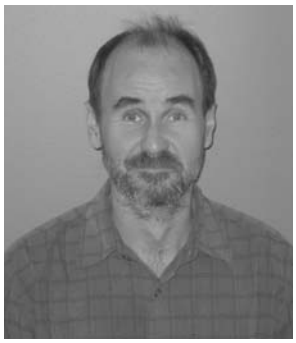
Adopting Standards-based XML File Formats in Open Source Localisation



James M. Hogan

By Asgeir Frimannsson and James M. Hogan, School of Software Engineering and Data Communications, Queensland University of Technology, Brisbane, Australia.

Asgeir Frimannsson

In recent years, major localisation vendors and key standards organisations have agreed on open XML-based standards for storage and exchange of data in the localisation process: The Translation Memory eXchange (TMX) file format for exchanging translation memory data, the TermBase eXchange (TBX) format for terminology exchange, and the XML Localisation Interchange File Format (XLIFF) for extracting and storing locale-dependent resources in a common file format. Up until recently, very few open source tools have supported these formats, and hence very few open source projects have adopted them in the localisation process. A majority of open source applications continue to use Gettext and the Portable Object (PO) file format. This paper evaluates the case for adopting XLIFF in localisation processes currently based on the Gettext toolkit, and examines the usefulness of other standards such as TMX, TBX and Translation Web Services (TWS) in these processes.

1 Introduction

LOCALISATION OF SOFTWARE IN OPEN SOURCE PROJECTS IS USUALLY HANDLED BY GETTEXT, a set of tools from the GNU project. This toolkit contains tools for extracting and merging messages from source code for localisation, as well as libraries for loading the translated messages from resource files at runtime. Gettext uses its own file format, the Portable Object (PO) format, for storing resources in the localisation process. With open source desktop environments — such as GNOME and KDE — today having translation teams for over 80 different languages, it is evident that Linux and open source software (OSS) is reaching out to a global market, and is not limited to English-speaking cultures and communities. As industry and governments around the world continue to embrace open source, localisation has become a critical factor in OSS adoption. Thus, it is increasingly important that OSS localisation processes become aligned with industry standards — allowing seamless integration of commercial and open source processes when developing for open source platforms.

In the main contribution of this paper, we present the results of working with open source contributors in defining an *XLIFF Representation Guide for Gettext PO* – a work now submitted to the XLIFF Technical Committee in line with their programme to develop canonical XLIFF profiles for common file formats. We propose a bridge between current localisation practices in open source and established localisation industry standards. As open source localisation is based around the common PO resource format, this paper will focus on XLIFF as a possible replacement for PO as the common format in the OSS localisation process; further, we provide standards alignment through a canonical representation of the PO file format within the XLIFF standard. XLIFF was designed principally to address needs in commercial localisation processes, especially the localisation of Microsoft Windows-based resource formats. This research aims to identify aspects of Gettext and other PO-based localisation processes for which XLIFF lacks support, and to address these deficiencies.

Building on this foundation, we then examine whether the adoption of other localisation standards may further improve OSS localisation practices. This evaluation considers the handling of terminology, translation memories and localisation workflows — areas that have that have up until now only been addressed on an ad hoc basis in open source localisation. We will investigate how successfully standards such as TMX and TBX might be incorporated into OSS localisation processes, and discuss the need for a service-based architecture, evaluating emerging standards such as TWS in open source localisation.

We have chosen to provide detailed coverage of the case for XLIFF adoption, but only a high-level overview of the case for other standards such as TMX, TBX and TWS. This is a natural focus at this time, as XLIFF is currently the only format for which there is an existing 'equivalent' in OSS localisation processes. Usage of Translation Memory technologies, structured Terminology Databases and service-based architectures is very limited in the open source community, as open source tools for these technologies do not exist at present. Thus, when we present models incorporating these standards, we therefore build on best practices from the wider industry and not current practices from OSS localisation alone.

This research is a first look at how open source can benefit from standards-based localisation formats; hence, much of the focus of the paper is to identify areas needing further research. Nevertheless, this paper builds a solid foundation for further research, providing a solution for representing the PO format in XLIFF, and arguing the merits of XLIFF as a common resource format in all open source localisation processes.

This paper is organised as follows: in Section 2 we provide a thorough survey of the relevant localisation and internationalisation standards, and current practices in open source localisation. Section 3 presents our main result, a canonical mapping of the PO format to XLIFF. We then look at how this format can be incorporated into existing build systems and development processes (Section 4). Section 5 builds on these foundations, discussing the need for other exchange standards in open source localisation. We conclude, in Section 6, by summarising the findings of this paper, and by examining the future of OSS localisation.

2 Background

2.1 Open Source Localisation

The term 'open source' relates to the practice of freely sharing access to the source code of a product, allowing anyone to extend or modify a piece of software. The open source development process has been described as a user-driven, just-in-time approach, driven by a global developer community (Berglund and Priestly, 2001). Development is driven by demand for the product within this community, and new features are implemented as a result of requests from the user base. Raymond (2000) describes the process as a 'bazaar' where software is released early and often, the software process is open and transparent and work is delegated as much as possible. In contrast, traditional software is developed by closed teams having long release cycles and is only released after being thoroughly tested. Surveys reveal that fewer than one in five open source participants are paid for their involvement (Hars and Ou, 2001). These characteristics imply that localisation of open source software is an ongoing process, driven primarily by voluntary efforts, in response to constantly changing source code.

Localisation of *software messages* is handled in most open source projects by Gettext — as specified in the OpenI18N Globalisation Specification (Free Standards Group, 2003). The Gettext library is based around two file formats:

- The Portable Object (PO) file format: a simple string table for storing translation units in the localisation process
- The Machine Object (MO) file format: a binary representation of a string table — used by an application to retrieve translated strings at runtime

Whilst the majority of open source projects use Gettext for internationalisation support, several projects do not rely on this toolkit. The Mozilla project (which includes the popular Firefox web browser) and the OpenOffice.org office suite use custom resource formats, and many projects rely only on the platform support provided by Java and .NET.

2.1.1 Gettext and the PO format

According to GNU Standards, software, including software messages, is to be written in American English (Free Software Foundation, 2004). By default, the original software messages are not externalised to resource files, but stored in source code, thereby allowing the application to run in the default language (English) without needing any resource files. The Gettext toolkit is then used to extract strings marked for localisation from source code, as depicted in the following diagram:

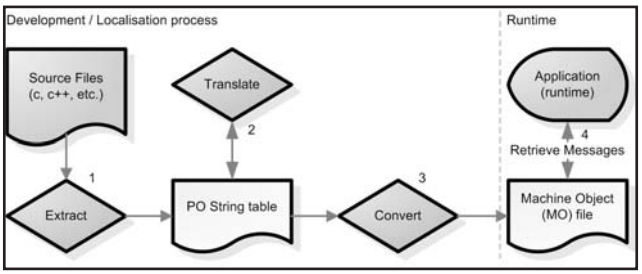


Figure 1: *Typical Gettext-based localisation process*

Gettext tools extract localisable strings from source code into PO string tables (1). Normally, each application only uses a single String table, easing the work for translators in the localisation process. Translators then translate (2) these files using PO-based localisation tools. After translation, the PO files are converted to binary Machine Object (MO) resource files (3). Applications then retrieve (4) translated strings from the language-specific binary MO files at runtime. When no translation is available for a given string, the original (English) string is used.

Gettext provides mechanisms for updating and merging string tables, much needed in open source development processes characterised by short release cycles and rapid change.

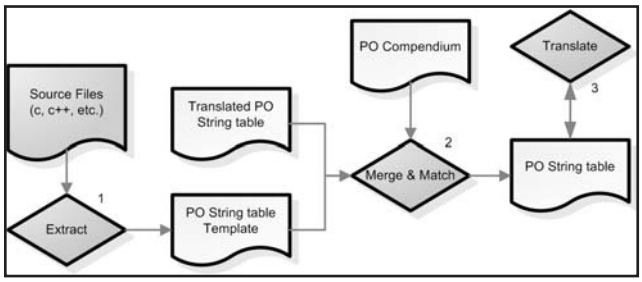


Figure 2: *Gettext-based workflow — merging translations*

As depicted in Figure 2, Gettext tools first extract localisable strings from source code into a PO string table (1). Next, the newly extracted PO string table is merged with an existing PO string table containing translations (2). In addition, new entries are matched against entries of a PO Compendium — a string table acting as a translation memory, holding translations combined from multiple PO Files. Translators then translate and review changes in the updated PO string table (3) before the PO file is converted to MO for use at runtime.

