Open Source Software Localization
Process Report

1. Introduction 3
2. OSS Localization Objective 3
3. OSS Localization Team Structure 3
   3.1. Developers 3
   3.2. Linguists 3
   3.3. Project Coordinator 3
4. OSS Localization Process 4
   4.1. Selection of OSS to Localize 4
      4.1.1. Encoding support: 4
      4.2.2. Localization support: 4
      4.3.3. Cross-platform support: 5
   4.2. Selection of Localization Tools 5
      4.2.1. OmegaT (Translation Memory Application): 5
   4.3. Localization Registration 7
   4.4. Translation Resources 7
   4.5. String Extraction 7
   4.6. Translation Process 7
      4.6.1. Initial translation: 8
      4.6.2. New terminology lookup: 8
      4.6.3. Glossary extension meeting: 8
      4.6.4. Translation completion: 8
   4.7. Translation Review and Incorporation 9
   4.8. Localization Testing and Quality Assurance 9
   4.9. Final Localized Product 10
5. Translation Issues 10
   5.1. Translation Selection 10
   5.2. Translation Consistency 11
   5.3. Distinct Translation Mapping 11
   5.4. Translation Rate 11
   5.5. Translation Error Corrections 11
6. Case Study – SeaMonkey (ur-PK) Localization 11
   6.1. OSS Selection 12
   6.2. Localization Tools 12
   6.3. Localization Registration 12
   6.4. Translation Resources 12
      6.4.1. Core Glossary: 12
      6.4.2. Mozilla Suite 1.5 ur-PK glossary: 12
      6.4.3. Firefox 1.0.6 ur-IN glossary: 12
      6.4.4. Thunderbird 1.0.7 ur-IN glossary: 13
      6.4.5. OpenOffice 2.0.3 ur-IN glossary: 13
      6.4.6. Online technical terminology translations (English to Urdu): 13
      6.4.7. Dictionaries: 13
      6.4.8. Miscellaneous: 13
   6.5. String Extraction 13
   6.6. Translation Process 13

6.7. Translation Review and Incorporation 13
6.8. Localization Testing and Quality Assurance 14
6.9. Final Localized Product 14
7. Conclusion 14

1. Introduction

This report describes a process for the localization of Open Source Software (OSS). The process is currently being used for the Urdu localization of OpenOffice, SeaMonkey and NVu. The process is described in a generic manner and can be customized and used for the localization of a variety of OSS. A team structure is also proposed for localization. At the end, an OSS localization case study (Urdu localization case study) using the described process is also presented.

2. OSS Localization Objective

Access to and generation of information through Information and Communication Technologies (ICTs) are yielding massive benefits, but since the medium of most of this information is English, it cannot be utilized by people who do not speak, read or write English. To bridge this digital divide and to truly harness the potential of ICTs a primary step is to localize available technology in indigenous languages. Localization of software involves adapting it for use by a particular culture. The core of any culture is its language so language adaptation is a major part of the localization process. Localization of OSS can play a vital role in making ICTs available for indigenous populations because once the technology is available, it will support both the generation of content in local languages and access to it as well.

3. OSS Localization Team Structure

This section describes the proposed team structure recommended for use with the localization process.

3.1. Developers

Developers will have the responsibility of all the technical aspects of the localization such as surveying localizable software, isolating the items of the software that need to be localized, looking into all aspects of the software being localized and creating builds, installers and language packs for the localized products. Developers will also be responsible for managing the translations (this includes assigning translations to linguists) and should also have some management skills. Along with the technical skills necessary for these tasks, it will be advantageous for developers to have some language skills (both in the source and target language) as well since the core effort of localization goes into translation. Developers should be able to assist in the interpretation and translation of technical terminology. Developers will also be responsible for preliminary testing (final testing will be done by end-users) of the localized product.

A minimum cumulative team of two developers is recommended for the localization of one application, so that a trained backup is available at all times. This number may change according to the size and complexity of the application being localized or the time available for localization as required.

3.2. Linguists

Linguists will have the responsibility of translating all the content of the application into the target language. As the translations are of a technical nature, and Computer Aided Translation (CAT) tools will be used during the process, it is essential for linguists to have at least some experience using computers and a variety of software. Translators will also have to assist in the testing of the localized product.

One linguist is recommended for the localization of one application. This number may change, but it is recommended that for every developer there is one linguist. Additional linguists may be required for translating application help, documentation and for developing additional training material for users. I think this is not sufficient.

The number of linguists can vary according to the size of the application being localized, although a minimal cumulative team of two linguists per application is recommended to ensure the availability of a trained backup at all times and for consultation purposes.

3.3. Project Coordinator

One project coordinator is required to oversee the deployment of the localized application and facilitation of it's usage by end-users. The coordinator will also be required to oversee the development any additional training

material that is to be developed for the application. The coordinator should be of a technical background, with leadership, management and communication skills, and good linguistic skills in both source and target languages will be advantageous.

4. OSS Localization Process

This section describes the basic OSS localization process as illustrated in Figure 1.

