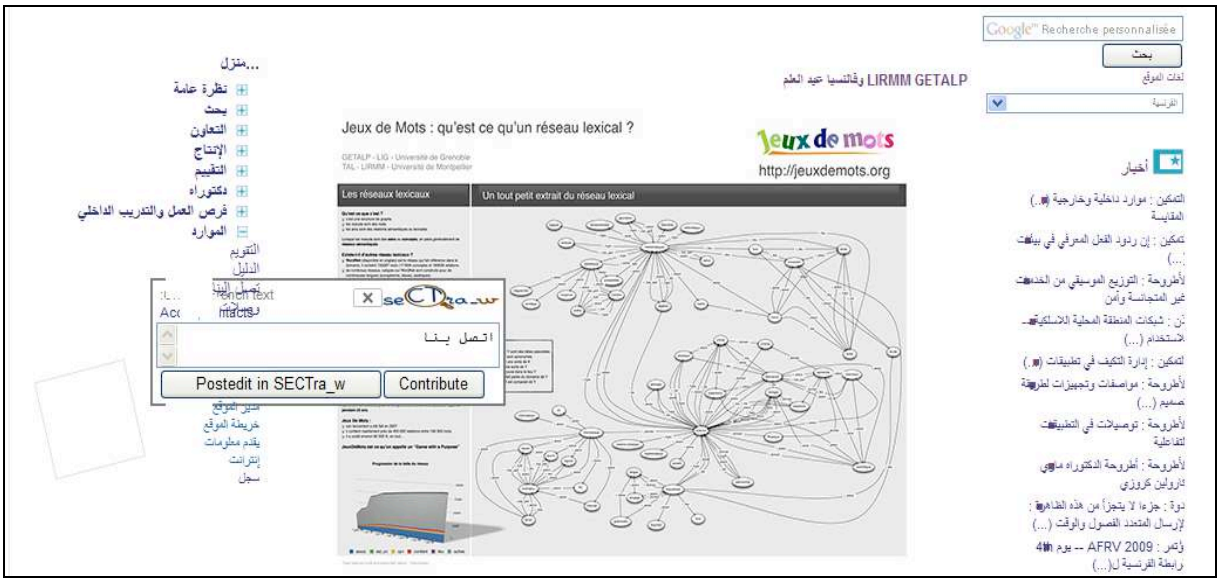


Figure 48 : *Amélioration des traductions existantes de la page Web par l'internaute*

3.1.3 Limites des projets existants

Tous les projets existants, concernant la localisation des applications, que nous avons cité ci-dessus présentent certaines limites.

3.1.3.1 Traduction hors contexte

Dans le cadre des projets de localisation collaborative mentionnés ci-dessus, exceptant iMAG, les traductions encore sont faites hors contexte. En effet, dans le cas de Mozilla

comme dans celui d'Ubuntu, les contributeurs traduisent des listes de segments et ils n'ont aucun accès au contexte dans lequel les éléments textuels sont utilisés. Or, comme nous l'avons expliqué dans le chapitre précédent, 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.

3.1.3.2 Long délais de mise à jour

Dans le cas de Mozilla comme dans celui d'Ubuntu, les utilisateurs ne peuvent inclure leurs nouvelles propositions en temps réel dans leurs applications. En effet, toutes les nouvelles propositions de traduction sont contrôlées puis validées uniquement par les personnes qui font partie de la communauté des administrateurs. Bien entendu, la communauté des administrateurs et les processus de traduction ne sont pas exempts de conflits et de tensions.

Certains se résorbent et d'autres non, notamment quand il y a des contributeurs qui ne sont pas d'accord avec les traductions qui ont été validées par la communauté des administrateurs. En effet, dans les cas ou les contributions sont rejetées par cette communauté, les contributeurs ne verront jamais leurs propositions de traduction acceptées.

3.2 De la traduction discontinue, coordonnée et hors contexte à la traduction continue, non coordonnée et en contexte

Le modèle actuel de traduction, utilisé dans la localisation des logiciels commerciaux ou libres, est fortement discontinu, coordonné et hors contexte. Le processus est discontinu car les traductions sont disséminées uniquement lors des mises à jour périodiques des produits. Il s'ensuit qu'une fois publié, le logiciel devient non modifiable (intouchable) jusqu'au prochain assemblage "build" de la version localisée. Cela implique qu'une chaîne qui a été détectée comme mal traduite, après la publication du logiciel, ne sera corrigée qu'au moment de l'assemblage d'une prochaine version localisée du logiciel.

Ce processus est *coordonné* car le modèle de traduction actuel nécessite une certaine coordination et organisation entre les deux entités principales à savoir, le donneur d'ordres qui est le *coordinateur* (l'éditeur du logiciel dans le cas des logiciels commerciaux et la communauté des administrateurs dans le cas des logiciels libres) et les traducteurs (professionnels ou bénévoles).

Enfin, comme mentionné avant, la traduction des éléments textuels des interfaces utilisateur est faite hors contexte. Ainsi, on aboutit parfois à une mauvaise qualité de traduction car le contexte dans lequel apparaît une chaîne de l'interface contient de nombreuses informations qui sont utiles pour choisir la bonne traduction.

3.2.1 De la traduction discontinue à la traduction continue

La solution que nous proposons consiste à passer du modèle de localisation discontinue à un modèle de localisation continue. L'idée consiste à renoncer à l'idée de traduction parfaite, et de publier une traduction « brute » de qualité variable dont la qualité augmentera avec le temps, c'est-à-dire que la traduction se poursuivra et s'améliorera de manière continue. Il s'agit donc de permettre à toute personne, souhaitant participer à la localisation de l'application, de traduire à n'importe quel moment certains éléments textuels

de l'application. Cela permet donc de résoudre le problème de délais dans la mesure où les contributeurs peuvent télécharger à n'importe quel moment une version partiellement localisée voire non localisée du logiciel. De la même façon, l'éditeur de logiciels pourra publier une première version partiellement localisée qui va être totalement localisée au fur et à mesure pendant la vie du logiciel.

3.2.2 De la traduction coordonné vers la traduction non coordonnée

Dans le domaine de localisation des logiciels commerciaux ou libres, la traduction est fortement coordonnée. Cela veut dire que, quelque soit le modèle de localisation utilisé, la localisation est faite sous forme d'un projet de localisation qui est contrôlé par le coordinateur. Le scénario est le suivant :

1. le coordinateur (éditeur de logiciel dans le cas des logiciels commerciaux ou la communauté d'administrateur dans le cas des logiciels libres) annonce aux différents types de traducteurs (traducteurs professionnels ou contributeurs bénévoles) le début d'un projet de localisation.
2. les traducteurs traduisent et envoient leurs traductions au coordinateur.
3. le coordinateur contrôle et intègre les nouvelles traductions.

Cependant, l'apparition du concept de Wiki au milieu des années 90, qui s'est surtout fait connaître avec l'encyclopédie Wikipedia (fondée en 2001), a révolutionné les pratiques de la traduction. On parle alors de *traduction non coordonnée*. Il s'agit de publier des ébauches de traduction qui seront améliorées au fur et à mesure par les internautes. Cela implique, que la traduction continue pendant la vie du document. En effet, le concept Wiki permet de publier des documents partiellement traduits, voire non traduits, et qui vont être traduits à n'importe quel moment par n'importe quel internaute. Le succès du site Wikipedia en est un exemple, il contient des millions de pages très riches et très diverses, de bonne qualité de traduction.

Pour l'instant ce modèle de traduction *non coordonnée* est appliqué uniquement pour traduire des documents sur Internet. Notre solution consiste donc à appliquer ce modèle à la localisation des logiciels. Il s'agit de permettre à n'importe quelle personne, souhaitant participer à la localisation du logiciel, de traduire de façon spontanée certains éléments textuels du logiciel.

3.2.3 De la traduction hors contexte à la traduction en contexte

Comme mentionné dans le chapitre 2, le problème de la qualité, dû au fait que les traductions sont faites *hors contexte*, est un problème majeur du processus de localisation actuel. Pour résoudre ce problème récurrent, nous proposons une nouvelle méthode, la localisation *en contexte*.

Il s'agit de rendre accessible la localisation directement à partir des interfaces utilisateur des logiciels. Cela permettra aux contributeurs d'accéder au contexte dans lequel les éléments textuels sont utilisés et donc de produire des traductions de meilleure qualité de traduction. D'autre part, afin de réduire le coût et le délai de la localisation, nous souhaitons faire participer plus que les traducteurs professionnels. Ainsi, les contributeurs peuvent être des contributeurs bénévoles, des β-testeurs, etc. Cependant, dans le cas de produits commerciaux, il n'y a pas actuellement de contributeurs bénévoles. Ainsi, une autre catégorie de contributeurs pourrait dans ce cas être constituée d'utilisateurs finals connaissant l'anglais, (la langue source du logiciel mais pas souvent) dont la langue maternelle est la langue visée, et

qui, pendant l'utilisation des produits, traduiraient certains éléments textuels des interfaces utilisateur, ou amélioreraient la traduction proposée par des systèmes de traductions automatiques (TA) ou des mémoires de traductions (MT). L'éditeur de logiciel pourrait alors les *gratifier* d'une façon ou d'une autre, par exemple en leur donnant des points permettant d'avoir des licences gratuites, etc.

Conclusion

Nous avons présenté dans cette première partie de mémoire la situation actuelle de la localisation des logiciels commerciaux et libres en décrivant le processus métier de localisation actuel et en identifiant les problèmes liés à ce processus anciens et nouveaux. Nous avons présenté par la suite les solutions que nous préconisons pour améliorer le processus de localisation actuel. Ces solutions consistent à mettre en place une nouvelle méthode de localisation *en contexte*.

Partie 2 : Localisation interne du code source des applications existantes

Les logiciels existants ne sont pas prévus pour être localiser *en contexte*, nous présentons donc dans cette partie notre solution de localisation *interne* pour permettre la localisation *en contexte* des logiciels existants.

Dans le chapitre 4 nous présentons un état de l'art sur les choix usuels des développeurs des applications commerciales et libres en ce qui concerne la fabrication et l'affichage des interfaces graphiques et plus particulièrement la fabrication et l'affichage des éléments textuels des interfaces utilisateur. Nous présentons ensuite notre solution de localisation *interne* du code source pour permettre la localisation *en contexte* des logiciels commerciaux et libres existants.

Dans le chapitre suivant, nous présentons une première expérimentation de notre approche de localisation *en contexte* sur un logiciel libre Notepad-plus-plus. Nos résultats prouvent la faisabilité technique de la solution que nous avons proposée dans le chapitre précédent : localisation *interne* du code source.

Enfin, dans un dernier chapitre, nous présentons une deuxième expérimentation complémentaire sur un deuxième logiciel libre, Vuze, qui utilise une technologie différente de celle utilisée dans Notepad-plus-plus. Le but de cette deuxième expérimentation est d'évaluer la généricité de notre approche ainsi que le délai de réalisation par rapport à la première expérimentation.

Chapitre 4 : Solution pour la localisation en contexte de logiciels existants

Introduction

La localisation *en contexte* des éléments textuels des interfaces utilisateur de logiciels consiste à rendre localisable les éléments textuels directement à partir des interfaces graphiques. Cependant les logiciels existants ne sont pas prévus pour être localisés en contexte. Nous présentons dans ce chapitre une solution permettant la localisation en contexte de logiciels existants.

Pour cela, nous commençons par faire un état de l'art sur les choix usuels des développeurs concernant la fabrication ainsi que l'affichage des interfaces utilisateur, et plus particulièrement les éléments textuels des interfaces graphiques, afin de déterminer l'approche la plus générique possible nous permettant de proposer un tel contexte de localisation lors de l'utilisation de logiciels par les utilisateurs finals. Nous présenterons ensuite la solution que nous avons adoptée pour permettre la localisation en contexte de logiciels existants.

4.1 Etat de l'art sur les choix usuels des développeurs des logiciels commerciaux et libres en ce qui concerne la fabrication et l'affichage des éléments textuels des interfaces utilisateur

Nous étudions d'abord les choix usuels des développeurs de logiciels libres ou commerciaux en ce qui concerne la fabrication des éléments textuels des interfaces utilisateur. Ensuite, nous exposerons les mécanismes usuels d'affichage des éléments textuels des interfaces utilisateur.

4.1.1 Fabrication des éléments textuels des interfaces utilisateur

Lors de la création d'un logiciel, le développeur doit choisir la façon avec laquelle il souhaite fabriquer les éléments textuels des interfaces utilisateur de son logiciel. Ce choix dépend de la prise en considération ou non du facteur internationalisation du logiciel. Pour cela nous allons étudier dans un premier temps la fabrication des éléments textuels des interfaces dans le cas des logiciels non internationalisés, puis nous détaillerons les choix des développeurs dans le cas des logiciels internationalisés.

4.1.1.1 Fabrication des éléments textuels des interfaces dans le cas des logiciels non internationalisés

Certains développeurs oublient ou ne pensent pas forcément au moment de la fabrication des interfaces utilisateur, et plus particulièrement des éléments textuels des interfaces, à l'internationalisation de leurs logiciels. C'est bien dommage parce que coder en

intégrant ce facteur simplifie le travail du localiseur pour adapter le logiciel à une langue cible et à sa spécificité locale, lui évitant ainsi des requêtes fastidieuses et des contacts aléatoires avec le développeur de l'application. Dans cette catégorie de logiciels, les éléments textuels sont généralement fabriqués dans le code et même parfois fabriqués dans des images bitmap.

4.1.1.1.1 Eléments textuels fabriqués dans le code source

Dans le cas des logiciels non internationalisés, la plupart des éléments textuels que nous voyons apparaître dans les interfaces utilisateur (message d'erreur, instruction, menu,…) sont directement décrits dans le code source du logiciel (Figure 49).

```
import java.awt.*;
import javax.swing.*;
public class  UserInterface extends JFrame {
   private JPanel  contentPane;
   private JLabel  paysLabel = new JLabel();
   private JLabel  dateLabel = new JLabel();
   private JLabel  tempsLabel = new JLabel();
   private JLabel  montantLabel = new JLabel();
   private JTextField paysText = new JTextField();
   private JTextField dateText = new JTextField();
   private JTextField tempsText = new JTextField();
   private JTextField montantText = new JTextField();
   public UserInterface()  {
      contentPane =  (JPanel) this.getContentPane();
      contentPane.setLayout(new GridLayout(4, 2));
      contentPane.add(paysLabel);
      contentPane.add(paysText);
      contentPane.add(dateLabel);
      contentPane.add(dateText);
      contentPane.add(tempsLabel);
      contentPane.add(tempsText);
      contentPane.add(montantLabel);
      contentPane.add(montantText);
      paysLabel.setText("Pays");
      dateLabel.setText("Date");
      tempsLabel.setText("Temp");
      montantLabel.setText("Montant");
      setTitle("Exemple de fenêtre non internationalisée");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setResizable(true);
      //set Size(293,123);
      setVisible(true);
   }
   public static  void main(String[] args) {
      new  UserInterface();
   }
}
```

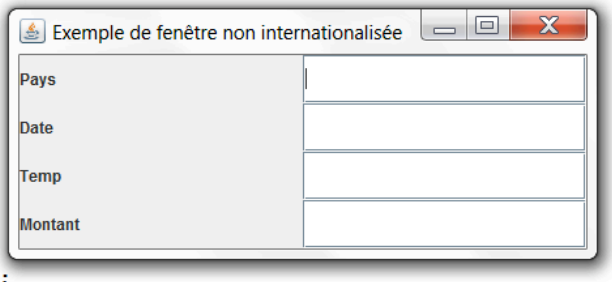


Figure 49 : *Exemple d'une application dont les éléments textuels sont fabriqués dans le code source*

Comme nous le montre l'exemple ci-dessus, tous les éléments textuels de l'interface utilisateur sont codés en dur. Ainsi, traduire cette interface revient à traduire les chaînes dans le code source. Cela n'est pas du tout évident quand le logiciel est constitué de millions de lignes de code, écrites dans différents langages, et quand la compilation est longue et fastidieuse.

4.1.1.1.2 Elément textuels fabriqués au format bitmap

Certains éléments textuels des interfaces utilisateur peuvent également être affichés sous format bitmap (Figure 50).

Figure 50 *: Exemple de ressources au format bitmap*

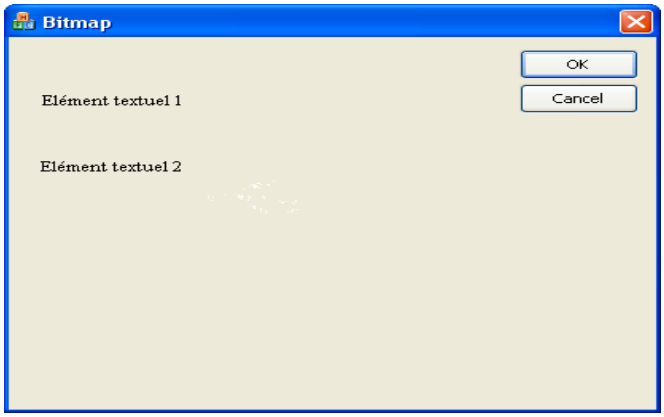


Figure 51 *: Interface utilisateur fabriquée à partir de la ressource bitmap de la Figure 50*

Pour traduire un tel texte, il convient d'éditer la ressource bitmap et de la modifier. Le travail est long et fastidieux, et la qualité graphique de la ressource bitmap localisée n'est pas toujours garantie.

Dans ce premier paragraphe, nous avons donné les deux choix les plus fréquents pour la fabrication des éléments textuels des interfaces graphiques dans le cas des logiciels non internationalisés. Ces deux exemples nous montrent la difficulté de traduire ces éléments textuels.

En effet, traduire une interface utilisateur, dans ces deux cas, nécessite une intervention au niveau du code source. Cela suppose que tout le travail de localisation soit fait avant la compilation du code source.

4.1.1.2 Fabrication des éléments textuels des interfaces dans le cas des logiciels internationalisés

Fabriquer des interfaces en prenant en considération l'internationalisation du logiciel impose la séparation entre les éléments textuels statiques des interfaces utilisateur et le code source de l'application. Ainsi, chaque langage de programmation offre aux développeurs des outils permettant de séparer le code source des éléments textuels. Le principe est le même pour tous les langages de programmation : il s'agit de stocker les chaînes de caractères dans

des fichiers externes appelés Ressources, ces chaînes étant ensuite affichées lors de l'exécution du programme.

Les éditeurs le souhaitant peuvent également implémenter leur propre mécanisme de gestion de ressources. Les choix usuels de ces derniers reposent le plus souvent sur les mêmes mécanismes que ceux proposés nativement par les langages de programmation et les environnements de développement (IDE).

Dans le monde des logiciels commerciaux, le langage de programmation le plus utilisé est Visual C++ sous environnement Windows, alors que dans le cas des logiciels libres les développeurs utilisent plutôt le langage de programmation Java. Nous présentons maintenant la fabrication des éléments textuels des interfaces utilisateur en Visual C++ et en Java sous environnement Windows.

4.1.1.2.1 Fabrication des éléments textuels des interfaces utilisateur en Visual C++

Les développeurs Visual C++ utilisent d'habitude les fichiers de ressources RC pour séparer les éléments textuels du code source. Un fichier RC est un fichier de script permettant de décrire différents types d'éléments des interfaces utilisateur, tels que les bitmaps, les curseurs, les boîtes de dialogues, les polices, les icônes, les menus contextuels, etc. Pour illustrer cela, nous allons créer un fichier de ressources MyResource.RC (Figure 52) qui décrit une boîte de dialogue nommée DIALOG_1.

```
#include "ressource.h"

DIALOG_1 DIALOG 14, 30, 212, 227
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "DIALOG_1"
FONT 8, "MS Sans Serif"
{
 CONTROL "Ok", IDOK, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTO
✂
P, 56, 193, 36, 14
 CONTROL "Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | W
✂
S_TABSTOP, 115, 193, 38, 14
 CONTROL "", IDC_FILENAME, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER |
✂
 WS_TABSTOP, 26, 18, 91, 12
 CONTROL "", IDC_TEXT, "EDIT", ES_LEFT | ES_MULTILINE | WS_CHILD | WS_VISIBLE |
✂
WS_BORDER | WS_TABSTOP, 18, 49, 175, 119
 CONTROL "Lire", IDC_LIRE, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_
✂
TABSTOP, 136, 18, 50, 14
}
```

Figure 52 : *Le fichier de ressources MyResource.RC décrivant la boîte de dialogue «DIALOG_1»*

La figure 4 montre le fichier de ressources MyResource.RC, qui décrit la boîte de dialogue DIALOG_1. Cette boîte contient plusieurs contrôles (un bouton «OK», un bouton «Cancel», deux zones de texte et un bouton «Lire»). Nous remarquons, que le fichier de ressources RC contient non seulement les éléments textuels, mais aussi la taille des composants graphiques, leurs types, leurs identifiants, leurs positions, etc.

Voyons maintenant comment ces ressources sont utilisées par le programme C++. Avec Visual C++, la façon la plus courante d'afficher une boîte de dialogue et ses contrôles est de faire appel aux primitives de base *Win32*. Par exemple, pour créer une boîte de dialogue, les développeurs utilisent la primitive CreateDialog() qui prend en paramètre une structure de données représentant le dialogue à créer. Cette structure peut être renseignée par le développeur directement dans le code. Il a donc à sa charge la description de toutes les caractéristiques de sa boîte de dialogue (taille, style, titre...) ainsi que de tous les composants graphiques qu'elle contient. Cette structure peut aussi être lue directement depuis les ressources RC, ce qui est le cas de la ressource RC décrite par la Figure 52.

Pour mieux comprendre, nous commençons par créer la boîte de dialogue de la Figure 52. Pour cela, nous utilisons la fonction CreateDialog() (Figure 53).

```
Hdlg = CreateDialog ( hInstance, (LPCTSTR) IDD_DIALOG1, NULL, (DLGPROC) MainProc );
```

Figure 53 : *La primitive Win 32 CreateDialog () qui permet la création d'une boîte de dialogue*

Le paramètre hInstance est un identifiant du module contenant la ressource RC qui décrit la boîte de dialogue. Le deuxième paramètre de la fonction IDD_DIALOG1 est l'identifiant de la ressource que nous voulons créer, ici DIALOG_1. Le troisième paramètre est l'identifiant de la fenêtre parent. Dans le cas de l'exemple, la boîte de dialogue n'a pas de fenêtre parent (NULL). Le dernier paramètre est la fonction MainProc qui est un pointeur sur la fonction de procédure du dialogue (qui est déclenchée par des événements comme la réception d'un message, etc.).

4.1.1.2.2 Fabrication des éléments textuels des interfaces en Java

Les développeurs Java utilisent les fichiers properties pour séparer les éléments textuels du code source. Il s'agit des fichiers texte qui contiennent une liste d'association code_chaîne = valeur_chaîne avec une association par ligne (Figure 54). Le code_chaîne permet d'identifier chaque chaîne de caractères dans ce fichier. À chaque fichier properties sont associés une langue et un encodage de caractères. La langue et l'encodage par défaut sont configurés dans le fichier de configuration de l'application.

```
Key_titre = Exemple d'internationalisation;
key_pays = Pays;
Key_Date = Date;
Key_Temps = Temps;
Key_Montant = Montant;
```

Figure 54 : *Exemple d'un fichier properties*

Pour accéder aux fichiers properties, les développeurs utilisent la classe java.util.ResourceBundle (Figure 55).

```
Private ResourceBundle bundle =  ResourceBundle.getBundle("MyResource");
```

Figure 55 : *Création d'un objet RessourceBundle pour charger les éléments textuels à partir du fichier properties MyResource*

Le chargement des chaînes de caractères se fait par la méthode getString() de la classe ResourceBundle prenant en paramètre l'identifiant de la chaîne (Figure 56).

```
bundle.getString( Key_titre) ;
```

Figure 56 : *Récupération des chaînes de caractères en utilisant la méthode getString() de la classe ResourceBunde*

Reprenons maintenant l'exemple du code Java de la Figure 49 et modifions-le pour qu'il lise les éléments textuels à partir du fichier properties. Le code complet est donné par la Figure 57.

```
8    private JLabel  dateLabel = new JLabel();
9    private JLabel  tempsLabel = new JLabel();
10   private JLabel  montantLabel = new JLabel();
11   private JTextField paysText = new JTextField();
12   private JTextField dateText = new JTextField();
13   private JTextField tempsText = new JTextField();
14   private JTextField montantText = new JTextField();
15   // on ajoute un objet de type ResourceBundle qui  chargera les messages à
16   // partir du fichier DynExemple.properties
17   private ResourceBundle bundle = ResourceBundle.getBundle("MyResources");
18   public UserInterface()  {
19     contentPane =  (JPanel) this.getContentPane();
20     contentPane.setLayout(new GridLayout(4, 2));
21     contentPane.add(paysLabel);
22     contentPane.add(paysText);
23     contentPane.add(dateLabel);
24     contentPane.add(dateText);
25     contentPane.add(tempsLabel);
26     contentPane.add(tempsText);
27     contentPane.add(montantLabel);
28     contentPane.add(montantText);
29    // On remplace les textes statiques par la méthode
30   //"static" getString qui retourne le texte
31   // correspondant à une clé donnée.
32
33     paysLabel.setText(bundle.getString("Key_Pays"));
34     dateLabel.setText(bundle.getString("key_Date"));
35     tempsLabel.setText(bundle.getString("key_Temp"));
36     montantLabel.setText(bundle.getString("Key_Montant"));
37     setTitle(bundle.getString("Key_titre"));
38     setDefaultCloseOperation(EXIT_ON_CLOSE);
39     setResizable(true);
40     //set Size(293,123);
41     setVisible(true);
42   }
```

Figure 57 : *Code de la Figure 49 modifié pour charger les éléments textuels à partir du fichier properties*

4.1.2 Affichage des éléments textuels des interfaces utilisateur

Bien comprendre le principe de fonctionnement d'une application est essentiel avant de présenter les différentes méthodes conduisant à l'affichage des éléments graphiques et essentiellement des éléments textuels des interfaces utilisateur.

Tout d'abord, il convient de bien garder à l'esprit qu'un programme gérant une fenêtre doit rester en dialogue permanent avec le système d'exploitation. Le programme ne connaît rien sur son environnement. C'est le système d'exploitation qui signale à l'application si elle doit redimensionner le contenu d'une de ses fenêtres, ou encore si l'utilisateur essaie de fermer la fenêtre. Cette communication se fait au travers des messages que le système d'exploitation envoie à chaque application concernée. C'est au programme d'intercepter ces messages et de les transmettre aux fonctions gérant les différentes fenêtres.

Chaque fenêtre est associée à une fonction ou procédure de fenêtre (dans le cas de Windows c'est la procédure Window Proc). Parfois, plusieurs fenêtres peuvent être associées à une même procédure. Chaque message intercepté est transmis à la procédure correspondante qui se chargera de traiter ce message (redessiner la fenêtre, la redimensionner, afficher un caractère saisi par l'utilisateur, etc.).

4.1.2.1 Affichage des éléments textuels des interfaces utilisateur en Visual C++

En Visual C++, les éléments textuels des interfaces utilisateur sont stockés dans des fichiers de ressources RC. Dans un premier temps, nous verrons comment les ressources RC sont incorporées à l'application. Dans un deuxième temps, nous verrons comment les éléments textuels qui en sont issus sont affichés lors de l'exécution.

4.1.2.1.1 Incorporation des ressources RC dans l'exécutable

L'incorporation se réalise en deux étapes. La première consiste à compiler les ressources RC en données binaires .RES. Les fichiers de ressources sont ainsi transformés en un ensemble de structures de données binaires représentant les différents éléments d'interface des ressources RC (Figure 58).

```
#include "ressource.h"

DIALOG_1 DIALOG 14, 30, 212, 227
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "DIALOG_1"
FONT 8, "MS Sans Serif"
{
 CONTROL "Ok", IDOK, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTO
P, 56, 193, 36, 14
 CONTROL "Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | W
S_TABSTOP, 115, 193, 38, 14
 CONTROL "", IDC_FILENAME, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER |
 WS_TABSTOP, 26, 18, 91, 12
 CONTROL "", IDC_TEXT, "EDIT", ES_LEFT | ES_MULTILINE | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 18, 49, 175, 119
 CONTROL "Lire", IDC_LIRE, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_
TABSTOP, 136, 18, 50, 14
}
```

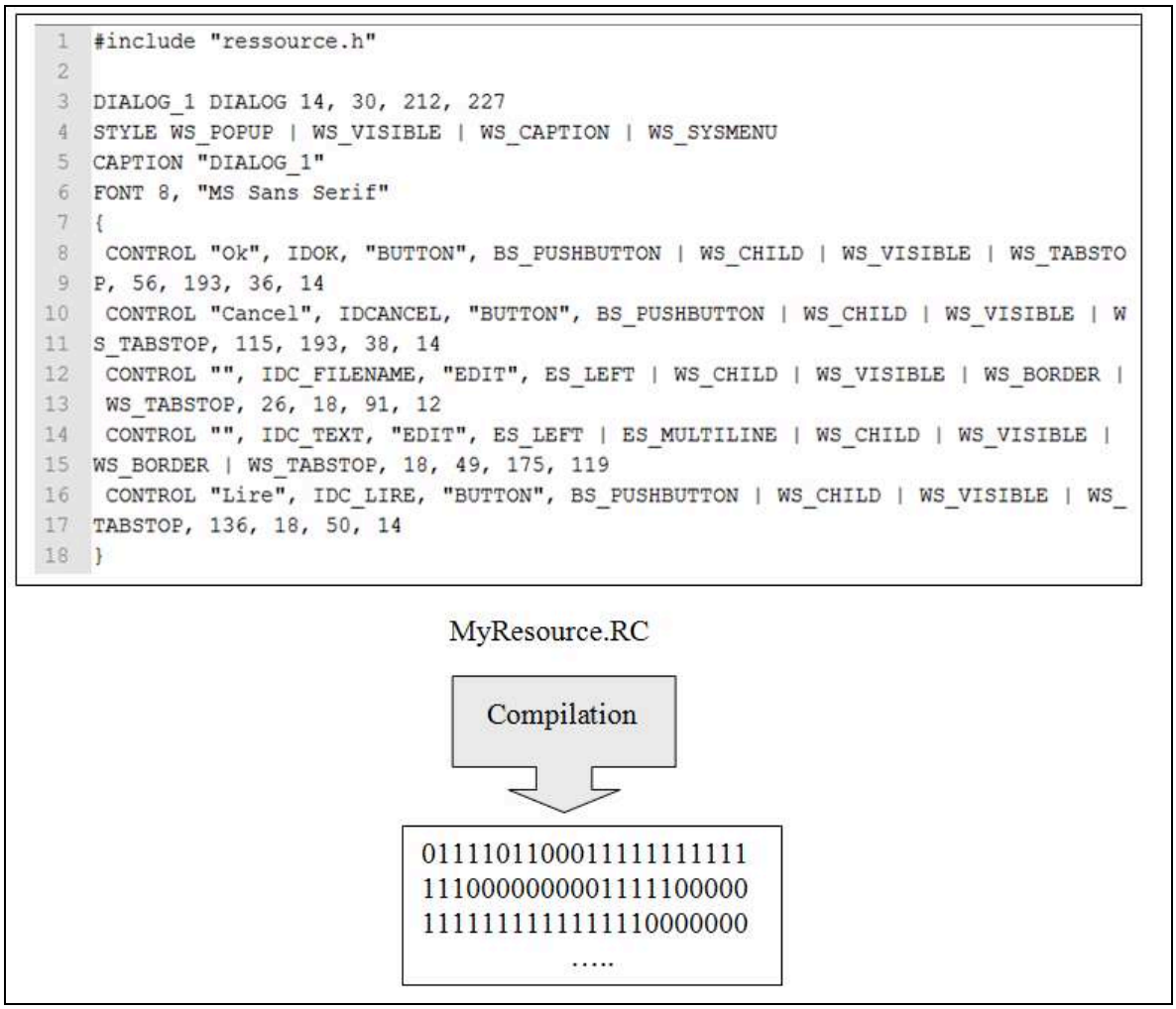


Figure 58 : *Compilation du fichier de ressources RC par un compilateur de ressources*

La deuxième étape est réalisée par l'éditeur de lien qui incorpore les fichiers binaires des ressources à l'exécutable de l'application. Il s'agit d'une copie intégrale du binaire résultant de la compilation des fichiers de ressources RC (.RES) dans le binaire de l'application (Figure 59). Lors de l'exécution du programme, l'éditeur de lien devra sur demande mettre en mémoire la structure binaire de la ressource.

Prenons l'exemple du fichier de ressources RC décrit par la Figure 52, qui, comme nous l'avons expliqué, décrit la boîte de dialogue DIALOG_1. Au moment de l'exécution du programme et plus particulièrement de la fonction CreateDialog (…, DIALOG_1,…), le programme appelle l'éditeur de lien en lui donnant l'identifiant de la ressource; ce dernier charge la structure binaire de la boîte de dialogue et la copie dans le code binaire de l'application. Lors du chargement de la boîte de dialogue, l'éditeur de lien n'a donc aucune information sur le contenu de la ressource et la seule information dont il dispose est l'identifiant de la ressource. En effet, les éléments textuels de la ressource sont encapsulés dans la structure binaire, et leur identification est très difficile, voire impossible.