While the PO format was originally intended only for software messages, many open source projects have taken advantage of the tool support for the format in the localisation of other content types, including DocBook-based documentation. There is no standard for the translation of such files; different projects use different approaches for various content types in an adhoc fashion. In KDE and the Fedora Linux distribution, translatable segments are extracted to PO files by custom filters, as shown in the following diagram:

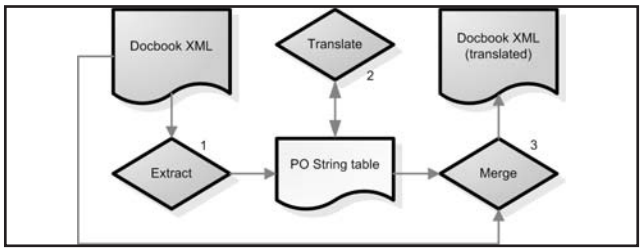


Figure 3: *DocBook files converted to PO for translation*

Here, DocBook files are converted to PO for translation (1). After translation (2), files are converted back into language-specific DocBook documents (3). By using this approach, translators can use the same tools for translating software messages as they do for translating documentation. Custom filters are also used for other file formats, including XML-based User Interface Dialogs and Desktop Entry files, where translatable segments are added to the applications PO file in the localisation process, and merged back after localisation.

The PO file format lies at the heart of Gettext, and has three different variants: PO Template files (POTs), Regular bilingual PO files, and PO Compendium files. Entries are extracted from source code, using the Gettext 'xgettext' tool, and stored in POT files. POT files are initialised (using the 'msginit' tool) or merged with existing translated PO files (using the 'msgmerge' tool) and stored in a language-specific PO file, which are then in turn used in the translation process. In addition, translation entries from across projects are stored in PO Compendiums, providing simple translation memory abilities.

Each PO file generally maps to a Gettext domain, where a domain is a collection of unique translation units stored in a single MO file. Most commonly, and a de facto standard in projects such as GNOME and KDE, each application uses a single Gettext domain,

the name of that domain being generally the same as that of the application.

The following example shows the basic structure of Gettext PO files:

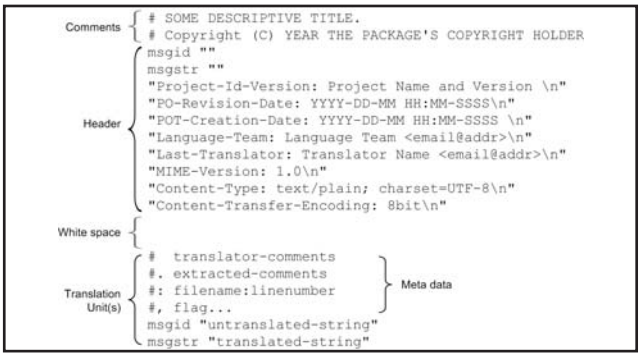


Figure 4: *PO file format structure*

Each PO file contains a set of translation units, each having a source (`msgid`) and target (`msgstr`) field, where the first translation unit generally contains header metadata. Each translation unit can, in addition to an original and translated string, include *translator comments*, *extracted comments* from source code, source code *references* (source filename and line-number) and Gettext-specific *flags*.

PO translation units support parameter-determined *plural form* translation units. These translation units contain the *singular* English form in the `msgid` field, and the *plural* form in the *msgid_plural* field. As the target, these translation units have an array of type `msgstr`, representing the number of forms in the target language:

```
msgid "You have %d file"
msgid_plural "You have %d files"
msgstr[0] "Du har %d fil"
msgstr[1] "Du har %d filer"
```

Figure 5: *PO Plural Forms*

The target language may have one or more forms (Japanese has one form, while Polish has three), and the selection logic is defined in a PO header field, where `nplurals` defines the number of forms and `plural` defines a C-language expression for evaluating which item in the `msgstr` array to use at run time:

```
"Plural-Forms: nplurals=2; plural=(n != 1);\n"
```

Figure 6: *Germanic Plural Forms header*

This is a typical example for a Germanic language, which has a special case when `n = 1`. A more complex example is Polish, which has special cases for when `n = 1`, and for a subset of numbers ending in `2`, `3` or `4`:

```
"Plural-Forms: nplurals=3; plural=n==1 ? 0 :
n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ?
1 : 2;"
```

Figure 7: *Polish Plural Forms header*

The selections are made using standard C ternary conditionals of the form: "`condition ? true_value : false_value`", where condition is a Boolean `expression`. In the second example, condition is "`n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20)`", which if `true` yields a result index of 1, and if `false` an index of 2. At run time, Gettext will use the `msgstr` associated with the index returned from this expression.

Another unique 'feature' of Gettext and the PO format, is that it uses the source string (`msgid`) as the primary identifier. This is different from other common resource formats such as Java Resource Bundles (Figure 8) and .NET Resources, which use some sort of logical key to map to the actual source string.

```
button_saveAs=Save As..
msg_usernameInvalid=Invalid Username. Please try
again.
```

Figure 8: *Java .properties fragment — use of logical ids*

Although use of the source string as id makes it easier for developers, as they do not have to handle external resource files, there are some disadvantages with this approach. First of all, there cannot be multiple translations of the same string within the same Gettext domain. This has led to 'hacks', where developers prefix the string with some context information, and this context information is then removed at runtime. In addition, if the original (English) string changes slightly because of spelling mistakes, fuzzy matching is required if tools are to update the translated PO files.

Source string-based ids are a challenge when updating the resource files after the source code has changed. If a string message has changed in the source code, there is no way for the tool to know which translation units are affected, and so existing translation units are simply discarded and new ones created. This makes change-tracking and version control on a translation unit cumbersome and error-prone, relying upon fuzzy string matching when a key would prevent no such concerns.

If source strings are used as ids, there is also no way to support different translations of similar strings in different contexts within the same catalogue. Some projects (for example KDE and the GNOME Glib library) combine the context and the source string to form unique ids, and have custom functions to extract the context information from the source string at runtime. Recent development versions of Gettext have however added an extra message context (`msgctx`) field to the PO format — supporting different contexts for identical strings.

2.1.2 Tool support in open source localisation

Open source localisation tools are mainly based around the PO file format. The most advanced of these is KBabel from the KDE project (Diehl et al., 2002). KBabel supports advanced features such as handling of plural forms in PO files, internal translation memory, dictionaries and catalogue management (Costin and Kiefer, 2004). Other popular PO editors include GTranslator, Emacs PO Mode and poEdit.

Each translation project (for example KDE, GNOME and Fedora) has its own set of web-based status pages, displaying statistics and information relating to the translation process. Recently, there has been a shift towards web-based translation *portals*, where contributors can, in addition to viewing statistics, also translate using browser-based translation workbench clients. The more sophisticated of these are Pootle and Rosetta (Figure 9). While Pootle is an independent project, covering translations from projects like OpenOffice, Mozilla and KDE, Rosetta is the translation portal for the Ubuntu Linux distribution.

Figure 9: *Web-based translation portals such as Ubuntu's Rosetta significantly reduces the technical skill needed to contribute.*

The Gettext tools have some support for translation memory through PO compendium files (Free Software Foundation, 2002). These files are of similar format to PO files and contain previously translated entries. PO compendia are used to ensure consistency across applications and projects by leveraging previously translated segments. KBabel provides an internal translation memory as well as support for PO compendia and TMX — although TMX support is limited to retrieval.

Present Gettext-based localisation processes suffer from very limited translation reuse. This is mainly due to limitations of the PO format, providing little support for segmentation and abstraction of inline codes.

There are no agreed standards for handling terminology in open source localisation, and different language teams and projects use different approaches. Most commonly, teams share a glossary list, which sometimes contain definitions of terms, but usually only contain the original English term and the translated term. These glossaries are shared through collaborative wikis and web pages, and tool support is limited or non-existent.

As previously mentioned, most active open source projects have short release cycles. Localisable data is stored together with source code in version control systems such as CVS (Concurrent Versions System) and SVN (Subversion). Batch jobs take care of updating PO files from the original sources as updated source code is stored in the versioning system (KDE.org Team, 2004). Translators can view the status of translations though user friendly status pages on the web which are automatically generated from the PO files stored in the repository (Diehl et al.).

2.1.3 Limitations

Localisation of open source software is more or less limited to translation of software messages. Other aspects of the user interface such as icons, images, button sizes and layout are currently hard to localise efficiently (Van Schouwen, 2004). This is largely due to the development model adopted by most open source projects, in which the application is distributed in US English, and additional language packs can be downloaded, each containing the Gettext message catalogue in the desired language.