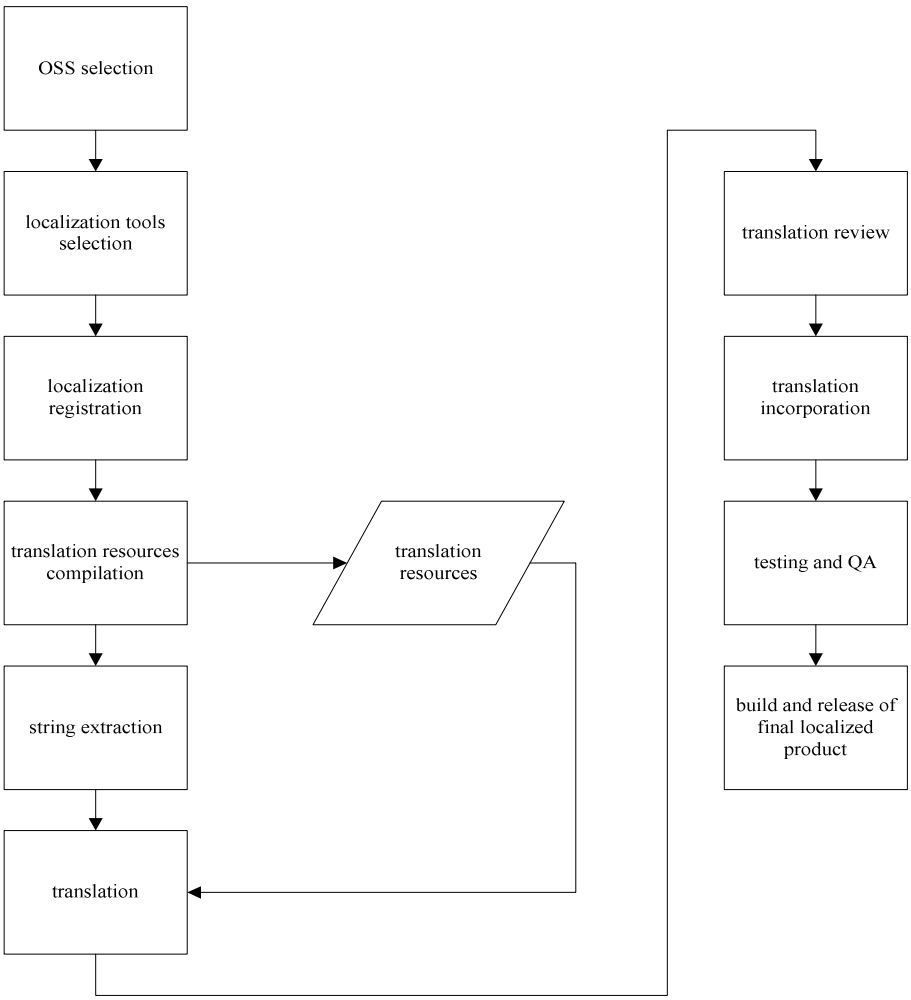

Figure 1: OSS Localization Process Flow Diagram

4.1. Selection of OSS to Localize

After the type of software that has to be localized is decided upon (e.g., a word processor, a web browser etc.), open source software of that type needs to be selected. At this stage three points need to be taken into consideration, as described in the following sections.

4.1.1. Encoding support: The application should support the character set encoding required by the language that it is being localized for. In case the localization language text direction is right-to-left, it is also necessary to ensure that the application provides adequate support.

4.2.2. Localization support: A variety of technology is available using which localizable open source software is being developed; examples are GNU gettext based applications (http://www.gnu.org/software/gettext/), XUL (extensible user interface language) based applications (http://www.mozilla.org/projects/xul/) and Qt based (http://www.trolltech.com/products/qt/) applications. These technologies enable the development of an application in a manner such that the localizable parts of the application (mainly the strings that are to be translated) are isolated from the rest of it and can be modified easily. A large selection of open source software

based on these technologies is available and can be readily localized. Some GNU gettext based applications include OpenOffice (http://www.openoffice.org/) an office suite, VLC Media Player (http://www.videolan.org/vlc/), Enigma (http://www.nongnu.org/enigma/) a puzzle game. Examples of XUL based applications are Firefox (http://www.mozilla.com/en-US/firefox/) a browser, Thunderbird (http://www.mozilla.com/en-US/thunderbird/) an e-mail client, SeaMonkey (http://www.mozilla.org/projects/seamonkey/) an Internet suite and NVu (http://www.nvu.com) a web development application. An example of a Qt based application is Psi (http://psi-im.org/) an instant messaging/chat client. The effort that goes into localizing the three types of applications is basically the same; all provide a specially formatted file containing the strings that need to be translated.

4.3.3. Cross-platform support: Selecting software that is supported across multiple platforms is also advantageous, since in this case the localization effort that goes into localizing the application for one platform does not have to be repeated for other platforms, the majority of it will be re-usable. The XUL and Qt technologies mentioned above also support the development of applications for multiple platforms, so applications developed using this should be multi-platform.

All the applications (OpenOffice, SeaMonkey, NVu, Firefox, Thunderbird, Psi, VLC Media Player and Enigma) mentioned in this section support Unicode, are localizable and are available for Windows, Linux and Mac OS X, and are therefore good candidates for localization.

4.2. Selection of Localization Tools

As noted earlier, XUL, GNU gettext and Qt based software provide files containing the strings to be translated. Different tools are available that can aid in the translations for these files and also in building the final localized product. For example, for localization of Qt based applications, Qt Linguist (http://www.trolltech.com/images/products/qt/qt-linguist/view) is used to translate the strings in the applications and to create an installable language pack. For localization of GNU gettext based applications, poEdit (http://www.poedit.net/) may be used.

These tools help in translation management, and also provide support for maintaining translation consistency. However, it should be noted that when multiple applications are being localized, consistency must also be maintained across the application set. For example, the term “file” is very common in software applications. When localizing an application, it must be ensured that for every time it occurs, it is translated in the same manner. However, when a set of applications is being localized, it is necessary to ensure some level of consistency across the application set as well. A mechanism is required where every member of the translation team has access to all the translations. For this purpose it would be convenient if all the translators are using the same tool for translation which facilitates exchange of the translations as well.

A very useful tool that can be used for conversion between the different localization file formats is translate-toolkit (http://translate.sourceforge.net/). There are also localization management systems online using which translations can be managed on the web, e.g, Pootle (http://pootle.wordforge.org/) and Launchpad (https://launchpad.net/).