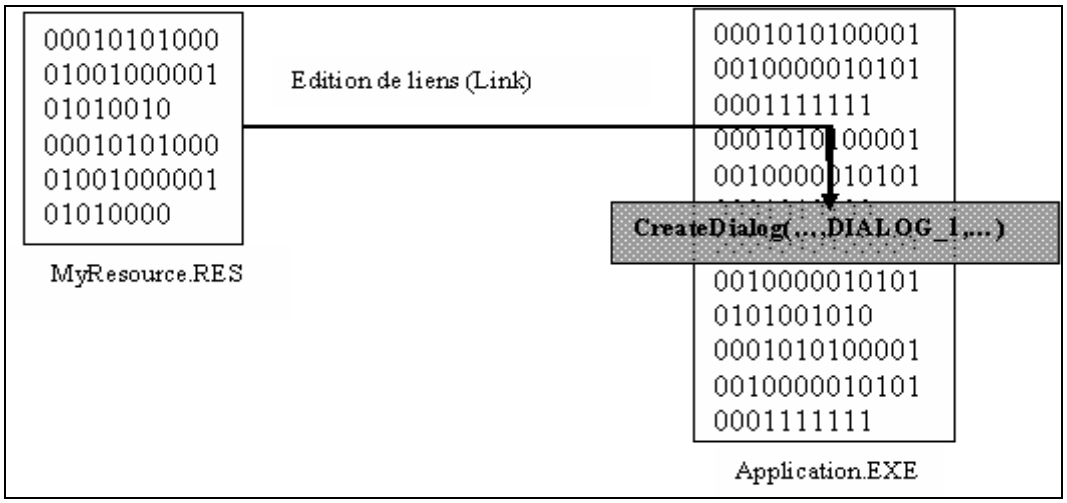


Figure 59 : *Édition des liens entre le fichier de ressource binaire MyResource.RES et l'exécutable de l'application*

4.1.2.1.2 Appels des primitives système

Une fois les liens édités, et la structure binaire de ressource copiée dans le code binaire de l'application, celle-ci fait appel aux primitives système pour construire tous les éléments graphiques. En effet, la structure de données contient, entre autres, comme attributs un nom de classe. Ce nom correspond à la classe C++ de base de Windows qui sait créer, afficher et gérer l'élément graphique qu'elle décrit. La primitive CreateDialog() lira l'ensemble des structures de données contenues dans la ressource et décrivant le dialogue que l'on cherche à afficher. Par exemple, la structure représentant le dialogue lui-même (oublions un instant son contenu), aura pour attribut la classe Window. Une instance de Window sera donc créée avec ses attributs initialisés à partir de ceux de la structure.

Le même traitement est réalisé pour les différents éléments d'interface de la fenêtre. Les instances des objets correspondants seront des fils de celle-ci. Ces différents objets ont en particulier dans leurs attributs les chaînes qu'ils doivent afficher. Ces chaînes sont donc issues des ressources.

En fonction de la position de la fenêtre et de son rang (*z-order*) par rapport aux autres fenêtres (de la même application ou des autres applications), le système d'exploitation (Windows) demande si nécessaire le dessin de la fenêtre. Dans ce cas, l'application *win32* reçoit un certain nombre de messages provoquant son affichage. Dès lors, les différents objets précédemment instanciés par le biais d'une procédure unique par type d'objet (Window Proc), solliciterons le code de dessin de chacun des éléments d'interfaces. Ce code de dessin fait appel à des primitives de bas niveau qui, provoquent in fine la création point par point de la fenêtre et de son contenu. Ainsi, le texte d'un bouton, stocké comme membre d'une instance de la classe Button, sera lu par le code de dessin de cette classe et affiché via une primitive de bas niveau qui dessine le texte à partir de sa valeur, de la police et du dispositif de rendu (l'écran dans notre cas).

4.1.2.2 Affichage des éléments textuels des interfaces en Java

Comme nous l'avons expliqué ci-dessus, les éléments textuels des interfaces graphiques Java sont stockés dans des fichiers properties. Nous verrons dans la suite de ce

paragraphe comment ces éléments sont incorporés puis affichés lors de l'exécution de l'application.

4.1.2.2.1 Incorporation des fichiers properties à l'exécutable

Contrairement aux fichiers de ressources RC, les fichiers properties ne nécessitent aucune compilation et ils sont utilisés dans leur format d'origine (format textuel) par la machine virtuelle Java (Figure 60). En effet, lors de l'exécution des fichiers .class, et plus particulièrement de la ligne du code suivante :

```
ResourceBundle bundle = ResourceBundle.getBundle(« MyResources ») ;
```

La machine virtuelle Java accède au fichier des ressources indiqué par la classe ResourceBundle et charge le contenu du fichier properties en mémoire. Ainsi, une fois l'application lancée, le chargement des éléments textuels des différentes interfaces utilisateur se fait à partir de la zone mémoire de la machine virtuelle et pas à partir du fichier properties.

Figure 60 : *Fichier properties de l'application UserInterface*

4.1.2.2.2 Affichage des éléments textuels lors de l'exécution

Comme dit ci-dessus, lors de l'exécution de l'application, les éléments textuels des interfaces utilisateur sont chargés en mémoire par la machine virtuelle. Dans cette partie, nous allons étudier la façon dont ils sont affichés. En effet, le processus d'affichage des éléments textuels dépend de la bibliothèque graphique qui a été utilisée pour fabriquer les interfaces utilisateur. Java offre principalement quatre bibliothèques graphiques, à savoir AWT, Swing, SWT et JFace. Nous allons étudier les deux bibliothèques graphiques les plus fréquemment utilisées par les développeurs Java, qui sont AWT et Swing.

4.1.2.2.2.1 Affichage des éléments textuels des interfaces utilisateur avec AWT

Dans les premières versions de Java, les API graphiques étaient rangées dans la bibliothèque AWT (***A**bstract **W**indowing **T**oolkit*). Cette bibliothèque est bien entendu portable. Cependant, AWT utilise les classes de base du système d'exploitation pour l'affichage des éléments graphiques. Ainsi, tous les composants graphiques d'AWT (Frame, Button, TextField, Label,…) dérivent des classes de base du système d'exploitation.

Prenons l'exemple d'un bouton. Pour le dessiner, la machine virtuelle aura instancié un objet Button de Windows. C'est alors le code de dessin correspondant à cet objet de base

du système d'exploitation qui sera appelé. Plus précisément, pour les éléments textuels, le système procédera à leur affichage à partir de l'objet de base correspondant instancié. Dès lors, l'élément textuel affiché n'aura plus aucun lien avec la ressource dans laquelle il était initialement décrit.

Les bibliothèques SWT et JFace ont le même mode de fonctionnement qu'AWT.

4.1.2.2.2.2 Affichage des éléments textuels des interfaces utilisateur avec Swing

Swing propose des composants graphiques totalement pris en charge et dessinés par la machine virtuelle Java. En effet, pour une application classique, seule la classe de base JFrame (décrivant la fenêtre principale de l'application) dérive de la classe de base du système d'exploitation.

Illustrons cela dans le cas de Windows. Si nous créons une fenêtre principale qui contient des étiquettes, des boutons, des zones de textes, cette fenêtre dérive de l'objet Windows correspondant. Cela lui permet de recevoir du système d'exploitation les messages nécessaires à son exécution (demande de dessin par exemple…). Le reste des composants (étiquettes, boutons, etc.) est dessiné par la machine virtuelle. Chaque composant graphique possède son propre code de dessin qui ne s'appuie pas sur le code de dessin des composants graphiques de base. Les primitives de base de code de dessin sont réalisées par la machine virtuelle par des appels à des primitives du système. Cela veut dire que, pour dessiner par exemple un bouton, Java a son propre code de dessin du bouton, et que ce code de dessin utilise des primitives de base de dessin du système d'exploitation comme par exemple : drawline(), etc.

Revenons plus précisément sur l'affichage des éléments textuels. Chaque composant graphique (bouton, étiquette, zone de texte, …) existe donc dans le code sous la forme d'une instance d'un objet Swing. Chacun de ses objets possède un attribut texte qui contient l'élément textuel qui sera affiché par le code de dessin du composant. Cet attribut, de façon usuelle, aura été évalué par l'intermédiaire d'un ResourceBundle. Lors de l'affichage, la machine virtuelle exécutera le code de dessin Swing du composant correspondant. Dans ce code de dessin, les éléments textuels calculeront donc leur valeur à partir de l'attribut des composants correspondants, et non depuis les fichiers properties.

4.2 Solution adoptée pour la localisation en contexte : localisation interne du code source

Nous avons exposé les choix usuels des développeurs concernant la fabrication et l'affichage des éléments textuels des interfaces graphiques. Ces choix dépendent du langage de programmation, de l'environnement de développement, et du système d'exploitation.

Notre étude a porté sur les deux langages de programmation les plus utilisés, à savoir C++ sous environnement VisualC++ et Windows (langage et environnement le plus fréquemment rencontré dans le cas des logiciels commerciaux) et Java avec ses bibliothèques AWT et SWING (rencontré dans le cas des logiciels libres, mais aussi pour certains logiciels commerciaux).

Comme nous souhaitons traiter les deux contextes précédents par une approche générique, permettant la localisation *en contexte* des éléments textuels des interfaces utilisateur, nous ne pourrons pas intervenir lors du dessin des éléments textuels pour afficher une chaîne différente de celle de départ. Il est donc impossible de rendre localisable *en contexte* les chaînes des interfaces de façon externe (sans modifier le code source de l'application). En effet, le code affichant les éléments textuels peut être entièrement pris en charge par des bibliothèques système (comme c'est le cas pour Visual C++ et pour Java avec

AWT) ou peut être complètement redéfini par les bibliothèques graphiques (Java avec SWING).

Dans les deux cas, ce sont des primitives de dessin de bas niveau qui réalisent l'affichage point à point des éléments textuels, et ces primitives n'ont plus aucun lien avec les ressources dans lesquelles les éléments textuels ont été définis par les développeurs. Notre solution consiste donc à procéder à une localisation *interne* du code source pour permettre la localisation *en contexte* des éléments textuels des interfaces utilisateur.

Cette solution consiste à attribuer à tous les éléments textuels des interfaces utilisateur un nouveau comportement adapté à la localisation *en contexte*. Pour cela, nous avons besoin (1) de surcharger les classes de base des IHM : l'architecture graphique de logiciels (écrits en C++ ou en Java) est structurée selon des liens d'héritage (Figure 61). Cela veut dire que nous n'avons pas une classe par interface graphique mais une ou deux classes génériques dont dérivent toutes les interfaces graphiques de l'application. D'une manière générale, les développeurs utilisent une classe générique par type d'interface (une classe générique pour les menus contextuels par exemple et une classe pour les boîtes de dialogue, etc.). Pour être le plus générique possible et toucher le moins possible au code source, les modifications doivent porter uniquement sur les classes génériques des IHM. Tous les éléments textuels des interfaces graphiques hériteront alors de ce même comportement. Nous avons besoin aussi (2) d'implémenter un module de localisation en contexte pour gérer l'édition des éléments textuels lors de l'exécution de l'application et l'intégrer dans l'architecture logicielle de l'application à localiser. Ce module est intégré au niveau des classes génériques des IHM, la communication et l'échange de données entre le module de localisation et l'application s'effectuera via un fichier XLIFF que nous détaillerons le contenu dans la section suivante. L'intérêt de ce fichier XLIFF est d'enregistrer toutes les nouvelles propositions de traductions des contributeurs. L'application chargera et affichera donc les chaînes éditées depuis ce fichier XLIFF et non pas depuis les fichiers de ressources initiales (Figure 62).

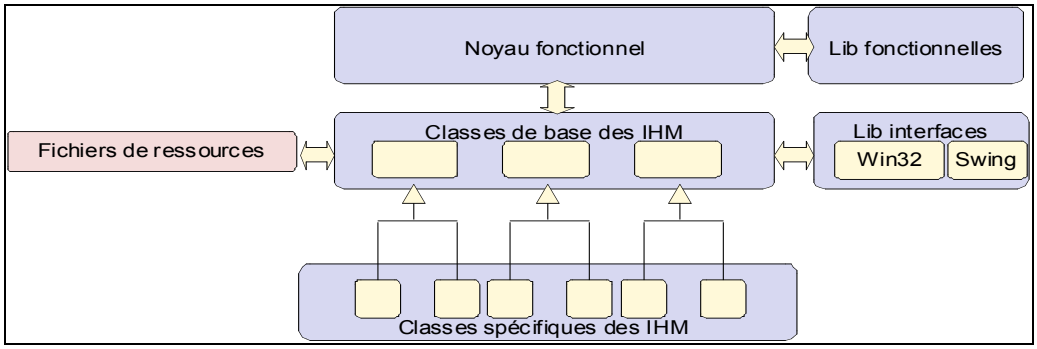


Figure 61 : *Architecture logicielle initiale d'une application*

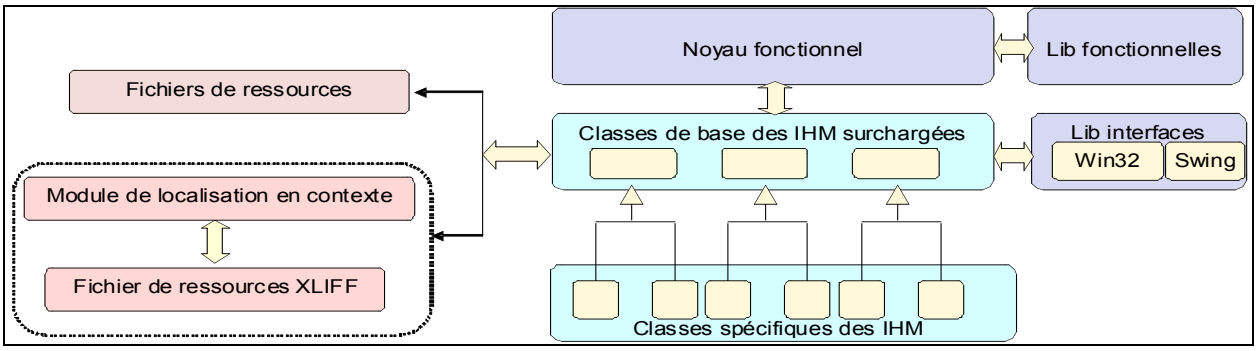


Figure 62 : *Architecture logicielle après localisation interne de l'application*

Nous présentons dans la suite les différentes étapes de la localisation interne du code source.

4.2.1 Construction du fichier XLIFF

4.2.1.1 Présentation du format XLIFF

XLIFF (**X**ML **L**ocalization **I**nterchange **F**ile **F**ormat), introduit en 2001 par OASIS (OASIS, 2001), 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).

4.2.1.2 Structure du fichier XLIFF

Un document XLIFF commence, comme tout document XML, par une déclaration XML (Figure 64). Après la déclaration XML vient le contenu du document, celui-ci est encapsulé dans l'élément <*xliff*>. Ce contenu se compose d'une ou de plusieurs sections, chacune encapsulé dans un élément <*file*>. Un élément <*file*> est constitué d'un élément <*header*>, qui contient des métadonnées à propos de <*file*>, et d'un élément <*body*>, qui contient les données traduisibles extraites de <*file*>. Les données traduisibles au sein d'éléments <*trans-unit*> sont organisées en éléments couplés <*source*> et <*target*>. Ces éléments <*trans-unit*> peuvent être regroupés dans des éléments <*group*> (Figure 64).

En outre, XLIFF offre la possibilité de conserver les informations concernant le traitement du fichier via l'élément <*phase*>. Les traductions possibles d'un élément <*source*> spécifiques peuvent être générées à partir d'un nombre quelconque de systèmes de traductions automatiques, et stockées en tant que frères de l'élément <*source*> dans des éléments <*alt-trans*>.

Il est vivement recommandé que le contenu au sein de l'élément <*file*> soit uniformément bilingue. En d'autres termes, chaque élément <*source*> et <*target*> qui sont fils de l'élément <*trans-unit*> sont, respectivement de la même langue que l'attribut *source-language* et *target-language* de l'élément <*file*>. On ne doit pas utiliser l'attribut `xml:lang` dans ces éléments.

```
<xliff version='1.2'
      xmlns='urn:oasis:names:tc:xliff:document:1.2'>
<file original='hello.txt' source-language='en' target-language='fr'
      datatype='plaintext'>
<body>
<trans-unit id='hi'>
<source>Hello world</source>
<target>Bonjour le monde</target>
</trans-unit>
</body>
</file>
</xliff>
```

Figure 63 : *Structure d'un fichier XLIFF*

La Figure 64 montre un exemple d'un fichier XLIFF qui contient les éléments textuels d'une boîte de dialogue. Le fichier contient donc les chaînes source qui sont en anglais, ainsi que les chaînes traduites en français.

```
<?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 64 : *Exemple de fichier XLIFF contenant les éléments textuels d'une boîte de dialogue*

Nous utilisons donc le format XLIFF comme format d'échange pour communiquer et échanger des données avec l'application. Initialement le fichier XLIFF est vide, il est installé avec l'application sur l'ordinateur de l'utilisateur. Chaque fois qu'une nouvelle proposition de traduction est saisie, un élément *<trans-unit>* est créé dans le fichier XLIFF. Lors 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 initiales de l'application.

4.2.2 Identification des classes génériques des IHM

Le principe de notre solution de localisation *interne* consiste à intervenir uniquement au niveau des classes de base produisant les IHM. Il est donc important, dans un premier temps, d'identifier les classes génériques d'IHM dont dérivent toutes les interfaces utilisateur de l'application.

De façon générale, pour chaque type d'objet d'une IHM (Boîte de dialogue, Menu, etc.), il existe une seule classe générique. Pour identifier ces classes, nous devons étudier l'architecture graphique de l'application en examinant les différentes classes du code source et en s'aidant de la documentation développeur si elle existe. Généralement, les noms des classes peuvent donner aussi une idée sur leur contenu.

Une fois ces classes identifiées, nous procédons à leur localisation *interne* en leur ajoutant, par héritage, un nouveau comportement adapté à la localisation *en contexte* des éléments textuels des interfaces utilisateur.

4.2.3 Intégration du nouveau comportement adapté à la localisation en contexte des éléments textuels des interfaces utilisateur dans les classes de base des IHM

Pour intégrer ce nouveau comportement au niveau des classes génériques des IHM nous rajoutons les deux comportements : mise en édition et mise à jour en temps réel des éléments textuels des interfaces.

4.2.3.1 Ajout du comportement mise en édition

La première modification apportée consiste à rajouter un comportement *mise en édition* aux éléments textuels. Cela permet aux utilisateurs d'éditer n'importe quelle chaîne de l'interface graphique. Pour cela, il faut identifier en premier lieu un événement qui n'est pas utilisé par l'application (Clic-droit, Clic-droit+Ctrl+Shift, etc.), et qui permettra de déclencher l'édition des éléments textuels des interfaces utilisateur. Lorsque l'événement déclencheur est choisi, nous l'interceptons sur la classe générique dont dérivent toutes les interfaces graphiques de l'application. Nous recherchons ensuite la structure de données représentant le contrôle IHM (ou l'entrée de menu) situé sous le curseur de la souris. En réaction à cet événement, nous demandons à l'application la mise en édition de l'élément textuel identifié.

4.2.3.2 Ajout du comportement de mise à jour des IHM par intégration des contributions de localisation présentes dans le fichier XLIFF

Le comportement à ajouter doit permettre la mise à jour de l'IHM en temps réel. Pour cela, nous intervenons au niveau des classes génériques. À la réception du message provoquant l'affichage d'une interface graphique, nous récupérons l'identifiant de l'interface et parcourons le fichier XLIFF, recherchons la présence de nouvelles propositions de traduction des éléments textuels constituant l'interface graphique en question. Si de nouvelles traductions sont présentes dans le fichier XLIFF, nous mettons à jour, dans la structure de données modélisant les éléments textuels, les valeurs textuelles des éléments d'IHM correspondants.

4.2.4 Implémentation du module de localisation en contexte « LocalnContext »

Nous avons implémenté le module LocalnContext pour gérer la localisation en contexte des éléments textuels lors de l'exécution de l'application. Ce module a été implémenté en Visual C++. Ce choix a été motivé par la première application sur laquelle nous avons intégré

notre module, i.e. Notepad-plus-plus, lui-même développé en Visual C++. Le module LocaInContext gère donc les deux comportements de mise en édition ainsi que celui de la mise à jour en temps réel des éléments textuels que nous avons intégré dans les classes de base des IHM (paragraphe 4.2.2.3).

4.2.4.1 Implémentation de la mise en contexte

À l'appel du module pour la mise en contexte, l'application fournit en paramètre l'identifiant de la chaîne qui doit rentrer en édition. Le module, via un analyseur du fichier XLIFF que nous avons développé, collecte toutes les propositions de traduction déjà présentes dans le fichier XLIFF, puis il redirige l'utilisateur vers le contexte d'édition de la chaîne. Ce dernier est une IHM que nous avons développé en Visual C++ articulé autour d'un ListView qui représente les informations du XLIFF que l'analyseur nous a fournit.

Les fonctionnalités principales du contexte d'édition sont les suivantes :

- Possibilité de visualiser les traductions disponibles à partir du fichier XLIFF.
- Choisir une traduction parmi celles qui sont disponibles ou saisir une nouvelle traduction.
- Valider sa traduction et mettre à jour son interface.

Cette IHM contient d'autres fonctionnalités plus avancées que nous détaillerons plus loin dans ce mémoire.

4.2.4.2 Implémentation de la mise à jour en temps réel

Une fois l'édition terminée, la nouvelle proposition de traduction est enregistrée dans le fichier XLIFF.

L'application est alors avertie qu'il faut rafraichir son IHM. Elle appelle alors le module LocaInContext, en lui passant en paramètre l'identifiant de l'élément textuel qu'elle rafraichit pour récupérer la traduction choisie par l'utilisateur depuis le fichier XLIFF. Si aucune traduction n'est trouvée, on ne remet pas à jour l'IHM.

4.2.5 Intégration du module « LocaInContext » dans l'application

Le module LocaInContext est intégré dans l'architecture logicielle de l'application, et plus particulièrement au niveau des classes de base des IHM (Figure 65).

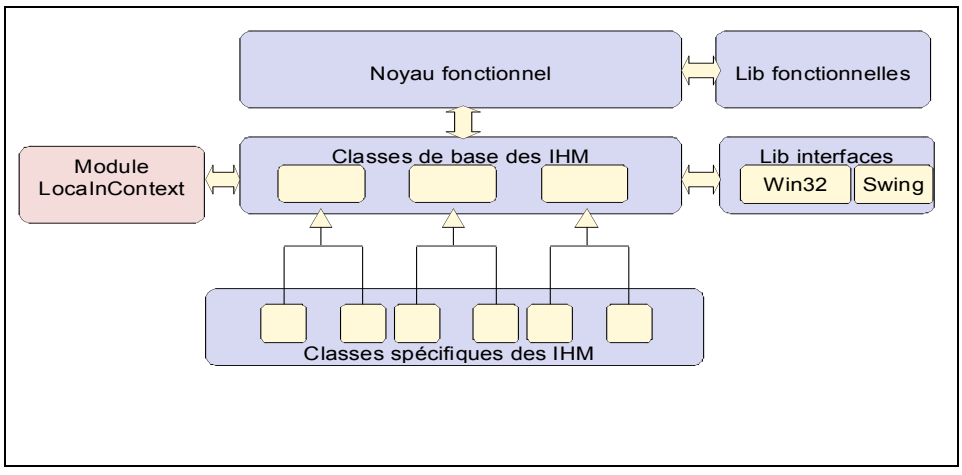


Figure 65 : *Intégration du module LocaInContext dans l'architecture logicielle d'une application*

L'application interagit avec le module LocaInContext lors de la localisation d'un élément textuel de l'interface par l'utilisateur, ainsi qu'au moment de la mise à jour de l'interface.

4.2.5.1 Interaction entre l'application et le module « LocaInContext » au moment de l'édition d'une chaîne de l'interface par l'utilisateur

Au moment de la demande d'édition d'une chaîne de l'interface utilisateur (clic-droit sur la chaîne, Ctrl+alt+clic-droit sur la chaîne, etc.), l'application récupère l'identifiant de la chaîne et appelle une méthode du module LocaInContext qui va provoquer la mise en édition de la chaîne d'interface correspondante. 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. Lorsque l'édition est terminée, le fichier XLIFF est mis à jour et l'application est avertie afin qu'elle mette à jour son interface.

4.2.5.2 Interaction entre l'application et le module « LocaInContext » au moment de la mise à jour de l'interface utilisateur

Les bibliothèques graphiques les plus connues et les plus utilisées en Java sont Swing, AWT, SWT (Sun, 2008). En C++, il s'agit de Win32 (sous Windows) (Microsoft, 2008). Le principe de fonctionnement de ces bibliothèques 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 LocaInContext 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.

Conclusion

Nous avons présenté dans ce chapitre un état de l'art sur les choix usuels des développeurs concernant la fabrication ainsi que l'affichage des interfaces graphiques et plus particulièrement les éléments textuels des interfaces utilisateurs. Cela nous a permis de montrer que, pour permettre la localisation *en contexte* des logiciels existants, nous devons intervenir sur le code source et procéder donc à une localisation *interne*. Nous avons détaillé par la suite le principe de cette localisation interne.

Chapitre 5 : Expérimentation de la localisation en contexte sur un logiciel libre

Introduction

Nous avons mené une expérience complète avec Notepad-plus-plus (logiciel libre à code source ouvert 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 Adobe PhotoShop que nous avions étudiés auparavant. Cela nous a permis de tenter sa *localisation interne* nous-mêmes, sans faire appel à des collaborateurs extérieurs. Nous illustrons en premier temps le scénario de localisation *en contexte* sur l'exemple de Notepad-plus-plus, puis nous détaillons la localisation *interne* de son code source.

5.1 Étude et recherche de logiciel pour expérimentation

Nous présentons dans cette section l'étude et la recherche que nous avons effectuée avant de faire notre choix de logiciel. Étant donné que ma thèse a été effectuée dans le cadre d'une convention CIFRE avec WinSoft. Cela nous a permis d'étudier de près certains logiciels propriétaires d'Adobe et de FileMaker localisés par WinSoft. Cette étude a porté aussi sur des logiciels libres à code source ouvert.

5.1.1 Logiciels propriétaires étudiés

Dans le cadre de sa collaboration avec Adobe et FileMaker, WinSoft a accès à certaines parties du code source des logiciels. En outre, certaines parties du code source, à disposition de WinSoft, peuvent être modifiées à leur gré. Cela nous a permis d'avoir accès au code source de certaines applications et de tenter leur localisation *interne* afin de les rendre localisable *en contexte*. Nous détaillons donc dans la suite nos deux expérimentations que nous avons effectuées sur les deux logiciels Adobe PhotoShop 11 et FileMaker 10, puis nous fournissons un bilan de ces expériences.

5.1.1.1 FileMaker 10

FileMaker 10 est le premier logiciel que nous avons étudié. Il contient environ 5 millions de lignes de code, écrit entièrement en langage C++. Les ressources textuelles des interfaces utilisateur sont stockées dans des fichiers de ressources de type RC. Elles sont donc compilées avec l'application et incorporées dans des fichiers binaires (.RES). L'architecture logicielle de FileMaker est sous forme d'une arborescence avec des liens d'héritage entre les différentes classes. Nous avons donc pu identifier les classes génériques qui produisent les différentes interfaces graphiques de FileMaker. Elles sont au nombre de deux. Pour des raisons de confidentialité, nous ne pouvons pas donner plus de détail sur l'architecture graphique de FileMaker. WinSoft n'étant pas le propriétaire du code source, nous n'avons pas pu mener une expérimentation complète. En effet, l'accès et la modification du code source sont soumis à des contrats bien définis entre WinSoft et Adobe.

5.1.1.2 Adobe PhotoShop 11

Le deuxième logiciel propriétaire que nous avons étudié est Adobe PhotoShop 11. Il s'agit d'un code de taille plus importante, plus de 7 millions de lignes de code écrits en C++. Comme nous l'avons expliqué dans le premier chapitre de ce mémoire, pour des raisons de sécurité, Adobe utilise ses propres formats de ressources EVE. Cela permet d'avoir des ressources difficilement décryptables. Comme pour FileMaker, nous avons commencé par étudier l'architecture logicielle de l'application afin d'identifier les classes de base de l'IHM. Nous avons pu identifier trois classes génériques qui produisent les différentes interfaces graphiques du logiciel. Vu la taille de Adobe PhotoShop 11, l'étude de son architecture logicielle et l'identification des classes produisant les IHM nous ont pris environ un mois. WinSoft étant sous-traitant d'Adobe, toute modification de code source doit faire l'objet d'un contrat précis entre les deux entités. À l'heure de la rédaction de ce mémoire, les modalités de la localisation *interne* des logiciels d'Adobe par WinSoft sont encore en cours de définition. Ainsi, nous n'avons pas pu procéder à la localisation *interne* d'Adobe PhotoShop 11.

5.1.1.3 Bilan de ces deux expériences

Avant de démarrer l'investigation sur Adobe Photoshop 11 et FileMaker 10, Winsoft a pris contact avec Adobe et FileMaker afin de définir les modalités d'un contrat qui nous permettrait d'avoir accès et de modifier le code source de ce logiciel. Les discussions sont encore à ce jour en cours.

La tentative d'expérimentation de localisation *en contexte* sur les deux logiciels propriétaires FileMaker 10 et Adobe PhotoShop 11 nous a permis d'identifier des problèmes et contraintes spécifiques aux logiciels commerciaux. En effet, en plus du fait qu'il s'agit de logiciels à code source fermé, certains éditeurs de logiciels comme Adobe utilisent leurs propres API graphiques pour fabriquer leurs interfaces utilisateur ainsi que leurs propres formats de ressources (comme EVE pour Adobe).

Dans le cas d'Adobe, même si nous avons accès au code source des applications, nous n'avons pas accès au code source des API graphiques utilisées. Ainsi, nous n'avons aucune idée sur le fonctionnement de ces bibliothèques graphiques et plus particulièrement sur la façon dont les éléments textuels sont fabriqués et affichés. Cela implique que, sans l'autorisation et la collaboration de l'éditeur du logiciel, nous ne pouvons mener une expérimentation complète sur ce type de logiciel.

5.1.2 Logiciels libres étudiés et choix de Notepad-plus-plus

Après nos deux études des logiciels propriétaires, nous nous sommes orientés vers les logiciels libres à code source ouvert. Les deux logiciels propriétaires (FileMaker et Adobe PhotoShop 11) que nous avons étudiés sont écrits en C++. Nous avons donc essayé de trouver un logiciel libre à code source ouvert écrit en C++. Nous avons, en particulier, étudié VLC[15] et Notepad-plus-plus[16].

Nous avons retenu Notepad-plus-plus car sa taille était raisonnable (60.000 lignes de code) et aussi il est connu et utilisé par de nombreux utilisateurs. Notepad-plus-plus est un éditeur de code source qui prend en charge plusieurs langages (Java, C++, HTML, XML, etc.) (Figure 66).

[15] http://vlcplayer.2010-fr.com/fr/
[16] http://notepad-plus.sourceforge.net/fr/site.htm

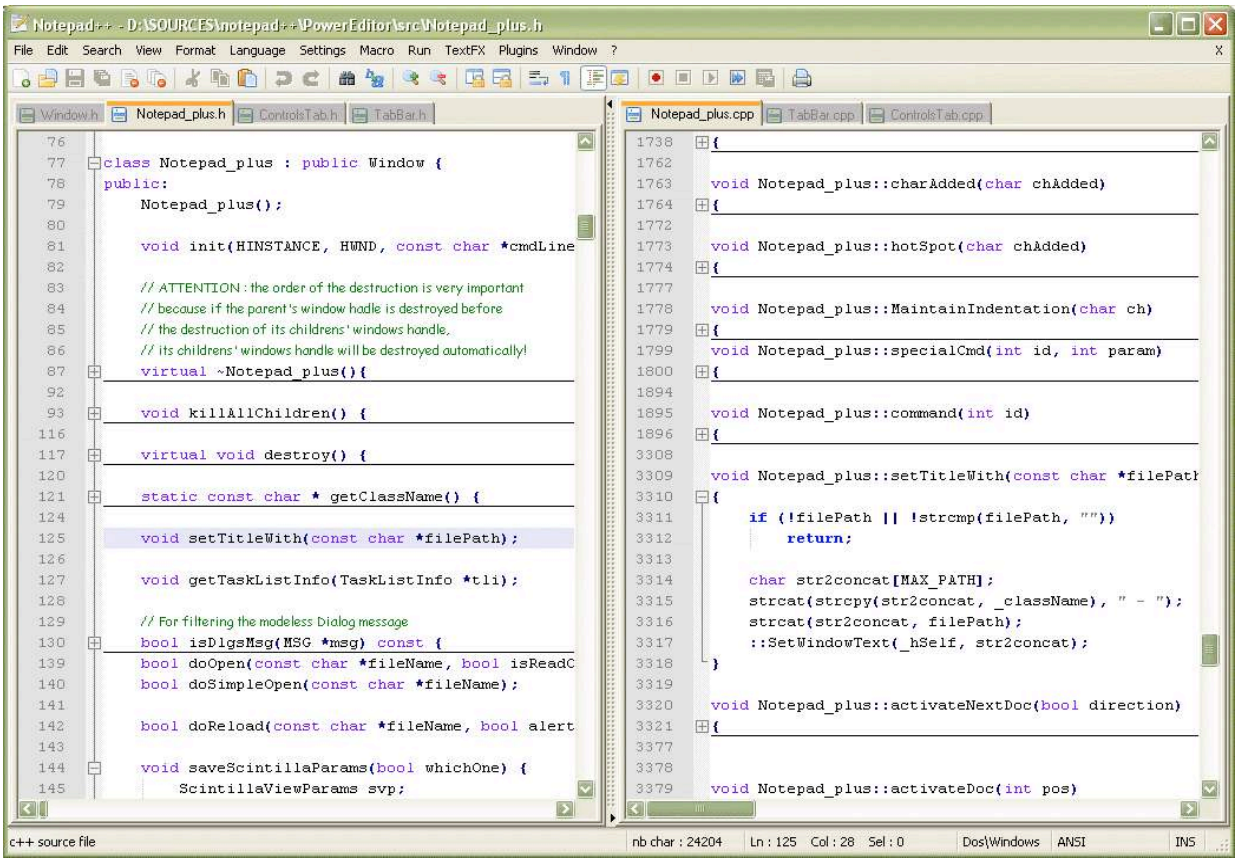


Figure 66 : *Interface principale de l'outil Notepad-plus-plus*

5.2 Illustration du scénario de la localisation en contexte sur Notepad-plus-plus

Le scénario de la localisation *en contexte* doit permettre à tout type de contributeur (utilisateur final, traducteur bénévole, β-testeur, etc.), quelle que soit sa maîtrise de l'outil informatique, de participer de la façon la plus simple et la plus efficace possible à la localisation du logiciel. Cependant, dans le cas des logiciels commerciaux, l'esprit de la contribution bénévole est moins important que pour les logiciels libres. Ainsi, afin de motiver les utilisateurs finals à contribuer, le scénario ne doit nécessiter aucun effort supplémentaire de la part des utilisateurs.

Le scénario de localisation *en contexte* comporte 3 étapes principales.

5.2.1 Étape 1 : éditer une chaîne de l'interface en cours d'utilisation du logiciel

Durant l'utilisation du logiciel, le contributeur peut éditer n'importe quelle chaîne de l'interface (bouton, étiquette, titre de fenêtre, icone, liste, etc.). Dans le cas de Notepad-plus-plus, nous avons choisi l'événement clic-droit sur une chaîne d'interface comme événement déclencheur d'édition. Ainsi par un simple clic-droit, le contributeur peut éditer n'importe quelle chaîne de l'interface. Prenons l'exemple de la boîte de dialogue "Column Editor" du logiciel Notepad-plus-plus (Figure 67). Pour traduire la chaîne anglaise "Text to insert" en français, il suffit de faire un clic-droit sur la chaîne "Text to insert" pour passer dans un contexte d'édition qui contient par défaut :

- la chaîne source sur laquelle l'utilisateur a cliqué.
- une liste des traductions disponibles produites par des systèmes de traductions automatiques ou à partir de dictionnaires locaux.

L'utilisateur peut saisir une nouvelle traduction ou en choisir une parmi la liste des traductions disponibles.

Les traductions disponibles (produites par les systèmes de TA ou à partir de dictionnaires) sont installées avec l'application dans des fichiers de ressources locales. Nous détaillons le contenu de ces ressources plus loin dans ce mémoire.

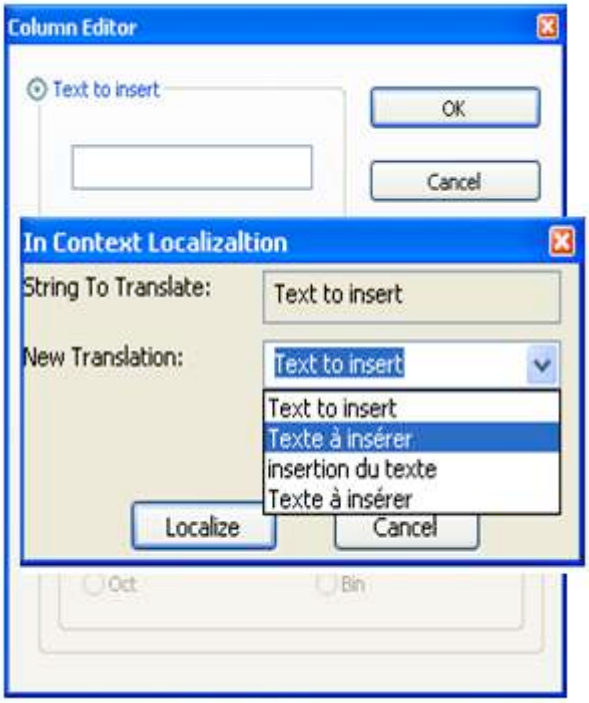


Figure 67 : *Localisation en contexte des éléments textuels de la boîte de dialogue "Column Editor" du logiciel Notepad-plus-plus*

5.2.2 Étape 2 : proposer et valider une nouvelle proposition de traduction

Une fois la nouvelle proposition de traduction saisie (Figure 68), pour valider sa traduction, le contributeur doit cliquer sur le bouton "Localize" de la boîte de dialogue "In Context Localization". La validation d'une traduction permet d'enregistrer en local la nouvelle proposition de traduction dans le fichier XLIFF et de mettre à jour en temps réel l'interface utilisateur.

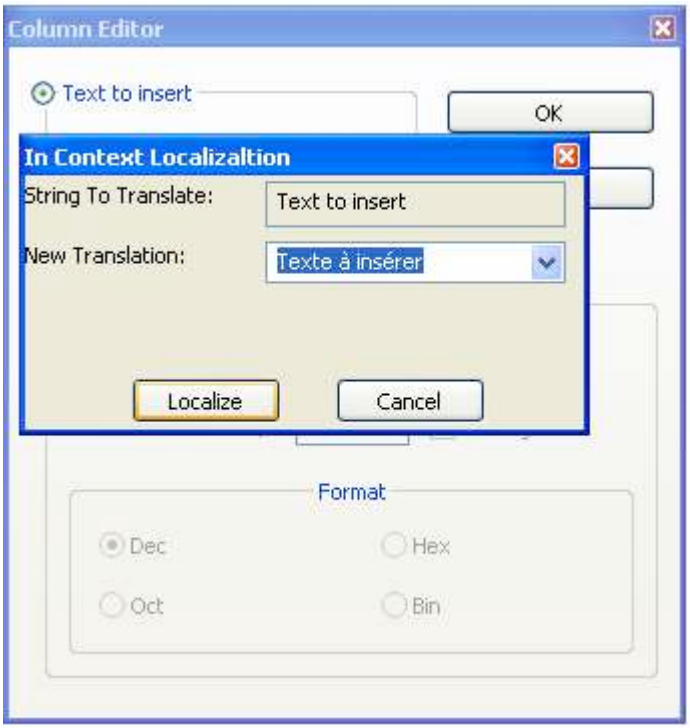


Figure 68 : *Traduction de la chaîne "Text to insert" de la boîte de dialogue "Column Editor" par « Texte à insérer »*

5.2.3 Étape 3 : Mise à jour en temps réel de l'interface utilisateur

Une fois enregistrée, la nouvelle proposition de traduction apparaît en temps réel dans l'interface utilisateur. La mise à jour de l'interface ne nécessite aucun redémarrage de l'application (Figure 69). L'utilisateur pourra donc continuer à utiliser le logiciel normalement ou traduire d'autres chaînes de l'interface.

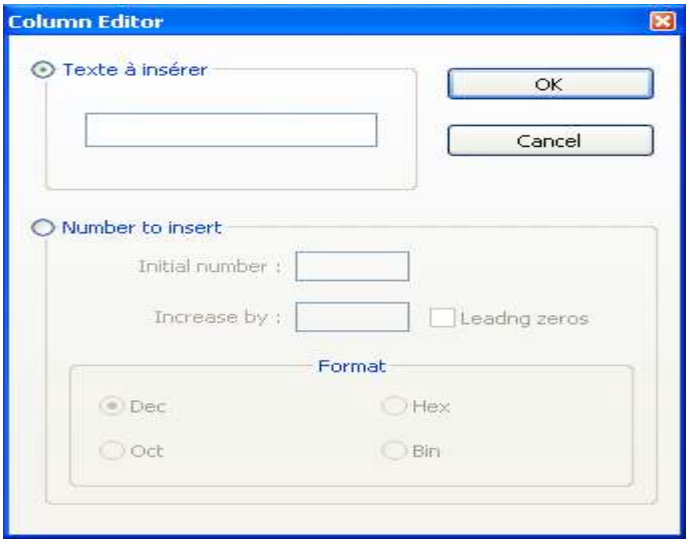


Figure 69 : *Apparition de la nouvelle proposition de traduction « Texte à insérer » en temps réel*

5.3 Localisation interne du code source de Notepad-plus-plus

Comme expliqué dans le chapitre précédent, nous avons choisi la localisation *interne* comme solution pour permettre la localisation en contexte des logiciels existants. Notre méthode de localisation interne consiste à modifier le code de manière minimale très localement, et systématiquement de la même manière pour toutes les applications.

5.3.1 Construction du fichier XLIFF

Comme nous l'avons expliqué dans le chapitre 4, nous avons besoin de construire un fichier XLIFF. Initialement, ce fichier est vide, durant la localisation en contexte, toutes les nouvelles propositions de traduction sont stockées dans ce fichier. Dans le cas de Notepad-plus-plus, les éléments textuels des interfaces graphiques sont stockés, initialement, dans un seul fichier de ressource NativeLang.xml (Figure 70). Lors du stockage d'une nouvelle proposition de traduction dans le fichier XLIFF, nous récupérons aussi, à partir du fichier NativeLang, toutes les informations relatives à la chaîne éditée (identifiant, chaîne source ainsi que l'identifiant de l'IHM contenant la chaîne). Ces informations nous seront utiles pour pouvoir identifier la chaîne lors de la mise à jour de l'interface.