Though widely used and accepted in the open source communities — and successfully deployed in activities well outside its original brief — the simple string table PO file format is reaching its limits, being unable to support the rich structures inherent in modern localisation. More specifically:

- PO lacks support for advanced metadata, pre-translation and workflows
- PO-based localisation is limited to translation of textual content such as documentation and software messages, and the format does not support icons or binary images
- While PO has been successfully used for localisation of DocBook documents, the format is not a natural fit for the localisation of paragraph-based text. In particular, the format has no support for segment-based translation reuse, metadata support is very limited, and inline tags cannot be extracted

In addition, localisation of open source software and documentation is highly technical, and translators presently need advanced technical skills to be able to contribute to the process.

2.2 XLIFF and Localisation Industry Standards

The XLIFF Technical Committee (2003a) describes the purpose of XLIFF as "*to define, through XML vocabularies, an extensible specification for the interchange of localisation information.*" As shown in the following diagram, XLIFF enables tool vendors to focus on a *single file format* in contrast to traditional localisation tools which have to support a multitude of native formats.

(Yves Savourel, Adapted from Jewtushenko, 2004)

Figure 10: *XLIFF localisation workflow.*

Vendors can create XLIFF *filters* for their proprietary file formats, extracting localisable data to the standard XLIFF format. In the extraction process, non-localisable material can be stored in a XLIFF *skeleton* file, which can then be used later by a merging tool to recreate the original translated file. With XLIFF, localisation engineers are not bound to use tools that support the vendors' proprietary format, but can use any tool that supports the XLIFF standard.

XLIFF supports custom defined workflow metadata during the localisation process. During the different stages of localisation, the XLIFF document might include data from Translation Memory, Machine Translation and Term Bases. Segments can be marked as 'signed-off', 'needing review' or similar, and documents can go through several *localisation phases* (e.g. rough translation, review).

Similar to TMX, XLIFF allows for abstraction of inline codes such as text markers, inline images and references (Savourel, 2003). This means that metadata and markup such as XML tags can be included in translations, allowing tools to display these as non-translatable markup elements.

XLIFF is a young standard, and since its introduction in 2002, some vendors have added support for parts of the standard to their tools, but many tools still provide no special XLIFF support — treating it as just another XML format (The XLIFF Technical Committee, 2003a). Many high-end localisation tools such as TRADOS and Alchemy Catalyst already have very good support for proprietary file formats such as Windows resources and Microsoft Word documents, so the need for XLIFF is not as great if customers adhere to these file formats.

Viswanadha and Scherer (2004) describe two distinct approaches to handling XLIFF in the development and localisation process:

1. *The transient approach:* Convert source formats to XLIFF in the localisation process, and only keep the source files in source control systems. On each iteration of the localisation process, an XLIFF roundtrip is needed, and XLIFF is simply a transient format used only in the localisation process.
2. *The persistent approach:* Keep XLIFF files in source control systems, and convert back to the original format at build time. Updated source files are merged with existing XLIFF documents, and metadata in the XLIFF files are preserved. This approach requires intelligent filters to handle updates and merging of XLIFF files.

Some further work is being undertaken to improve the XLIFF standard. The XLIFF Segmentation Sub Committee have been working on a standard for representing *segmentation metadata* in XLIFF to improve translation memory effectiveness (Jewtushenko, 2004), and this will be part of the upcoming XLIFF 1.2 specification. The existing XLIFF standard doesn't specify how text is segmented into translation units and as different vendors and file formats use different algorithms for segmentation, it becomes increasingly difficult to leverage useful data from translation memory systems. In addition, The XLIFF Technical Committee is working on guides for *canonical representations of common file formats* (such as Java Properties and .NET resource bundles), promoting greater interoperability between tools.

2.2.1 XLIFF Document Structure

In this section, we provide an overview of the structure of an XLIFF document, and details of the key elements of its XML specification. XLIFF is structured as a group of `<file>` elements, each representing an extracted document. Each `<file>` element contains a `<header>` element for storing metadata relating to the original

document, as well as localisation metadata. Following the <header> element is a <body> element containing the localisable material, which may include textual or binary translation units. Textual data is represented as <trans-unit> elements, and binary data as <bin-unit> elements. Translation units can be grouped together using <group> elements, supporting hierarchical data structures such as menus.

```
<xliff>
  <file>
    <header>
      metadata related to original document and
      localisation process
    </header>
    <body>
      <trans-unit> or <bin-unit> translation-unit
      elements in optional
      <group> element hierarchies.
    </body>
  </file>
  additional <file> elements...
</xliff>
```

Figure 11: *Common XLIFF 'shell'*

Most elements and attributes in XLIFF are optional, making it possible to create very simple XLIFF files. This enables tool vendors to add XLIFF features to tools and filters incrementally.

```
<xliff version="1.1">
  <file original="myfile.ext" datatype="plain-
text"
    source-language="en-US" target-language="nb-
    NO">
    <body>
      <trans-unit id="#1">
        <source>Have a nice day</source>
        <target>Ha en hyggelig dag</target>
      </trans-unit>
    </body>
  </file>
</xliff>
```

Figure 12: *Minimal XLIFF document*

Translation Units. Localisable material is segmented into translation units, represented as <trans-unit> elements (for textual content) and <bin-unit> elements (for binary content). As Gettext and other open source localisation solutions at present only deal with textual data, we will not explain handling of binary content in any further detail.

Each <trans-unit> element contains a <source> and a <target> element, representing the original and the translated resource. As a translator usually only handles the translation to one target language, an XLIFF document is strictly bilingual, allowing only a single <target> element for a translation unit. Other language translation suggestions (usually from translation memory or machine translation), stored in <alt-trans> elements (Figure 15), can however be included to provide guidance to translators.

XLIFF allows for abstraction of markup and other data in <source> and <target> elements. The following example shows the abstraction of the HTML <b> element within a translation unit:

```
<trans-unit id="#1">
  <source>This is a <bpt
  id="1">&lt;b&gt;</bpt>bold <ept
  id="1">&lt;b&gt;</ept> example</source>
</trans-unit>
```

Figure 13: *Abstraction of inline codes*

Context Information. Additional context information for translation units can be stored in <context> elements, grouped in <context-group> elements. In hierarchical XLIFF files, it is also possible to define context groups for a set of translation units:

```
<group id="#1">
  <trans-unit id="#1">
    <source>How are you?</source>
    <context-group name="location data" pur-
    pose="location">
      <context context-type="linenumber">11</con-
      text>
    </context-group>
  </trans-unit>
  <trans-unit id="#2">
    <source>Have a nice day</source>
    <context-group name="location data" pur-
    pose="location">
      <context context-type="linenumber">12</con-
      text>
    </context-group>
  </trans-unit>
  <!-- Group level context information  -->
  <context-group name="location data"
  purpose="location">
    <context context-
    type="sourcefile">myfile.txt</context>
  </context-group>
</group>
```

Figure 14: *Adding Context Information*

Context groups can be used for a range of purposes, including translation memory lookups ('purpose' attribute set to 'match'), translator information ('purpose' attribute set to 'information') and location information ('purpose' attribute set to 'location').

Workflow Information. XLIFF maintains processing and localisation workflow information in <phase> elements. The different phases are defined in the XLIFF header, as <phase> elements.

```
<xliff>
  <header>
    <phase-group>
      <phase name="review"
      process-name="Translation"
      contact-name="Joe Bloggs"
      contact-email="joebloggs@example.com"/>
      <phase name="pre-trans#1" process-name="TM
      Matching"/>
    </phase-group>
  </header>
  <body>
    <trans-unit id="#1">
    <source>Have a nice day</source>
    <target phase-name="review">
      Ha en hyggelig dag
    </target>
    <alt-trans match-quality="70%" tool="company-
    TM">
      <source>Have a nice time</source>
      <target xml:lang="nb-NO" phase-name="pre-
      trans#1">
        Ha det gøy
      </target>
      </alt-trans>
      <alt-trans tool="company-MT">
        <source>Have a nice day</source>
        <target xml:lang="da" phase-name="pre-
        trans#1">
```

continued on next page

continued from previous page

```
        ha en fin dag
      </target>
    </alt-trans>
    <note from="John">blah</note>
  </trans-unit>
 </body>
</xliff>
```

Figure 15: *Workflow tracking through <phase> elements*

Metadata. XLIFF provides a large set of standard attributes to support metadata such as restrictions on string length and translator comments, as well as specifying data type and resource type for a translation unit.

Extensibility. XLIFF can be extended using custom namespaces, and the standard provides a number of extension points. In addition, standard attributes can be extended by prefixing attribute values with 'x-'. While XLIFF extensibility is valuable, it is recommended that standard specification-provided elements and attribute values be used, enabling greater tool interoperability.