A survey of several localization tools indicates that these tools would have to be utilized at two levels. At the first level, the tool used would be specific to the application being localized, e.g., if SeaMonkey is being localized, Mozilla Translator (http://sourceforge.net/project/showfiles.php?group_id=18842) can be used by the developer for management of the strings that need to be translated. At the second level, where the actual translation occurs, a common application would be used by all the linguists (who may be working on the same or different applications). This application should basically be a translation memory application, which should keep track of the translations done, and should also allow the use of pre-existing translations or glossaries. A good candidate for this purpose is OmegaT (www.omegat.org) and is described in the next section.

4.2.1. OmegaT (Translation Memory Application): OmegaT (www.omegat.org) is an open source translation memory tool written in Java that can be used with Windows, Linux or Mac OS X. It has been selected because it fulfills most translation management requirements.

First and foremost, being a translation memory application, it maintains a translation memory of all the translations done by the linguists in TMX (Translation Memory eXchange) files. TMX is an XML standard for the exchange of translation memory between different translation memory and CAT applications. OmegaT is a single user application, so linguists have to be provided with an updated translation memory on a daily basis (or

as required), which they add to their OmegaT projects (as external translation memories). As a result, all linguists have access to all the translation memory that builds up over the course of time. Matches from external translation memories are labeled, so each linguist can easily identify the author of translations that come up as matches. This also facilitates an informal peer review process, since in this way linguists are able to critique each others work as well, and may detect translation errors that could possibly pass through the localization process undetected.

Secondly, OmegaT allows the addition of external glossaries, which is necessary incase there are pre-existing glossaries or translations that can be used.

Figure 2 shows a sample OmegaT project for English to Urdu translation. Four source files for translation, five glossaries and three external translation memories (in TMX format imported from other OmegaT projects; translation memories in TMX format from other translation memory applications may also be imported into OmegaT projects). One source file has been opened for translation, and a string "Minimum font size" has been selected (in the window on the left). As soon as a string is selected, matches from the translation memories and glossaries are shown in the windows on the right.

The top window titled "Fuzzy Matches" shows fuzzy matches (only those that have a match of over 30%) from external and internal translation memories. The "Fuzzy Matches" window shows five matches, all of which are from the external translation memories. The translation memories have been named after the linguist they were obtained from, and this name can be seen at the end of each match along with the match percentage. Matches not included in the "Fuzzy Matches" window (that fall below the 30% match criteria) can be accessed using "Search Project" from the "Edit" menu.

The bottom window on the right shows matches from the glossary, along with the name of the glossary where the match was found.

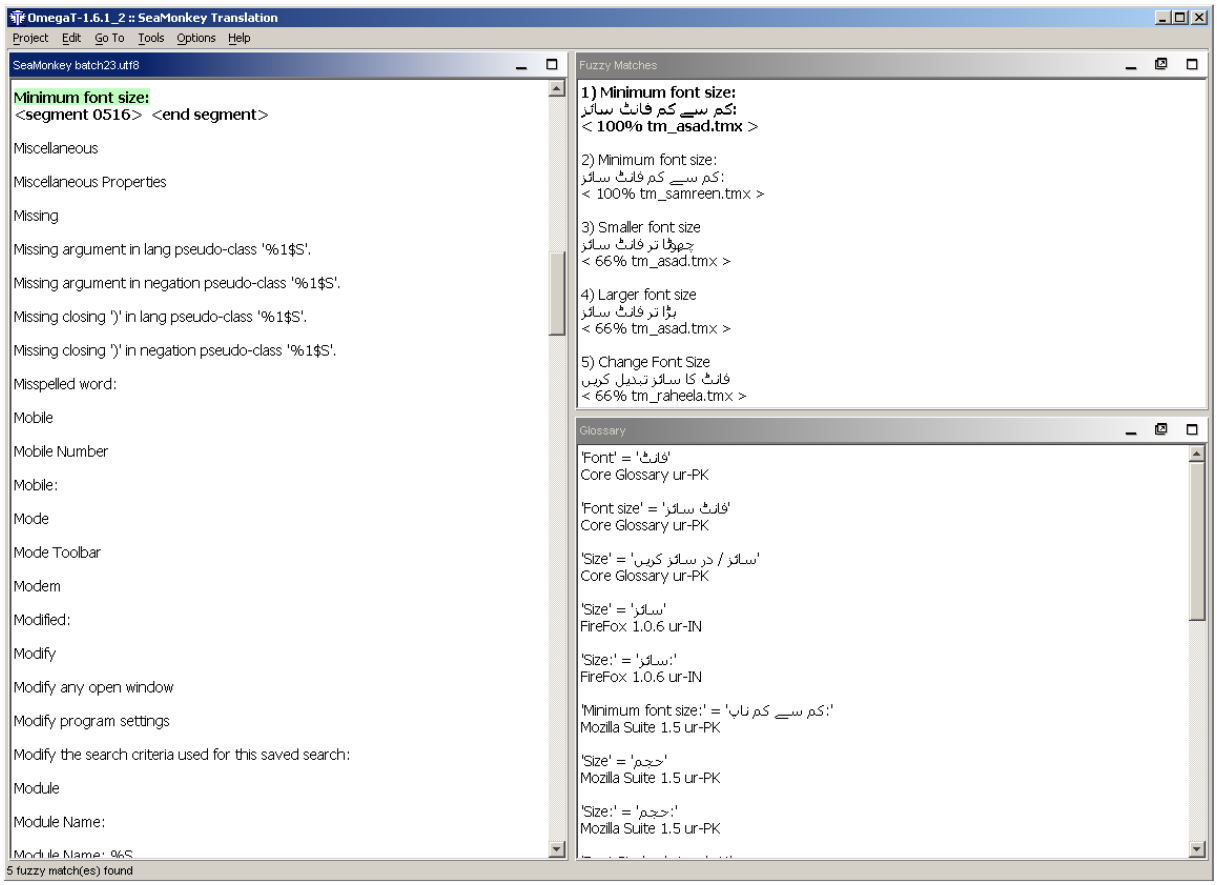
OmegaT-1.6.1_2 :: SeaMonkey Translation
Project Edit Go To Tools Options Help
SeaMonkey batch23.utf8
Minimum font size:
<segment 0516> <end segment>
Miscellaneous
Miscellaneous Properties
Missing
Missing argument in lang pseudo-class '%1$S'.
Missing argument in negation pseudo-class '%1$S'.
Missing closing ')' in lang pseudo-class '%1$S'.
Missing closing ')' in negation pseudo-class '%1$S'.
Misspelled word:
Mobile
Mobile Number
Mobile:
Mode
Mode Toolbar
Modem
Modified:
Modify
Modify any open window
Modify program settings
Modify the search criteria used for this saved search:
Module
Module Name:
Fuzzy Matches
1) Minimum font size:
کم سے کم فانٹ سائز:
< 100% tm_asad.tmx >
2) Minimum font size:
کم سے کم فانٹ سائز:
< 100% tm_samreen.tmx >
3) Smaller font size
چھوٹا تر فانٹ سائز
< 66% tm_asad.tmx >
4) Larger font size
بڑا تر فانٹ سائز
< 66% tm_asad.tmx >
5) Change Font Size
فانٹ کا سائز تبدیل کریں
< 66% tm_raheela.tmx >
Glossary
'Font' = 'فانٹ'
Core Glossary ur-PK
'Font size' = 'فانٹ سائز'
Core Glossary ur-PK
'Size' = 'سائز / در سائز کریں'
Core Glossary ur-PK
'Size' = 'سائز'
FireFox 1.0.6 ur-IN
'Size:' = 'سائز:'
FireFox 1.0.6 ur-IN
'Minimum font size:' = 'کم سے کم ناپ:'
Mozilla Suite 1.5 ur-PK
'Size' = 'حجم'
Mozilla Suite 1.5 ur-PK
'Size:' = 'حجم:'
Mozilla Suite 1.5 ur-PK
5 fuzzy match(es) found