```
<?xml version="1.0" encoding="Windows-1252" ?>
<NotepadPlus>
    <Native-Langue name = "Français">
        <Menu>

        <Dialog>
            <Find title = "" titleFind = "Rechercher" titleReplace = "Remplacer" titleFindInFiles = "Rechercher dans les fichiers">
            <GoToLine title = "Allez à la Ligne #">

            <Run title = "Exécuter...">

            <StyleConfig title = "Configurateur de coloration syntaxique">

            <UserDefine title = "Langage Défini par l'utilisateur">
            <Preference title = "Préférences">
            <MultiMacro title = "Exécuter une macro en boucle">
            <Window title = "Document">
            <ColumnEditor title = "Editeur de Colonne">
                <Item id = "2023" name = "Texte à insérer"/>
                <Item id = "2033" name = "Nombre à insérer"/>
                <Item id = "2030" name = "Nombre initial :"/>
                <Item id = "2031" name = "Augmenter de :"/>
                <Item id = "2035" name = "Nombre de zéros"/>
                <Item id = "2032" name = "Format"/>
                <Item id = "2024" name = "Dec"/>
                <Item id = "2025" name = "Oct"/>
                <Item id = "2026" name = "Hex"/>
                <Item id = "2027" name = "Bin"/>
                <Item id = "1" name = "OK"/>
                <Item id = "2" name = "Annuler"/>
            </ColumnEditor>
        </Dialog>
    </Native-Langue>
</NotepadPlus>
```

Figure 70 *: Extrait du fichier NativeLang contenant les chaînes d'interface de la boîte de dialogue « Éditeur de colonne »*

```
<?xml version="1.0" encoding="UTF-8" ?>
<group restype="dialog" resname=" Editeur de colonne">
    <trans-unit id = "2023">
        <source name = "Texte à insérer"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2033">
        <source name = "Nombre à insérer"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2030">
        <source name = "Nombre initial :"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2031">
        <source name = "Augmenter de :"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2035">
        <source name = "Nombre de zéros"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2032">
        <source name = "Format"/>
        <target name=""/>
    </trans-unit>
    <trans-unitid = "2024">
        <source name = "Dec"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2025">
    <trans-unit id = "2026">
    <trans-unit id = "2027">
    <trans-unit id = "1">
        <source name = "OK"/>
        <target name=""/>
    </trans-unit>
    <trans-unit id = "2">
        <source name = "Annuler"/>
        <target name=""/>
    </trans-unit>
</group>
```

Figure 71 : *Extrait du fichier XLIFF contenant les chaînes d'interfaces de la boîte de dialogue « Éditeur de Colonne »*

5.3.2 Identification des classes de base produisant les IHM

Il est important de comprendre l'architecture logicielle de l'application à localiser et plus particulièrement l'architecture graphique afin d'identifier les classes génériques d'IHM sur lesquelles nous devrons intervenir.

Dans le cas de Notepad-plus-plus, nous avons identifié deux classes génériques d'IHM dont dérivent toutes les interfaces de l'application, la classe Static_Dialog dont dérivent toutes les boîtes de dialogue de l'application, et la classe Static_Menu dont dérivent tous les menus contextuels de l'application (Figure 72). Toutes nos modifications portent donc exclusivement sur ces deux classes.

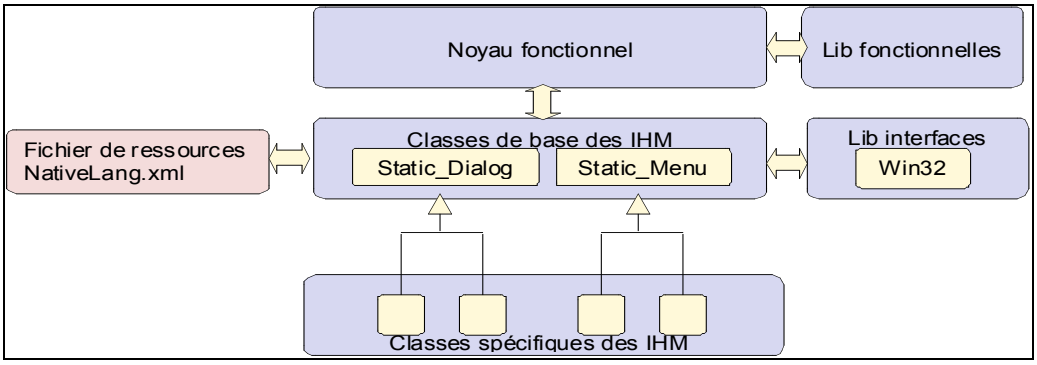


Figure 72 : *Architecture logicielle initiale de Notepad-plus-plus*

La classe Static_Dialog est la classe générique dont dérivent toutes les boîtes de dialogue du logiciel et plus particulièrement tous les éléments textuels des boîtes de dialogue. Elle confère à toutes ses classes filles, des comportements de dessin (arrière-plan des dialogues, police utilisée, etc.), et le même mécanisme de chargement de ses données textuelles depuis le fichier de ressources NativeLang.

La classe Static_Menu est la classe générique dont dérivent tous les menus contextuels de Notepad-plus-plus. Outre le comportement de dessin, un mécanisme de chargement des données textuelles, similaire à celui de Satic_Dialog, est conféré à toutes ses classes filles.

5.3.3 Intégration du nouveau comportement adapté à la localisation en contexte

Nous intégrons donc le nouveau comportement adapté à la localisation *en contexte* au niveau des deux classes génériques Static_Dialog et Static_Menu.

5.3.3.1 Intégration du nouveau comportement au niveau de la classe Static_Dialog

Nous surchargeons la classe générique Static_Dialog en rajoutant le comportement de mise en édition, de façon à ce que tous les éléments textuels héritent de ce comportement. Pour cela, nous interceptons au niveau de la classe Static_Dialog le clic-droit sur les éléments textuels. Nous recherchons ensuite la structure de données représentant l'élément textuel qui se trouve sous le curseur de la souris. Cela nous permet de récupérer l'identifiant de l'élément textuel en question. Nous demandons ensuite à l'application d'éditer l'élément textuel identifié.

De la même façon, nous rajoutons ensuite le comportement de mise à jour en temps réel des éléments textuels édités. Pour cela, nous interceptons au niveau de la classe Static_Dialog les messages provoquant l'affichage de nouvelles boîtes de dialogue. Au moment ou la classe Static_Dialog reçoit un message pour afficher une boîte de dialogue, nous récupérons l'identifiant de la boîte de dialogue. Puis, nous parcourons le fichier XLIFF, via l'identifiant de l'interface, pour vérifier s'il existe de nouvelles propositions de traduction des éléments textuels de la boîte de dialogue. Si c'est le cas, nous demandons à l'application de mettre à jour la structure de données des éléments textuels concernés avant d'afficher la boîte de dialogue.

5.3.3.2 Intégration du nouveau comportement au niveau de la classe Static_Menu

Nous procédons de la même manière pour la classe Static_Menu que pour la classe Static_Dialog. Nous rajoutons d'abord le comportement de mise en édition au niveau de la classe Static_Menu afin que tous les éléments textuels des menus contextuels puissent hériter de ce même comportement. Pour cela, nous interceptons le clic-droit sur les chaînes des menus au niveau de la classe Static_Menu. Nous demandons ensuite à l'application de récupérer l'identifiant de l'entrée de menu qui se trouve sous le curseur de la souris. Une fois l'élément textuel identifié, nous demandons à l'application de le passer en édition.

Ensuite, comme pour la classe Static_Dialog, nous rajoutons le comportement de mise à jour en temps réel aux chaînes des menus contextuels en interceptant le message provoquant l'affichage des menus au niveau de la classe Static_Menu. À la réception d'un message pour afficher un menu contextuel dans la classe Static_Menu, nous parcourons le fichier XLIFF pour vérifier si l'élément textuel du menu contextuel admet une nouvelle proposition de traduction. Si c'est le cas, nous demandons à l'application, avant d'afficher le menu, de mettre à jour la valeur de la structure de données représentant l'élément textuel.

5.3.4 Intégration du module de localisation en contexte dans l'architecture logicielle de Notepad-plus-plus

Nous intégrons le module de localisation en contexte : LocaInContext au niveau des classes génériques des IHM : Static_Dialog et Static_Menu (Figure 73). Le module LocaInContext communique avec la classe Static_Dialog dans le cas de la localisation en contexte d'un élément textuel qui appartient à une boîte de dialogue et avec la classe Static_Menu quand il s'agit d'une chaîne d'un menu contextuel.

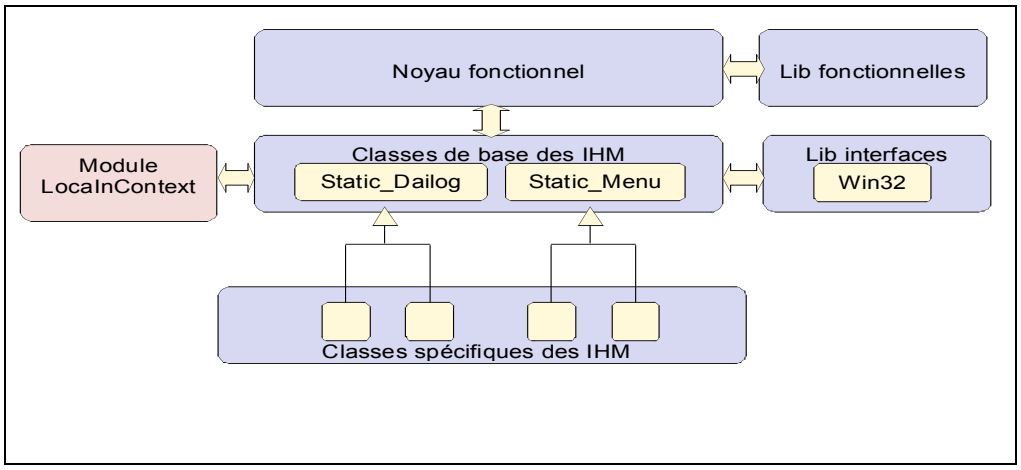


Figure 73 : *Intégration du module LocaInContext dans l'architecture logicielle de Notepad-plus-plus*

5.4 Bilan de la localisation interne du code source de Notepad-plus-plus

Avant de procéder à la localisation interne du code source de Notepad-plus-plus, nous avons mis environ 28h pour étudier l'application et identifier les classes génériques des IHM. La localisation interne nous a pris environ 25h.

Une fois la localisation interne du code source du logiciel terminée, nous avons testé la localisation en contexte des éléments textuels des interfaces graphiques pour vérifier si les chaînes d'interface ont bien hérité du comportement de localisation en contexte que nous avons rajouté aux deux classes génériques.

5.4.1 Chaînes d'interfaces héritant du comportement de la localisation en contexte

Les chaînes qui ont hérité du comportement de localisation en contexte sont : les titres de fenêtre, les étiquettes, les boutons, les boutons-radio, les icônes, les onglets, les menus, les listes, etc.

Pour résumer, tout contrôle qui ne surcharge pas l'affichage de sa partie textuelle a hérité de ce comportement, soient 586 chaînes sur 600 chaînes en total.

5.4.2 Chaînes d'interfaces n'ayant pas hérité du nouveau comportement de localisation en contexte

Certains éléments textuels des interfaces utilisateur n'ont pas hérité du comportement de localisation en contexte. Cela est dû au fait que ces éléments:

- sont codés en dur dans le code source de l'application. Dans ce cas, ces éléments ont bien hérité du comportement autorisant l'édition, mais ils ne peuvent pas être mis à jour.
- ou bien ils changent de valeur durant l'exécution de l'application.

Dans le cas de Notepad-plus-plus, ce sont les messages d'erreur qui sont codés en dur, et certains contrôles dont le texte est formaté par code avant affichage. Le nombre total de ces chaînes est de 14.

Conclusion

Nous avons expérimenté dans ce chapitre la localisation en contexte sur Notepad-plus-plus. Cela nous a permis de prouver la faisabilité technique de notre approche de localisation en contexte en procédant à une localisation interne du code source de l'application. Cette expérimentation nous a permis de localiser en contexte, de l'anglais vers le français, toutes les chaînes des interfaces graphiques de Notepad-plus-plus, sauf celles qui sont codées en dur dans l'application, où dont la valeur peut changer lors de l'exécution.

Chapitre 6 : Expérimentation complémentaire sur un autre logiciel libre

Introduction

Nous présentons dans ce chapitre une expérimentation complémentaire de notre méthode de localisation *en contexte* sur un deuxième logiciel libre Vuze[17]. Le but de ce chapitre est de valider notre approche sur un logiciel qui utilise une technologie différente de celle utilisée dans Notepad-plus-plus, ainsi que d'évaluer le délai de localisation interne du deuxième logiciel par rapport au premier.

6.1 Localisation interne de Vuze

6.1.1 Pourquoi Vuze ?

Comme annoncé dans le chapitre 4, notre approche de localisation interne peut être appliquée sur tout logiciel écrit en C++/Win32 et JAVA/SWING-AWT-SWT. Notre première expérimentation ayant porté sur le logiciel Notepad-plus-plus écrit en C++ et utilisant la bibliothèque graphique Win32, nous avons décidé de choisir un logiciel écrit en JAVA/SWING-AWT-SWT afin d'expérimenter notre méthode sur une autre technologie.

La Figure 75 montre la liste des logiciels que nous avons étudiés avant de choisir Vuze. Les critères retenus pour la sélection des logiciels candidats à la localisation en contexte ont été de deux sortes. Nous avons d'une part recherché des logiciels possédant une documentation développeur afin de faciliter la compréhension du code, et d'autre part nous avons sélectionné des logiciels ayant des ressources textuelles externalisées.

Après l'étude de quelques logiciels comme Eclipse, nous nous sommes rendu compte que, pour pouvoir effectuer la localisation interne de ce type de logiciel dans des délais raisonnables, la documentation développeur n'est pas suffisante, la collaboration avec les développeurs de l'application est indispensable. En effet, même dans les cas où il existe une documentation développeur, elle est souvent mal écrite voire non mise à jour. La contrainte de temps étant forte, nous avons fait le choix de prendre un logiciel de taille raisonnable, même s'il ne possède pas de documentation développeur.

Vuze est un logiciel de P2P conçu pour le protocole Bittorrent (Figure 74). Il contient environ 80.000 lignes de code, et ayant ses ressources textuelles externalisées. Il est écrit en JAVA et utilise SWT comme bibliothèque graphique. Il répond donc à nos deux critères. Une étude de son architecture graphique a confirmé le choix de ce logiciel. En effet, les classes produisant les différents IHM de l'application sont liées entre elles par des liens d'héritage. Ce qui est important pour la localisation interne du code source dans la mesure où nous intervenons uniquement sur les classes génériques des IHM. Dans le cas contraire notre

[17] Logiciel de P2P conçu pour le protocole Bittorrent.
www.vuze.com

intervention sera beaucoup plus lourde car nous devrions procéder à la localisation interne de toutes les classes graphiques.

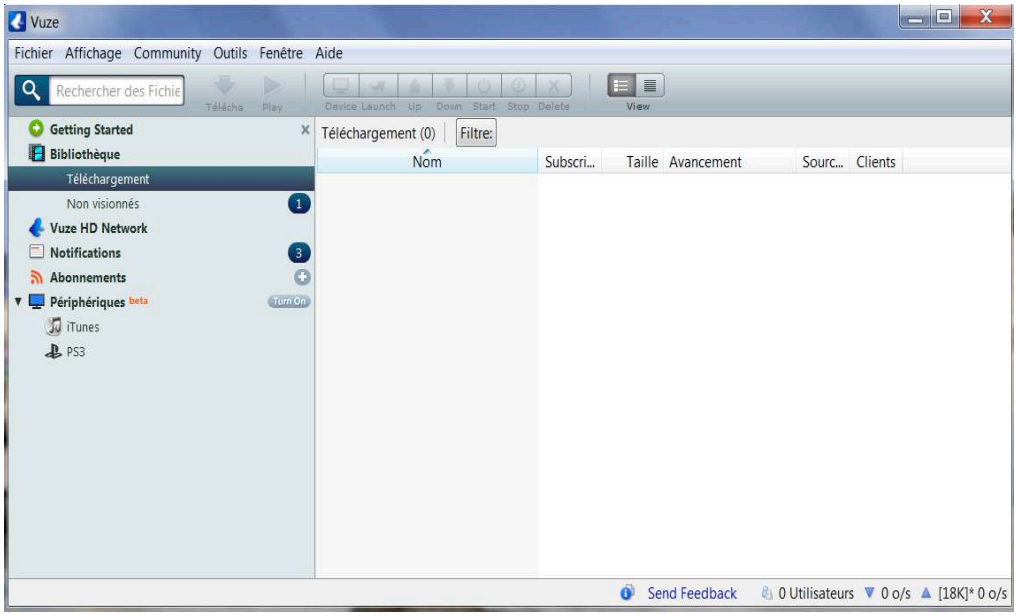


Figure 74 : *Interface principale de Vuze*

	Petit	Gros
Pas de documentation développeur	Vuze	
Avec documentation développeur	VLC	Eclipse

Figure 75 : *Listes des logiciels libres à code source ouvert étudiés*

6.1.2 Localisation interne du code source de Vuze

Comme nous l'avons déjà dit, la localisation interne du code source comporte trois étapes principales : l'étude de l'application, l'identification des classes de base produisant les IHM de l'application, et enfin l'intégration du nouveau comportement de localisation en contexte au niveau des classes génériques des IHM précédemment identifiées.

6.1.2.1 Étude de l'application

L'étude de l'application est la première étape de notre processus de localisation interne. Il s'agit d'étudier le format des ressources textuelles de l'application, la bibliothèque graphique utilisée, ainsi que son principe de fonctionnement.

6.1.2.1.1 Fichiers de ressources de Vuze

Dans le cas de Vuze, toutes les ressources textuelles sont stockées dans des fichiers texte properties. Comme expliqué dans le chapitre 4, les fichiers properties sont des fichiers texte utilisés par les développeurs Java. Leurs contenu est sous forme d'une liste d'association identifiant_chaîne = valeur_chaîne. Les fichiers properties sont installés avec l'application dans leur format d'origine (texte) et les chaînes sont donc chargées au moment de l'exécution de l'application.

Comme le montre la Figure 76, les identifiants des chaînes, choisis par les développeurs, indiquent l'endroit d'apparition de la chaîne dans les interfaces graphiques. Prenons l'exemple suivant :

MainWindow.menu.file = &Fichier

Nous pouvons déduire que la chaîne « &Fichier » apparaît dans le menu menu de la fenêtre principale MainWindow. Le seul intérêt d'avoir des identifiants sous cette forme est d'avoir un fichier de ressources plus lisible du fait que les identifiants ont un sens.

```
Main.parameter.maxUploads=Nombre max d'envois simultan\u00e9s
Main.parameter.maxSpeed=Vitesse d'envoi max en octets/s
MainWindow.menu.file=&Fichier
MainWindow.menu.file.open=&Ouvrir
MainWindow.menu.file.create=&Cr\u00e9er un torrent
MainWindow.menu.file.create.fromfile=\u00c0 &partir d'un fichier
MainWindow.menu.file.create.fromdir=\u00c0 partir d'un r\u00e9perto&ire
MainWindow.menu.file.export=&Exporter un torrent en XML...
MainWindow.menu.file.import=&Importer un torrent XML...
MainWindow.menu.file.closetab=Fermer l'ongle&t
MainWindow.menu.file.closewindow=Fermer la &fen\u00eatre
MainWindow.menu.file.exit=&Quitter
MainWindow.dialog.choose.file=Choisissez le fichier torrent
MainWindow.menu.file.folder=R\u00e9pe&rtoire
MainWindow.dialog.choose.folder=Choisissez le r\u00e9pertoire contenant les .torrent
MainWindow.menu.view=&Affichage
MainWindow.menu.view.show=Afficher
MainWindow.menu.view.mytorrents=M&es torrents
MainWindow.menu.view.configuration=O&ptions...
MainWindow.menu.closealldetails=Fermer tous les &d\u00e9tails
MainWindow.menu.closealldownloadbars=Fermer toutes les &mini-barres
MainWindow.menu.language=&Langue
ConfigView.section.language=Langue
MainWindow.menu.window=&Fen\u00eatre
MainWindow.menu.window.minimize=&Minimiser
MainWindow.menu.window.alltofront=A&fficher tout devant
MainWindow.menu.help=Aid&e
MainWindow.menu.help.about=\u00c0 &propos...
MainWindow.about.title=\u00c0 propos de Vuze
MainWindow.about.section.developers=D\u00e9veloppeurs
MainWindow.about.section.translators=Traducteurs
MainWindow.about.section.system=Syst\u00e8me
MainWindow.about.section.internet=Sur Internet
MainWindow.about.internet.homepage=Page d'accueil de Vuze
MainWindow.about.internet.sourceforge=Page Sourceforge
MainWindow.about.internet.sourceforgedownloads=T\u00e91\u00e9chargements sur Sourceforge
MainWindow.about.internet.bugreports=Rapporter un bogue
MainWindow.about.internet.forumdiscussion=Forum g\u00e9n\u00e9ral
```

Figure 76 : *Extrait du fichier de ressources MessagesBundle_fr_FR.properties de l'application Vuze*

6.1.2.1.2 Bibliothèque graphique SWT : principe de fonctionnement

Les développeurs de Vuze utilisent la bibliothèque graphique SWT pour fabriquer les interfaces graphiques de l'application. La compréhension du principe de fonctionnement de cette API est importante avant de procéder à la localisation interne des classes de base des IHM.

SWT est une bibliothèque graphique Java développée par IBM pour Eclipse (IBM, 2000,2006), (Horn, 2005). Celle-ci est distribuée librement en open source par IBM et peut donc être utilisée par les développeurs pour créer des plugins pour Eclipse ou des applications autonomes.

SWT à la particularité de procurer des composants graphiques de base d'une manière indépendante du système d'exploitation.

L'API SWT est implémentée sur différentes plates-formes en utilisant une combinaison de Java et de JNI[18] spécifique à chaque plate-forme. SWT est développée en totalité en Java : le JNI est uniquement utilisé pour appeler le système d'exploitation. Il y a une bijection entre les méthodes de base Java et les appels au système d'exploitation.

Il y a donc une implémentation des composants SWT pour chaque plate-forme mais la signature des méthodes publiques reste la même. L'application qui utilise les composants SWT est donc bien multiplate-forme.

La Figure 77 montre la hiérarchie des classes du package SWT (Eclipse, 2008). On voit que tous les composants graphiques (étiquette, onglet, bouton, menu, liste, etc.) dérivent de la classe Control, qui elle-même dérive de la classe de base Widget.

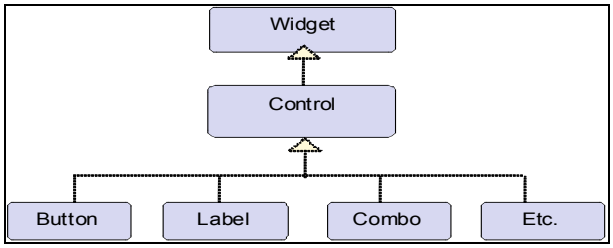


Figure 77 : *Hiérarchie des classes SWT*

6.1.2.2 Identification des classes de base produisant les IHM et gestion des ressources textuelles

Nous présentons ici les classes génériques des IHM de Vuze ainsi que la façon dont les ressources textuelles sont gérées.

6.1.2.2.1 Identification des classes de base produisant les IHM

Dans le cas de Vuze, les développeurs ont utilisé directement les classes de base de SWT (Button, Label, etc.). Cela veut dire qu'ils n'ont pas surchargé leurs propres classes d'IHM qui dérivent des classes de base de SWT.

Pour illustrer cela, prenons l'exemple de la ligne de code de la Figure 78, qui montre la façon dont les boutons sont créés dans Vuze. Il s'agit de la façon la plus classique et la plus simple, puisqu'on utilise directement la classe de base Button de SWT. De même, pour n'importe quel autre composant graphique, les développeurs ont utilisé directement les classes de base de SWT pour fabriquer les interfaces graphiques de Vuze.

Cela implique que nous devons procéder à la localisation interne directement au niveau des classes de base de SWT.

```
final Button config_button = new Button( controls, SWT.NULL );
```

Figure 78 : *Ligne de code de l'application Vuze pour créer un bouton.*

[18] http://java.sun.com/j2se/1.5.0/docs/guide/jni/

La question qui se pose maintenant est de savoir pourquoi nous ne procédons pas directement à la localisation interne des classes Control ou Widget, puisque les classes Button, Label, TabFolder, etc., héritent toutes de la classe Control, qui elle-même hérite de Widget (Figure 77).

Si nous avions choisi de surcharger la classe Widget, nous devrions procéder à la localisation interne du code source de la librairie graphique SWT. Cela est possible que dans le cas où le code source de la bibliothèque graphique est ouvert, ce qui est loin d'être le cas des logiciels commerciaux qui utilisent, dans la plupart des cas, leurs propres bibliothèques graphiques, basées sur des bibliothèques graphiques libres ou non, pour fabriquer leurs interfaces utilisateur.

Visant, la majorité des logiciels libres et commerciaux, nous avons donc choisi de nous placer au niveau des classes de base des IHM utilisées directement par l'application.

6.1.2.2.2 Gestion des ressources textuelles dans Vuze

Dans le cas de Vuze, c'est la classe Messages.java qui gère les ressources textuelles des différents composants graphiques de l'application, et plus particulièrement les deux méthodes setLangageText() et updateLangageFromData() de la classe Messages.java.

La classe Widget dont dérivent tous les composants graphiques de l'application admet un membre DATA de type Object qui permet aux développeurs de stocker des données. Dans notre cas, ce membre est utilisé pour stocker les identifiants des chaînes d'interface.

La méthode setLangageText() (Figure 79) permet de mémoriser l'identifiant d'une chaîne dans le membre DATA, puis elle appelle la méthode UpdateLangageFromData() pour charger la chaîne à partir du fichier de ressources (Figure 80).