2.2.2 Translation Memory and Open Standards

Translation memory (TM) allows for reuse of previously translated segments. Translations are stored in a database, and as new documents are translated, similar or identical entries are suggested to translators based on these previous translations. Effective leveraging of previous translations relies upon a *high degree of repetition* in the projects undertaken; otherwise, only a limited degree of reuse is possible (Iverson, 2003). Successfully deployed TMs significantly reduce turnaround time on translation projects, reducing cost and time-to-market.

Translation Memory eXchange (TMX) is a mature XML standard for lossless exchange of TM data regardless of TM vendor and tool. Two levels of certification exist:

- *Level 1* where no inline codes or markup are stored
- *Level 2* which incorporates inline codes

A large number of vendors are now supporting the standards, and an increasing number of tools have been certified by LISA (The Localisation Industry Standards Association) (Zetzsche, 2003).

Effective leveraging of translations relies on a common *segmentation* standard. This has recently been accomplished through the Segmentation Rule eXchange standards (Localisation Industry Standards Association, 2004), which tightly coupled with the TMX standard provide a complete standards-based mechanism for exchanging translation memory data.

Future work to improve the TMX standard includes support for *XML schemas (Localisation Industry Standards Association, 2004)*, allowing for extensible non-TMX markup inside TMX documents through the use of XML namespaces.

The internet has opened up possibilities for global sharing of TMs. (Levitt, 2003) predicts internet-shared TMs will become a standard, as this may allow higher overall consistency and shorter turnaround time compared to traditional TM systems in which each translator has their own local TM.

TMX and XLIFF make decoupling of translation tools and TMs possible (Foster, 2004b). XLIFF documents can be fed with TM entries before they are sent to translators by inserting alt trans elements. Translators will then not need access to a TM since all relevant information is stored in the XLIFF document. When translation is complete, the translated entries can be included in the TM by converting the XLIFF document to TMX using a standard XML-based transformation script.

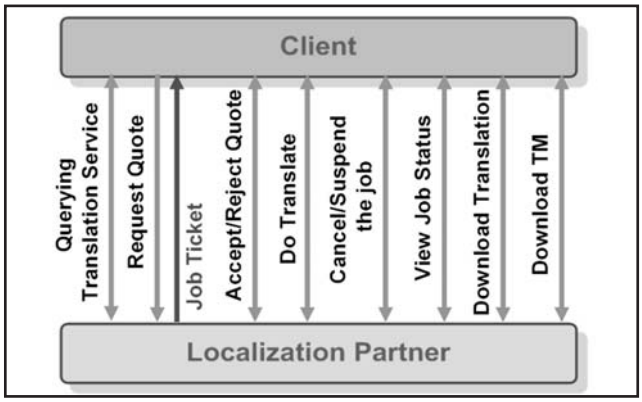
2.2.3 Terminology Management

Whereas translation memory is all about reusing translations, terminology management is about defining and managing concepts occurring in translatable material. Warburton (2005a) identifies the need for terminology management as early as in the content creation process. If terms are identified, defined and translated early, cost can be reduced and translators will have access to a consistent bilingual dictionary throughout the localisation process. The TermBase eXchange (TBX) standard provides a common file format for the exchange of terminology data. As with TMX, the standard prevents vendor lock-in, as the format provides a rich framework for importing and exporting terminology data from terminology management systems.

To allow better linkage of terms in XML documents to TBX entries, LISA is currently drafting the TBX Link specification. As this specification matures, it is reasonable to believe that XLIFF will incorporate support for it, providing direct support for linking terms in XLIFF translation units with corresponding entries in a TBX document.

2.2.4 Translation Web Services

The OASIS Translation Web Services (TWS) technical committee is working on a specification for automating the communication between parties in the localisation process (The Translation Web Services Technical Committee, 2005). This specification includes methods for requesting localisation quotes, retrieving and submitting localisation jobs, and querying the status of localisation jobs. In addition, the specification covers querying language service providers for services and languages supported.



(Palas and Karásek, 2003)

Figure 16: *Translation Web Services*

2.3 Advantages of XLIFF and TMX over PO

2.3.1 Metadata

XLIFF allows for various types of metadata to be stored in documents. While PO files allow a limited amount of metadata (e.g. general comments and references for each individual segment, various header-specific metadata), XLIFF has an extensive range of possible elements to specify for each translation segment as well as for the XLIFF file as a whole (The XLIFF Technical Committee, 2003b). By using `maxwidth` and `minwidth`, size constraints can be specified for strings as well as other measurable elements such as images. Each segment and phase can have a translator associated with it, thus providing direct support for workflows and integrated version management. If PO files are used, this information is only available in CVS logs.

2.3.2 Workflows

The PO format does not allow for workflows beyond marking entries as fuzzy for later revision. In contrast, XLIFF specifies elements and attributes for workflow information, and localisation can pass through multiple, defined phases (e.g. pre-translation, rough translation, review), in which changes are documented in the XLIFF sources (The XLIFF Technical Committee, 2003a). This can be utilised in, for example, release planning and quality assurance processes in open source projects. Workflows can also integrate translation memory as explained below.

2.3.3 Translation Memory Improvements

XLIFF and TMX offer the possibility of shared translation memories in open source projects. At present this is hard to accomplish because of the limited information stored in the PO format. With XLIFF, multiple TM matches can be stored in the document, eliminating the need for client side TM technologies (Raya, 2004b). Sharing of TM is important to open source projects since contributors are spread geographically. Upon 'checking out' a file for translation, TM suggestions can be automatically inserted. When a translator has completed work and returns the file to the repository, a TMX document containing the approved translations can be automatically processed and new pairs imported into the TM.

2.3.4 Decoupling of localisation technologies

Open source localisation is currently based on the Gettext utilities, and most localisation tools in this domain are focused solely around the PO file format (Foster, 2004a). By building open source localisation tools that support XLIFF, software projects can use technologies other than Gettext without this affecting the existing localisation process.

2.3.5 XML-based processing

One advantage of XML over other file formats is the range of open and free tools and technologies available to process this format (Savourel, 2001). Many facilities exist to define parsing and transformation tools for specific formats. These can be used in the open source environment in a wide range of contexts; for example presenting summaries and data for the translation status pages of the various projects. By using simple XSLT and other XML transformation languages, intuitive summaries and reports can readily be generated from TMX and XLIFF files. These technologies can also be used by the translation tool to present user friendly reports at various stages of the localisation process e.g., printable HTML documentation can be straightforwardly rendered from XLIFF sources for comparison and proofreading.

2.3.6 Localisation of non-textual elements

XLIFF is not limited to localisation of textual content, but can also handle binary data. This opens up a lot of possibilities for future enhancements of the OSS localisation process. GNOME and KDE User Interface dialogs are currently stored in XML formats and can be encapsulated in XLIFF documents and localised using visual XLIFF tools, similar to the processing of Windows Resource files in tools such as Alchemy Catalyst (The XLIFF Technical Committee, 2003a). This however, requires architectural changes to the way localisation is handled at runtime.

3 XLIFF representation of PO

While earlier versions of the XLIFF standard provided considerable flexibility in the permissible attribute values — with many allowing free-form parsed character data — the XLIFF technical committee has recognised that this flexibility acts as a barrier to intelligent resource matching and translation reuse. Recent revisions to the standard have limited this practice, placing greater reliance on explicit enumeration of attribute values. In the subsequent discussion, there arise a number of situations in which the shell of our proposed mapping is readily defined and specified, but the resulting framework permits a range of implementations consistent with this specification. In each case we have tried to specify a model or reference implementation which we believe best represents the additional information contained in the PO format, while remaining consistent with the intent of the XLIFF standard.

The XLIFF standard and its structure were examined in some detail in Section 2.2.1, and similarly, a high-level overview of the PO format was presented in Section 2.1.1. While the details of the actual mapping between PO and XLIFF are now available from the OASIS XLIFF Technical Committee document repository, we will present here the core findings and challenges identified in the process.