Figure 2: Sample OmegaT project

OmegaT also provides Unicode (UTF-8) support and bidirectional support for right-to-left languages.

OmegaT is designed for professional translators and supports a large variety of file formats for translation including plain text, HTML, OpenDocument and PO files (see OmegaT documentation for more details). In the localization process that is described here OmegaT is used with plain text files. As a result, translations are assigned to linguists in the form of UTF-8 encoded text files, with a .utf8 extension, and are translated using OmegaT which generates a UTF-8 encoded text file of the translations.

4.3. Localization Registration

The next step is to register your OSS localization. Registration procedures vary and may be as simple as sending an e-mail. If someone else has already been working on that localization, the locale will be registered already and in that case some sort of collaboration will have to be set up. At this point, it is also recommended to join relevant mailing lists and forums for the application and its localization. This will help in keeping up-to-date with the development of the application itself (latest versions, bug fixes, release information etc.), and also localization issues. This is also essential since some applications may have a release schedule to be followed if you intend for your localization to be an official part of their release. These requirements vary from application to application, therefore it is essential to keep in touch and up-to-date with the development and localization efforts (of other languages) of the application being localized.

4.4. Translation Resources

A survey of all translation resources that are available for the localization language should be conducted and these should be made available for the translators. If possible, they should be incorporated into the translation memory tool being used (OmegaT for this process). Possible resources for translations are: any official technical terminology compilation, dictionaries, localizations of previous versions of the application, localized versions of other applications etc. The actual translation is a major and time-consuming effort so any previous work that has been done should be used for efficiency and consistency. It is also recommended to have a core glossary and to use the resources as references, where additions can be made to the core glossary. The core glossary may also be based on a resource, e.g., an official terminology translation compilation may be considered the core glossary, and it may be extended as required using the rest of the translation resources.

4.5. String Extraction

In the string extraction process the developer extracts the strings from the application being localized that need to be translated and assigns them to linguists. Strings are extracted and divided into batches for better management. Each batch may contain about 600-700 words. The number of strings in each batch will vary according to the number of words per string. Each batch is handed over to the linguist as a UTF-8 encoded text file, where strings are separated by line breaks. A linguist can normally complete the translation of four batches in about a week. Strings to be translated come from three sources in the application: 1) the GUI, 2) the application help, and 3) any other application documentation.

4.6. Translation Process

Each linguist has an OmegaT project for translation of the batches assigned to him/her (each linguist has a separate project). Each file to be translated is added to the project. The OmegaT project contains a core glossary that is to be used for translation, any available reference glossaries and also the translation memory of all the linguists in the team (updated on a daily basis or as required).

For translation purposes each word in a string first has to be classified as either a functional or a content word. All nouns, verbs, adjectives and adverbs are functional words; words that fall into any other category, e.g., prepositions, conjunctions etc. are functional words. For each string to be translated, the general rule to be followed is that the translation of functional words is left to the discretion of each individual linguist, but translations of content words are to be taken from the core glossary only (which is developed with the mutual consent of linguists and developers). For example, in the following strings, the content words are in bold: "**Failed** to **remove** this **account**."; "**Filters associated** with this **folder** will be **updated**."; "**Horizontal scrolling**"; "**New languages** can be **configured using** the **Languages Panel**."