```
private static void
setLanguageText(Widget widget, String key, String[] params, boolean setTooltipOnly) {
  widget.setData(key);
  if(!setTooltipOnly)
    updateLanguageFromData(widget, params);
  updateToolTipFromData(widget);
}
```

Figure 79 : *Méthode setLangageText() de la classe Messages.java de Vuze.*

```
1
2   private static void updateLanguageFromData(Widget widget,String[] params) {
3     if (widget == null || widget.isDisposed()) {
4         return;
5     }
6
7       if (widget.getData() != null) {
8         String key = null;
9         try {
10          key = (String) widget.getData();
11        } catch(ClassCastException e) {
12        }
13
14        if(key == null) return;
15        if(key.endsWith(".tooltip")) return;
16
17        String  message;
18
19        if ( params == null ){
20
21            message = MessageText.getString((String) widget.getData());
22        }else{
23
24            message = MessageText.getString((String) widget.getData(), params);
25        }
26
27        String sTransFromXliff = LocaInSituParser.GetTranslation((String) widget.getData());
28        if (!sTransFromXliff.isEmpty())
29            message = sTransFromXliff;
30
31        if (widget instanceof MenuItem) {
32            final MenuItem menuItem = ((MenuItem) widget);
33            boolean indent = (menuItem.getData("IndentItem") != null);
34
35            if(Constants.isOSX)
36                message = HIG_ELLIP_EXP.matcher(message).replaceAll("\u2026"); // hig style - ellipsis
37
38            menuItem.setText(indent ? "   " + message : message);
39
40            if(menuItem.getAccelerator() != 0) // opt-in only for now; remove this conditional check to allow accelerators for arbit
41                KeyBindings.setAccelerator(menuItem, (String)menuItem.getData()); // update keybinding
42        }
43        else if (widget instanceof TableColumn) {
44            TableColumn tc = ((TableColumn) widget);
45          tc.setText(message);
46        } else if (widget instanceof Label)
47            // Disable Mnemonic when & is before a space.  Otherwise, it's most
48            // likely meant to be a Mnemonic
49          ((Label) widget).setText(message.replaceAll("& ", "&& "));
50        else if (widget instanceof CLabel)
51          ((CLabel) widget).setText(message.replaceAll("& ", "&& "));
52        else if (widget instanceof Group)
53            ((Group) widget).setText(message);
54        else if (widget instanceof Button)
55            ((Button) widget).setText(message);
56        else if (widget instanceof CTabItem)
57            ((CTabItem) widget).setText(message);
58        else if (widget instanceof TabItem)
59            ((TabItem) widget).setText(message);
60        else if (widget instanceof TreeItem)
61          ((TreeItem) widget).setText(message);
62        else if(widget instanceof Shell)
63          ((Shell) widget).setText(message);
64        else if(widget instanceof ToolItem)
65            ((ToolItem) widget).setText(message);
66        else if(widget instanceof Text)
67            ((Text) widget).setText(message);
68        else{
69          Debug.out( "No cast for " + widget.getClass().getName());
70        }
71      }
72  }
```

Figure 80 : *Méthode updateLangageFromData() de la classe Messages.java de Vuze*

6.1.2.3 *Intégration du nouveau comportement adapté à la localisation en contexte au niveau des classes de base des IHM*

Comme nous l'avons mentionné, dans le cas de Vuze, les développeurs ont utilisé directement les classes Button, Label, etc. de SWT. Ainsi, nous avons intégré le nouveau comportement de localisation en contexte au niveau de ces classes de base.

Nous rajoutons dans un premier temps le comportement de mise en édition des éléments textuels appartenant aux différentes boîtes de dialogue. Pour cela, nous interceptons au niveau des classes Button, Label, Canvas, etc. le clic-droit sur les éléments textuels (Figure 81).

```
1  package LocaInContextBaseClassVuze;
2
3  import locaInContext.LocaInContextMenuDetectListener;
4
5  import org.eclipse.swt.widgets.Button;
6  import org.eclipse.swt.widgets.Composite;
7
8  public class LocaInContextButton extends Button {
9
10     public LocaInContextButton(Composite parent, int style) {
11         super(parent, style);
12         this.addMenuDetectListener(new LocaInContextMenuDetectListener());
13     }
14
15     @Override
16     protected void checkSubclass() {
17         // TODO Auto-generated method stub
18         //super.checkSubclass();
19     }
20 }
```

Figure 81 : *Intégration d'un listner dans le constructeur de la classe LocaInContextButton qui dérive de la classe de base Button.*

Le listener interceptant l'événement correspondant a accès à l'objet que l'on souhaite mettre en édition. Nous avons alors accès grâce à son membre DATA à l'identifiant du composant graphique.

Comme exposé plus haut, c'est la méthode UpdateLangageFromData() qui se charge d'affecter une valeur textuelle à tout contrôle de l'application. Nous avons donc modifié cette méthode pour qu'elle vérifie via l'identifiant de la chaîne s'il existe de nouvelles propositions de traduction dans le fichier XLIFF (Figure 82). Si c'est le cas, la méthode affecte au contrôle en question la nouvelle traduction. Sinon, elle se contente d'affecter la valeur définie dans le fichier de ressources initiale de l'application (RessourceBundle_fr_FR.properties dans notre cas).

```
private static void updateLanguageFromData(Widget widget,String[] params) {
    if (widget == null || widget.isDisposed()) {
        return;
    }

      if (widget.getData() != null) {
        String key = null;
        try {
          key = (String) widget.getData();
        } catch(ClassCastException e) {
        }

        if(key == null) return;
        if(key.endsWith(".tooltip")) return;

        String  message;

        if ( params == null ){

            message = MessageText.getString((String) widget.getData());
        }else{

            message = MessageText.getString((String) widget.getData(), params);
        }

        String sTransFromXliff = LocaInSituParser.GetTranslation((String) widget.getData
        if (!sTransFromXliff.isEmpty())
            message = sTransFromXliff;

        if (widget instanceof MenuItem) {
            final MenuItem menuItem = ((MenuItem) widget);
            boolean indent = (menuItem.getData("IndentItem") != null);

            if(Constants.isOSX)
                message = HIG_ELLIP_EXP.matcher(message).replaceAll("\u2026"); // hig sty

            menuItem.setText(indent ? "   " + message : message);

            if(menuItem.getAccelerator() != 0) // opt-in only for now; remove this condi
                KeyBindings.setAccelerator(menuItem, (String)menuItem.getData()); // upd
        }
        else if (widget instanceof TableColumn) {
            TableColumn tc = ((TableColumn) widget);
          tc.setText(message);
        } else if (widget instanceof Label)
            // Disable Mnemonic when & is before a space.  Otherwise, it's most
            // likely meant to be a Mnemonic
          ((Label) widget).setText(message.replaceAll("& ", "&& "));
        else if (widget instanceof CLabel)
          ((CLabel) widget).setText(message.replaceAll("& ", "&& "));
        else if (widget instanceof Group)
           ((Group) widget).setText(message);
        else if (widget instanceof Button)
           ((Button) widget).setText(message);
        else if (widget instanceof CTabItem)
           ((CTabItem) widget).setText(message);
        else if (widget instanceof TabItem)
           ((TabItem) widget).setText(message);
        else if (widget instanceof TreeItem)
          ((TreeItem) widget).setText(message);
        else if(widget instanceof Shell)
          ((Shell) widget).setText(message);
        else if(widget instanceof ToolItem)
            ((ToolItem) widget).setText(message);
        else if(widget instanceof Text)
            ((Text) widget).setText(message);
        else{
          Debug.out( "No cast for " + widget.getClass().getName());
        }
      }
  }
```

Figure 82 : *Méthode updateLangageFromData () de la classe Messages.java après intégration de nos modifications*

Sans autre intervention de notre part, nos modifications sur la classe UpdateLangageFromData() ne permettent la mise à jour des chaînes éditées que lors des prochaines réouvertures des boîtes de dialogues correspondantes.

Pour permettre la mise à jour en temps réel des chaînes, sans être obligé de fermer et réouvrir la boîte de dialogue, nous faisons appel à la méthode setLangageText() qui permet de rafraîchir le contrôle venant d'être édité.

6.2 Bilan des expériences réalisées

6.2.1 Bilan sur Vuze

Nous étudions ici les résultats de la localisation interne du code source de Vuze en termes de nombre de chaînes qui ont hérité du nouveau comportement de localisation en contexte et qui sont donc devenues localisables en contexte puis en termes de coût de réalisation.

6.2.1.1 Bilan quantitatif : nombre de chaîne localisables en contexte

Une fois la localisation interne terminée, nous avons testé la localisation en contexte des différentes interfaces graphiques de Vuze. Nous avons constaté que l'application est devenue entièrement localisable en contexte. En effet, toutes les chaînes d'interfaces de Vuze, soit 1895 chaînes, ont hérité du nouveau comportement de localisation en contexte.

Certaines interfaces utilisateur sont difficiles à faire apparaître, pour s'assurer donc que nous avons testé toutes les interfaces utilisateur nous avons utilisé des outils qui permettent de défiler toutes les interfaces graphiques d'un logiciel comme Routinebot[19] ou Ranorex[20]. Ce type d'outils est utilisé généralement par les Bêta-testeurs pour tester toutes les interfaces graphiques d'un logiciel avant sa commercialisation. Cela nous a permis de vérifier que tous les éléments textuels de l'application ont bien hérité du comportement de la localisation en contexte.

La Figure 83 montre par exemple, la localisation en contexte de la chaîne « Ajouter Fichiers » de la boîte de dialogue « Ouvrir le(s) torrent(s) » de l'application Vuze du français vers le vietnamien.

[19] http://www.routinebot.com/
[20] http://www.ranorex.com/

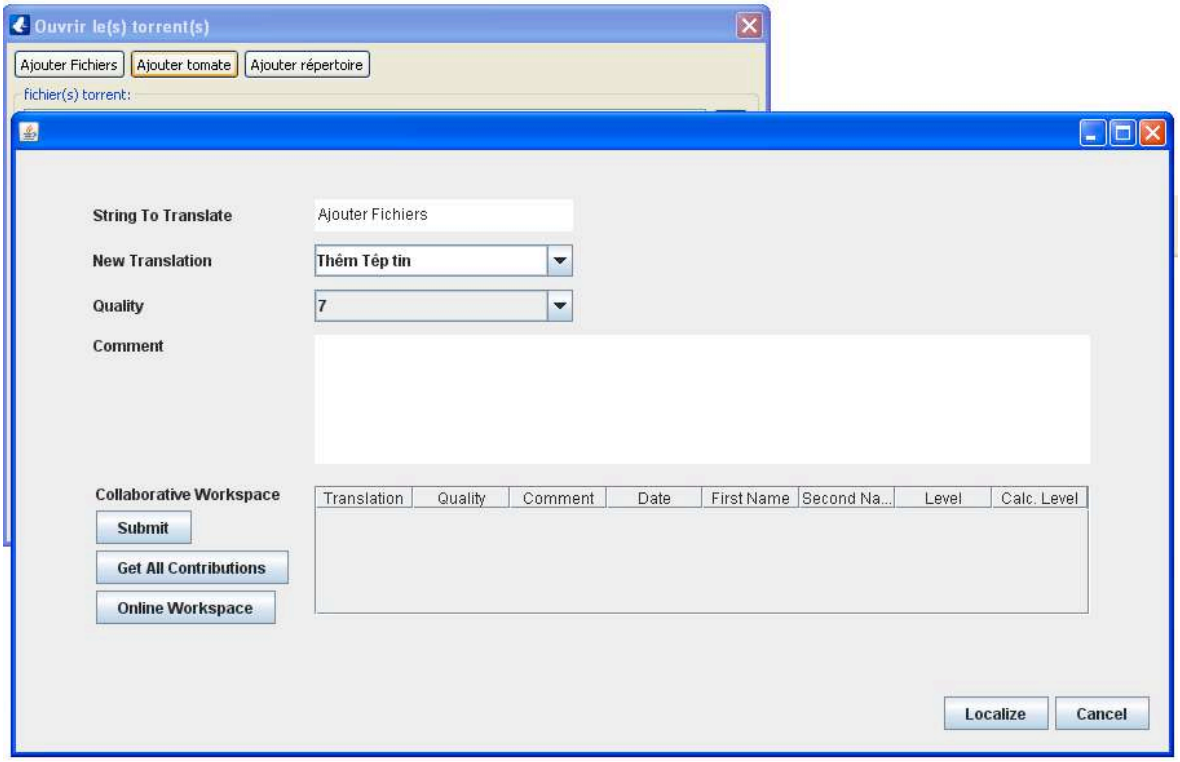


Figure 83 : *Localisation en contexte de la chaîne "Ajouter Fichiers" du français vers le vietnamien*

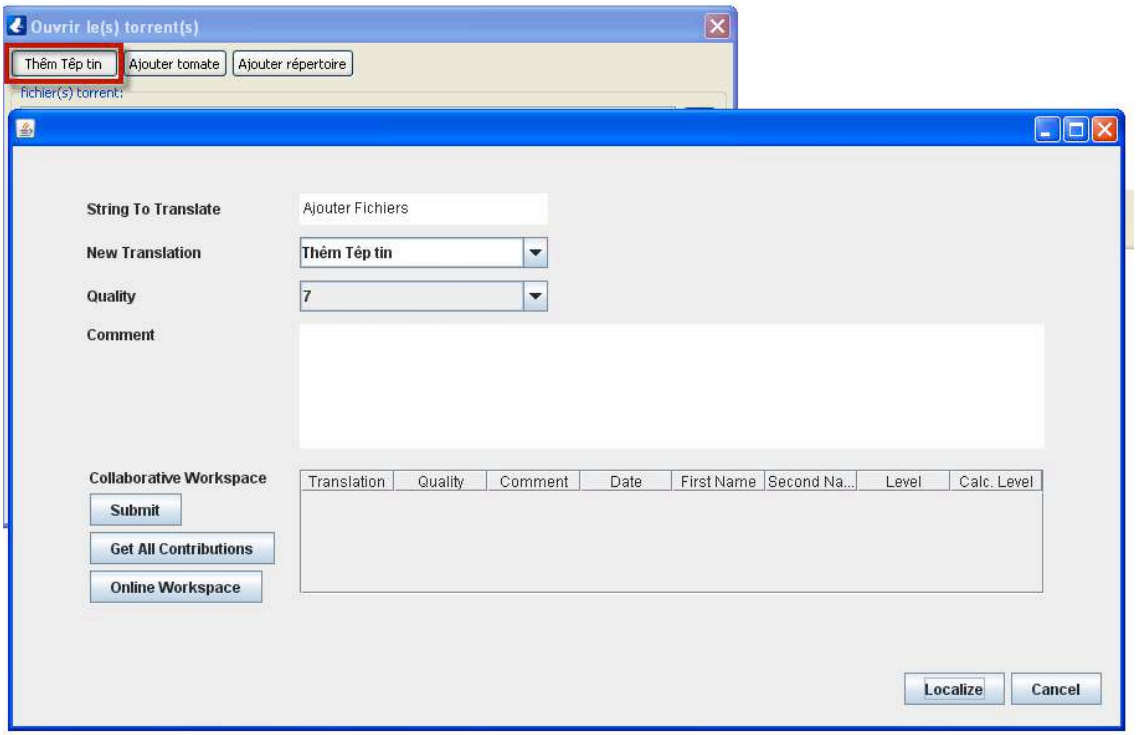


Figure 84 : *La chaîne "Ajouter Fichiers" est remplacée en temps réel par la nouvelle proposition de traduction "Thêm Tệp tin"*

Nous sommes en train de localiser l'application Vuze en entier de l'anglais vers le vietnamien. Nous avons choisi le vietnamien comme langue cible parce que, d'une part, il n'existe pas de version localisée de Vuze en vietnamien et que, d'autre part, nous avons dans notre équipe de recherche (GETALP-LIG) plusieurs personnes maîtrisant le vietnamien.

6.2.1.2 Bilan temporel

Nous présentons ici le coût de la localisation interne de Vuze en termes du temps. Vuze compte 80.000 lignes de code. Pour le rendre localisable en contexte, la localisation interne du son code source nous pris en total environ 28h. En effet, l'étude de l'application pour comprendre l'architecture logicielle et plus particulièrement l'architecture graphique afin d'identifier les classes génériques des IHM nous a pris environ 16h. À la fin de cette étude nous avons donc pu identifier 7 classes de base des IHM. La localisation interne de ces classes nous a pris 12h.

6.2.2 Bilan des deux expériences sur Vuze et Notepad-plus-plus

Nous présentons dans ce paragraphe une étude comparative des délais de réalisation de la localisation interne de Notepad-plus-plus et de Vuze.

Les critères qui sont pris en compte dans notre étude comparative sont la taille de l'application à localiser en termes de nombre de ligne de code, et la documentation développeur qui peut nous aider à comprendre le code source plus vite, ainsi que le nombre de classes à surcharger.

	Vuze	Notepad-plus-plus
Taille (lignes de code)	80.000	60.000
Documentation développeur	Pas de documentation	Pas de documentation
Nombre de classes surchargées	**7**	2
Durée de l'étude de l'application	16h	28h
Durée de la localisation interne du code source	12h	25h
Durée total	28h	53h

Figure 85 : *Tableau comparatif illustrant les délais de localisation interne de Vuze et Notepad-plus-plus*

Comme le montre la Figure 85, les deux applications Notepad-plus-plus et Vuze ne disposent pas d'une documentation développeur. Cela ne facilite pas l'étude et la compréhension du code source.

Bien que les deux applications soient à peu près de même taille, cette étape nous a pris environ 28h pour Notepad-plus-plus contre 16h uniquement pour Vuze. Le délai a donc été divisé presque par deux. Cela s'explique par le fait que, lors de la deuxième expérimentation, nous avions acquis un certain savoir-faire nous permettant d'identifier plus rapidement les classes de base des IHM d'une application dont nous ne sommes pas propriétaire du code source.

Le temps de cette étape peut être énormément raccourci dans le cas où nous disposons d'une documentation développeur ou dans le cas ou nous collaborons directement avec les développeurs de l'application.

La deuxième étape est celle de la localisation interne des classes génériques des IHM identifiées dans la première étape. Dans le cas de Vuze, nous avons mis environ 12h, contre 25h pour Notepad-plus-plus. Ici, bien que le nombre de classes de base surchargées de Vuze soit plus grand que celui de Notepad-plus-plus, nous avons aussi divisé le temps de réalisation par deux, (7 classes dans le cas de Vuze et 2 classes dans le cas de Notepad-plus-plus).

Ainsi, nous avons mis au total 28h pour la localisation interne de Vuze, contre 53h pour Notepad-plus-plus.

Conclusion

Nous avons présenté dans ce chapitre une expérimentation complémentaire de notre méthode de localisation interne et en contexte sur un deuxième logiciel libre. L'expérimentation exposée dans ce chapitre permet donc de valider la faisabilité de notre méthode de localisation interne sur des logiciels écrits en Java, comme Vuze. Notre approche est donc maintenant valide sur des logiciels écrits en Java et en C++, qui sont les deux langages les plus utilisés pour la réalisation de logiciels. Notre approche possède donc la généricité recherchée.

Partie 3 : Localisation en contexte dans un scénario collaboratif

La mise en œuvre de notre solution de localisation en contexte nécessite l'intégration d'un gestionnaire de flots de traductions afin de (1) gérer les propositions de traduction à provenances multiples et de (2) permettre aux différents contributeurs de collaborer ensemble.

Le but de cette partie et de mettre en œuvre notre solution de localisation en contexte dans un scénario collaboratif. Nous modélisons ce scénario collaboratif sous forme d'un processus de localisation tripartite faisant intervenir trois entités (le donneur d'ordres, les contributeurs, le site de gestionnaires de flots de traductions).

Nous présentons dans le premier chapitre les trois entités de ce scénario collaboratif. Puis nous présentons dans le deuxième chapitre les différentes interactions entres ces trois entités durant le processus de localisation. Enfin nous présentons une expérimentation de ce processus de localisation tripartite sur le logiciel Notepad-plus-plus.

Chapitre 7 : Nouveau processus de localisation tripartite

Introduction

Dans ce chapitre, nous présentons le processus de localisation en contexte sous forme d'un processus triparties[21] avec trois entités : contributeurs (utilisateurs finals, traducteurs bénévoles, β-testeurs, etc.), donneur d'ordres (l'éditeur de logiciel dans le cas des logiciels commerciaux ou la communauté d'administrateurs dans le cas des logiciels libres) et le site de gestion de flots de traductions « SECTra_w » (Huynh et al., 2008). Nous décrirons chaque acteur, en précisant son rôle dans le processus de localisation.

7.1 Le donneur d'ordres

Le donneur d'ordres est celui qui annonce le début du projet de localisation du logiciel et gère le déroulement du processus de localisation du début jusqu'à la fin. Dans le cas des logiciels commerciaux, il s'agit de l'éditeur du logiciel lui-même (c'est souvent le cas des petits éditeurs de logiciels) ou des sociétés de service de localisation (ce qui est le cas de WinSoft qui se charge de la localisation des logiciels d'Adobe et de FileMaker). Dans le cas des logiciels libres, il s'agit de la communauté d'administrateurs ou des responsables du projet de localisation. Par exemple, dans le cas de Mozilla, c'est "Mozilla Localization Project Staff" (Mozilla, 2006) qui annonce et gère le processus de localisation.

Avant le début d'un projet de localisation, le donneur d'ordres doit préparer les données sur lesquelles les contributeurs vont travailler. Dans le cas des logiciels commerciaux, le donneur d'ordres prépare les glossaires qu'il doit communiquer aux traducteurs et réviseurs professionnels. Dans le cas des logiciels libres comme Ubuntu, les responsables du projet de localisation mettent à disposition des contributeurs toutes les données qui doivent être localisées sur la plate-forme "Launchpad Translators" (Canonical Ltd, 2009) (Figure 44 du chapitre 3).

Une fois le processus de localisation démarré, le donneur d'ordres doit suivre son évolution. Chaque donneur d'ordres définit sa politique de validation qui lui permet de valider ou non les propositions de traduction des contributeurs. Par exemple, pour valider les traductions faites par les traducteurs professionnels, WinSoft a recours à des réviseurs professionnels qui sont censés avoir une meilleure connaissance du contexte d'utilisation des éléments textuels.

Nous présenterons donc dans la suite de cette section, les données qui doivent être fournies par le donneur d'ordres dans le cadre de notre approche de localisation en contexte. Nous exposerons ensuite la politique de validation et de labellisation des traductions que nous avons adoptée pour contrôler les différentes propositions de traduction des contributeurs.

7.1.1 Données à fournir

Nous présentons ici les données indispensables au processus de localisation en contexte. Elles doivent être fournies par le donneur d'ordres, et contenir toutes les ressources textuelles du logiciel à localiser (documentation technique, aide en ligne, éléments textuels

[21] Tripartite (*Adj.)* : composé de trois parties, trois éléments.

des interfaces utilisateur, etc.). Dans notre cas, nous mettons l'accent sur les données relatives aux éléments textuels des interfaces graphiques (étant donné que nous nous intéressons à la localisation des interfaces graphiques). Comme nous l'avons montré dans le premier chapitre, les données de ce type sont stockées en général dans des fichiers appelés glossaires. Les glossaires peuvent être accompagnés dans certains cas de bases terminologiques.

7.1.1.1 Glossaires

Comme expliqué auparavant, dans le domaine de la localisation des logiciels, un glossaire désigne l'ensemble des éléments textuels des interfaces utilisateur. Selon les éditeurs de logiciels, les glossaires peuvent avoir différents formats. Le format le plus utilisé est Excel. Généralement, les localiseurs (éditeurs de logiciels ou sociétés spécialisées dans la localisation) construisent un glossaire par langue cible et par produit. La Figure 86 montre l'ensemble des glossaires du logiciel Adobe Acrobat dans 32 langues.

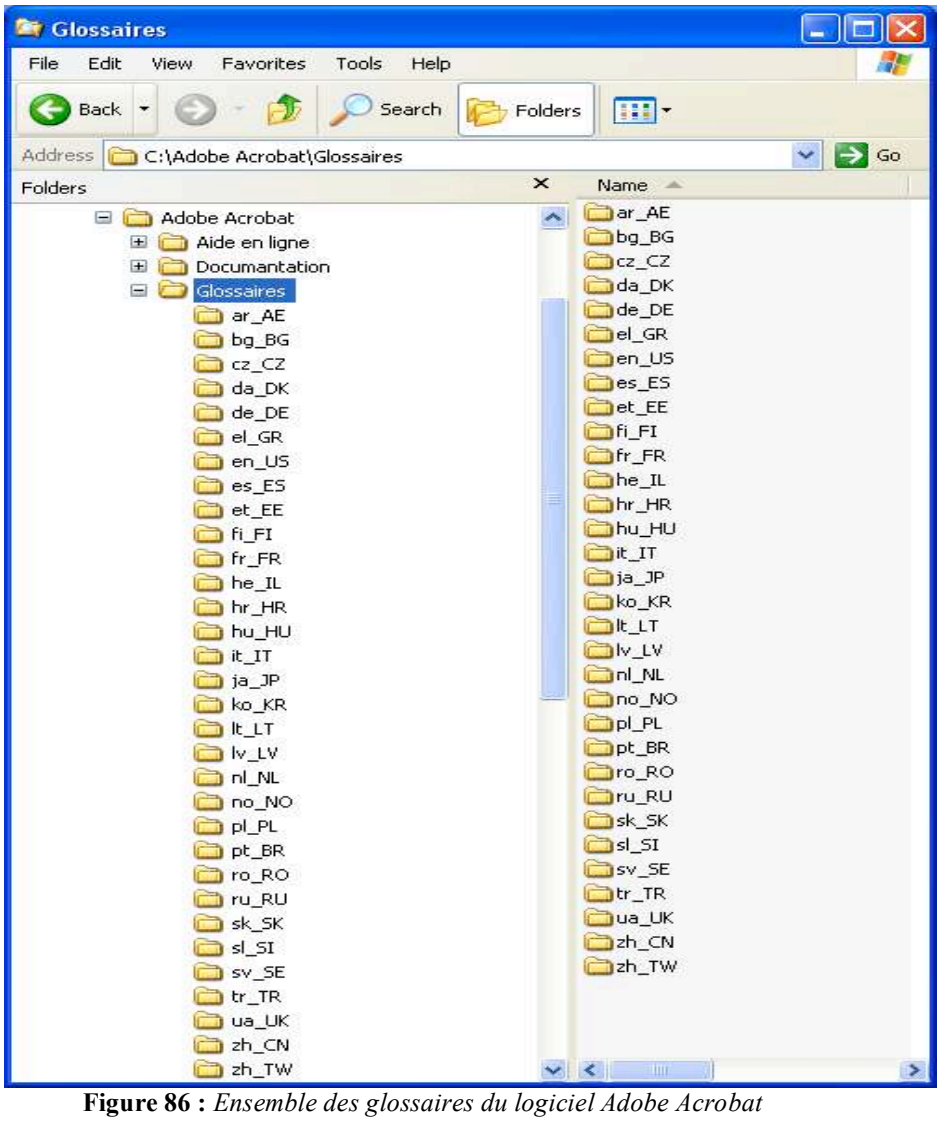


Figure 86 : *Ensemble des glossaires du logiciel Adobe Acrobat*

7.1.1.2 Base terminologique

Le terme base terminologique est aussi emprunté au domaine de la terminologie, mais il est loin de désigner la même chose que dans ce domaine. En effet, dans le domaine de la localisation de logiciel, une base terminologique désigne un fichier qui peut avoir plusieurs formats et qui contient tous les termes qui ont été validés auparavant par le donneur d'ordres et qui ne doivent pas changer dans les nouvelles versions du logiciel à localiser. Cela permet d'assurer une certaine cohérence de traduction entre les différentes versions d'un même logiciel. En particulier, dans le cas de WinSoft, il s'agit de la base MultiTerm qui est sous la forme d'un fichier Excel et qui contient les termes de base pour chaque produit Adobe dans chaque langue cible. Elle contient par exemple 5000 termes pour le logiciel Adobe Acrobat en arabe. Cette base terminologique peut être utilisée par le donneur d'ordres pour *prétraduire* les termes de base d'une nouvelle version du logiciel. Certains éditeurs de logiciels ou sociétés de service de localisation partagent cette base terminologique avec les traducteurs et les réviseurs professionnels.

7.1.1.3 Documentation technique et aide en ligne

La documentation qui accompagne le logiciel doit être aussi mise à disposition des contributeurs sous des formats éditables (HTML, TXT, etc.). Toutefois, comme nous nous intéressons uniquement à la localisation en contexte des éléments textuels des interfaces graphiques, nous n'avons pas mis à disposition des contributeurs les documents qui peuvent accompagner les logiciels.

7.1.2 Politique de validation et de labellisation des propositions de traduction faites par les contributeurs

Le principe de notre approche de localisation en contexte est de permettre à plusieurs types de contributeurs (utilisateurs finals, bêta-testeurs, traducteurs bénévoles, etc.) de participer au processus de localisation, de façon efficace et dynamique, en traduisant certaines chaînes des interfaces graphiques, durant l'utilisation du logiciel. Ainsi, cela implique qu'une seule chaîne source peut avoir plusieurs propositions de traduction proposées par différents participant au processus de localisation. Le donneur d'ordres doit donc adopter une politique de validation et de labellisation pour gérer toutes les propositions de traduction à provenances multiples.

7.1.2.1 Validation des propositions de traduction des différents contributeurs

Le donneur d'ordres ne maîtrisant pas toutes les langues cible doit avoir une politique de validation adaptée à l'approche de localisation en contexte afin de valider les différentes propositions de traduction. Nous proposons dans la suite les différentes solutions envisageables que le donneur d'ordres peut adopter. Toutefois, le donneur d'ordres peut toujours faire appel à un réviseur professionnel pour valider les différentes propositions de traduction proposées.

7.1.2.1.1 Cas des propositions de traduction convergeant vers une seule traduction

Dans le cas où les différentes propositions de traduction convergent vers une seule traduction, la validation est simple. Il s'agit de valider celle qui a été proposée par le plus grand nombre de contributeurs.

7.1.2.1.2 Cas des propositions de traduction divergentes

Dans le cas où les propositions de traduction ne convergent pas, le donneur d'ordres, ne connaissant pas la langue cible, peut choisir l'une des politiques de validation suivantes :

- *Validation par des réviseurs professionnels :* faire appel à un réviseur professionnel pour valider certaines propositions de traduction.
- *Validation automatique en se basant sur un système d'apprentissage :* construire un système d'apprentissage qui lui permettra d'effectuer des validations automatiques. Bien entendu, ce système doit se baser sur le profil utilisateur (langue maternelle, domaine professionnel, score utilisateur qui est calculé en fonction du nombre de traductions proposées et validées par l'utilisateur, etc.).
- *Validation par un système de vote :* soumettre les différentes propositions de traduction à un système de vote et valider la traduction élue.

7.1.2.2 Étiquette d'évaluation qualitative des propositions de traductions

Le donneur d'ordres peut associer à chaque traduction une étiquette indiquant son statut de validation. Les étiquettes seront visibles au contributeur au moment de la localisation en contexte de l'application. L'intérêt de ces étiquettes est de donner une idée à l'utilisateur sur la façon dont l'élément textuel a été validé. Nous adoptons comme valeur d'étiquette les *niveaux* de SECTra_w (Huynh et al., 2008):

***** 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), ni post-éditée ni certifiée.

7.2 Le site de la communauté des contributeurs à la localisation en contexte

La mise en place d'une telle approche de localisation a nécessité l'intégration d'un gestionnaire de flot de traductions (Albatal, 2005). Pour cela nous avons effectué un état de l'art sur les outils existants afin de choisir le mieux adapté à notre situation.

7.2.1 Étude et recherche d'outil

L'outil que nous recherchons doit satisfaire la liste des critères suivants :

- Il doit être un outil collaboratif permettant l'import, l'export et le partage des données multilingues,
- il doit permettre l'édition des contenus multilingues,
- il doit permettre l'évaluation et la validation des propositions de traduction
- il doit contenir des outils d'aide à la traduction (appel à des systèmes de traduction automatique, mémoire de traductions, aide dictionnairique, etc.)

L'état de l'art nous a permis d'identifier deux environnements candidats : « BeyTrans » (Bey, 2007) et « SECTra_w » (Huynh, 2009).

7.2.1.1 L'outil BeyTrans

« BeyTrans » "Better Environnement For Your Translation" (Bey, 2006a, 2006b, 2007, 2008a, 2008b), est une plate-forme en ligne permettant la traduction collaborative et incrémentale des documents par des traducteurs/utilisateurs bénévoles. Ces derniers peuvent charger et partager des documents avec d'autres traducteurs en mode collaboratif (Figure 87). L'environnement est ouvert à tous les traducteurs bénévoles en ligne. Tout traducteur/utilisateur intéressé par la traduction peut donc charger et partager des documents avec d'autres traducteurs en mode collaboratif.

Avant de commencer une traduction, le document source doit être préparé par le système. Cela dit, le texte source est extrait et segmenté en plusieurs unités de traduction. En effet, l'import d'un document à traduire crée une structure interne TMX[22] aidant à stocker efficacement les unités de traduction sources et cibles. Pour réaliser les traductions, les traducteurs peuvent appliquer sur les unités de traduction les fonctionnalités suivantes :

- Appel de systèmes de TA : les traducteurs ont le choix entre plusieurs systèmes de TA gratuits pour avoir une première version (*prétraduction*) (Systran, Reverso, etc.). Les *prétraduction*s des unités de traduction seront par la suite post-éditées et stockées dans une structure XML spéciale.
- Appel des MT : une fois trouvés, les segments détectés similaires à la source sont affichés, et leurs traductions sont proposées dans la zone cible. Cette méthode est plus efficace que la précédente si la similarité est élevée, parce que cela augmente la productivité et diminue le temps de traduction plus que si l'on part d'un résultat de TA « Web ».

Les résultats de traduction des unités de traduction sont synchronisés avec les unités de traduction source. Les traducteurs peuvent basculer du mode lecture au mode édition. L'éditeur de traductions multilingue offre une interface en ligne conçue à la « Excel », où toutes les unités de traduction source et cible sont synchronisées et visualisées en parallèle (Figure 88). Ainsi, il est possible d'exploiter le contenu des ressources linguistiques (dictionnaires, mémoires de traduction, etc.) par des fonctionnalités diverses de consultation.

Les fonctionnalités d'aide linguistique sont couplées aux événements déclenchés lors des manipulations des unités de traduction source via l'interface de traduction. Les fonctionnalités usuelles d'édition sont aussi disponibles : les suppressions, la fusion, l'ajout et la division des unités de traduction (des lignes dans l'interface) sont possibles. Un traducteur peut ajuster la segmentation par création de nouvelles unités de traduction ou par fusion de deux segments.

[22] http://www.lisa.org/fileadmin/standards/tmx1.4/tmx.htm

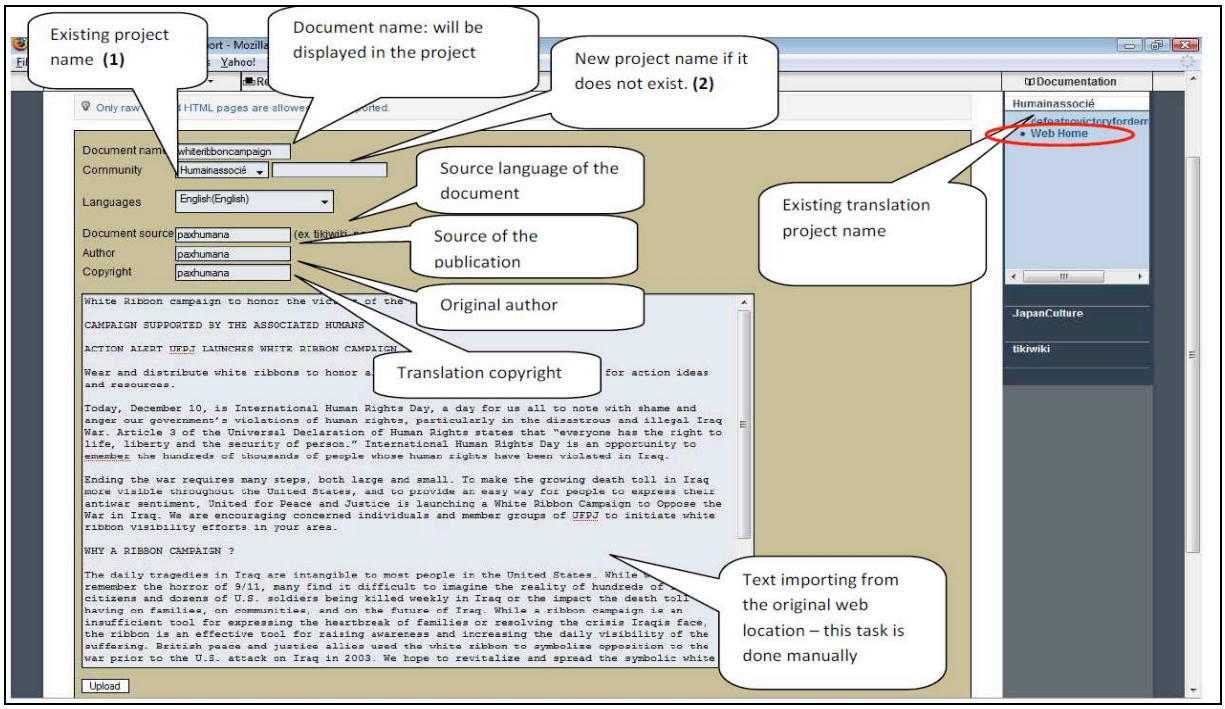


***Figure 87** : Interface d'import des documents de l'outil BeyTrans version 2.0*

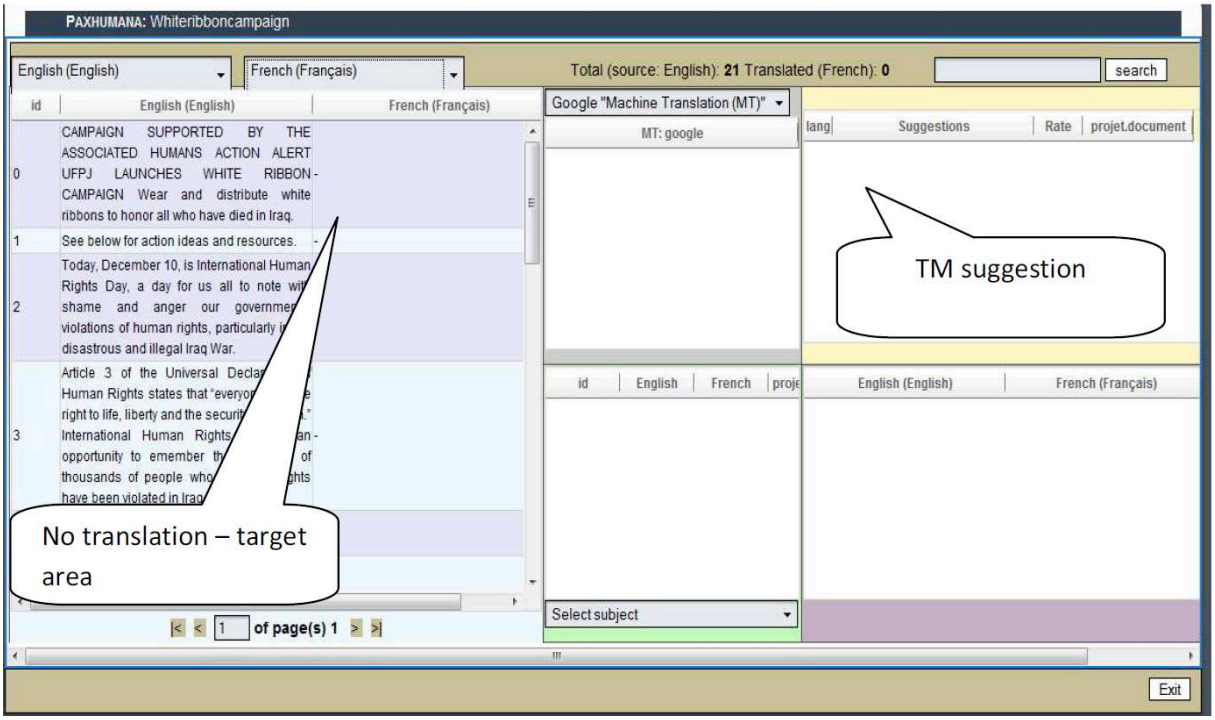


Figure 88 : *Interface de post-édition de l'outil BeyTrans version 2.0*

Toutefois, nous n'avons pas pu utiliser cet outil, car au moment de notre étude la version BeyTrans n'était pas encore assez stable.

7.2.1.2 *L'outil SECTra_w*

« SECTra_w » (Huynh et al., 2008) est le deuxième outil étudié, il s'agit d'un service Web qui vise de façon générale à permettre l'exploitation collaborative sur le Web de corpus de traductions multilingues, multiannotés et multimédias (Blanchon et al., 2009). Il contient de nombreuses fonctionnalités aidant considérablement la post-édition collaborative en ligne de documents (division de chaque document en « pages logiques », aides traductionnelles et dictionnairiques proactives, interface adaptée, vues parallèles synchronisées et cliquables pour aller dans l'interface de post-édition, etc.).

SECTra_w peut charger un document ou un ensemble de documents, les découper en « segments » (phrases ou titres) qu'il stocke dans une mémoire de traductions initialisée avec les *prétraductions* produites par des systèmes de TA tels que Reverso, Systran ou Google-Translate. L'interface de post-édition est une page Web (Figure 89) : les segments sont présentés les uns à la suite des autres verticalement dans une colonne (une quinzaine, ce qui correspond à une "page" de traduction), et associés horizontalement aux contenus d'autres colonnes. Une autre colonne (colonne de "propositions" de traductions) présente les résultats de traduction automatique par chaque système utilisé. Enfin, pour chaque segment, la colonne de post-édition contient une cellule éditable initialisée avec le contenu de la cellule correspondant à la « meilleure » *prétraduction* disponible.

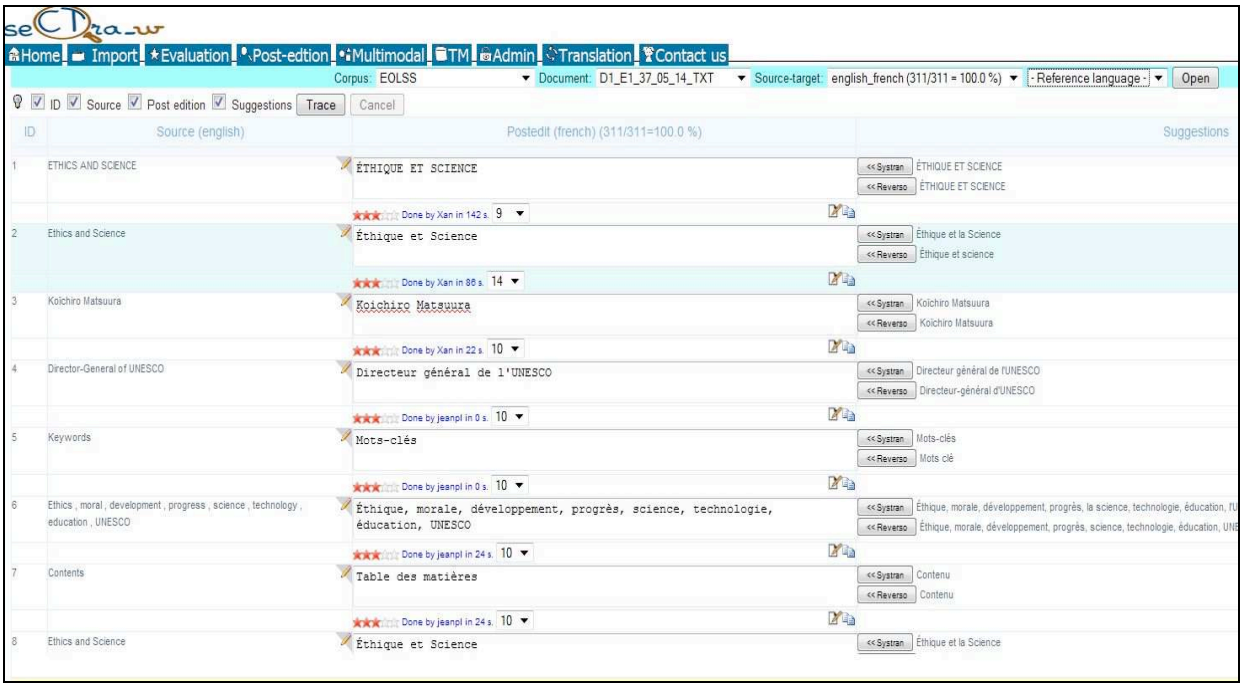


Figure 89 : *Interface de post-édition de SECTra_w*

Nous avons donc fait le choix de l'outil SECTra_w puisqu'il correspond parfaitement à nos besoins et répond à tous nos critères.

7.2.2 Choix de SECTra_w : Système d'Exploitation de Corpus de Traductions sur le Web

SECTra_w peut charger un document (Excel, Txt, Html, Xml) ou un ensemble de documents (ce qui convient parfaitement aux cas simples des éléments textuels des interfaces graphiques qui sont stockés généralement dans des glossaires sous format Excel), une fois découpé en « segments ». Dans notre cas, chaque ligne du glossaire correspond à un segment

source. Ils sont donc stockés directement dans la mémoire de traductions initialisée avec les *prétraductions* produites par des systèmes de TA.

7.2.2.1 Utilisation de SECTra_w pour la localisation

Les différentes étapes nécessaires à l'utilisation de SECTra_w dans un projet de localisation sont la création d'un projet, la configuration et l'import des données.

7.2.2.1.1 Création d'un nouveau projet de localisation

Avant de démarrer le projet de localisation, le donneur d'ordres doit créer un nouveau projet de localisation sous SECTra_w (Figure 90).

Figure 90 : *Interface de SECTra_w permettant la création d'un nouveau projet de localisation*

Pour pouvoir créer un nouveau projet sur SECTra_w, le donneur d'ordres doit être enregistré en tant qu'administrateur.

7.2.2.1.2 Configuration et définition des profils utilisateurs

Lorsque le projet est créé, le donneur d'ordres peut définir les profils des contributeurs qui vont participer à la localisation (Figure 91), par exemple, leur niveau d'expertise en traduction (débutant, intermédiaire, professionnel), leurs droits, etc.

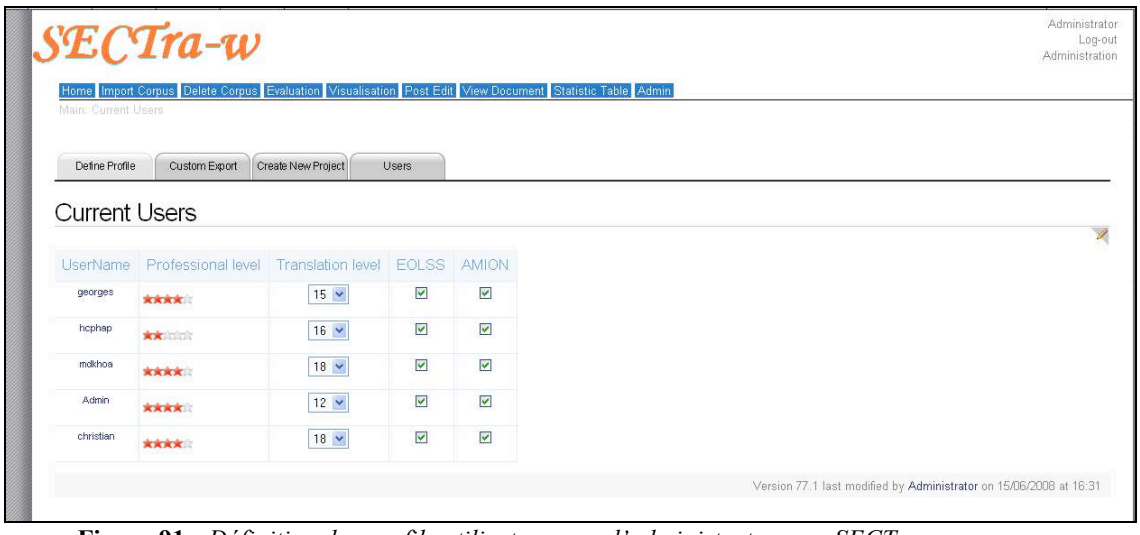


Figure 91 : *Définition des profils utilisateurs par l'administrateur sur SECTra_w*

Comme le montre la Figure 91, l'administrateur du projet peut attribuer, par défaut, à chaque utilisateur : le niveau d'expertise en traduction (de 3 à 4 étoiles), un score par défaut (de 0 à 20) ainsi que les projets auxquels il peut participer. Le score de traduction est attribué par défaut à tout segment post-édité par un contributeur, et peut être modifié par l'utilisateur à tout moment.

Prenons l'exemple de la Figure 91, l'administrateur a affecté à l'utilisateur « George » le score 15, cela veut dire que tous les segments source qui seront traduits par George auront 15 comme score par défaut. Toutefois, ce score peut être modifié par l'utilisateur. Par exemple, l'utilisateur peut se dire « *j'ai fait un bon travail sur ce segment* » et il se met 17/20, ou il se dit « *je ne suis pas sûr de ma proposition de traduction* » et il se met 10/20.

7.2.2.1.3 Import des données dans SECTra_w

Pour pouvoir importer des données dans SECTra_w, il faut qu'elles soient dans l'un des formats gérés par l'outil. Voyons maintenant la structure des corpus de données ainsi que les formats des fichiers gérés par SECTra_w.

7.2.2.1.3.1 Structure du corpus

Un corpus est formé de plusieurs documents. À chaque document est associé nécessairement un fichier source et optionnellement un fichier cible qui contient les *prétraductions* faites par des systèmes de TA (Figure 92).

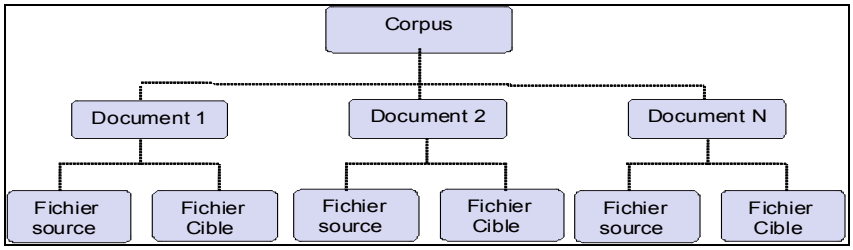


Figure 92 : *Structure du corpus sur SECTra_w*

7.2.2.1.3.2 Format du corpus

SECTra_w gère trois formats : Txt, Xml, ou Excel.

- *Le format Txt :* le contenu des fichiers est une liste d'association identifiant_chaîne = valeur_chaîne avec une association par ligne. L'identifiant_chaîne permet d'identifier chaque chaîne de caractères dans le fichier source (Figure 93).

Source file

Id1: source1
Id2: source2
Id3: source3
Id4: source4
Id5: source5
……

Figure 93 : *Contenu du fichier source sous format TXT*

- *Le format Excel :* le fichier contient deux colonnes, une colonne pour les identifiants et une colonne pour les chaînes source (Figure 94).

	A	B
1	ID	En
2	E-290.9-HE01-023/V-1/0013/01	An unpleasant path for a carriage, to the south of Korla
3	E-290.9-HE01-023/V-1/0015/01	An old East Turkish man in Schinnega
4	E-290.9-HE01-023/V-1/0016/01	Cool shade in Schinnega
5	E-290.9-HE01-023/V-1/0025/01	Orderly placed canoes in Kontje
6	E-290.9-HE01-023/V-1/0026/01	A fleet of canoes on the bank of Kontje-daria
7	E-290.9-HE01-023/V-1/0031/01	Lunch at kontje-daria. The author, Chen and Kung.
8	E-290.9-HE01-023/V-1/0032/01	Loading a canoe, Kontje-daria
9	E-290.9-HE01-023/V-1/0037/01	An old poplar tree with a blue heron nest, Kontje-daria
10	E-290.9-HE01-023/V-1/0038/01	A cooking unit of kontje-daria. Li, Chia Kwei, Gagarin and two rowers
11	E-290.9-HE01-023/V-1/0051/01	A wild boar with its cubs
12	E-290.9-HE01-023/V-1/0052/01	A fleet gliding down Kontje-daria
13	E-290.9-HE01-023/V-1/0054/01	No. 56 camp of April 8
14	E-290.9-HE01-023/V-1/0057/01	Overlooking Kontje-daria from Sai-tjeke
15	E-290.9-HE01-023/V-1/0058/01	A 76-year-old Muslim shepherd from Ak-basch
16	E-290.9-HE01-023/V-1/0061/01	Khodai Kullu
17	E-290.9-HE01-023/V-1/0062/01	Departure from Sai-tjeke
18	E-290.9-HE01-023/V-1/0067/01	Patients in a Hummel tent. Dilpar.
19	E-290.9-HE01-023/V-1/0068/01	The author in his own double canoe
20	E-290.9-HE01-023/V-1/0069/01	Sattma, a hut made of reed, at Dilpar

Figure 94 : *Fichier source sous format Excel*

- *Le format Xml :* chaque segment est représenté entre deux balise XML , le contenu du segment est le segment à traduire, et l'attribut "id" contient l'identifiant du segment (Figure 95).