3.1 Overview

We have mapped each PO file to an XLIFF `<file>` element, with the datatype attribute set to 'po' (Figure 17 below). The PO header is mapped to a translation unit, or is stored in the XLIFF skeleton. Each additional Gettext *domain* defined in the PO file is encapsulated in an XLIFF `<group>` element, with the restype attribute set to 'x-gettext-domain'.

```
<?xml version="1.0" ?>
<xliff version="1.1"
xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file original="filename.po" source-lan-
  guage="en-US" datatype="po">
    <body>

      <trans-unit>
        ... PO header for default domain...
      </trans-unit>
      ... translation units for default domain...

      <group restype="x-gettext-domain"
      resname="domain-name">
        ... header and translation units for
        domain 'domain-name'...
      </group>

    </body>
  </file>
</xliff>
```

Figure 17: *XLIFF representation of PO — General Structure*

3.2 The PO Header.

The PO header contains both technical and project-related metadata, and can also contain user-defined variables. As seen in Figure 18, the header is structured as a normal translation unit, with the source field (`msgid`) left empty, and the PO header elements contained within the target field (`msgstr`). Additional metadata, such as copyright and licensing information, is stored in comment fields.

```
# Translation for MyPackage.
# Copyright (C) 2005 Foo Bar
# This file is distributed under the same license
as MyPackage.
# Foo Bar <foo@example.com>, 2005.
#
msgid ""
msgstr ""
"Project-Id-Version: MyPackage 1.0\n"
"Report-Msgid-Bugs-To: foo@example.com\n"
"POT-Creation-Date: 2004-11-11 04:29+0900\n"
"PO-Revision-Date: 2005-02-01 12:00+0900\n"
"Last-Translator: Foo Bar <foo@example.com>\n"
"Language-Team: My Language <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=n>1;\n"
"X-Custom-Variable: value\n"
```

Figure 18: *PO Header*

PO supports a set of project metadata for which there is no existing XLIFF equivalent. XLIFF is merely interested in the localisable content of a file, whereas PO also contains metadata about the localisation project and technical metadata providing context.

In processes where XLIFF is simply used as an optional *transient* localisation format (Section 2.2), it is important that translators are able to change some of the information stored in the PO header —

such as adding their name as a translator and changing the PO revision date. In addition, translators must be able to retrieve information such as the email address for bug reports. To accommodate this, the easiest and most convenient approach is to store the PO header data in a <trans-unit> element, as shown in the following example:

```
<trans-unit id="#1" restype="x-gettext-domain-
header">
  <source>
    Project-Id-Version: MyPackage 1.0
    Report-Msgid-Bugs-To: foo@example.com
    POT-Creation-Date: 2004-11-11 04:29+0900
    PO-Revision-Date: 2005-02-01 12:00+0900
    Last-Translator: Foo Bar foo@example.com
    Language-Team: My Language LL@li.org
    MIME-Version: 1.0
    Content-Type: text/plain; charset=utf-8
    Content-Transfer-Encoding: 8bit
    Plural-Forms: nplurals=2; plural=n>1;
    X-Custom-Variable: value
  </source>
  <target>
    ... modified header ...
  </target>
  <note from="po-file">
  Translation for MyPackage.
  Copyright (C) 2005 Foo Bar
    This file is distributed under the same
    license as MyPackage.
    Foo Bar <foo@example.com>, 2005.
  </note>
</trans-unit>
```

Figure 19: *PO header as a* <trans-unit> *element*

While this approach provides a way to combine PO- and XLIFF-based localisation processes, it is not without limitations:

- Translators would need to know the purpose and format of PO headers even though they are using XLIFF-based localisation tools
- The PO header is not a conceptual translation unit, and therefore treating it as one is not consistent with the XLIFF Specification.

In processes where XLIFF is used as a persistent format throughout the development and localisation process, it is possible to eliminate the need for the PO header in the localisation process. Project- and localisation-related metadata can be stored elsewhere, and the technical metadata needed for converting PO to MO at build time can be automatically generated.

3.3 Translation Units

Not surprisingly, we suggest mapping each PO translation unit to an XLIFF <trans-unit> element, with the <source> element representing the PO `msgid` field, and the <target> element representing the PO `msgstr` field. While this base-case is trivial, there are several features of the PO format that require further consideration:

Plural forms. XLIFF does not have any concept of pluralisation of translation units, but we suggest simulating this through utilising XLIFF's hierarchical structure of <group> and <trans-unit> elements. As shown in Figure 20, a group of plurals can be grouped in an XLIFF <group> element, and each plural form can then be represented as one <trans-unit> element within that group.

```
<group restype="x-gettext-plurals">
  <trans-unit id="##1">
    <source>%d file was deleted</source>
    <target>translation form 0</target>
  </trans-unit>
  <trans-unit id="##2">
    <source>%d files were deleted</source>
    <target>translation form 1</target>
  </trans-unit>
  ...
  <trans-unit id="##n">
    <source>%d files were deleted</source>
    <target>translation form n</target>
  </trans-unit>
</group>
```

Figure 20: *Plural Translation Unit in XLIFF*

The fuzzy flag. The fuzzy flag is Gettext's way of marking a translation as unfinished, and denotes that it needs review by a translator. This flag can easily be mapped to the `approved` attribute in XLIFF, where this attribute is set to 'no' if the fuzzy flag is present or when the PO entry is not translated.

Comments. PO files have two types of comments: *Translator* comments and *extracted* comments. The extracted comments occur both in PO Templates and PO files, and represent comments related to a translation unit, which are added by developers in the source code. Translator comments occur only in PO files, as they are added by translators in the localisation process. We suggest mapping these to XLIFF <note> elements, and/or XLIFF <context> elements, as shown in the following example:

```
<trans-unit>
  <source>hello world</source>

  <!-- Comments as note elements -->
    <note from="po-translator">Translator
    Comment</note>
  <note from="developer">Extracted Comment</note>

  <!-- Comments as context information -->
  <context-group name="po-entry">
    <context context-type="x-po-
    transcomment">Translator Comment</context>
    <context context-type="x-po-
autocomment">Extracted Comment</context>
  </context-group>

</trans-unit>
```

Figure 21: *Handling of PO comments in XLIFF*

Source code references. Each PO entry contains a list of references, referring to the location (source file + line number) from which the translation unit was extracted. The current XLIFF specification has some support for this concept through a set of predefined values ('sourcefile' and 'linenumber') for the 'context-type' attribute in <context> elements:

```
<trans-unit>
  <source>hello world</source>

  <!-- Location Reference -->
  <context-group name="po-reference"
  purpose="location">
    <context context-
    type="sourcefile">myfile.c</context>
    <context context-type="linenumber">23</con-
    text>
  </context-group>

</trans-unit>
```

Figure 22: *XLIFF native support for references*

Abstraction of inline codes. To ease the work for translators and to improve translation memory matches, it is beneficial to abstract non-translatable inline codes and markup inside translation units. Gettext supports a vast number of programming languages, each having their own set of rules for how variable parameters are represented. Creating filters to support all these languages is a daunting task, and we believe the best solution is for filter implementers to incrementally add support for abstraction of inline codes in the XLIFF representation of PO files. The following example shows how parameters from C source code can be abstracted:

```
<trans-unit>
  <source>hello <ph id="1" ctype="x-c-
  param">%s</ph>, how are you?</source>
</trans-unit>
```

Figure 23: *Abstraction of inline codes when representing PO in XLIFF*

Generating unique resource identifiers. Many tools in XLIFF-based localisation processes rely on a unique static identifier, the 'resname' attribute, for translation units. With resource types such as Java .properties files this is easy, as they use logical ids for unique identifiers. PO, however, use the English source string as the unique identifier, and this value cannot be used as the 'resname' attribute for translation units, because of the limitation of XML attribute values. To overcome this, we suggest using hash values to create unique identifiers, using the following two rules:

- For non-plural Translation Units, use a string hash of `domain_name + "::" + msgid`. If the Translation Unit is in the default domain, use '`messages`' as the domain name
- For plural Translation Units, use a string hash of `domain_name "::" + msgid + "::plural[" + n + "]"`, where n is the plural index of `msgstr`

Handling Escape Sequences. In source code it is often necessary to use escape sequences to represent characters such as the newline character (`U+000A`), the horizontal tab character (`U+0009`) and non-ASCII Unicode characters. We suggest representing these in XLIFF using the intended characters, and not the escape sequences. For example:

```
msgid ""
"Please Enter the following Data:\n"
"\t- First Name\n"
"\t- Last Name\n"
msgstr ""
```

Figure 24: *PO entry with escape sequences*

This entry should be represented in XLIFF as follows, replacing the escape sequences with the intended characters:

```
<trans-unit>
  <source>Please Enter the following Data:
    - First Name
    - Last Name
</source>
</trans-unit>
```

Figure 25: *XLIFF translation unit with converted escape sequences*

4 XLIFF and the software process

Up until now we have primarily focused on a general mapping between PO and XLIFF. While this is beneficial in itself, the success of adopting XLIFF in open source localisation depends on how well this format can be integrated into development processes and tools. In this section, we first examine the most commonly used Gettext-based localisation workflow, and then extend our discussion to consider incorporating XLIFF into open source localisation. Finally we discuss the need for open source translation tools that support XLIFF and present recent developments in the area.