Keeping the above rule in mind, the linguists proceed with the translation of these batches in four stages as described in the following sections, and shown in Figure 3. (Each linguist is assigned four batches per week. Updated translation memories from the OmegaT projects of each linguist are provided to the others on a daily basis, and translation memory backups for each linguist are also taken on a daily basis)

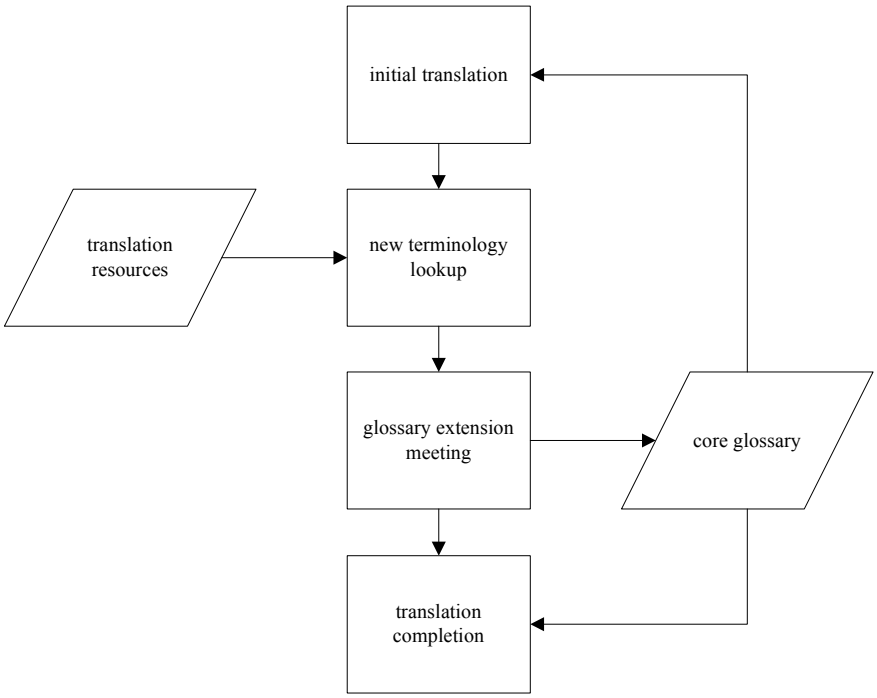

Figure 3: Translation process flow diagram

4.6.1. Initial translation: First, the linguists go through all four batches, translating those strings for which all content words have appropriate entries in the core glossary. When linguists come across a string which has a content word which is not included in the core glossary, they skip that string and make an entry for that word (including the word itself, the sentence in which it occurred, and its part-of-speech within that sentence) in a new terminology list that is compiled by each linguist each week. This activity should take about two days.

4.6.2. New terminology lookup: After the translation stage, the linguists go through their new terminology lists, looking up appropriate translations for the new terms they have identified that need to be translated. Linguists have access to the resources described in Section 4.4 during this step, and developers are also consulted when the context of a term cannot be determined. Developers may be required to locate a string as it occurs in the application and look up appropriate application documentation to determine how terms should be translated. At the end of this activity, each linguist has compiled a list of new terms that needs to be translated, along with a suggested translation for each. Alternate translations are also listed in case the suggested one is not acceptable. This activity should take about one day, although variations may occur depending on the coverage of the core glossary being used. It is expected that as the glossary is extended, the time taken for this activity will start to decrease.

4.6.3. Glossary extension meeting: When the new terminology lists have been compiled by the linguists, a meeting is held that is attended by linguists and developers. The purpose of this meeting is to go through the lists compiled by the linguists, finalize the new translations and add them to the core glossary. For each word, if the translation suggested by the linguist who reported it is accepted by the team it is added to the core glossary. If there is a disagreement, then it is resolved through discussion and possibly additional lookup activity. Issues raised can be from both linguistic and technical perspectives. From the linguistic perspective, a more appropriate translation may be suggested, and from the technical perspective incorrect senses and parts-of-speech for words used during translation may be identified. It is expected that as the glossary is extended, the time taken for this activity will start to decrease. This activity takes about one day and at the end of it the updated core glossary is provided to the linguists. Any other updations in the core glossary (fixing errors identified in the past few days) can be made during this meeting.

4.6.4. Translation completion: When the linguists receive the updated core glossary, the content words of all the batches that have been assigned to them for the week should have appropriate entries in the core glossary. Using the updated core glossary, the linguists complete the translations assigned to them for the week. This activity takes about one day, at the end of which the linguists hand back the files containing the translations to the developer.

The translation process described is flexible. For instance, during the translation step, if a linguist comes across a string from which one word is missing and has to be added to the glossary during a glossary extension meeting, the linguist may choose to translate the string and leave a placeholder for the missing word, or leave the translation of the string till the translation completion stage. Apart from this, while the process described includes one meeting per week, the meeting may be held every two weeks or as required, as long as at least four batches (as defined earlier) are completed per linguist each week.

4.7. Translation Review and Incorporation

For each file that developers assign for translation, a corresponding file is received which contains the translation of each string (separated by line breaks). The developers align the source and translation files and then review the translations. Any errors found are corrected, possibly after consultation with the relevant linguist. It may be necessary for the linguist to make the correction in the OmegaT project (where the translation was done) as well, since the error will also exist in the linguist's translation memory and, if left as is, may be repeated during translation memory usage. After the translations have been reviewed and finalized by the developer, they are incorporated into the application.

An additional task that is to be done during the incorporation phase is assignment of control and accelerator keys. These are shortcut keys for menus and menu items indicated to the user by underlining a character in a menu or menu item. For example the "File" menu in most applications has the "F" underlined, and it can be accessed by pressing Alt+F, in this case "F" is the accelerator key, and has to modified accordingly during translation incorporation. An example of a control key is Ctrl+S for the "Save" item (in the "File" menu), where the "S" is underlined. Control and accelerator keys both need to be set appropriately according to the translations.

4.8. Localization Testing and Quality Assurance

After translations have been incorporated into the application, developers check the applications to identify any problems with the localization, e.g., placeholders in strings not being displayed as expected. An example of the of this is shown in Figures 4, where the source string to be translated is "The web site %S does not support encryption for the page you are viewing.". Here "%S" is a placeholder, and may be misplaced during translation and review, as shown in Figure 4. The string inserted for the placeholder "www.google.com.pk" is appearing at an incorrect position. Errors of this type can occur due to linguistic (lack of knowledge about the nature of the placeholder may cause incorrect placement) or technical reasons (due to insufficient bidirectional support – only in the case of left-to-right languages - the placeholder in the translated string may appear in differently in the localization tools and in the application being localized).