```
<segment id="ID"> segment à traduire </segment>
```

Figure 95 : *Extrait d'un fichier source sous format Xml*

7.2.2.1.3.3 Import des corpus

Une fois les fichiers source construits, l'administrateur peut les importer sur SECTra_w (Figure 96) en précisant :

- le nom du corpus correspondant,
- le nom du document à importer,
- la langue source,
- la langue cible,
- le chemin d'accès du fichier source,
- le chemin d'accès du fichier cible s'il existe.

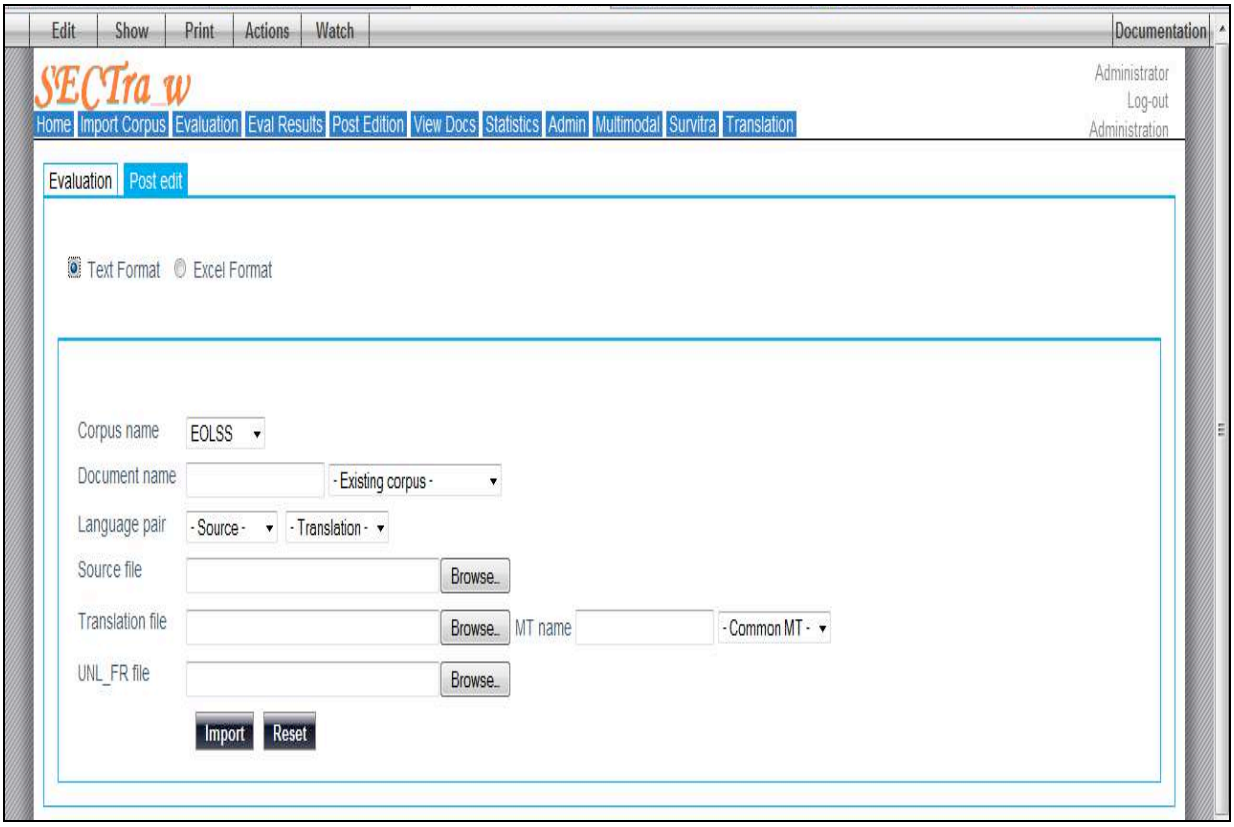


Figure 96 : *Interface d'import de SECTra_w*

7.2.2.1.4 Post-édition avec SECTra_w

SECTra_w offre une interface de post-édition riche en fonctionnalités et facile à utiliser. Pour post-éditer sur SECTra_w, il faut choisir dans les listes en haut de la page : le nom du corpus à post-éditer, le nom du document, la langue source ainsi que la langue cible, puis cliquer sur le bouton "Open" (Figure 97).

La Figure 97 montre un exemple où les *prétraductions* ont été produites par Google-Translate. Le post-éditeur peut, en cliquant sur l'icône "Show Suggestions from TM", visualiser les propositions de traduction à partir de la mémoire de traductions ainsi que leurs taux de coïncidence avec le segment source.

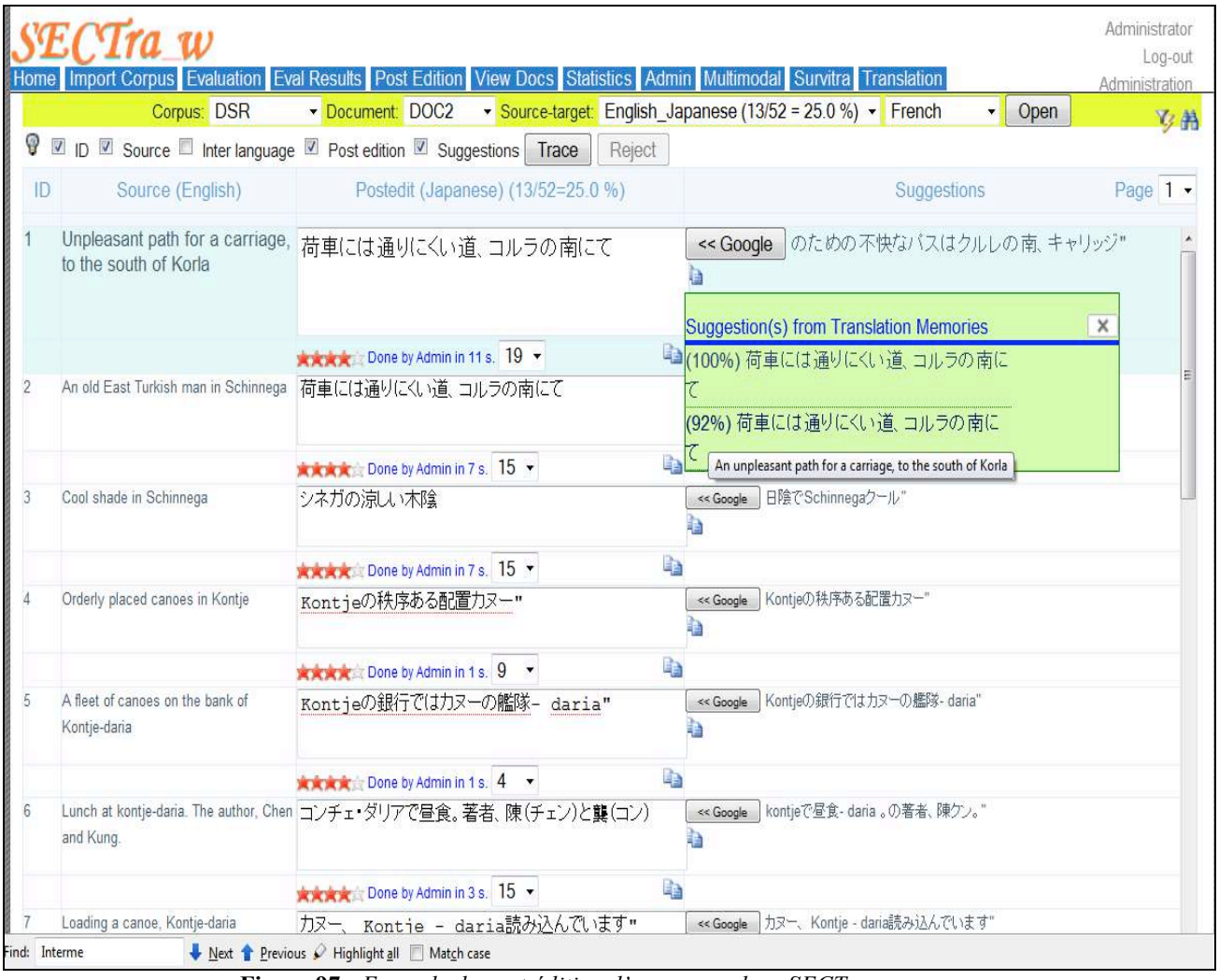


Figure 97 : *Exemple de post-édition d'un corpus dans SECTra_w*

SECTra_w permet aussi de gérer différentes versions de post-édition. En effet, un segment peut être post-édité plusieurs fois par plusieurs post-éditeurs. Pour visualiser l'historique des segments, il suffit de cliquer sur l'icône "Show Older Versions" (Figure 98).

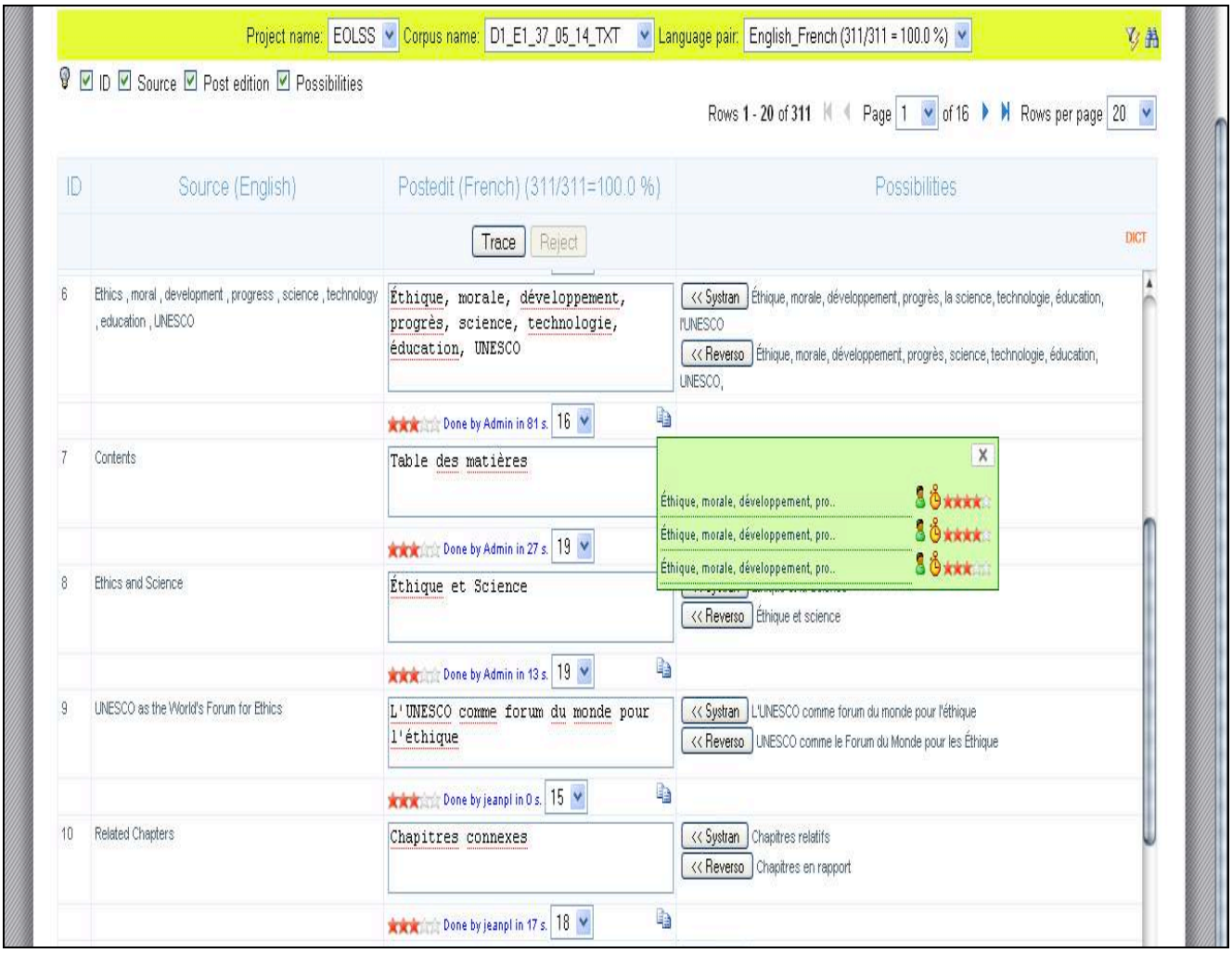


Figure 98 : *Visualisation de l'historique du segment sur SECTra_w*

7.3 Contributeur

La troisième entité de notre processus de localisation tripartite est le contributeur (utilisateur final, bêta-testeur, traducteur bénévole, etc.). Afin de participer de façon efficace au processus de localisation, le contributeur dispose de certaines données sur sa machine. Nous pouvons classer ces données selon deux catégories : des données indispensables à la localisation en contexte de l'application et des données optionnelles sous forme des ressources locales réduites d'aide à la traduction.

7.3.1 Données indispensables à la localisation en contexte de l'application

Lors de l'installation de l'application sur sa machine, l'utilisateur dispose de certaines données qui sont installées par défaut avec son application. Il s'agit essentiellement du fichier XLIFF de l'application et du module LocaInContext.

- *Fichier XLIFF contenant les éléments textuels de l'application :* il est installé par défaut avec l'application. Comme expliqué dans le chapitre 4, ce fichier contient toutes les nouvelles propositions de traduction du contributeur. Ce fichier est

installé directement dans le répertoire de l'application et il est indispensable au processus de localisation en contexte. Il permet d'une part d'enregistrer en local toutes les nouvelles propositions de traduction du contributeur et d'autre part la mise à jour des interfaces graphiques éditées.

- *Le module LocaInContext* : il est aussi installé par défaut avec l'application. Il est intégré directement dans l'exécutable de l'application.

7.3.2 Ressources locales réduite d'aide à la traduction

Au moment de l'installation de l'application, certaines ressources linguistiques sont installées par défaut avec l'application. Le but est de permettre au contributeur, même dans le cas ou il travaille en local, d'avoir accès à des ressources d'aide à la traduction. Il s'agit de deux types de ressources : ressources produites par des systèmes de traduction automatique (Google-Translate, Reverso) et ressources dictionnairiques.

7.3.2.1 Ressources produites par des systèmes de traduction automatique

Nous avons construit un deuxième fichier XLIFF Trad_Auto.xliff qui contient les *prétraductions* de tous les éléments textuels des interfaces graphiques de l'application, produites par les systèmes de traduction Google-Translate et Reverso (Figure 99). Au moment de la localisation en contexte et en local, le contributeur peut s'en servir comme point de départ. La structure ainsi que les identifiants des chaînes du fichier Trad_Auto sont les mêmes que ceux du fichier XLIFF de l'application.