Most GNU-based open source projects use automated build systems based on GNU Autotools. These systems support integration with Gettext, automatically extracting translatable strings from source-code and, in some cases, merging previously translated string-tables using fuzzy-matching. These build systems are composed of small unix tools, called from macros and build scripts. The Gettext toolkit comes with a set of tools for this purpose:

- `xgettext` — Extracts messages — from source code into PO Template (POT) files
- `msginit` — Initializes a POT file to a language specific PO file
- `msgmerge` — Merges a translated PO file with a newly extracted POT file by adding new translation units, marking obsolete entries (entries that have been removed from source code) and updating extracted comments and source code references
- `msgfmt` — Generates a binary MO file from a translated PO file

4.1 Current Gettext-based workflow

The following diagram gives an overview of how Gettext is integrated into the development process and build system:

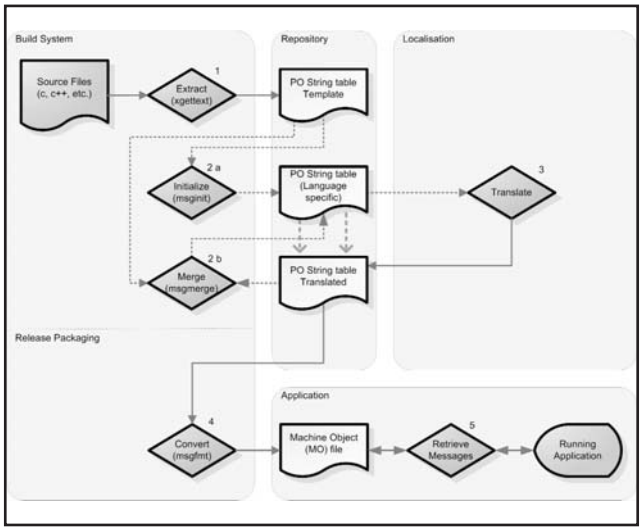


Figure 26: *Current PO-based workflow*

1. Messages are extracted from source files to PO String table Templates (POT files) using the xgettext tool.
2. For each language, previously translated PO files are merged (2b) with the freshly extracted PO Template file using the `msgmerge` tool. If there is no existing PO file for the target language, the POT file is initialised (2a) for the target language using the `msginit` tool. In addition, entries are at this stage in some cases matched against PO Compendium files containing completed translations from across projects.
3. Translators retrieve PO files from the repository, and translate using PO-based localisation tools. Translated PO files are then committed back to the repository.
4. When a maintainer creates a release, Machine Object (MO) files are generated from translated PO files using the `msgfmt` tool, and included in the distributed package.
5. Applications retrieve translated messages from MO files at runtime.

While this diagram gives a good overview of the process, it is important to note that it is only a representation i.e., a common way of using Gettext in the development process, and that most projects use *variations* of this workflow. For example, in many projects it is common that merging of PO files (Step 2) is done by translators in the translation process. In addition, many projects use the intltool package to extract strings to PO from other file types (Desktop entries, User Interface dialogs and XML files) for localisation, and this tool also takes care of re-merging the translated entries. It is also important to note that Steps 1 to 3 are often iterated many times in a single release cycle, providing a way for translators to incrementally translate an application while it is being developed.

4.2 Optional XLIFF workflow in current processes

One approach to incorporating XLIFF in open source is to leave the present processes and workflows untouched, and simply use XLIFF as an optional file format for use by translators in the localisation process. With this approach, PO files are converted to XLIFF for translation, and then back-converted to PO when translation is complete:

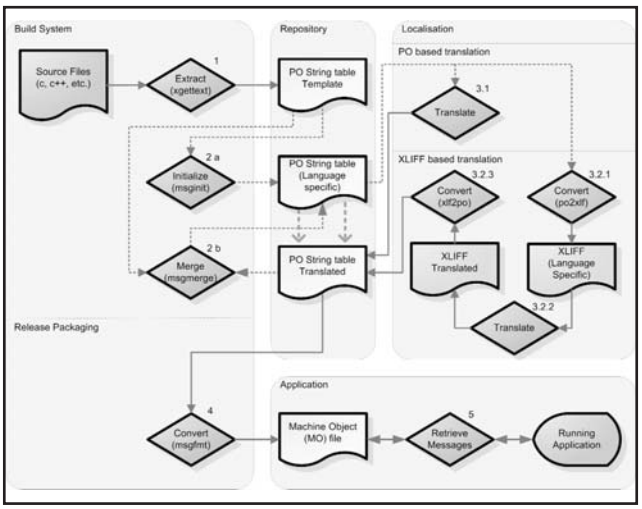


Figure 27: *XLIFF-based translation within a PO-based workflow*

1. and 2. As before (Figure 26).

3. At this point, translators have a choice of two approaches:
   - 3.1. As before, translate using PO tools, or
   - 3.2. Convert to XLIFF for translation, using the `po2xlf`[1] tool:
     - 3.2.1. First, translate using XLIFF editor.
     - 3.2.2. Then convert translated XLIFF back to PO using the `xlf2po` tool.

4. and 5. As before (Figure 26).

A limitation with this approach is that every time the PO sources are updated (this happens quite frequently in open source development processes), new XLIFF files would have to be generated, losing the rich metadata stored in the old XLIFF sources. With this approach, PO is the *persistent* file format (Section 2.2), and XLIFF is simply a *transient* file format used in the localisation process. To overcome these limitations, we need an approach in which XLIFF is stored in the repository as a persistent format.

4.3 Native XLIFF-based workflow

In contrast to the minimalist view of section 4.2, the following approach proposes changes to the build system to allow the use of XLIFF as the persistent file format in the build systems and localisation process, totally eliminating the need for the PO format in the *localisation process:*

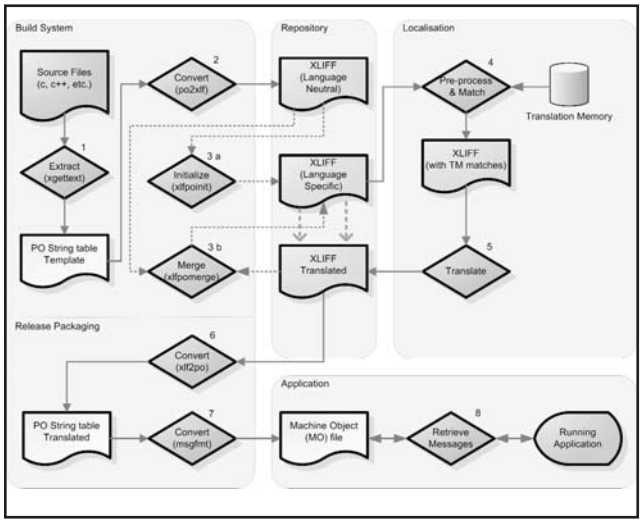


Figure 28: *Fully XLIFF-based workflow*

1. Extract messages from source code to PO template using the `xgettext` tool.
2. Convert the PO Template file to XLIFF using the `po2xlf` tool.
3. Initialize (a) (using `xlfpoinit`) or update (b) (using `xlf-pomerge`) the XLIFF file for each language.
4. Optionally pre-process the XLIFF file for each language (for example adding translation suggestions).
5. Translate XLIFF file for each language using XLIFF-based localisation tools.
6. When a release is prepared, convert the translated XLIFF files back to PO (using `xlf2po`).
7. Create MO files from PO files for distribution, using the msgfmt tool.

In this new workflow we have replaced PO with XLIFF as the persistent format, storing XLIFF files in the version control repository. The need for PO — other than as an intermediate format for communicating with Gettext — is eliminated. PO files are never edited by developers or translators, being used only as a temporary format in the software development process. Developers, localisation engineers and translators can then all benefit from XLIFF in all stages of the development process.

To support this approach, we have introduced two new tools: `xlfpoinit` and `xlfpomerge`. These tools serve the same purpose as their PO equivalents (`msginit` and `msgmerge`), and provide a way to manage updating sources in the localisation process.

By using this approach, open source projects can leverage all the benefits of XLIFF-based localisation, but still use Gettext internally as the resource format. This approach also builds a platform for further expansion of the localisation process in open source projects, where other source formats (such as DocBook and Desktop Entry files) can be converted to XLIFF for localisation.