After the translations are complete or possibly at an earlier stage, the localized application may be deployed for user testing. Any strings that seem awkward or cannot be understood will be reported with a severity level for each string, and also details of how the string was encountered. Apart from this translation testing, any other errors (such as unusual application behavior) will also be reported, along with the severity level and a description of how the error was encountered. These errors will be fixed by the developers.

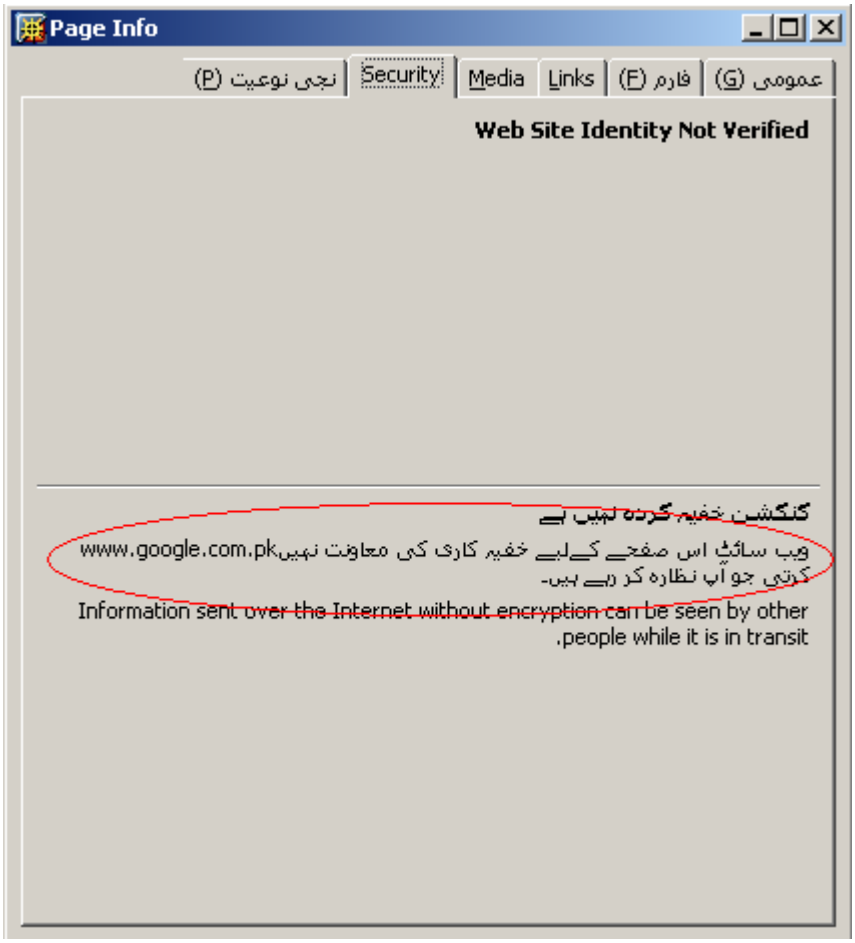

Figure 4: Misplacement of placeholder in translated string

4.9. Final Localized Product

After translation and testing is complete, the localized product can be added to the release for the application. The procedure for this varies for different applications, and some may require application specific testing cycle to be followed. The final localized product may also vary across applications. There may be an installer for the localized version, a language pack which can be added to an installation, or it may be incorporated into a single release for the application which includes all localizations (in this case, when the installer is run, users are asked to select a language). In all cases the release schedule of the application may need to be followed, if an official localization is required.

5. Translation Issues

The translation step is the most important in the whole process. It is essential that the translations produced are appropriate for the locale selected. If the translations are incorrect, or inappropriate in any way (e.g., too difficult for an end user to understand) it may render the localized product unusable. This section includes some of the issues that normally occur during the translation process.

5.1. Translation Selection

When selecting a translation for a word, the simplest and most appropriate translation should be selected. New and unfamiliar terminology should only be introduced if a word needs to be translated for which there is no commonly used word in the target language. One way that this is encouraged in the localization process defined in this report is through the linguist-developer collaboration, specifically during the glossary extension meetings. Linguists may tend to suggest literary translations, whereas developers may tend to suggest translations that are dependent on English terminology or too technical for laymen who are using the application. During these meetings the whole team has a say in the translation that is selected. While the primary goal is to translate such that linguistic conventions of the target language (as used by end users) are followed, in some cases due to

application constraints, such as limited space in the GUI for a string, some compromise may be required. Linguist-developer collaboration is essential in cases like these as well.

5.2. Translation Consistency

Translation consistency, both within a localized application and across localized applications is also essential. As much consistency as possible should be maintained so that if a user switches from one localized application, the terminology being used in each is similar and the transition is smooth. A simple example is the set of menus that are common to almost all applications, and also occur multiple times within applications ("File", "Edit", "View" etc.). It is necessary that these are translated in the same manner in localized applications. The core glossary, its extension process and the shared translation memory between linguists as described in the localization process ensures this to a large degree.

However, there can be exceptions and in some cases consistency may not be desired. A word may need to be translated differently in different contexts, since the source language word may have multiple senses that are not directly mapped in the target language. This situation should not arise frequently, and should probably never arise in translation of strings used in menus, but both linguists and developers should be aware of this especially when translating long message strings.

5.3. Distinct Translation Mapping

This translation issue is related to the translation consistency issue defined above. Just as each source language term must be translated in the same manner whenever it occurs, it must be ensured that two distinct terms in the source language (which may be related to each other) should never be translated into the same term in the target language. Examples of this include the set "Find" and "Search" and the set "Show" and "Display". It must be ensured that each member of a set such as these is assigned a distinct translation and that the translations in each set are as meaningful as the source language terms. This can be a difficult tasks, since sometimes no appropriate translation can be found in the target language, so it is recommended for this rule to be followed essentially only when the set of terms being translated are significant terms that occur frequently, e.g., in menus. For longer message strings, this rule may be compromised if no solution can be found.