```
<xliff version='1.2'>
<file original='NativeLang.xml' source-language='en' target-language='fr'>

<body>

<group restype="dialog" resname="ColumnEditor">
    <trans-unit id="2604">
        <source name="Number to insert" />
        <target System="Google-Translate"> Nombre d'insérer </target>
        <target System="Reverso"> Le numéro(nombre) pour insérer </target>
    </trans-unit>
    <trans-unit id="2605">
        <source name="Text to insert" />
        <target System="Google-Translate"> Texte à insérer </target>
        <target System="Reverso"> Le texte pour insérer </target>
    </trans-unit>
    <trans-unit id="2606">
        <source name="Cancel" />
        <target System="Google-Translate"> Annuler </target>
        <target System="Reverso"> Abandonner </target>
    </trans-unit>
    <trans-unit id="2607">
        <source name="OK" />
        <target System="Google-Translate"> d'accord </target>
        <target System="Reverso"> OK </target>
    </trans-unit>
```

Figure 99 : *Extrait du fichier contenant des traductions produites par les systèmes de traduction automatique Google-Translate et Reverso*

7.3.2.2 Ressources dictionnairiques

En plus des *prétraductions* produites par les systèmes de traduction automatique, nous fournissons au contributeur des ressources dictionnairiques (Sérasset, 1994, 1995). Il s'agit d'un fichier XML en local qui a été construit à partir de la base Jibiki (Sérasset, 2004, 2006), (Mangeot, 2002). La plate-forme Jibiki a été construite dans le cadre de projet Papillon (Boitet et al. 2002), (Mangeot, 2001) et (Sérasset, 2001). Il s'agit d'une base lexicale multilingue comprenant entre autres l'allemand, l'anglais, le français, le japonais, le malais, le lao, le thaï, le vietnamien et le chinois (Boitet, 2005).

Pour construire le fichier XML à partir de la plate-forme Jibiki, nous avons utilisé un client écrit en JAVA "REST Application programming Interface" (Figure 100), qui permet de communiquer avec la base Jibiki via des requêtes HTTP (GET, POST, PUT, DELETE). Nous avons utilisé en particulier la méthode GET de ce client en lui passant en paramètre le nom du dictionnaire, la langue source, la langue cible et le préfixe (peut être composé d'une ou plusieurs lettres). Cela nous permet de récupérer en retour un fichier XML avec toutes les entrées du dictionnaire correspondant à notre requête. Une fois ce fichier récupéré, nous l'analysons afin d'enlever les informations inutiles à l'utilisateur. Ainsi, lors de la localisation *en* contexte en local, les traductions issues du dictionnaire auront une étiquette avec une seule étoile indiquant qu'il s'agit d'une traduction venant d'un dictionnaire.

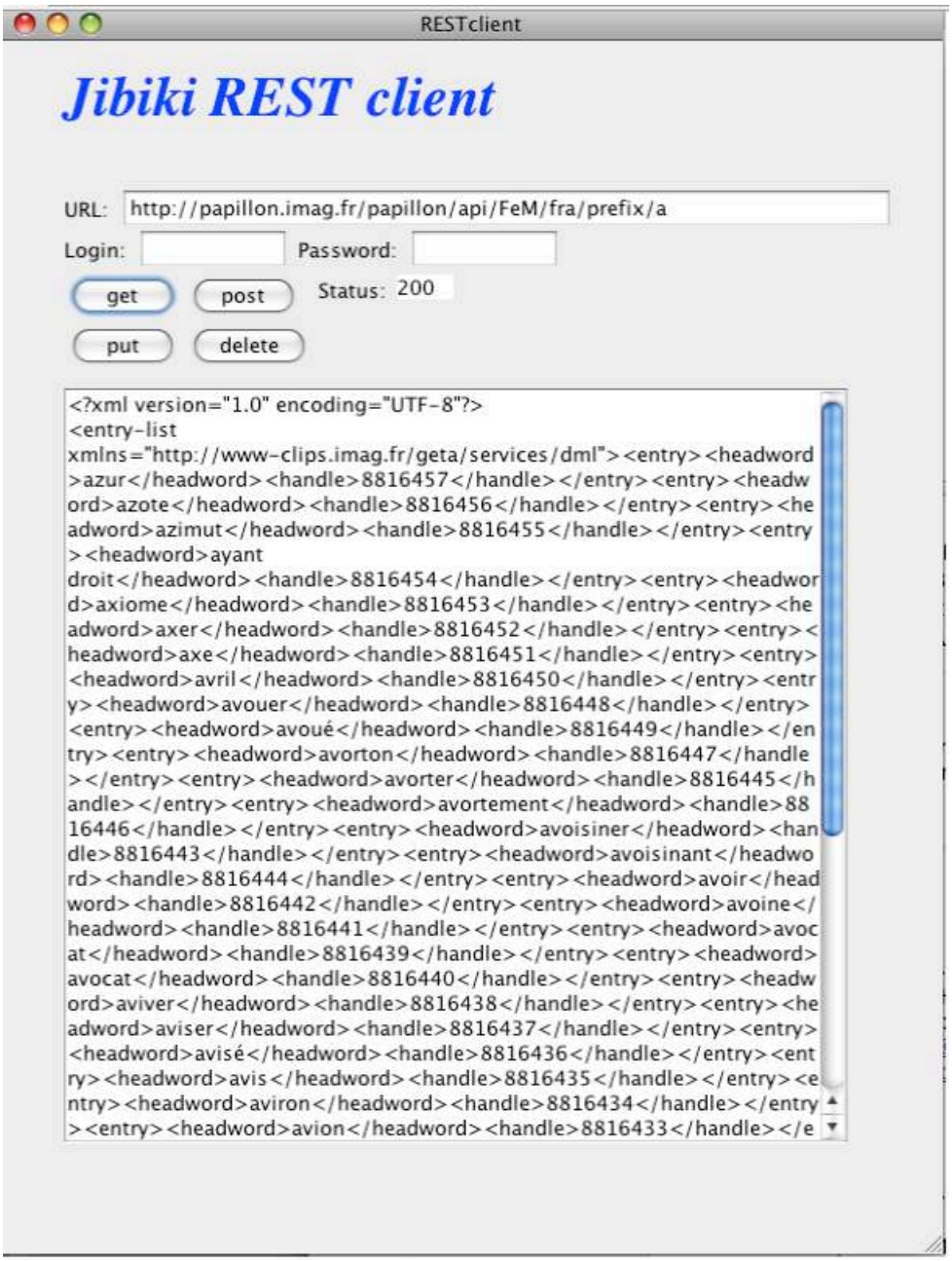


Figure 100 : *Interface du client JAVA affichant le résultat de la méthode GET sur la base Jibiki des termes ayant comme préfixe la lettre 'a'.*

Chapitre 8 : Interactions entre les trois entités du processus de localisation tripartite

Introduction

Nous avons présenté dans le chapitre précédent le nouveau processus de localisation en contexte sous forme d'un processus tripartite avec les trois entités suivants : donneur d'ordres (Éditeur de logiciel dans le cas des logiciels commerciaux et communauté d'administrateurs dans le cas des logiciels libres), contributeur (utilisateur final, traducteur bénévole ou professionnel, β-testeur, etc.) et le site SECTra_w. Ces trois entités interagissent tout au long du processus de localisation. Nous présentons donc dans ce chapitre, les différentes interactions entre ces trois entités.

8.1 Donneur d'ordres <---> site SECTra_w

Nous avons classé les interactions entre le donneur d'ordres et le site SECTra_w selon deux catégories principales : interactions permettant les échanges de données entre les deux entités et interactions permettant la suivie, le contrôle et la validation des données.

8.1.1 Échanges des données entre le donneur d'ordres et SECTra_w

Plusieurs échanges de données entre le donneur d'ordres et SECTra_w sont effectués durant le processus de localisation. Ces échanges de données sont faits sous forme d'import et d'exports des documents source et cible.

8.1.1.1 Import du corpus source

Avant le démarrage du projet de localisation, le donneur d'ordres doit importer le corpus source correspondant sur SECTra_w. Dans notre cas, le corpus source est constitué d'un ou plusieurs fichiers sources, qui contiennent les éléments textuels des interfaces graphiques de l'application à localiser. Le corpus source est donc construit, à partir des fichiers de ressources de l'application. Selon le format des fichiers de ressources, le donneur d'ordres extrait automatiquement ou manuellement les éléments textuels des interfaces pour construire les fichiers source. Il peut faire le choix de rassembler tous les éléments textuels des différents fichiers de ressources dans un même fichier source. Il peut aussi associer à chaque fichier de ressources un fichier source. Prenons l'exemple d'un logiciel qui a 3 fichiers de ressources, nous pourrions avoir un corpus source composé soit de 3 fichiers source, soit d'un seul fichier source (qui contient les éléments textuels des 3 fichiers de ressources). Les fichiers source doivent être sous l'un des formats qui sont pris en compte par SECTra_w (Excel, Txt, XML) (Figure 101).

```
1  Dialog_Find@1=Suivant;
2  Dialog_Find@2=Annuler;
3  Dialog_Find@1620=Recherche :
4  Dialog_Find@1603=Mot entier &amp;amp;uniquement;
5  Dialog_Find@1604=Respecter la &amp;amp;casse;
6  Dialog_Find@1605=&amp;amp;Expression régulière;
7  Dialog_Find@1606=&amp;amp;Boucler;
8  Dialog_Find@1612=&amp;amp;Haut;
9  Dialog_Find@1613=&amp;amp;Bas;
10 Dialog_Find@1614=Compter;
11 Dialog_Find@1615=Rechercher tout;
12 Dialog_Find@1616=Marquer les lignes;
13 Dialog_Find@1617=Colorer les mots trouvés;
14 Dialog_Find@1618=Purger à chaque fois;
15 Dialog_Find@1621=Direction;
16 Dialog_Find@1611=Rem&amp;amp;placer par :;
17 Dialog_Find@1608=&amp;amp;Remplacer;
18 Dialog_Find@1609=Remplacer &amp;amp;tout;
19 Dialog_Find@1623=Transparence;
20 Dialog_Find@1687=à la perte du focus;
21 Dialog_Find@1688=persistante;
22 Dialog_Find@1632=Dans sélection;
23 Dialog_Find@1633=Purger;
```

Figure 101 : *Extrait d'un corpus source sous format textuel*

Une fois construit, le corpus source est importé par le donneur d'ordres dans SECTra_w. L'import du corpus source peut se faire soit en le rajoutant à un projet de localisation existant, soit en créant un nouveau projet de localisation.

8.1.1.2 Import des termes de base dans la mémoire de traductions de SECTra_w

En plus du corpus source, le donneur d'ordres peut aussi importer dans SECTra_w les termes de base qui ont été déjà traduits et validés dans des versions antérieures du logiciel. WinSoft par exemple, rassemble tous les termes de base de chaque logiciel, dans chaque langue cible, dans la base MultiTerm (par exemple pour le logiciel Adobe Acrobat en arabe la base MultiTerm contient 5000 termes de base). Pour importer dans SECTra_w ces données, il faut construire un fichier, par application et par langue cible, qui rassemble la terminologie de base du logiciel à localiser. Ces données peuvent donc être utilisées pour *prétraduire* les termes de base qui figurent dans les fichiers source.

8.1.1.3 Import des fichiers cible dans SECTra_w

SECTra_w permet aussi l'import des fichiers cible s'ils existent. Souvent il s'agit des fichiers qui contiennent des *prétraductions* des segments source obtenues à partir de systèmes de traduction automatique ou de mémoires de traduction. WinSoft par exemple utilise la mémoire de traductions TRADOS pour *prétraduire* les segments qui ont un taux de coïncidence de 90%-100%. Pour le reste des segments, certains localiseurs utilisent les systèmes de traduction automatique. Cela permet de construire des fichiers cible qui sont initialisés à partir des propositions de traduction issues des systèmes de traduction automatique (Systran, Google, Reverso, etc.). Pour importer les fichiers cible dans SECTra_w, le donneur d'ordres doit indiquer le nom du corpus (dans le cas de la Figure 102 il s'agit du corpus EOLSS), le nom du document source (D1_E1_37_05_14_TXT), la langue source, la

langue cible, le chemin d'accès du fichier cible à importer et le nom du système de TA (dans le cas ou le fichier cible a été construit à partir d'un système de TA).

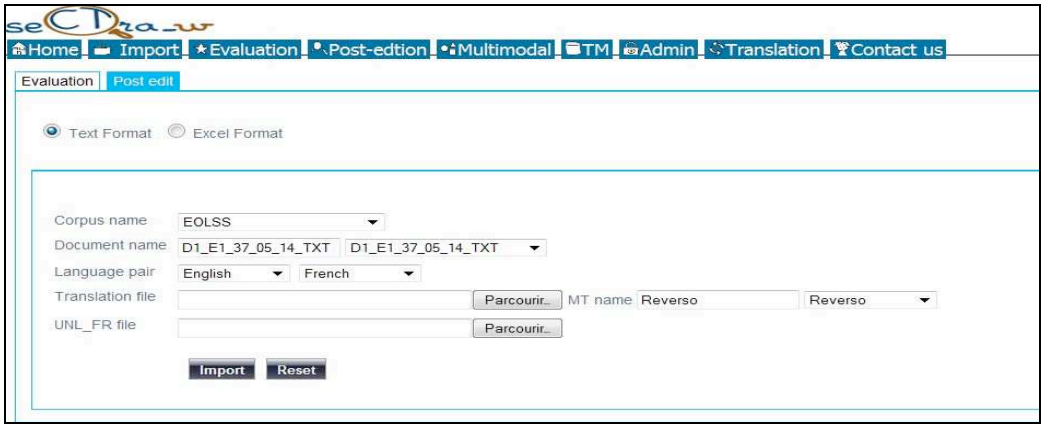


Figure 102 : *Interface de SECTra_w permettant l'import des fichiers cible*

8.1.1.4 Export des corpus cible

Le donneur d'ordres peut exporter à tout moment les fichiers cible qui contiennent les propositions de traduction qui ont été faites par les contributeurs. Comme le montre la Figure 103, le donneur d'ordres peut choisir d'exporter tous les fichiers du corpus ou sélectionner uniquement le fichier cible qu'il souhaite exporter.

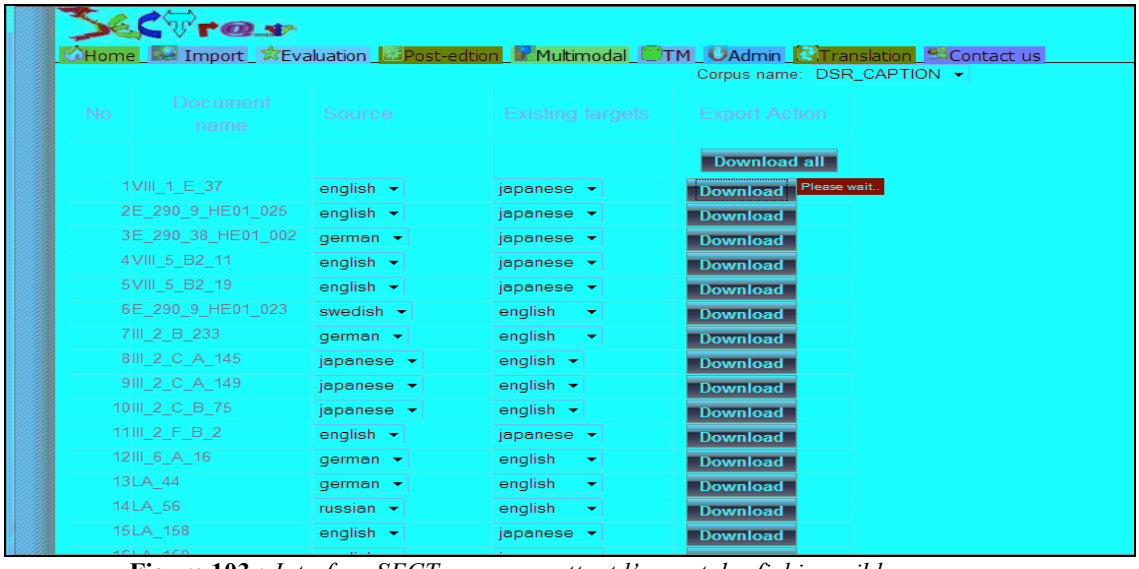


Figure 103 : *Interface SECTra_w permettant l'export des fichiers cible.*

8.1.2 Procédure et politique de suivi, contrôle et validation des propositions de traduction

Comme nous l'avons expliqué dans le premier chapitre, dans le processus de localisation actuel, les traductions sont faites uniquement par des traducteurs professionnels et contrôlées puis validées par des réviseurs professionnels. Ainsi, en aucun cas, le donneur

d'ordres n'intervient pour contrôler ou valider les traductions. Cependant, la mise en place de la nouvelle approche de localisation en contexte, qui fait intervenir plusieurs types d'utilisateur, suppose l'adoption d'une nouvelle procédure et politique de suivi, contrôle et validation des différentes propositions de traduction à provenances multiples. Le donneur d'ordres, ne maîtrisant pas toutes les langues cible, est incapable de contrôler et valider toutes les propositions de traduction dans toutes les langues cibles.

Nous proposons donc deux solutions, la première est une validation humaine et la deuxième est une validation automatique.

- *Validation humaine :* le donneur d'ordres peut associer à chaque projet de localisation, dans chaque langue cible, un modérateur qui a pour rôle de suivre, contrôler, post-éditer et valider les nouvelles traductions. Le modérateur peut être un collaborateur externe ou un simple utilisateur qui a été choisi par le donneur d'ordres (en se basant sur certains critères comme son profil, son implication dans le projet de localisation, etc.).

  Le modérateur peut donc post-éditer et valider directement sur SECTra_w en enregistrant dans la mémoire de traductions les traductions validées. Les traductions validées auront donc l'étiquette ***** (traduction certifiée par un traducteur agréé par l'éditeur).

  Toutefois, l'évaluation peut aussi être faite par d'autres types de contributeurs. Dans ce cas, les traductions porteront l'étiquette **** (traduction certifiée par un traducteur, non agréé par l'éditeur).

- *Validation automatique :* elle consiste à valider les traductions en se basant sur un système d'apprentissage.

L'évaluation de la qualité de traduction est un axe de recherche à part entière. Étant donné que notre problématique de recherche ne porte pas sur l'évaluation de la qualité et la validation des traductions, nous ne sommes pas allés plus loin dans la mise en place de ces solutions.

Dans notre cas, nous avons considéré que toutes les traductions qui ont été proposées par les contributeurs et qui ont un score de qualité supérieur à 10/20 sont valides. Les traductions validées sont exportées par le donneur d'ordres pour être intégrées dans les fichiers de ressources de l'application et ainsi générer de nouvelles versions localisées. Les traductions sont exportées avec leurs étiquettes, qui seront visibles pour l'utilisateur au moment de la localisation en contexte.

8.2 Site SECTra_w <---> Contributeurs

Nous présentons ici les interactions entre les contributeurs et le site SECTra_w. Tout au long du processus de localisation en contexte, les contributeurs interagissent de façon permanente avec le site SECTra_w sur le Web pour s'échanger et partager des données entre eux. Les interactions entre les contributeurs et SECTra_w se font via des requêtes HTTP directement à partir de l'application.

8.2.1 Localisation en contexte et en ligne sur SECTra_w

Nous avons mis en place deux scénarios permettant la localisation *en contexte* et en ligne avec SECTra_w. Le premier scénario permet à l'utilisateur de localiser son application en contexte en interagissant avec le site SECTra_w. Le deuxième scénario permet à l'utilisateur de post-éditer, dans un premier temps, directement sur SECTra_w, puis, dans un deuxième temps, de récupérer et d'intégrer toutes ses nouvelles propositions de traduction dans son application. Nous décrivons dans la suite de ce paragraphe ces deux scénarios et nous les illustrons sur l'exemple de Notepad-plus-plus dans le chapitre suivant.

8.2.1.1 Scénario 1 : Localisation en contexte intégrant l'appel à SECTra_w depuis l'application

L'utilisateur peut localiser son application *en contexte* tout en interagissant avec SECTra_w (Figure 104). Ce scénario permet à l'utilisateur, durant l'utilisation de son application, par un simple clic-droit sur n'importe quel élément textuel de l'interface graphique, de passer dans un contexte d'édition qui communique avec SECTra_w. En effet, une fois passé dans le contexte d'édition, l'utilisateur peut récupérer à partir de SECTra_w toutes les propositions de traduction qui ont été faites par les autres utilisateurs. La requête de récupération des traductions doit contenir les informations suivantes : le projet de localisation correspondant, le nom du corpus source, l'identifiant de la chaîne éditée et pour laquelle l'utilisateur souhaite récupérer les propositions de traduction des autres utilisateurs, la langue source ainsi que la langue cible.

L'utilisateur peut aussi envoyer, depuis son application, ses nouvelles propositions de traduction. Comme pour la récupération des traductions, la requête d'envoi doit contenir le nom du projet de localisation, le nom du corpus source, l'identifiant de la chaîne, la langue source et la langue cible.

Dans les deux cas, les requêtes sont gérées par notre module de localisation *en contexte*. Nous illustrons ce scénario sur le logiciel Notepad-plus-plus dans le chapitre suivant.

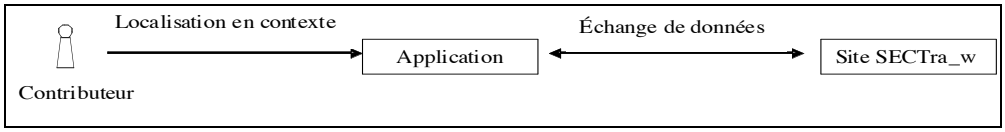


Figure 104 : *Localisation en contexte intégrant l'appel à SECTra_w depuis l'application*

8.2.1.2 Scénario 2 : Localisation en contexte directement sur SECTra_w

Ce scénario (Figure 105) permet dans un premier temps à l'utilisateur d'aller post-éditer directement sur SECTra_w le(s) segment(s) source(s) qu'il souhaite traduire. Dans un deuxième temps, il peut revenir dans son application et récupérer toutes les nouvelles propositions qu'il a effectuées sur SECTra_w.

Comme nous l'avons expliqué dans le chapitre précédent, l'interface de post-édition de SECTra_w est une page Web qui contient des segments les uns à la suite des autres verticalement dans une colonne (une quinzaine, ce qui correspond à une *page* e traduction), et associés horizontalement aux contenus d'autres colonnes. Ainsi, au moment de son passage sur SECTra_w, l'utilisateur est redirigé directement sur la page qui contient le segment source qu'il souhaite éditer. Une fois dans SECTra_w, l'utilisateur peut post-éditer s'il le souhaite d'autres segments source et revenir à tout moment dans son application pour récupérer ses propositions de traduction à partir de SECTra_w. Les nouvelles propositions de traduction sont donc enregistrées dans le fichier XLIFF de l'application.

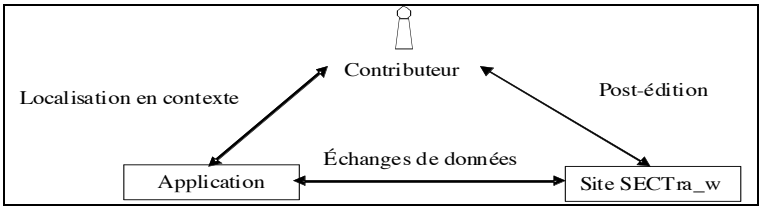


Figure 105 : *Localisation en contexte directement sur SECTra_w*

8.2.2 Synchronisation des ressources locales depuis l'application

Comme nous l'avons mentionné dans le chapitre précédent, nous avons mis à disposition des utilisateurs des ressources minimales locales qui peuvent servir comme point de départ à la traduction. Ces ressources sont constituées initialement, uniquement, des propositions de traduction issues des systèmes de traduction automatique (Systran, Reverso, Google). Ainsi, au fur et à mesure de l'avancement du processus de localisation, l'utilisateur peut demander la synchronisation des ses ressources locales depuis SECTra_w. Cette synchronisation permet d'une part à l'utilisateur d'envoyer toutes les propositions de traduction qu'il a faite en local, et d'autre part de récupérer les traductions faites par les autres utilisateurs en les rajoutant aux fichiers de ressources locales.

8.2.2.1 Envoi des propositions de traduction de l'utilisateur

L'utilisateur peut faire le choix de localiser en local son application et de se synchroniser avec le site SECTra_w plus tard. Dans ce cas, toutes ses traductions sont stockées dans le fichier XLIFF de l'application. Ainsi, lors de la synchronisation des ressources locales, les traductions locales de l'utilisateur sont envoyées à SECTra_w à partir du fichier XLIFF. La question qui se pose maintenant est de savoir comment identifier les traductions que l'utilisateur a déjà envoyées de celles qu'il n'a pas envoyées ?

Pour ce faire, nous avons associé à chaque élément textuel dans le fichier XLIFF un attribut submitted qui prend comme valeur YES ou NO. Ainsi, à chaque synchronisation de l'utilisateur, l'attribut submitted est mis à jour.

Lorsque les nouvelles propositions de traduction sont identifiées, elles sont envoyées sur SECTra_w avec les informations correspondantes (contexte (boîte de dialogue ou menu auxquelles elles appartiennent), nom de l'utilisateur, date, score de qualité attribué par l'utilisateur, commentaire, etc.).

8.2.2.2 Récupération des nouvelles traductions qui ont été proposées par les autres utilisateurs

Lors de la synchronisation des ressources locales, l'utilisateur récupère toutes les nouvelles propositions de traduction qui ont été faites par les autres utilisateurs ou celles qui ont été validées par le donneur d'ordres. Les traductions récupérées sont enregistrées dans le fichier XLIFF de l'application et plus précisément dans la balise <user-Translation> du fichier XLIFF (Figure 106).

```
<trans-unit id = "2023">
    <source name = "Text to insert"/>
    <target name= "Texte à insérer" status= "approved by editor"/>
    <alt-Trans>
        <Google-Translation name = "Texte à insérer"/>
        <user-Translation name= "Texte à insérer" author= "Fraisse" date= "30-12-2009" quality="18"/>
        <user-Translation name= "Texte pour insérer" author= "hung" date= "05-12-2009" quality="15"/
    </alt-Trans>
</trans-unit>
```

Figure 106 : *Extrait du fichier XLIFF de l'application Notepad-plus-plus*

Une fois récupérées, les propositions de traduction des autres utilisateurs sont visibles pour l'utilisateur au moment de la localisation *en contexte*.

8.3 Contributeur <---> Donneur d'ordres : synchronisation lors des connexions normales de mise à niveau de l'application / sur requête du contributeur

Les interactions entre le contributeur et le donneur d'ordres sont sous forme des requêtes de synchronisation entre le PC du contributeur et le site du donneur d'ordres.

Actuellement, le processus classique de synchronisation se fait lors des mises à jour périodiques de l'application. Ainsi, il faut compter en moyenne un délai minimum de 6 mois entre les différentes "release" de l'application. Ce processus de synchronisation n'est donc pas adapté à notre approche de localisation en contexte qui est caractérisée par un modèle de localisation continue, et donc incrémentale. En effet, parmi les solutions que nous avons proposées dans le chapitre 3, et qui peuvent être mise en œuvre grâce à notre approche de localisation *en contexte*, l'une d'entre elles permet au donneur d'ordres de publier une version partiellement localisée voir non localisée qui va être localisée au fur et à mesure pendant la vie du logiciel.

Ce modèle de localisation continue permet donc d'avoir un processus de localisation incrémentale en termes de quantité ainsi qu'en termes de qualité. Nous proposons un processus de synchronisation qui permet à l'utilisateur d'avoir accès à tout moment à la dernière version localisée du logiciel.

La fréquence de génération des nouvelles versions localisées dépend du donneur d'ordres. Ce dernier peut faire le choix de générer une nouvelle version tous les jours, tous les deux jours, tous les trois jours, toutes les semaines, tous les mois, etc.

8.3.1 Analyse du fichier XLIFF de l'application

L'analyse du fichier XLIFF est faite suite à la demande de synchronisation de l'utilisateur. Cela permet d'identifier le logiciel que l'utilisateur souhaite mettre à jour, la langue cible, ainsi que la dernière date de synchronisation qui a été effectuée par l'utilisateur. L'intérêt de cette analyse est de vérifier que toutes les propositions de traduction de l'utilisateur ont été bien envoyées à SECTra_w. Si ce n'est pas le cas, le donneur d'ordres peut récupérer les traductions de l'utilisateur pour les importer par la suite dans SECTra_w.

```
<?xml version="1.0" encoding="UTF-8" ?>
<NotepadPlus version="V5.0">
<file original="NativeLang.xml" source-language="en" target-language="fr" Last-Synchronisation-Date= "02-01-2010">
    <body>
        <group restype= "dialog" resname= "Editeur de colonne">
            <trans-unit id = "2023">
                <source name = "Text to insert"/>
                <target name= "Texte à insérer" status= "approved by editor"/>
                <alt-Trans>
                    <Google-Translation name = "Texte à insérer"/>
                    <user-Translation name= "Texte à insérer" author= "Fraisse" date= "30-12-2009" quality="18"/>
                </alt-Trans>
            </trans-unit>
            <trans-unit id = "2033">
                <source name = "Number to insert"/>
                <target name="Nombre à insérer" status = "approved by editor"/>
                <alt-Trans>
                    <Google-Translation name =" Nombre d'insérer"/>
                    <user-Translation name= "Nombre à insérer" author="Fraisse" date=" 30-12-2009" quality="18"/>
                </alt-Trans>
            </trans-unit>
```

Figure 107 : *Extrait du fichier XLIFF de l'application Notepad-plus-plus*

8.3.2 Visualisation des nouvelles traductions qui ont été validées par l'éditeur depuis la dernière synchronisation de l'utilisateur

Avant de procéder à la mise à jour de son application, l'utilisateur peut visualiser les nouvelles propositions de traduction qui ont été validées par le donneur d'ordres. Il a le choix d'accepter toutes les nouvelles traductions validées ou seulement celles sur les quelles il est d'accord.

8.3.2.1 Synchronisation totale : accepter toutes les nouvelles traductions validées par le donneur d'ordres

L'utilisateur peut choisir d'accepter toutes les nouvelles traductions validées par le donneur d'ordres. Il s'agit alors d'un processus de synchronisation totale. Dans ce cas, toutes les nouvelles propositions de traduction validées par le donneur d'ordres sont intégrées dans le fichier XLIFF de l'application. Les interfaces utilisateur contenant les éléments textuels correspondants sont donc mises à jour.

Le processus de synchronisation totale est complètement implicite, dans la mesure où l'utilisateur est incapable d'identifier les nouvelles traductions qui ont été intégrées dans sa nouvelle version localisée. Cela implique que, si l'utilisateur souhaite mettre à jour son application, il est obligé d'accepter toutes les nouvelles traductions qui ont été validées par le donneur d'ordres. Lorsque la mise à jour est effectuée, l'utilisateur découvre au fur et à mesure de l'utilisation de son application, les traductions qui ont été rajoutées ou modifiées. Ainsi, dans le cas ou l'utilisateur n'est pas d'accord sur certaines traductions, il est trop tard pour revenir en arrière. Dans ce cas, la seule chose que l'utilisateur peut faire est d'envoyer un mail à l'éditeur de logiciel pour lui exprimer son mécontentement.

8.3.2.2 Synchronisation partielle : sélectionner et valider quelques traductions

Nous avons mis en place une nouvelle approche de synchronisation qui permet d'inclure l'utilisateur. En effet, au moment de la synchronisation, l'utilisateur peut demander la visualisation de toutes les nouvelles traductions qui ont été validées par le donneur d'ordres depuis sa dernière synchronisation. Il peut donc sélectionner uniquement les traductions avec lesquelles il est d'accord. Une fois sélectionnées, les nouvelles traductions sont intégrées dans le fichier XLIFF de l'application et les interfaces graphiques seront mises à jour lors des prochains affichages par l'utilisateur.

Chapitre 9 : Expérimentation du processus de localisation tripartite sur Notepad-plus-plus

Introduction

Nous avons présenté, dans les deux chapitres précédents, le processus de localisation *en contexte* sous forme d'un processus tripartie avec les trois entités : contributeurs, donneur d'ordres et SECTra_w. Nous avons aussi présenté les différentes interactions entre ces trois entités. Nous présentons maintenant une expérimentation de ce processus tripartite sur le logiciel Notepad-plus-plus.

9.1 Préparation et import des données de Notepad-plus-plus sur SECTra_w

Comme expliqué dans les deux chapitres précédents, afin de permettre la localisation *en contexte* et en ligne d'un logiciel, le donneur d'ordres doit préparer puis importer sur SECTra_w, les données que les contributeurs vont s'échanger et partager durant le processus de localisation *en contexte*. Dans notre cas, il s'agit de tous les éléments textuels des interfaces graphiques de l'application à localiser.

Nous décrivons donc dans ce paragraphe la préparation ainsi que l'import des données de Notepad-plus-plus dans SECTra_w.

9.1.1 Corpus source de Notepad-plus-plus

Nous avons construit un corpus source que nous avons nommé NOTEPAD. Il contient tous les éléments textuels des interfaces utilisateur. Nous décrirons dans la suite de ce paragraphe le format et la structure de ce corpus ainsi que la façon dont nous l'avons construit.

9.1.1.1 Format et structure

De façon générale, un corpus source est composé dans SECTra_w d'un ou plusieurs fichiers qui contiennent les segments source à traduire. Les formats acceptés par SECTra_w sont : Txt, XML ou Excel. Dans le cas de Notepad-plus-plus, nous avons choisi le format Txt comme format pour les fichiers source.

Le corpus source de Notepad-plus-plus contient deux fichiers: un fichier pour les éléments textuels des menus contextuels de l'application, et un fichier pour les chaînes des différentes boîtes de dialogues de Notepad-plus-plus. Nous détaillons ci-dessous la construction de ces deux fichiers.

9.1.1.2 Construction du corpus source NOTEPAD à partir du fichier de ressources « NativeLang » de Notepad-plus-plus

Dans le cas de Notepad-plus-plus, initialement, tous les éléments textuels des interfaces graphiques (boîtes de dialogues et menus contextuels) sont stockés dans un seul fichier de ressources, NativeLang.xml. Nous nous sommes basée sur ce fichier de ressources pour construire le corpus NOTEPAD. Le fichier NativeLang est sous format XML et contient au total 600 chaînes d'interface. Comme le montre la Figure 108, selon son endroit d'apparition, une chaîne appartient soit à la balise <Menu> soit à la balise <Dialog>.

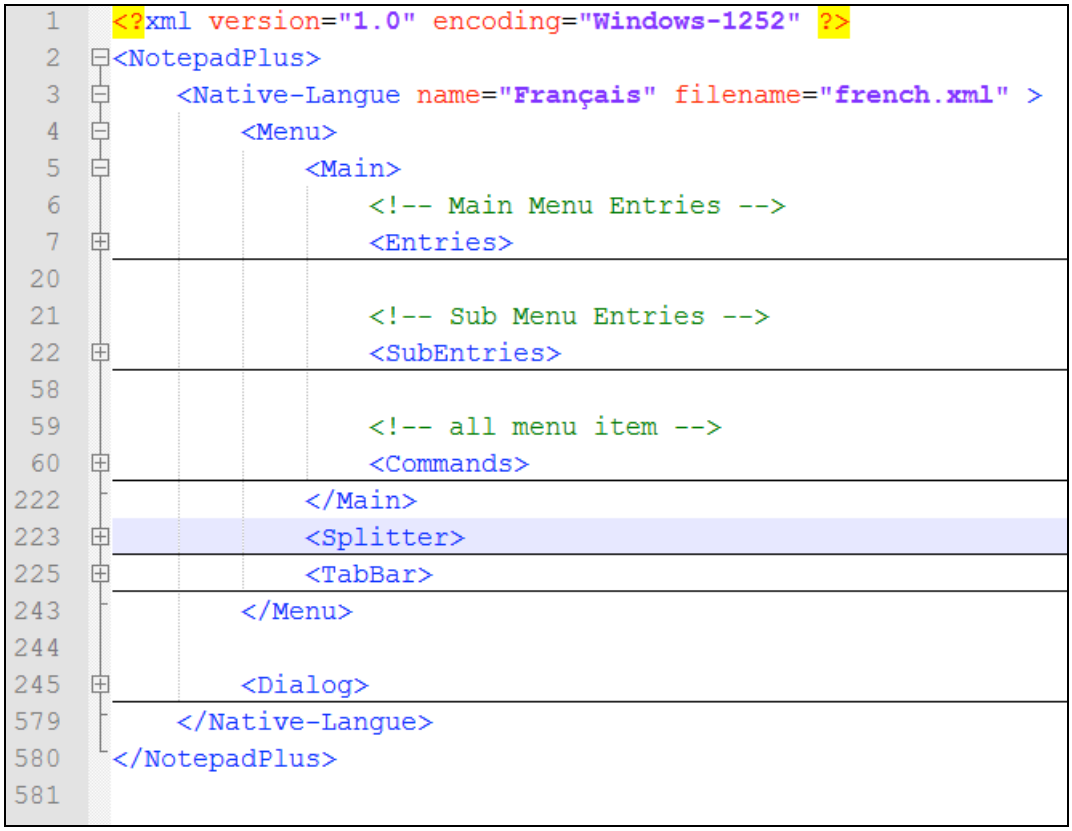


Figure 108 : *Structure du fichier NativeLang du logiciel Notepad-plus-plus*

La Figure 109 montre un extrait du fichier NativeLang qui contient les éléments textuels du menu principal de Notepad-plus-plus. Chaque élément textuel est représenté par une balise <Item> qui contient deux attributs : un attribut "id" qui contient l'identifiant de la chaîne et un attribut "name" qui contient la valeur de la chaîne.

Figure 109 : *Extrait du fichier NativeLang contenant les éléments textuels du menu principal de Notepad-plus-plus*

La Figure 110 illustre le contenu de la balise principale <Dialog> qui contient au total neuf boîtes de dialogue (Find, GoToLine, Run, StyleConfig, UserDefine, Preference, MultiMacro, Window et ColumnEditor).