4.4 Gettext Integration

Building on the approach presented above, it would now be possible to integrate XLIFF within the Gettext toolkit, eliminating the need for custom XLIFF filters and tools in the build system and development process:

[1] Prototypes of `po2xlf`, `xlf2po`, `xlfpoinit` and `xlfpomerge` have been developed as part of this research, and are available through the XLIFF Tools Project (http://xliff-tools.freedesktop.org)

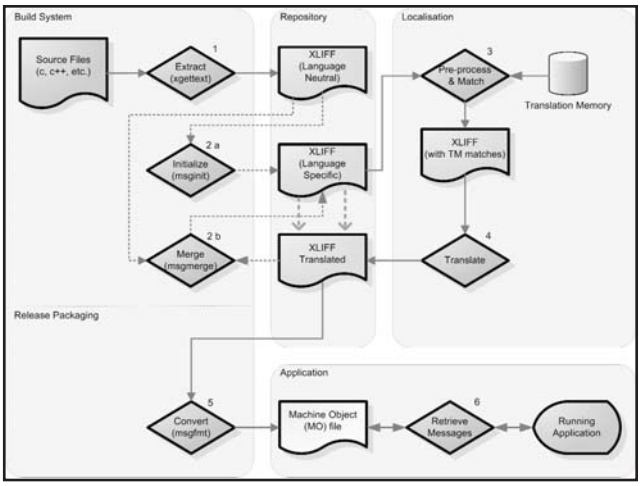


Figure 29: *XLIFF Gettext integration*

1. Extract messages from source code to a language-neutral XLIFF file, using the modified `xgettext` tool.
2. Initialize (a) or update (b) the XLIFF file for each language, using the modified `msginit` and `msgmerge` tools.
3. Optionally pre-process the XLIFF file for each language (for example adding translation suggestions).
4. Translate XLIFF file for each language using XLIFF-based localisation tools.
5. Create MO files from XLIFF files for distribution, using the modified `msgfmt` tool.

This approach totally eliminates the need for the PO format, as the Gettext tools would work natively with XLIFF. We have not prioritised further researching this approach, as our goal with this research is to propose XLIFF as a common resource format for all open source processes, and the 'native approach' described above meets our needs in this area.

5 Open source Localisation and Open Standards

In this section we present possible solutions to the more challenging issues facing open source localisation — the actual localisation process. We do this by further examining common open source localisation processes (background presented in Section 2.1), and propose high level solutions based on standards such as TMX, TBX and Translation Web Services (TWS). Some of these standards are still in draft form (Translation Web Services, TBX), while others have been around for some time and have received broad industry acceptance (TMX). Common to all these standards, is the fact that they have emerged as a result of needs in the commercial localisation industry, and promote best practices in the field.

Some of the themes we discuss in this section are not revolutionary or new in commercial localisation settings (in fact, we draw examples from some of the available proprietary tools). They are however new in the setting of open source and collaborative localisation, and contribute to meeting many of the challenges facing open source localisation:

1. Limited Translation reuse.
2. Ad hoc Terminology Management.
3. High level of technical skills needed to contribute.
4. Error prone due to highly manual processes.
5. No standardised way of contributing to the localisation process — different projects use different approaches, but often with the same people contributing.
6. Limited support for quality control in the localisation process.

We will propose solutions for each of these issues. In Section 5.1 we discuss how we can increase translation reuse (1) through centralised translation memory repositories. Section 5.2 presents possible solutions for improving terminology management (2) in open source, through the adoption of standards such as TBX within an XLIFF-based localisation process. Finally, in Section 5.3 we discuss the need for service-based localisation workflows, and simplification of the localisation process through process automation and well defined communication protocols (4, 5 and 6). The main aim of this discussion is not to present the *right* solution (at this point we present 'sketches' rather than 'full solutions'), but to identify ways in which workflows based on open standards can meet the challenges facing existing open source localisation processes.

5.1 Shared Translation Memory Repositories

As described in Section 2.1.3, the current Gettext-based localisation practices provide very limited support for translation reuse, mainly due to the limited support for segmentation and abstraction of inline codes in the PO format. However, large open source projects, like KDE, GNOME and the Linux distributions, are ideal ecosystems for centralised translation memory databases, due to the vast quantity of translatable data available and the large contributor base.

It is instructive to look at each of these large open source projects from an industry-wide viewpoint. Localisation projects are *Localisation Providers;* community contributors are *translators* 'working' for the provider, and software maintainers are localisation *customers* consuming services from the provider. Providers retrieve sources for translation from customers, and ideally, these sources should be matched against the providers' translation memory, and sent to translators for completion. When a translation is complete, providers should incorporate the updated translation within their TM, increasing its value. This translation memory is the critical intellectual property of the provider.

However, in contrast to commercial localisation providers, open source TM assets could be shared across projects, further increasing the likelihood of a positive match. In addition, TM assets could be made available for download in the form of TMX archives, and imported into client localisation tools supporting the TMX format.

The following figure depicts a typical workflow for incorporating a centralised Translation Memory in the localisation process:

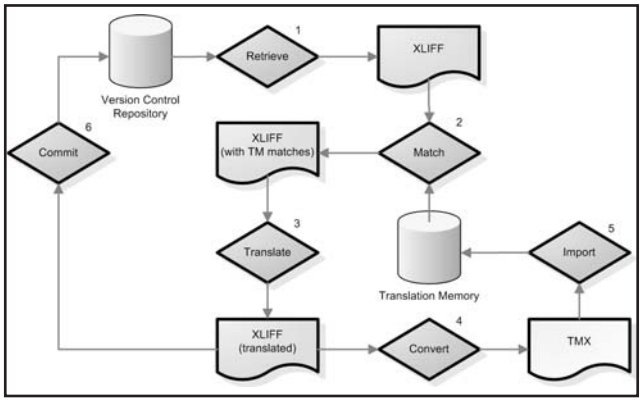


Figure 30: *XLIFF workflow with TMX-based Translation Memory*

In the above figure, XLIFF documents are retrieved from the version control repository (1), and submitted to the TM system for pre-translation (2). In the pre-translation process, exact and fuzzy matches are added to the XLIFF documents. In the translation process, the translator can then accept or reject the translation suggestions, and further translate the XLIFF documents (3). Approved entries in the translated XLIFF documents are then exported to TMX (4), and imported to the TM repository (5). Independent of this workflow, translated documents are committed to the version control repository.

The XLIFF standard will shortly be revised to provide extensive support for segmentation, in addition to the existing support for abstraction of inline codes. Combining XLIFF with the Segmentation Rules eXchange (SRX) standard and TMX, it will now be increasingly feasible to implement efficient Translation Memory support based on open XML standards. With widely published rules for text segmentation (based on the SRX standard), efficiently leveraging translations from across project boundaries becomes possible, as data from different TMX-enabled translation memories can share common segmentation rules.

5.2 Terminology Management

As in the case of TM matching, little standardisation of terminology management has been undertaken in open source localisation. Warburton (2005b) identifies the advantages of early identification and definitions of terms in the localisation process. The XLIFF format provides good support for this process through the `<mrk>` element:

```
<trans-unit id="1">
  <source>Please make sure the path separator is
  set to '/'</source>
</trans-unit>
```

Figure 31: *Translation Unit **without** terms identified*

Here we could identify the term "*path separator*" by encapsulating it in a `<mrk>` element:

```
<trans-unit id="1">
  <source>Please make sure the <mrk
  mtype="term">path separator</mrk> is set to
  '/'</source>
</trans-unit>
```

Figure 32: *Translation Unit **with** terms identified*

Identification of terms should ideally occur at a very early stage in the localisation process. This makes it possible for all language team members to standardise terminology prior to any translation work being done. As English is used as the source language in most open source projects, identification of terms can be done synchronously with the development effort, in a process similar to that depicted in the following diagram:

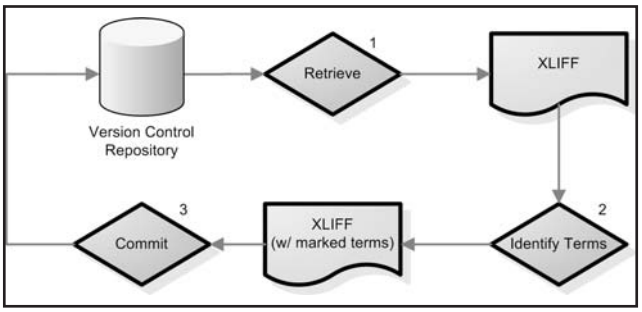


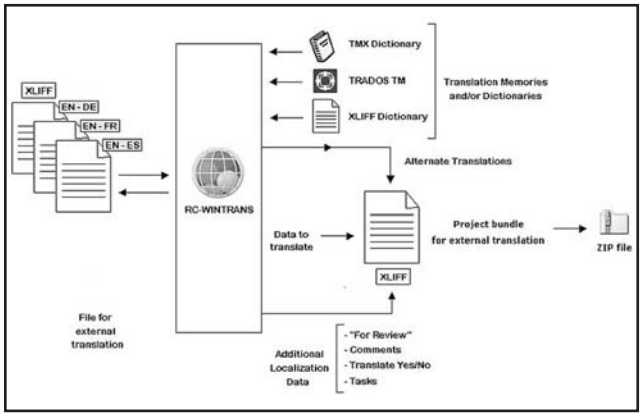
Figure 33: *Marking terms in XLIFF documents*

Here, an XLIFF document (ideally a language-neutral document) is retrieved from the version control repository (1). Terms are identified and marked in the XML document (2), before the file is again committed to the repository (3). In addition, a mechanism should exist that allows terms to be identified after a translation process is started.