5.4. Translation Rate

Another translation issue to be considered is its rate. A very convenient way to estimate the amount of work to be done is by the number of strings to be translated. However this creates an unbalanced estimate, since it is assumed that single word strings and longer word strings (which can be up to 40 words or more) will take the same time to translate. Whereas it has been observed that it takes linguists less time to translate five single words than a single five-word string (assuming that the core glossary contains the individual words). There is increase in translation complexity with increase in number of words per string. Thus the translations are assigned to linguists by counting words rather than strings. Also, the number of string given in a single batch could vary from 600-700 depending on the average string length in a batch, because it adds further complexity.

5.5. Translation Error Corrections

In the localization process defined, up to the incorporation stage the translation errors can be detected in two ways. The first is when the developer reviews the translations during incorporation and the second is when linguists are using each others translation memory. It is essential that when an error is identified in either of these ways, all concerned linguists and developers are informed so that the error could be fixed in the translation memory (to avoid further repetition of the error) and in any application where it may have been incorporated. In smaller teams this may be done by word of mouth or e-mail, but for larger teams it is recommended that a more rigorous process is used.

6. Case Study – SeaMonkey (ur-PK) Localization

This study presents the localization process described above as applied to an Urdu (ur-PK) localization for SeaMonkey (http://www.mozilla.org/projects/seamonkey/). The team for the SeaMonkey localization consisted of one developer and one linguist, and it is estimated that the localization of the GUI should take about three and a half months.

6.1. OSS Selection

After it was decided that localization of an application set that facilitated access to Internet content was required, SeaMonkey was selected because: 1) it was a complete Internet suite including a web browser and an e-mail client (because a browser and e-mail client was integrated into a single application, the overall localization effort would be less as compared to localizing a browser and e-mail client separately), 2) it had Unicode (UTF-8) and bidirectional language support which was required for Urdu, 3) it was localizable, and 4) it was supported on Windows, Linux and Mac OS X. SeaMonkey is an extension of the Mozilla Suite based on which the popular Firefox browser and Thunderbird e-mail client are being developed. Firefox and Thunderbird are standalone applications whereas SeaMonkey integrates several Internet applications. It was also ensured that SeaMonkey was a well-supported application based on activity in user and developer forums and mailing lists.

For Urdu, right-to-left text and bidirectional support had to be considered in addition to the encoding issue, because Urdu text (which uses Arabic script) is written from right to left but its numerals are written from left to right direction (making the script bidirectional). Also, strings to be translated may also include placeholders (e.g., "%S", where a string will be inserted by the application at run-time) which are also written from left-to-right.

6.2. Localization Tools

The localization tools selected were Mozilla Translator (http://sourceforge.net/project/showfiles.php?group_id=18842) and OmegaT (www.omegat.org). Both adequately provided the required language support for Urdu. Mozilla Translator was used for string management and translation incorporation and OmegaT was used for translation management, as described in Section 4.2.

6.3. Localization Registration

Currently the SeaMonkey registration process only requires making an addition to the Mozilla Wiki for SeaMonkey localization teams (http://wiki.mozilla.org/SeaMonkey:Localization_Teams), but as noted the process will be changing in the future and re-registration may be required when it does. Subscribing to the mailing lists for SeaMonkey development (dev-apps-seamonkey@lists.mozilla.org) and Mozilla localization (dev-l10n@lists.mozilla.org) is also recommended. Help can also be obtained at the IRC channels #seamonkey and #l10n at irc.mozilla.org.

6.4. Translation Resources

The following resources are being used by the linguists during the translation process:

6.4.1. Core Glossary: This is a terminology glossary based on the "Electronic Dictionary of Localization of Computer Applications (English - Urdu)", by the National Language Authority Islamabad, Pakistan. This is being extended with the collaborative effort of both linguists and developers. This is the main glossary that the linguists refer to when translating as it represents the recommended standard for Pakistan. It was converted into the glossary format required by OmegaT, and made available to the linguists in their OmegaT projects. This required typing up the whole book, and a dedicated typist was hired for this task.

6.4.2. Mozilla Suite 1.5 ur-PK glossary: This is the glossary that has been extracted from a Mozilla Suite 1.5 localization for the ur-PK (Urdu, Pakistan) locale (available at: http://www.qern.org/it/mozilla/ur-PK.html). The Mozilla Suite is an older version of SeaMonkey, so translations for many of the strings that need to be translated for SeaMonkey can be found in this glossary. The Mozilla Suite 1.5 ur-PK glossary is only used as a reference glossary. When linguists find a suitable translation here, it can only be used after the terminology it uses has been added to the core glossary. The Mozilla Suite 1.5 ur-PK glossary was converted into the glossary format required by OmegaT, and made available to the linguists in their OmegaT projects.

6.4.3. Firefox 1.0.6 ur-IN glossary: This is the glossary that has been extracted from a Firefox 1.0.6 localization for the ur-IN (Urdu, India) locale. Firefox is based on a component of the Mozilla Suite, and since SeaMonkey is the current version of the Mozilla Suite, translations for many of the strings that need to be translated for SeaMonkey can be found in this glossary. The Firefox 1.0.6 ur-IN glossary is only used as a reference glossary. When linguists find a suitable translation here, it can only be used after the terminology it uses has been added to

the core glossary. The Firefox 1.0.6 ur-IN glossary was converted into the glossary format required by OmegaT, and made available to the linguists in their OmegaT projects.

6.4.4. Thunderbird 1.0.7 ur-IN glossary: This is the glossary that has been extracted from a Thunderbird 1.0.7 localization for the ur-IN (Urdu, India) locale. Thunderbird is based on a component of the Mozilla Suite, and since SeaMonkey is the current version of the Mozilla Suite, translations for many of the strings that need to be translated for SeaMonkey can be found in this glossary. The Thunderbird 1.0.7 ur-IN glossary is only used as a reference glossary. When linguists find a suitable translation here, it can only be used after the terminology it uses has been added to the core glossary. The Thunderbird 1.0.7 ur-IN glossary was converted into the glossary format required by OmegaT and made available to the linguists in their OmegaT projects.