```
  1  <?xml version="1.0" encoding="Windows-1252" ?>
  2  <NotepadPlus>
  3      <Native-Langue name="Français" filename="french.xml" >
  4          <Menu>
244
245          <Dialog>
246              <Find title="" titleFind="Rechercher" titleReplace="Remplacer" titleFindInFiles="Rech
286              <GoToLine title="Aller à...">
295
296              <Run title="Exécuter...">
302
303              <StyleConfig title="Configurateur de coloration syntaxique">
335
336              <UserDefine title="Langage utilisateur">
391              <Preference title="Préférences">
546              <MultiMacro title="Exécuter une macro en boucle">
554              <Window title="Documents">
564              <ColumnEditor title="Édition en colonnes">
578          </Dialog>
579      </Native-Langue>
580  </NotepadPlus>
581
```

Figure 110 : *Extrait du fichier NativeLang du logiciel Notepad-plus-plus-plus contenant les différentes boîtes de dialogue de l'application.*

La Figure 111 montre le contenu de la boîte de dialogue GoToLine. Comme pour les menus, chaque chaîne est représentée par une balise <Item> qui contient l'identifiant ainsi que la valeur de la chaîne sous forme d'attribut.

```
  1  <?xml version="1.0" encoding="Windows-1252" ?>
  2  <NotepadPlus>
  3      <Native-Langue name="Français" filename="french.xml" >
  4          <Menu>
244
245          <Dialog>
246              <Find title="" titleFind="Rechercher" titleReplace="Remplacer"
286              <GoToLine title="Aller à...">
287                  <Item id="1" name="&amp;amp;Zyva !"/>
288                  <Item id="2" name="Je vais nulle part"/>
289                  <Item id="2004" name="Vous êtes ici :"/>
290                  <Item id="2005" name="Vous allez à :"/>
291                  <Item id="2006" name="Pas plus loin que :"/>
292                  <Item id="2007" name="Ligne"/>
293                  <Item id="2008" name="Position"/>
294              </GoToLine>
```

Figure 111 : *Extrait du fichier NativeLang du logiciel Notepad-plus-plus-plus illustrant les chaînes de la boîte de dialogue GoToLine*

Pourquoi nous n'avons pas importé directement le fichier NativeLang dans SECTra_w ?

Parce que la structure du fichier NativeLang est différente de la structure des fichiers XML qui sont pris en compte par SECTra_w. De plus, au moment où nous avons fait l'import du corpus source, la version courante de SECTra_w ne supportait que les fichiers au format Txt ou Excel.

Afin de construire le corpus source, nous nous sommes donc basée sur le fichier NativeLang. Pour cela, nous avons construit un analyseur pour extraire et stocker toutes les chaînes qui figurent dans le fichier NativeLang dans des fichiers source Txt. Comme nous l'avons expliqué dans le chapitre 7, pour que le contenu des fichiers Txt soit reconnu et importé correctement sur SECTra_w, il faut que le fichier Txt soit sous la forme d'une liste d'association identifiant_chaîne = valeur_chaîne avec une association par ligne.

Afin de garder une information sur le contexte d'apparition de chaque chaîne, nous avons choisi comme identifiant la concaténation de l'identifiant initial de la chaîne avec celui de l'interface graphique qui le contient (Figure 112).

```
1  Dialog_Find@1=Find Next;
2  Dialog_Find@2=Close;
3  Dialog_Find@1620=Find what :;
4  Dialog_Find@1603=Match &whole word only;
5  Dialog_Find@1604=Match &case;
6  Dialog_Find@1605=Regular &expression;
7  Dialog_Find@1606=Wrap aroun&d;
8  Dialog_Find@1612=&Up;
9  Dialog_Find@1613=&Down;
10 Dialog_Find@1614=Count;
11 Dialog_Find@1615=Find All;
12 Dialog_Find@1616=Mark Line;
13 Dialog_Find@1617=Style found token;
14 Dialog_Find@1618=Purge for each search;
15 Dialog_Find@1621=Direction;
16 Dialog_Find@1611=Re&place with :;
17 Dialog_Find@1608=&Replace;
18 Dialog_Find@1609=Replace &All;
19 Dialog_Find@1623=Transparency;
20 Dialog_Find@1687=On lose focus;
21 Dialog_Find@1688=Always;
22 Dialog_Find@1632=In selection;
23 Dialog_Find@1633=Clear;
```

Figure 112 : *Extrait du fichier source de Notepad-plus-plus contenant les chaînes de la boîte de dialogue "Find"*

La Figure 112 illustre un extrait du fichier source de Notepad-plus-plus, il s'agit des chaînes de la boîte de dialogue "Find". Nous avons choisi le caractère "@" pour séparer l'identifiant de la chaîne de celui de l'interface graphique qui la contient.

9.1.2 Corpus cible de Notepad-plus-plus

Comme pour le corpus source, nous pouvons importer aussi un corpus cible, s'il existe, sur SECTra_w.

9.1.2.1 Format et structure

Tout corpus cible possède le même format ainsi que la même structure que le corpus source. En effet, un corpus cible contient un ou plusieurs fichiers cible par langue. Dans le cas de SECTra_w, un fichier source peut avoir un ou plusieurs fichiers cibles par langue cible.

9.1.2.2 Prétraduction du corpus source

Pour construire le corpus cible, nous avons utilisé le système de traduction automatique "Google-Translate" pour *prétraduire* les fichiers source de Notepad-plus-plus de l'anglais vers le français. Pour cela, nous avons implémenté en Java un robot qui permet d'interroger en ligne le système de traduction "Google-Translate". Ce robot prend comme paramètre le fichier source, la langue source ainsi que la langue cible vers laquelle nous souhaitons *prétraduire* le fichier source et retourne le fichier cible (Figure 113).

```
1  Dialog_Find@1=Suivant;
2  Dialog_Find@2=Annuler;
3  Dialog_Find@1620=Recherche :
4  Dialog_Find@1603=Mot entier &amp;amp;uniquement;
5  Dialog_Find@1604=Respecter la &amp;amp;casse;
6  Dialog_Find@1605=&amp;amp;Expression régulière;
7  Dialog_Find@1606=&amp;amp;Boucler;
8  Dialog_Find@1612=&amp;amp;Haut;
9  Dialog_Find@1613=&amp;amp;Bas;
10 Dialog_Find@1614=Compter;
11 Dialog_Find@1615=Rechercher tout;
12 Dialog_Find@1616=Marquer les lignes;
13 Dialog_Find@1617=Colorer les mots trouvés;
14 Dialog_Find@1618=Purger à chaque fois;
15 Dialog_Find@1621=Direction;
16 Dialog_Find@1611=Rem&amp;amp;placer par :;
17 Dialog_Find@1608=&amp;amp;Remplacer;
18 Dialog_Find@1609=Remplacer &amp;amp;tout;
19 Dialog_Find@1623=Transparence;
20 Dialog_Find@1687=à la perte du focus;
21 Dialog_Find@1688=persistante;
22 Dialog_Find@1632=Dans sélection;
23 Dialog_Find@1633=Purger;
```

Figure 113 : *Extrait du fichier cible en français de Notepad-plus-plus contenant les chaînes de la boîte de dialogue "Find"*

9.1.3 Import du corpus Notepad-plus-plus sur SECTra_w

Une fois construits, les fichiers source de Notepad-plus-plus sont importés sur SECTra_w via l'interface d'import. Avant d'importer le corpus source et cible de Notepad-plus-plus, nous avons créé un nouveau projet de localisation nommé Notepad.

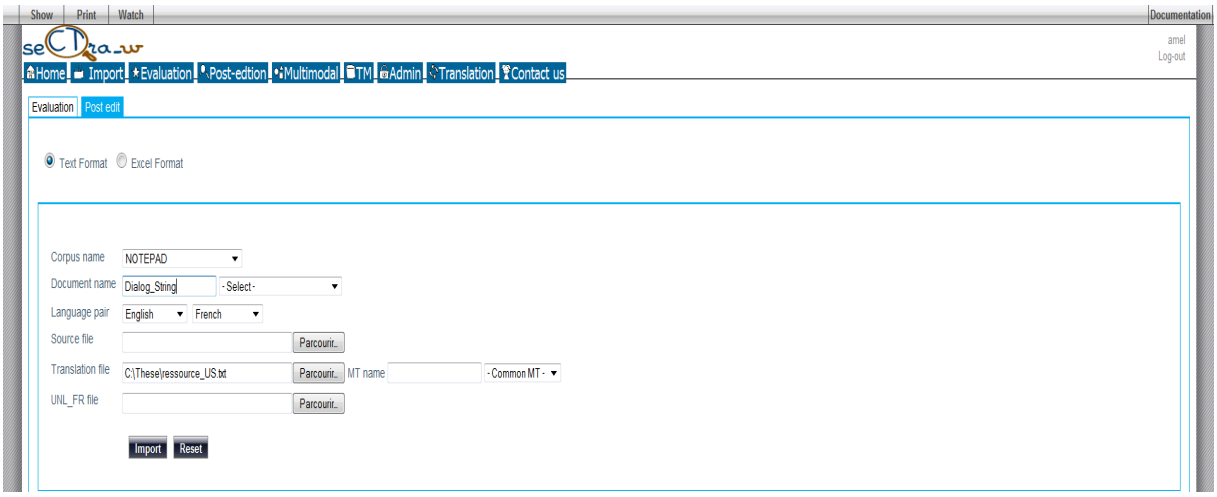


Figure 114 : *Interface d'import des corpus source et cible de Notepad-plus-plus*

9.2 Localisation en contexte et en ligne de Notepad-plus-plus

Nous détaillons dans ce deuxième paragraphe l'expérimentation de la localisation *en contexte* et en ligne de Notepad-plus-plus. Comme mentionné dans le chapitre précédent, nous avons mis en place deux scénarios qui permettent aux utilisateurs de localiser *en contexte* et en ligne leurs applications. Le premier scénario permet à l'utilisateur de localiser son application *en contexte*, sur sa machine, en communiquant avec SECTra_w. Le deuxième scénario permet à l'utilisateur de localiser directement sur SECTra_w et de récupérer par la suite ses traductions dans son application.

9.2.1 Scénario 1 : Localisation en contexte intégrant l'appel à SECTra_w depuis Notepad-plus-plus

Ce scénario permet à l'utilisateur de localiser une application en contexte en interagissant avec SECTra_w. Les interactions avec SECTra_w se font directement depuis l'application. En effet, sans quitter son application, l'utilisateur peut envoyer ses propositions de traduction sur SECTra_w et récupérer les traductions des autres utilisateurs. Cela est indispensable pour motiver l'utilisateur et faciliter son travail de localisation, surtout dans le cas des logiciels commerciaux. Nous détaillons dans la suite de ce paragraphe les différentes étapes de ce premier scénario.

9.2.1.1 Étape 1 : éditer une chaîne de l'interface au cours d'utilisation de Notepad-plus-plus

Au cours d'utilisation du logiciel, par un simple clic-droit, l'utilisateur peut éditer n'importe quel élément textuel de l'interface graphique (entrée du menu, onglet, bouton, étiquette, etc.). Prenons l'exemple de la Figure 115, suite au clic-droit sur la chaîne "Text to insert", l'utilisateur a accès au contexte d'édition de la chaîne qui communique avec SECTra_w.

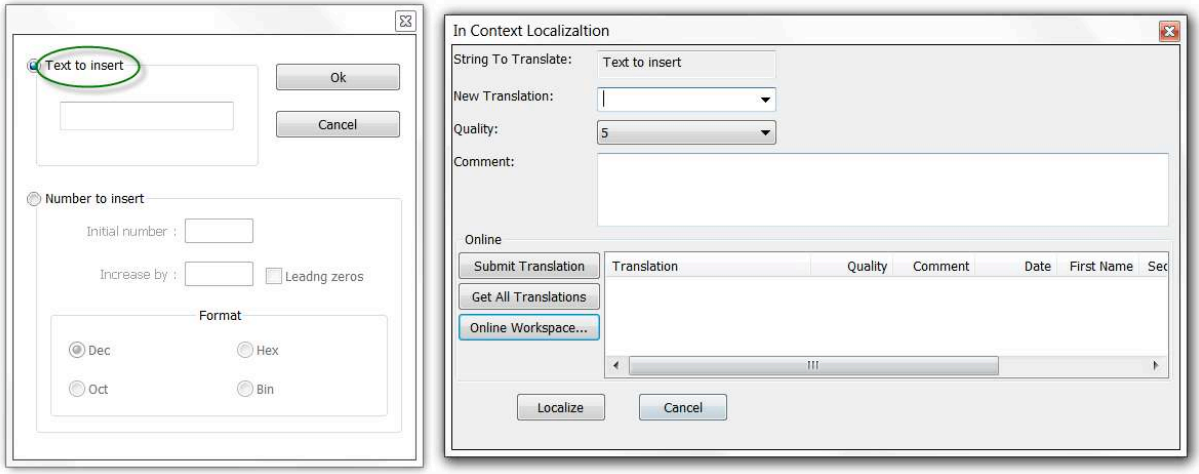


Figure 115 : *Édition de la chaîne d'interface "Text to insert" de Notepad-plus-plus*

La Figure 116 illustre le contexte d'édition de l'entrée du menu "Find Next".

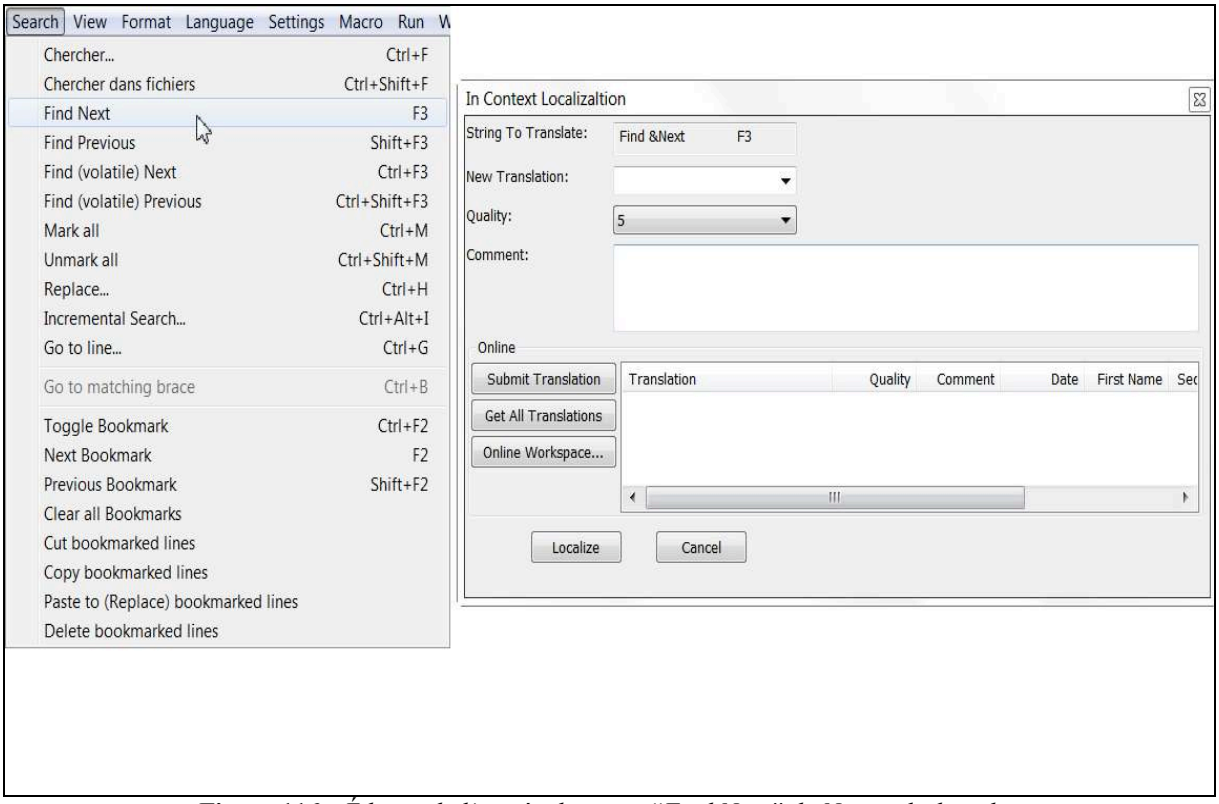


Figure 116 : *Édition de l'entrée du menu "Find Next" de Notepad-plus-plus*

9.2.1.2 Étape 2 : Interagir avec SECTra_w

Quand il est dans le contexte d'édition, l'utilisateur peut interagir avec SECTra_w pour :

- *Récupérer et visualiser les différentes propositions de traduction :* en cliquant sur le bouton "Get All Translations", l'utilisateur peut récupérer toutes les propositions de traduction qui ont été faites par les autres utilisateurs pour la même chaîne et dans le même contexte d'utilisation (dans le cas de cet exemple, le contexte d'utilisation de la chaîne "Text to insert" est la boîte de dialogue "Find" de Notepad-plus-plus).

Comme le montre la Figure 117, les propositions de traduction sont affichées accompagnées:

- du nom de l'utilisateur qui a proposé la traduction,
- de la qualité que l'utilisateur attribué à sa proposition de traduction,
- de la date,
- du commentaire,
- etc.

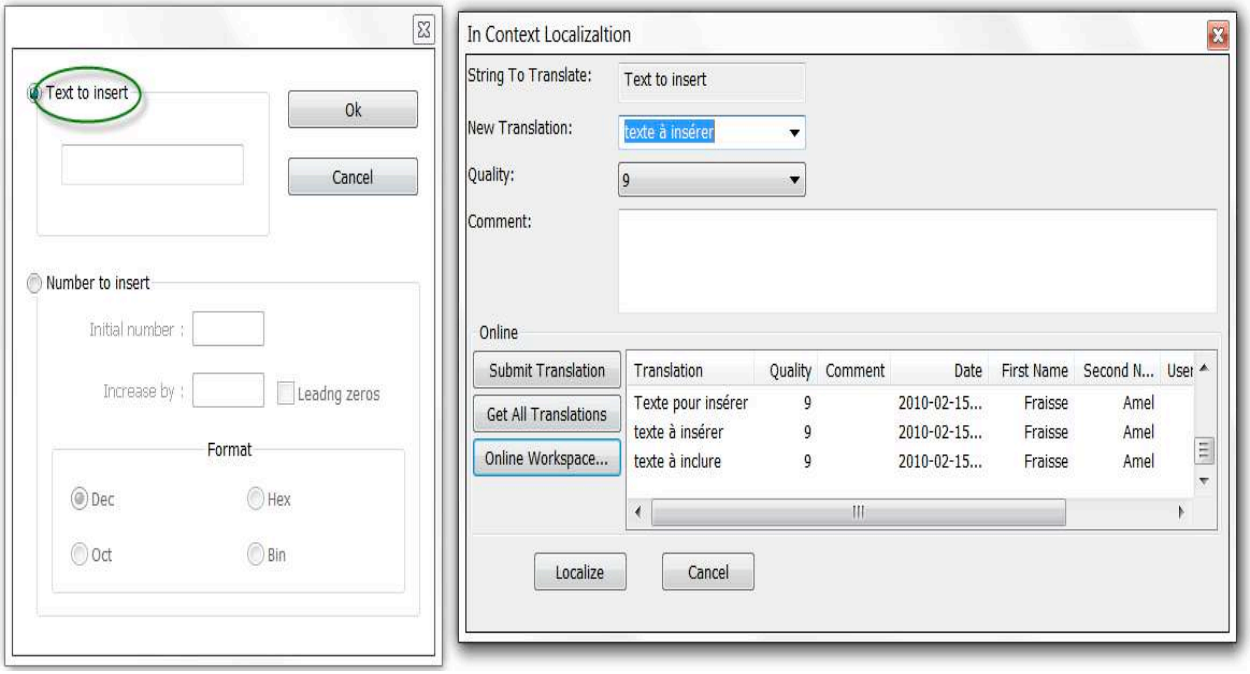


Figure 117 : *Récupération à partir de SECTra_w de toutes les propositions de traduction de la chaîne "Text to insert" de la boîte de dialogue "ColumnEditor" de Notepad-plus-plus.*

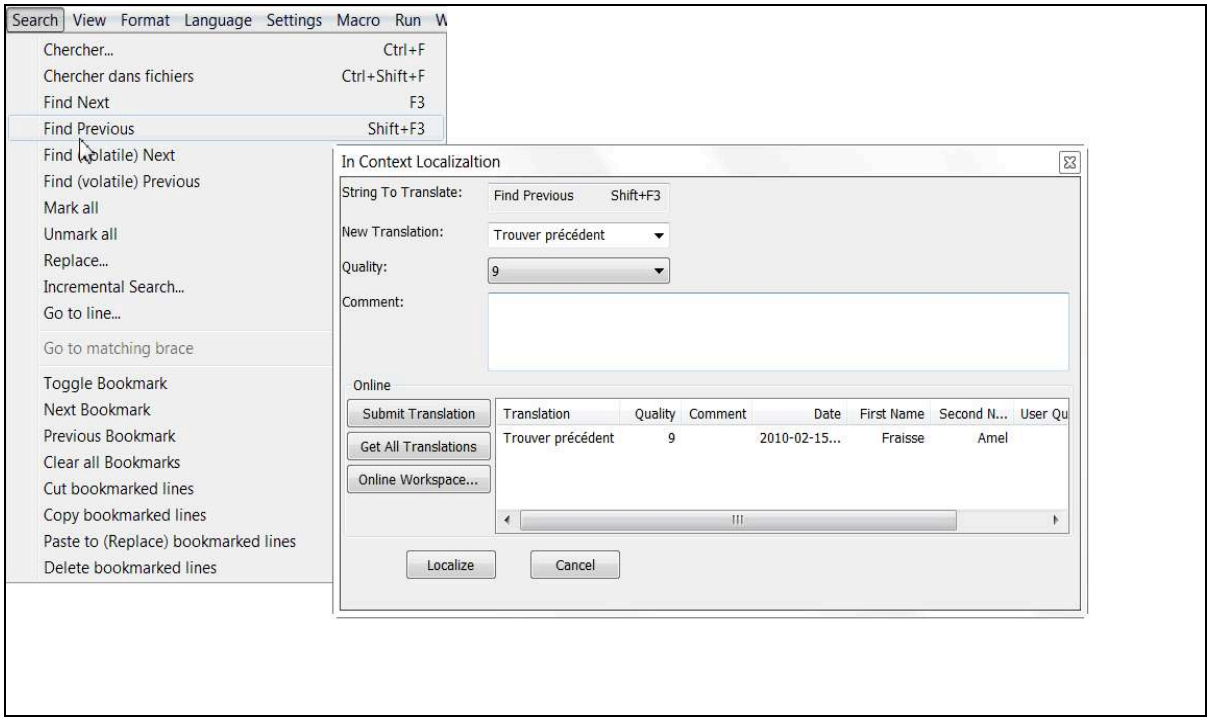


Figure 118 : *Récupération à partir de SECTra_w de toutes les propositions de traduction de la chaîne "Find Previous" du menu "Search"de Notepad-plus-plus.*

L'utilisateur peut donc choisir une proposition de traduction parmi celles qui sont disponibles ou une saisir nouvelle proposition de traduction.

- *Envoyer sa nouvelle proposition de traduction :* l'utilisateur peut envoyer sa proposition de traduction sur SECTra_w en cliquant sur le bouton "Submit Translation".

9.2.1.3 Étape 3 : mettre à jour son interface graphique

Après qu'une nouvelle proposition de traduction est saisie, pour mettre à jour son interface graphique et revenir dans son application, l'utilisateur doit appuyer sur le bouton "Localize". Il peut donc voir apparaître en temps réel sa nouvelle proposition de traduction dans son application.

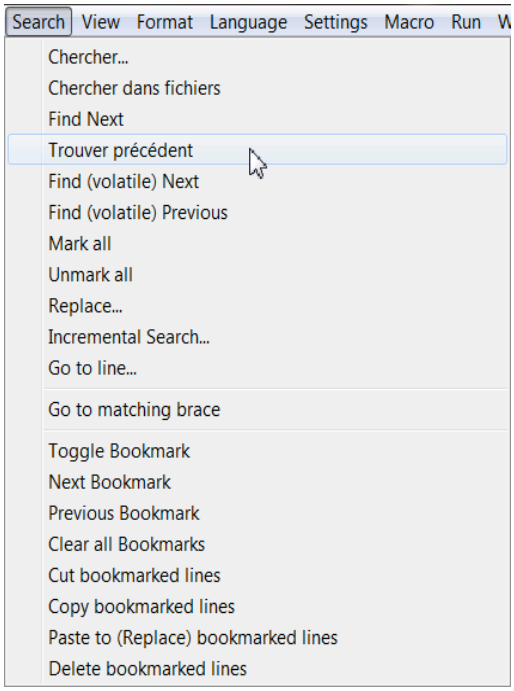


Figure 119 : *L'entrée du menu "Find previous" est remplacée par la nouvelle proposition de traduction de l'utilisateur "Trouver précédent"*

9.2.2 Scénario 2 : Localisation en contexte de Notepad-plus-plus directement sur SECTra_w

L'utilisateur peut aussi choisir de localiser directement sur le site SECTra_w puis revenir dans son application et récupérer ses propositions de traduction.

9.2.2.1 Étape 1 : éditer une chaîne de l'interface au cours d'utilisation du logiciel

Comme pour le premier scénario, durant l'utilisation de l'application, l'utilisateur peut faire un clic-droit sur n'importe quel élément textuel de l'interface graphique. Cela lui permet d'accéder en premier temps au même contexte d'édition que celui du premier scénario.

9.2.2.2 Étape 2 : post-éditer sur SECTra_w

Une fois dans le contexte d'édition, l'utilisateur peut cliquer sur le bouton "Online Workspace" pour aller éditer directement le segment source sur SECTra_w. Cela lui permet d'ouvrir directement la page de post-édition qui contient l'élément textuel à éditer (Figure 120). Sur SECTra_w, nous avons fait en sorte que chaque page de post-édition corresponde à une interface graphique. Cela veut dire que les chaînes appartenant à une même interface graphique sont affichées dans la même page de post-édition sur SECTra_w (Figure 121).

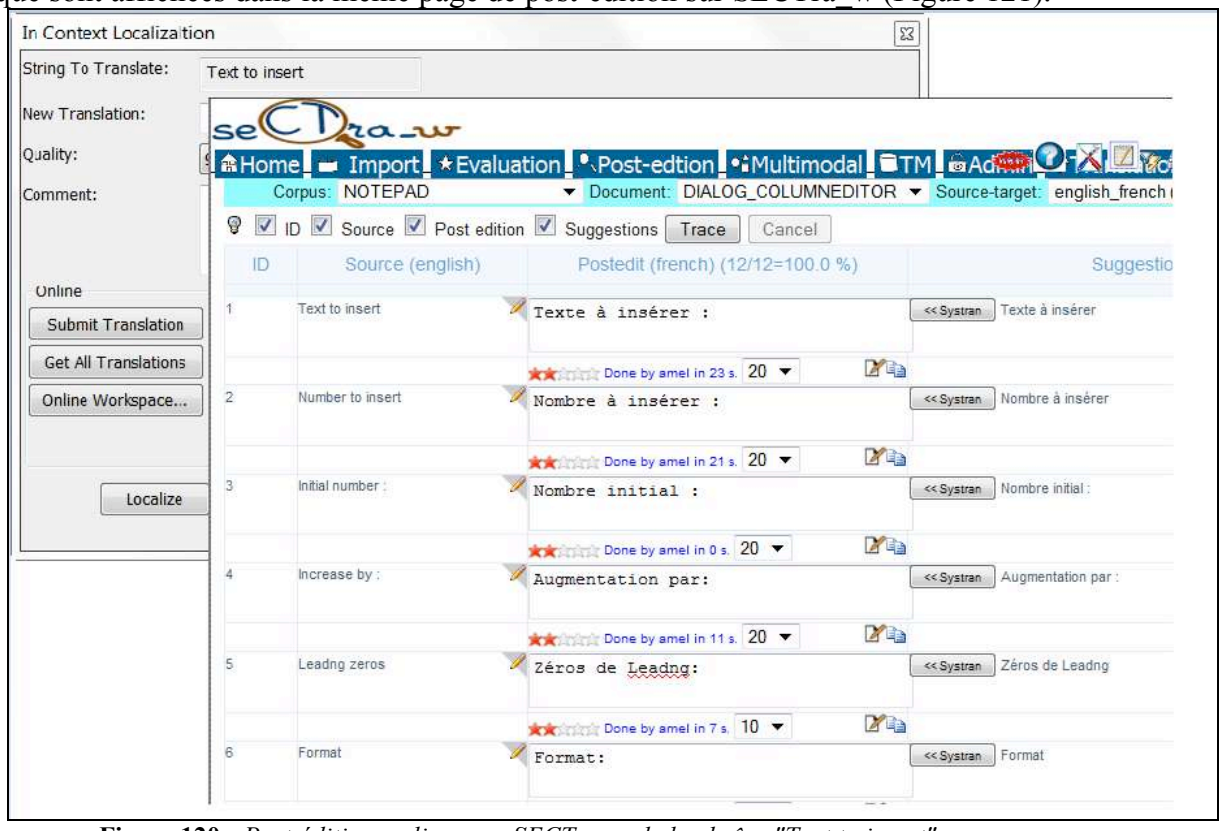


Figure 120 : *Post-édition en ligne sur SECTra_w de la chaîne "Text to insert"*

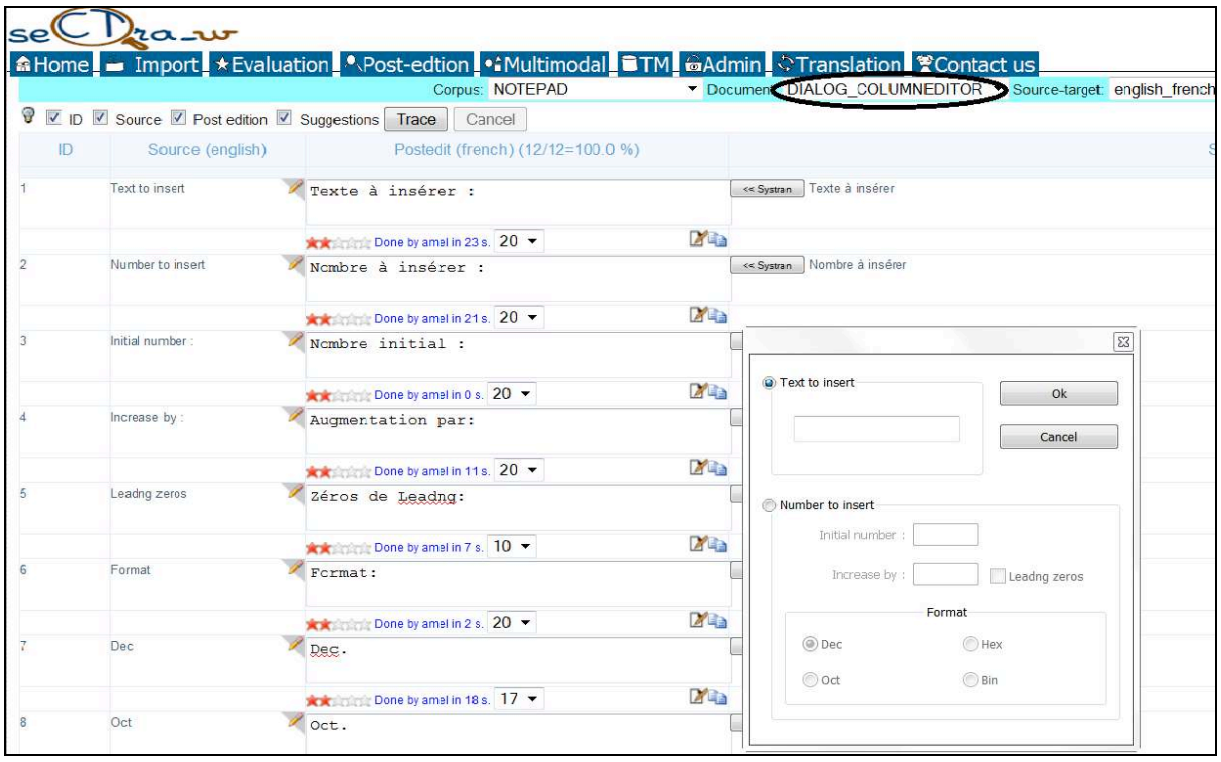


Figure 121 : *Interface de post-édition de SECTra_w contenant les chaînes de la boîte de dialogue "ColumnEditor" de Notepad-plus-plus*

L'utilisateur peut donc consulter les autres propositions de traduction, post-éditer un ou plusieurs segments source sur SECTra_w, etc.

9.2.2.3 Étape 3 : retourner à l'application et demander mise à jour des interfaces utilisateur

Une fois l'édition terminée, l'utilisateur peut donc retourner dans son application et demander la mise à jour des éléments textuels qu'il a édités sur le site SECTra_w.

9.2.3 Résultats

Notepad-plus-plus comporte environ 600 chaînes d'interface. Il est localisé initialement vers 55 langues. Nous avons choisi de le localiser en vietnamien à partir du français étant donné qu'il n'existe pas encore dans une telle version.

Les utilisateurs sollicités ont été au nombre de 3. Il s'agit de simples utilisateurs de Notepad-plus-plus qui ont comme langue maternelle le vietnamien et qui se sont portés volontaires pour participer à la localisation du Notepad-plus-plus. Nous avons donc installé sur la machine de chacun une version de Notepad-plus-plus qui est localisable *en contexte* puis nous avons créé un compte sur SECTra_w pour chacun d'entre eux.

Nous avons observé par la suite l'avancement de leurs travaux de localisation en collectant tous les soirs les nouvelles propositions de traduction et en leur demandant de nous communiquer le temps passé chaque jour sur la localisation. Nous avons pu établir les résultats illustrés dans la Figure 122.

	1er jour		2éme jour	
	Nombre d'heures passées	Nombre de segments traduits	Nombre d'heures passées	Nombre de segments traduits
Utilisateur 1	2h	80	1h	48
Utilisateur 2	15mn	25	1h30	126
Utilisateur 3	1h	47	30mn	37

Figure 122 : *Résultats de la localisation en contexte et en ligne de Notepad-plus-plus*

Ainsi sur ces 500 chaînes d'interface, 475 chaînes ont été traduites. Les 25 chaînes restantes correspondent aux messages d'erreur qui sont codées en dur dans le code source.

Conclusion

Comme exposé dans ce chapitre, nous avons pu expérimenter le processus de localisation tripartite sur le logiciel Notepad-plus-plus. Les résultats obtenus en termes de délai de localisation sont encourageants. En effet, nous avons pu générer une version Notepad-plus-plus localisée en vietnamien en seulement 2 jours. Se rajoute à cela le fait que les traductions sont de bonne qualité car elles ont été proposées d'une part par des utilisateurs qui ont le vietnamien comme langue maternelle, et en plus ils avaient accès au contexte des chaînes.

Conclusion & perspectives

Dans le cadre de cette thèse, nous avons dressé un bilan précis sur la situation actuelle de la localisation des logiciels. Cette étude nous a permis d'identifier les problèmes anciens et nouveaux qui lui sont liés et pour lesquelles il n'existait aucune solution. Actuellement, la traduction des ressources textuelles de logiciels est confiée uniquement à des professionnels. Cela rend le processus de localisation long, coûteux et quelquefois de mauvaise qualité car les traducteurs professionnels n'ont pas accès au contexte d'utilisation des éléments textuels. 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.

Notre méthode de localisation en contexte nous permet donc de :

- *Améliorer la qualité de la traduction.* Les éléments textuels des interfaces utilisateur ne sont pas extraits de leur contexte et ils sont traduits directement dans les interfaces graphiques qui les contiennent.
- *Réduire le coût de la localisation.* Notre méthode permet de faire participer plusieurs types de contributeurs bénévoles (utilisateurs finals, traducteurs bénévoles, Bêta-testeurs, etc.) dans le processus de localisation.
- *Réduire les délais de la localisation.* Le donneur d'ordres peut publier une version partiellement localisée voire non localisée qui va être totalement localisée au fur et à mesure pendant la vie du logiciel.
- *Localiser vers plus de langues cible.* Actuellement, les éditeurs de logiciels traduisent uniquement vers les langues « rentables », avec notre méthode de localisation en contexte, n'importe quel utilisateur peut localiser et utiliser le logiciel dans sa langue maternelle dans la mesure où la langue cible est prise en compte par l'application. Les utilisateurs n'ont plus à attendre longtemps avant de pouvoir utiliser le logiciel dans leur langue maternelle.
- *Améliorer la satisfaction de l'utilisateur final.* Actuellement, quand l'utilisateur détecte une chaîne mal traduite, la seule chose que l'utilisateur peut faire c'est d'envoyer un mail à l'éditeur de logiciel. Ce dernier ne peut modifier la traduction que lors de la prochaine version du logiciel. La localisation en contexte permet à l'utilisateur de mettre à jour son interface en temps réel en attendant qu'elle soit validée par le donneur d'ordres.

Nous pouvons citer les axes suivant comme perspectives de nos travaux :

- *Unification de la localisation en contexte*. Il s'agit d'unifier le traitement de la localisation de l'interface utilisateur, de l'aide en ligne et de la documentation technique, de façon à ce qu'on puisse toujours savoir prendre en compte une

modification terminologique faite sur l'un de ces éléments au niveau des 2 autres, ce qui est essentiel pour la cohérence des éléments textuels, elle-même indispensable pour que l'application n'apparaisse pas incohérente à ses utilisateurs. Il faut donc unifier la gestion des ressources linguistiques communes (mémoires de traductions et lexique), et concevoir un outillage pour l'évaluation de l'impact des contributions sur tout l'environnement.

Exemple: si l'on décidait aujourd'hui de changer l'item de menu “vertical symmetry” dans Photoshop en “horizontal symmetry”,

- que faudrait-il changer dans tout le logiciel, dans l'aide en ligne, et dans la documentation?
- devrait-on propager cette inversion dans toute nouvelle langue?

De façon plus générale, comment rendre sensible la fiabilité de la traduction des éléments textuels d'un logiciel?

- *Intégration du comportement de la localisation en contexte dans des bibliothèques graphiques libres.* Il s’agit d’intégrer ce nouveau comportement directement dans les bibliothèques graphiques libres comme Swing et AWT de façon à ce que les nouveaux logiciels qui utiliseront ces bibliothèques seront localisables en contexte.

Bibliographie

(Abekawa, 2007) Abekawa Takeshi et Kageura Kyo, *A Translation Aid System with a Stratified Lookup Interface*. In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007) Demos and Posters, 2007.

(Albatal, 2005) Rami Albatal, *La prise en compte des flux de travaux pour la construction collaborative des bases lexicales multilingues*. Mémoire de M2R, UJF (Grenoble1), 2005, 62 p, 2005.

(Augar*, et al.*, 2004) Augar Naomi, Raitman Ruth and Zhou Wanli, *Teaching and Learning Online with Wikis*. In Proceedings of the 21st Australasian Society of Computers In Learning In Tertiary Education Conference, ASCILITE-04. Australia, pp. 95-104, 2004.

(Bellynck, *et al.,* 2005) Valérie Bellynck, Christian Boitet et John Kenwright, *ITOLDU, a Web Service to Pool Technical Lexical Terms in a Learning Environment and Contribute to Multilingual Lexical Databases*. Alexander F. Gelbukh (Ed.): Computational Linguistics and Intelligent Text Processing, 6th International Conference, CICLing 2005, Mexico City, Mexico, February 13-19, 2005, Proceedings, pp. 324-332.

(Berment, 2004) Vincent Berment, *Méthodes pour informatiser les langues et les groupes de langues « peu dotées »*. Thèse UJF (Grenoble 1), 2004, 277 p.

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

(Bey, *et al.*, 2006b) Youcef Bey, Kyo Kageura et Christian Boitet, *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, *et al.*, 2007) Youcef Bey, Kyo Kageura et Christian Boitet, *BEYTrans: a Free Online Collaborative Wiki-Based CAT Environment Designed for Online Translation Communities*. PACLIC21, The 21st meeting Pacific Asia Conference on Language, Information and Computation, November 1-3, 2007, Seoul National University, Korea, 2007, 8 p.

(Bey, *et al.*, 2008a) Youcef Bey, Kyo Kageura et Christian Boitet, *BEYTrans: a Wiki-based*

Environment for Helping Online Volunteer Translators. In Topics in Language Resources for Translation and Localisation, 2008, Yuste Rodrigo, Elia (ed.), 135–150.

(Bey, *et al.*, 2008b) Youcef Bey, *Aides informatisées à la traduction collaborative bénévole sur le Web*. Thèse UJF, 2008 (Grenoble 1), 265 p.

(Blanchon, *et al.*, 2009) Hervé Blanchon, Christian Boitet et Cong-Phap Huynh, *a Web Service Enabling Gradable Post-edition of Pre-translations Produced by Existing Translation Tools: Practical Use to Provide High-quality Translation of an Online Encyclopedia*. MT Summit XII, Ottawa, Ontario, Canada, August 26-30, 2009, 8 p.

(Boitet, 1999) Christian Boitet, *a research perspective on how to democratize machine translation and translation aids aiming at high quality final output*. MT Summit VII, September 13-17, 1999, Kent Ridge Digital Labs, Singapore, 1999, pp. 125-133.

(Boitet, 2002) Christian Boitet, *a roadmap for MT : four « keys » to handle more languages, for all kinds of tasks, while making it possible to improve quality (on demand)*. International Conference on Universal Knowledge and Language (ICUKL2002), Goa, 25-29, November 2002, 8 p.

(Boitet, 2007) Christian Boitet, *Corpus pour la TA : types, tailles, et problèmes associés, selon leur usage et le type de système*. Revue française de linguistique appliquée. Vol. XII – 2007, pp. 25-38.

(Boitet, 2004) Christian Boitet, *Progress report on building the French BTEC and participating in the MT evaluation campaign (CSTAR project)*. (Rapport pour ATR), GETA, CLIPS, & ATR, 10 p, 2004.

(Boitet, 2005) Christian Boitet, *What will it take for Papillon to be concretely useful not only for humans, but for machines ?* Proc. SNLP-2005, Bangkok, Thailand, 2005, 10 p.

(Boitet, *et al.*, 2002) Christian Boitet, Mathieu Mangeot et Gilles Sérasset**,** *The Papillon project: cooperatively building a multilingual lexical data-base to derive open source dictionaries and lexicons*. Proceedings of the 2nd workshop on NLP and XML,volume 17, 2002, pp. 1-3.

(Boitet, 2003) Christian Boitet, *Messages avec variantes, automates finis contrôlés, et multilinguisme,* document interne, GETA, laboratoire CLIPS, IMAG, février 2003.

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

(Boitet, 2008) Christian Boitet, Valérie Bellynck, Mathieu Mangeot, et Carlos Ramisch, *Towards Higher Quality Internal and Outside Multilingualization of Web Sites*. In P. Bhattacharyya, editor, Summer Workshop on Ontology, NLP, Personalization and IE/IR ONII-08, page 8, IITB Bombay, India, 2008.

(Bryant*, et al.*, 2005) Bryant, Susan, Andrea Forte and Amy Bruckman, *Becoming Wikipedian: Transformation of Participation in a Collaborative Online Encyclopedia*.

In Proceedings of the GROUP International Conference on Supporting Group Work. Sanibel Island, Florida, US., 2005, pp. 1-10.

(Buffa, 2006) Buffa Michel, *Intranet Wikis*. In Proceedings of the Intranet Web Workshop (WWW Conference). Edinburgh, Scotland, UK, 2006, pp. 18-28.

(Buffa*, et al.*, 2006) Buffa Michel, Crova Gael, Gandon Fabien, Lecompte Claire et Passeron Jeremy, *SweetWiki: Semantic Web Enabled Technologies in Wiki*. In Proceedings of the 1st Semantic Wiki Workshop (SemWiki2006), pp. 69-78.

(Chenon, 2005) Christophe Chenon, *Vers une meilleure utilisabilité des mémoires de traductions, fondée sur un alignement sous-phrastique*. Thèse UJF (Grenoble 1), 28 octobre 2005, 228 p.

(Chenon, 2006) Christophe Chenon, *TransTree, a formalism to capture nested correspondences at sub-sentential level*. International Symposium on Parallel Treebanks, Stockholm, 21-22 September 2006, 8 p.

(Chen et Nie, 2000) Chen Jisong et Nie Jian-Yun, *Parallel Web Text Mining for Cross-Language Information Retrieval*. Actes du Colloque sur la Recherche d'Informations Assistée par Ordinateur (RIAO). Paris, France, 2000, pp. 62-77.

(Choumane*, et al.*, 2005) Choumane Ali, Blanchon Hervé, et Roisin Cécile, *Integrating Translation Services Within a Structured Editor*. In Proceedings of the ACM Symposium on Document Engineering (DocEng 2005). Bristol, UK. 2005. pp. 165-167.

(Crestani*, et al.*, 2002) Crestani Fabio, Girolami Mark et Van Rijsbergen C. J., *Building Bilingual Dictionaries from Parallel Web Documents*. In Proceedings of the 24th BCS-IRSG European Colloquium on IR Research: Advances in Information Retrieval. London, UK. 2002. pp. 303-323.

(Daille*, et al.*, 1994) Daille Béatrice, Gaussier Eric et Langé Jean Marc, *Towards Automatic Extraction of Monolingual and Bilingual Terminology*. In Proceedings of the 15th International Conference on Computational Linguistics (COLING'94) Kyoto, Japan. 1994. pp. 712-716.

(Daoud, 2007) Mohamed Daoud, *Vers des passerelles interactives d'accès multilingue (iMAG)*, Mémoire de Master Recherche, Université Joseph Fourier, Grenoble, France. 2007.

(Daoud, *et al.*, 2009) Mohammad Daoud, Daoud Daoud et Christian Boitet**,** *Collaborative Construction of Arabic Lexical Resources*. Khalid Choukri and Bente Maegaard (Ed.). In Proceedings of the Second International Conference on Arabic Language Resources and Tools, Cairo, Egypt. April 22-23 2009. pp. 119-124.

(Denoual, 2006) Denoual Etienne, *Méthodes en caractères pour le traitement automatique des langues*. Thèse UJF, GETA, CLIPS, IMAG, Grenoble. (Thèse préparée à ATR, Kyoto, Japan). 2006. 186 p.

(Désilets, *et al.*, 2006) Désilets Alain, Gonzalez Lucas, Paquet Sébastien et Stojanovic Marta, *Translation the Wiki Way*. In Proceedings of the WIKISym 2006. NRC 48736. Odense, Denmark. 2006. pp. 19-32.

(Doan-Nguyen, 1998) Doan-Nguyen H. *Accumulation of Lexical Sets: Acquisition of Dictionary Resources and Production of New Lexical Sets*. In Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL'98. Montréal, Canada. 1998. Vol. 1, pp. 330-335.

(Doan-Nguyen, 1998) Doan-Nguyen Hai, *Techniques génériques d'accumulation d'ensembles lexicaux structurés à partir de ressources dictionnairiques informatisées multilingues hétérogènes*. Thèse INPG, GETA, CLIPS, IMAG, Grenoble. 1998. 180 p.

(Drepper, *et al.*, 2002) Ulrich Drepper, Jim Meyering, François Pinard, and Bruno Haible, *GNU gettext tools, version 0.11.2*, Published by the Free Software Foundation, Avril 2002.

(Esselink, 2000) Esselink Bert, *a Practical Guide for Localization (2nde edition)*. John Benjamins. 2000. ISBN 1588110060.

(Everson, *et al.*, 2000) Michael Everson et Trond Trosterud, *Software localization into Nynorsk Norwegian,* rapport, 2000.

(Gotti, *et al.*, 2006) Gotti Fabrizio, Langlais Philippe et Coulombe Claude, *Vers l'intégration du contexte dans une mémoire de traduction sous-phrastique : détection du domaine de traduction*. In Proceedings of the Conference sur le Traitement Automatique des Langues Naturelles (TALN). Leuven, Belgium. 2006. pp. 483-492.

(Horn, 2005) François Horn, *Les bases graphiques: la librairie "Standad Widget Toolkit",* Rapport, 2005.

(Hutchins, 1998) Hutchins John, *The Origins of the Translator's Workstation*. Machine Translation. Vol. 13. 1998. pp. 287-307.

(Hutchins, 2003) Hutchins John, *Has Machine Translation Improved? Some Historical Comparisons*. In Proceedings of the MT Summit IX conference New Orleans, USA. 2003. pp. 181-188.

(Huynh, *et al.*, 2008) Cong-Phap Huynh, Christian Boitet et Hervé Blanchon, *SECTra_w.1: an online collaborative system for evaluating, post-editing and presenting MT translation corpora*. LREC 2008: 6th Language Resources and Evaluation Conference, Marrakech, Morocco, 26-30 May 2008. 6 p.

(Jones, *et al.*, 2006) Jones M. Cameron., Rathi Dinesh et Twidale Michael B., *Wikifying your Interface: Facilitating Community-Based Interface Translation*. In Proceedings of the 6th ACM Conference on Designing Interactive Systems. University Park, PA, USA. 2006. pp. 321-330.

(Jutras, 2000) Jutras Jean-Marc, *An Automatic Reviser: the TransCheck system*. In Proceedings of the 6th Conference on Applied Natural Language Processing. Seattle, Washington, USA. 2000. pp. 127-134.

(Kageura, 2006) Kageura Kyo, *Improving the Usability of Language Reference Tools for Translators*. In Proceedings of the 10th Annual Meeting of the Japan Association for Natural Language Processing. Japan. 2006. pp. 707-710.

(Lepouras, *et al.*, 2000) Giorgos Lepouras et George R.S Weir, *Mind your language: A study of language preference in Greek users*, International Journal of Human-Computer Studies, 2000.

(Leuf, *et al.,* 2001) Leuf Bo et Cunningham Ward, *The Wiki Way: Quick Collaboration on the Web*. Edited by Upper Saddle River, NJ, USA: Addison Wesley. 2001.

(Luong, *et al.*, 1995) Tuoc V. Luong, James S. H. Lok, David J. Taylor, Kevin Driscoll, *Internationalization: Developing software for Global Markets*, John Wiley. ISBN: 0-471-07661-9, Inc, New york, USA. 1995.

(Lustig, *et al.*, 1993) Myron W. Lustig et Jolene Koster, *Intercultural Competence – Interpersonal Communication Across Cultures*, Harper Collins College. ISBN: 0-06-044129-1, 1993.

(Mangeot, 2002) Mathieu Mangeot, *How to Import an Existing XML Dictionary into the Papillon Platform*. Proceedings of Papillon 2002 Workshop, 16-18 July 2002, NII, Tokyo, Japan. 2002. 10 p.

(Mangeot-Lerebours, *et al.*, 2002) Mangeot-Lerebours Mathieu et Sérasset Gilles *Frameworks, Implementation and Open Problems for the Collaborative Building of a Multilingual Lexical Database*. In Proceedings of the Building and Using Semantic Networks SEMANET-02 workshop (COLING). Taipei, Taiwan. 2002. pp. 9-15.

(Martin, 1990) Martin W. *User-orientation in Dictionaries: 9 Propositions*. In Proceedings of the BudaLEX´88 Conference. Budapest: Akadémiai Kiadó. 1990. pp. 393-399.

(Microsoft, 2002) Microsoft, *Developing International Software*. Microsoft Press. ISBN 0-735-61583-7. 2002.

(Murata, *et al.*, 2003) Toshiki Murata, Mihoko Kitamura, Tsuyoshi Fukui et Tatsuya Sukehiro *Implementation of Collaborative Translation Environment `Yakushite Net'*. Machine Translation Summit IX, September 23-27, 2003, New Orleans, Louisiana, USA. 2003. pp.479–482.

(Melby, 1982) Melby Alain K. *Multi-level Translation Aids in a Distributed System*. In Proceedings of the Coling 82 conference. Prague, Czech. 1982. pp. 215-220.

(Melby, 1983) Melby Alain K. *The Translation Profession and the Computer*. The Computer Assisted Language Instruction Consortium Journal. Vol. 1:(1). 1983. pp. 55-57.

(Muljadi, *et al.*, 2005) Muljadi Hendry, Takeda Hideaki, Araki Jiro, Kawamoto Shoko, Satoshi Kobayashi1, Yoko Mizuta1, Sven Minoru Demiya1, Satoshi Suzuki1, Asanobu Kitamoto, Yasuyuki Shirai, Nobuyuki Ichiyoshi, Takehiko Ito, Takashi Abe, Takashi Gojobori, Hideaki Sugawara, Satoru Miyazaki, et Asao Fujiyama1. *Semantic Mediawiki: A User-oriented System for Integrated Content and Metadata Management System*. In Proceedings of the IADIS International Conference on WWW/Internet 2005. Vol. 2, pp. 261-264.

(Nie, *et al.*, 1999) Jian-Yun Nie, Michel Simard, Pierre Isabelle, Richard Durand. *Cross-language Information Retrieval Based on Parallel Texts and Automatic Mining of Parallel Texts from the Web*. In Proceedings of the the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 1999. pp. 74-81.

(O'Donnell, *et al.*, 1994) O'Donnell et Sandra Martin. *Programming for the World: A Guide to Internationalization*, Prentice Hall. ISBN: 0-13-722190-8. 1994.

(Planas, *et al.*, 1999) Planas Emmanuel, and Furuse Osamu. *Formalizing Translation Memories*. In Proceedings of the Machine Translation Summit VII. Singapore. 1999. pp. 331-339.

(Planas, *et al.*, 2000) Planas E. and Furuse Osamu. *Multi-level Similar Segment Matching Algorithm for Translation Memories and Example-Based Machine Translation*. In Proceedings of the 18th conference on Computational linguistics (COLING-2000). Saarbruecken, Germany. 2000. Vol. 2, pp. 621-627.

(Pokorn, 2005) Nike K. Pokorn, *Translation into a non-mother* tongue. Challenging the Traditional Axioms. Benjamins Translation Library 62. 2005. 166pp.

(Sahama, *et al.*, 2004) Tony Sahama, Chris Ho-Stuart et James M. Hogan, *Developing and Delivering a Software Internationalisation Subject*. ACSW Frontiers, 199-204. 2004.

(Schmitz, 2002) Schmitz Klaus Dirk, *Lokalisierung: Konzepte und Aspekte*. Dans : Hennig und Tjarks-Sobhani (eds.), Schriften zur Technischen Kommunikation Tekom, Band 6, Lokalisierung von Technischer Dokumentation. 2002 Pages: 1-18.

(Schmitz, 2005) Schmitz Klaus Dirk, *Internationalisierung und Lokalisierung von Software*. Dans: Reineke and Schmitz (eds.), Einführung in die Softwarelokalisierung, Gunter Narr Verlag. Tübingen. 2005. Pages: 11-26.

(Sérasset, 1994) Gilles Sérasset, *SUBLIM : un système universel de bases lexicales multilingues et NADIA : sa spécialisation aux bases lexicales interlingues par acceptions*. Thèse UJF (Grenoble 1). 1994. 205 p.

(Sérasset, 1995) Gilles Sérasset *Interlingual Lexical Organisation for Multilingual Lexical Databases in NADIA*. COLING '94, 5-9 August 1994, Kyoto. Japan. 1994. pp. 278-282.

(Sérasset, 2004) Gilles Sérasset, *a Generic Collaborative Platform for Multilingual Lexical Database Development*. COLING 2004, Workshop on Multilingual Linguistic Resources, Geneva, Switzerland. Aug. 2004. pp. 73-79.

(Sérasset, 2006) Gilles Sérasset, *Multilingual legal terminology on the Jibiki platform: the LexALP project*. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Sydney, Australia. 2006, pp. 937-944.

(Souphavanh, 2005) Anousak Souphavanh & Theppitak Karoonboonyanan, *Free/Open Source Software Localization*, ISBN: 81-8147-754-5. 2005.

(Symmonds, 2002) Nick Symmonds, *Internationalization and Localization Using Microsoft.NET*, Apress. ISBN: 1-59059-002-3. 2002.

(Uren, *et al.*, 1993) Emmanuel Uren, Robert Howard et Tiziana Perinotti, *Software Internationalization and Localization: An Introduction*, Van Nostrand Reinhold. ISBN: 0-442-01498-8. 1993.

(Vo-Trung, 2004) Vo-Trung Hung, *Méthodes et outils pour utilisateurs, développeurs et traducteurs de logiciels en contexte multilingue*, thèse d'informatique, Institut national polytechnique de Grenoble. écembre 2004.

(Vo-Trung, 2004) Vo-Trung Hung, *Réutilisation de traducteurs gratuits pour développer des systèmes multilingues*. In Proceedings of the Conférence Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL). Fès, Maroc. 2004. pp. 111-117.

(Wagner*, et al.*, 1974) Wagner Robert A. and Fisher Michael J., *The String-to-String Correction Problem*. Journal of the ACM (JACM). Vol. 21:(1). 1974. pp. 168-173.

(Weir, *et al.*, 2001) George R.S Weir et Giorgos Lepouras, *Localisation and linguistic anomalies*. Universal Access In HCI: Towards an Information Society for All, Proceedings of HCI International 2001 (the 9th International Conference on Human-Computer Interaction), New Orleans, USA. 2001. Volume 3.

Netographie

(Adobe, 2007): Adobe, http://stlab.adobe.com/group__asl__tutorials__eve.html
(Microsoft, 2008a): Microsoft, http://msdn.microsoft.com/fr-fr/library/sxdy04be.aspx
(Microsoft, 2008b): Microsoft, http://msdn.microsoft.com/fr-fr/vstudio/cc138251.aspx
(Xeotrope, 2008) : Xeotrope, http://www.xoetrope.com/wiki/InSituTranslation
(Mozilla, 2009) : Mozilla,
http://www.mozilla.com/enUS/firefox/all.html?langsearch=&x=6&y=8
(EMC, 2009): EMC Content Management & Archiving,
http://www.emc.com/utilities/globalsiteselect.jhtml?checked=true
(Symantec, 2009): Symantec, *Community Translation of threatFire and Firewall Plus,*
http://together.pctools.com/
(Mozilla, 2006): Mozilla*, Localization and Leveraging tools*,
http://wwwarchive.mozilla.org/projects/l10n/mlp_tools.html
(CANONICAL LTD., 2008): CANONICAL LTD., *LoCoTeamList*,
https://wiki.ubuntu.com/LoCoTeamList
(CANONICAL LTD., 2009): CANONICAL LTD., *LaunchPad Translation*,
https://translations.launchpad.net/
(VISTAWIDE, 2009): VISTAWIDE,
http://www.vistawide.com/languages/top_30_languages.htm
(Inde Population, 2008): Inde Population,
http://www.indexmundi.com/fr/inde/population.html
(Ethnologue Estimation, 2005) : Ethnologue Estimation ,
http://www.ethnologue.com/ethno_docs/distribution.asp?by=size
(Wikipedia, 2010): Wikipedia,
http://fr.wikipedia.org/wiki/Localisation_%28informatique%29
(Sun, 2008) : Java ™ Platform, Standard Edition 6 API Specification
http://java.sun.com/javase/6/docs/api. 2008.
(Linus, 2005): Linux documentation translation. Traduc project. http://traduc.org/.
(Mozilla, 2006): Mozilla, *Mozilla project & Mozilla French Localization project*,
http://frenchmozilla.online.fr/
(Sun, 2008): Sun Micosystems Inc., *Building International Applications, documentation Sun*
http://docs.sun.com/db/doc/806-6663-01ee,
(Microsoft, 2008): Microsoft, *Win32 and COM Development,* http:// msdn.microsoft.com/en-us/library
(OASIS, 2001) OASIS, *XML Localisation Interchange File Format*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xliff
(Sourceforge, 2010) Sourceforge, http://.sourceforge.net/
(Mozilla, 2006) Mozilla, *Mozilla Localisation Project*,
http://www.archive.mozilla.org/projects/l10n/
Canoncal LTd (2009): Canoncal LTd, *Launchpad Translation in Launchpad*,
https://launchpad.net/rosetta.

(Eclipse, 2008): Eclipse, *Eclipse documentation*: http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/overview-summary.html.
(TRADOS, 2008) TRADOS**,** http://www.trados.com/en/.
(Google-Translation, 2009) Google-Translation**,** http://www.google.fr/language_tools?hl=fr.
(Jibiki, 2009) Plate-forme Jibiki**,** http://ligforge.imag.fr/projects/jibiki/.
(Papillon, 2009) Papillon**,** *Projet Papillon,* http://www.papillon-dictionary.org/.
(IToldU, 2009) IToldU**,** *IToldU,* http://domus.grenet.fr/itoldu/index.html.
(Yakushite.Net, 2008**) Yakushite.Net,** http://www.yakushite.net/.
(Humanitarian-Translation, 2006), un site de traductions humanitaires, *http://seattlepi.nwsource.com/local/184671_redcross03.html*.
(Lisa, 2008), *the Localization Industry Standards Association (LISA)*, *http://www.lisa.com/*.
(MTPostEditing, 2008*), la post-édition selon Jeff Allen*, *http://www.geocities.com/jeffallenpubs/*.
(Paxhumana, 2006), Paxhumana, *http://paxhumana.info*.
(Similis, 2005), Similis, *http://www.lingua-etmachina.com/*.
Termbase (2006), une base terminologique FR-EN des termes Internet, *http://home.uchicago.edu/~kuzmack/dictionary/*.
(Traduct, 2006), Traduct, *http://www.traduc.org*.
(Translationwiki, 2006), Translationwiki, *http://www.translationwiki.com*.
(Wiktionary, 2007), Wiktionary, *http://www.wiktionary.org*.
(XWiki, 2006), XWiki, *http://www.xwiki.com*

Annexe

Guide de localisation interne du code source d'une application

1. Télécharger le code source de l'application
2. Compiler le code source
3. Identifier les classes génériques des IHM : il faut parcourir toutes les classes de l'application afin d'identifier les classes de base produisant les IHM. Généralement, pour aller plus vite, nous pouvons se baser sur la documentation développeur si elle existe ainsi que les noms des classes ou des packages qui peuvent donner une idée sur leurs contenus.
4. Choisir un évènement déclencheur pour l'édition des éléments textuels des interfaces utilisateur : il faut choisir un événement qui n'est pas utilisé par l'application (généralement les événements qui sont composés de trois combinaisons par exemple : Ctrl+alt+clic-droit).
5. Créer le fichier XLIFF dans le répertoire dans l'application (Initialement le fichier XLIFF est vide, il contient uniquement la déclaration XML).
6. Surcharger les classes génériques des IHM
    a. Intercepter l'événement choisi dans 4 pour rajouter la mise en édition des éléments textuels
    b. Rajouter la mise à jour en temps réel des chaînes d'interfaces
7. Rajouter le module LocaInContext dans le package contenant les classes génériques des IHM.
8. Recompiler l'application
9. Tester la localisation en contexte des éléments textuels des interfaces graphiques

Résumé

Nous proposons une méthode novatrice pour permettre la localisation en contexte de la majorité des logiciels commerciaux et libres, ceux programmés en Java et en C++/C#. 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 aboutit à une mauvaise qualité car les traducteurs professionnels n'ont pas accès au contexte d'utilisation des éléments textuels. Dès que l'on sort du petit ensemble des quelques langues les mieux dotées, et que lon 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.

Notre méthode consiste à faire participer de façon efficace et dynamique les bêta-testeurs et les utilisateurs finals au processus de localisation : pendant qu'ils utilisent l'application, les utilisateurs connaissant la langue originale du logiciel (souvent mais pas toujours l'anglais) peuvent intervenir sur les éléments textuels d'interface que l'application leur présente dans leur contexte d'utilisation courant. Ils peuvent ainsi traduire en contexte les boutons, les menus, les étiquettes, les onglets, etc., ou améliorer la traduction proposée par des systèmes de traduction automatique (TA) ou des mémoires de traductions (MT). Afin de mettre en place ce nouveau paradigme, nous avons besoin d'intervenir très localement sur le code source du logiciel : il s'agit donc aussi d'un paradigme de localisation interne.

La mise en place d'une telle approche de localisation a nécessité l'intégration d'un gestionnaire de flot de traductions « SECTra_w ». Ainsi, nous avons un nouveau processus de localisation tripartite dont les trois parties sont l'utilisateur, l'éditeur du logiciel et le site collaboratif SECTra_w. Nous avons effectué une expérimentation complète du nouveau processus de localisation sur deux logiciels libres à code source ouvert : Notepad-plus-plus et Vuze.

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 sometimes 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 the source language of the software (Often but not always English) could modify strings of the user interface presented by the application in their current context. So, users could translate in context buttons, menus, labels, tabpage, etc. 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.

The implementation of such approach of localization required integration of a translation workflow built with SECTra_w. Thus, we have a new tripartite process of localization which parties are: the user, the software editor and the collaborative SECTra_w Web site. We have experimented our approach on Notepad-plus-plus and on Vuze, two open source applications.