6.4.5. OpenOffice 2.0.3 ur-IN glossary: This is the glossary that has been extracted from an OpenOffice localization for the ur-IN (Urdu, India) locale and contains translations of all the strings that need to be translated for OpenOffice. It is being used as a reference glossary as well, and translations from it can only be added after the terminology that is being used has been added to the core glossary. The OpenOffice 2.0.3 ur-IN glossary was converted into the glossary format required by OmegaT and made available to the linguists in their OmegaT projects.

6.4.6. Online technical terminology translations (English to Urdu): There are two English to Urdu technical technology translations available online. The first is the Urdu Word Bank (http://l10n.urduweb.org/dictionary/), which has a wiki-like functionality, i.e., the content is user generated. Users can look up translations for technical terminology, edit existing translations, add new translations, or put up requests for translations. The second is an Urdu technical terms glossary (http://www.qern.org/it/dict/urdu/dict_main.cgi) which also allows users to enter their own translations, but it is not as active as the first one.

6.4.7. Dictionaries: All major English to Urdu translation dictionaries have also been consulted in the process, e.g. Qaumi English-Urdu Dictionary published by National Language Authority of Pakistan.

6.4.8. Miscellaneous: Other than the resources listed above, frequently consulted resources include: 1) WordNet (available at: http://wordnet.princeton.edu/), an English lexical database; this is helpful when there is confusion about the sense or part-of-speech of a word being translated, 2) specialized terminology translations compiled by the National Language Authority Pakistan (e.g., mathematical terms, scientific terms etc.), and 3) various other online dictionaries and online documentation for the applications being localized.

6.5. String Extraction

The SeaMonkey GUI alone consists of over 10,000 strings. String extraction and preparation of batches could have proceeded in one of two ways, either proceed with each menu of the application (which is easily done as the translation files are organized that way) and translate 10,000 strings, or extract only unique strings (which can easily be done using Mozilla Translator and any spreadsheet application) which reduces the strings to a little over 6000. A disadvantage of the second way is that sometimes in the context of the application, an English word may possibly have different Urdu translations; but errors like this can be corrected at the testing stage, so it was decided to extract only the unique strings and translate them. A total of 55 batches for translation were created, where each batch contained about 600 words for translation.

6.6. Translation Process

The translation process proceeded as described in Section 4.6. As expected, the number of new terms that had to be added to the core glossary decreased at each successive glossary extension meeting. Starting with an average of 80 new terms per batch at the first meeting, after the translation of 30 batches, it has reduced to about 10 new terms per batch.

6.7. Translation Review and Incorporation

The translation process proceeded as described in Section 4.7. Most translation errors detected during translations reviews were due to misinterpretation of the source string. This misinterpretation was caused by the following reasons; firstly due to limited exposure to software in general linguists were not familiar with some types of sentence structures used in software GUIs; secondly because the linguists had not used the software being localized, they could not understand concepts specific to the software (e.g., the notion of tabbed browsing), and might translate them inappropriately. In the case of SeaMonkey these errors can possibly be reduced by

providing the linguists with SeaMonkey so that they can use it in place of the browsers and e-mail clients that they currently use.

6.8. Localization Testing and Quality Assurance

The testing and quality assurance process is proceeding as described in Section 4.8. Translation errors detected during the localization testing stage were caused by 1) strings with placeholders, e.g., the string “Renaming folder %S...”, when translated might have a problem due to the position of the placeholder “%S” either linguistically or even because one of the localization tools, Mozilla Translator did not provide proper bidirectional support and in some cases showed the “%S” in a position that is different from where it would actually appear in SeaMonkey, and 2) strings translated out of the context intended by the GUI. Although during the translation review stage developers were careful to double check the context of any ambiguous strings (by looking them up as they appeared in the GUI), a small number of errors of this type were also detected after translation incorporation, during the preliminary testing by the developers.

6.9. Final Localized Product

After the localization is complete an announcement of the completion of the localization will be made on one of the mailing lists mentioned in Section 6.3, and the localization will be provided and a request for it to be incorporated into the official project page for SeaMonkey will be made on the list. For SeaMonkey the finalized localized product can be installers for Windows, Linux or Mac OS X, or a language pack known as an XPI which can be installed on a pre-installed SeaMonkey (on either Windows, Linux or Mac OS X). An XPI will be made since it can be used with all the official installers provided on the official SeaMonkey website. An Windows installer may also be provided, since our intended end-users will be using a Windows environment.

7. Conclusion

The process described in this report has been in use for Urdu (ur-PK) localizations of SeaMonkey, NVu and OpenOffice over the past two months. Possible improvements include integrating the use of a versioning system into the process, so that error correction is less tedious. Also, some sort of formalized peer review process between linguists (currently translations done by linguists are officially reviewed by the developers only) may be devised which may result in higher quality translation.

Some improvements can also be made in the usage of OmegaT. The memory sharing mechanism (which currently consists of manually copying and pasting translation memory files) for OmegaT may also be automated. This may also require the determination of the frequency with which translations memory files should be updated, since when a file is added to an OmegaT project, the project has to be reloaded, which may take a considerable amount of time depending on the contents of the project (number of glossaries, number of translation memories, number of strings being translated, etc.). If memory updates and project reloads take place too frequently, it may disrupt the translation process.

After the localizations are complete, it is recommended that the localization and possibly a glossary for that particular localization should be made available so that when a newer version of the application is released, the localization can be updated quickly and conveniently (even if the ownership for the localization changes). There should be no string changes for minor releases, but the localized product still needs to be updated. Each time there is a major release, additional strings for translation should be expected. The glossaries can also be used for reference in other localization projects.