The process of extracting terminology could be automated, by checking a document against a list of pre-existing terms. It is important to note however, that different projects might use different definitions for identical terms, and that such a process does not take into account new terms that are not present in the predefined list. This could be improved by having a semi-automated process, where potential terms were automatically flagged, and a 'review' phase could be introduced to identify new terms and approve the terms automatically extracted in the process.

Obviously, the process of *identifying* terms is only valuable if definitions of these terms exist and are properly managed, which is the job of terminology management systems. Availability of such systems is becoming increasingly important in open source localisation, which involves geographically distributed contributors working on a common translation project. In open source, this is usually handled through sharing dictionaries using collaborative systems such as wikis.

Using web pages or wikis for sharing terminology databases is however not an optimal solution. The ideal situation would be to have the terminology data in a format that tools in the localisation process could easily access, exchange and use to automatically lookup terms that are identified in the source file. This provides a pressing need for the adoption in OSS of the TermBase eXchange standard (Section 2.2.3). By adopting the TBX standard for terminology management in open source software, it will be possible to automatically include definitions for terms present in an XLIFF document, and ship this with translations. This again highlights the advantage of XLIFF over PO, and utilisation of XLIFF as a format for *project bundles* is already proven by commercial practice, with localisation tools such as RC-WINTRANS (Figure 34 below) using this approach.



(Adopted from Schaudin, 2005)

Figure 34: *RC-WINTRANS' use of XLIFF project files*

5.3 Service-based Localisation Workflows

In the previous two sections we described an approach for incorporating centralised Translation Memories and Terminology Management in open source localisation. Current localisation practices in projects such as the Fedora and KDE localisation projects require translators to manually retrieve PO files from version control repositories for translation. Having translators retrieve source files directly from the development repositories eliminates the possibility of automated server side processing. In this section we identify a need for a higher level of abstraction in the localisation processes — automated workflow systems — to accommodate many of the aspects we have discussed so far.

We propose a generalised model (Figure 35 below) based on interoperable communication protocols such as XML-RPC or SOAP Web Services, that allows for communication between clients (rich clients like KBabel, or web-based systems like Pootle) and upstream translation projects (such as KDE, GNOME and Fedora). This approach removes the need for translators to engage directly with technical issues such as version control systems, instead providing a more abstract service model which allows better control and management of the process.

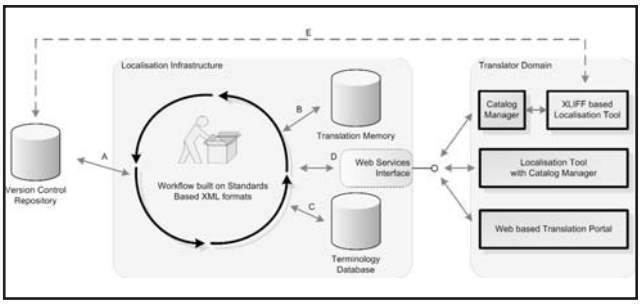


Figure 35: *Localisation Infrastructure based on standards-based XML formats*

This model is based on XLIFF files (and other project-related metadata) being stored in a version control repository — corresponding to the current practice of storing PO files in the repositories. Using this workflow, TM entries and Terminology definitions (as described in Section 5.1 and 5.2) could be added to the XLIFF archives before they are sent to translators (B and C); similarly new entries could be added to the TM before committing the translations to the repository (A).

Perhaps the most useful set of tools in current open source localisation practices are the *catalogue managers* (Section 2.1.2), in the form of translation status pages and tools like KBabel's Catalog Manager. These tools present translation tasks and status reports in user friendly and intuitive ways, and in addition often provide mechanisms for direct retrieval of sources from version control repositories. Recently, through portals such as Pootle, Rosetta and IRMA, these status pages have been extended to become fully fledged localisation solutions, allowing translations to be performed in a web-based environment. Extending these concepts, we propose a web services-based interface (D) to the Localisation Infrastructure system, enabling any client supporting the protocol (such as web-based translation portals, rich client localisation tools) to communicate with the upstream localisation project.

The OASIS Translation Web Services (TWS) committee aims to develop a web services-based interface for automating communication between localisation vendors and clients. This specification includes methods for requesting localisation quotes, retrieving and submitting localisation jobs, and querying the status of localisation jobs. In addition, the specification covers queries to language service providers to determine which services and languages are supported.

Open source localisation processes based on community contribution do not fit in very well within the scope of the TWS specification. In open source projects, translators (community contributors) are the active party, providing their services by translating an application to a specific language. There is however an emerging need for standardised and automated processes in open source localisation, to allow community contributors to query, retrieve and submit translations. Currently this can be done only by manually checking the translation status pages for a project — commonly via a web browser, or alternatively (in the case of the KDE project) by using a tool such as the KBabel Catalog Manager to interact with the version control system to retrieve this information.

In contrast, a web services-based protocol such as SOAP or XML-RPC, provides considerable flexibility in implementing clients. By implementing a standards-based interface, custom 'Catalog Manager' applications can be used to retrieve translation projects (collection of XLIFF files), which in turn can be translated in any XLIFF-enabled editor. The same interface could then be used to develop web-based translation portals, rich client catalogue managers, and localisation suites. Currently this is a big obstacle for translation portals such as Pootle, Rosetta and IRMA, as there is no standardised mechanism for upstream translation updates.

Defining a standard interface for interacting with open source localisation repositories also opens up possibilities of composing services in a unique way. This is especially useful for community driven Linux distributions such as Ubuntu and the Fedora Project — distributions that are composed by combining software from a multitude of projects. For example, in the Fedora project, it would be beneficial if translators could automatically retrieve translation tasks for the current GNOME or KDE release, without having to manually track each project independently.

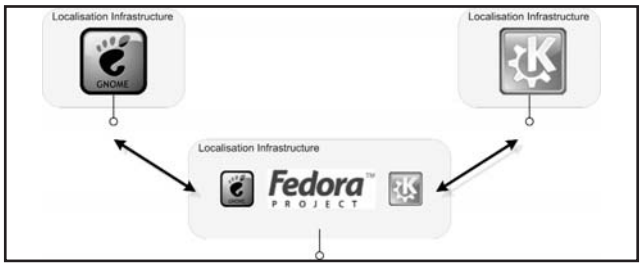


Figure 36: *Composing Localisation Projects*

In building an infrastructure targeting open source localisation, we cannot assert that localisation contributors have high-speed and reliable internet connections, or even have internet connections at all. In developing countries (which indeed have some of the most interesting open source localisation projects), it is often more feasible to arrange translation gatherings (nick-named 'translate-a-thons'), where contributors come together in school computer labs or similar and translate open source software using thin client web-based translation tools, with one computer acting as a server running a localisation portal such as Pootle. The proposed model also caters for this scenario, as local servers could be set up, or sets of translation tasks could be retrieved upfront.

In addition, the proposed localisation model (Figure 35) also supports the current practice of contributors directly interacting with the version control repositories, retrieving and committing sources for translation in their XLIFF-based localisation tool of choice (E). We see it as important to support this simple approach, as it would eliminate the danger of a single point of failure, and avoid the risk of losing contributors hesitant to change their current localisation practices.

The overall aim in proposing a service-based localisation infrastructure is to enhance and simplify the localisation process from a translator point of view. The level of technical knowledge needed to contribute to current open source localisation processes acts minimally to decrease efficiency, and at worst, to turn potential contributors away. With the advent of structured approaches to translation memory and terminology management, these issues must be addressed — otherwise these technologies may introduce yet another level of complexity for the translation contributors.

6 Conclusions

In Section 3 we presented the main contribution of this paper — a bridge between XLIFF (the industry standard for exchange of localisable data) and Gettext (the de facto standard in open source localisation). This was accomplished by developing an XLIFF representation guide for the Gettext PO file format, followed by a discussion of how best to incorporate XLIFF within present open source development processes. In this section we specifically targeted the PO format, as it is being used (indeed, exploited) as a common resource format for localisation of a number of file types in open source. Our goal, however, has not been simply to eliminate PO in favour of XLIFF in Gettext-based localisation, but rather to propose XLIFF as the standard resource format for all open source localisation, in much the same way as PO is being used (or indeed, overused) at present. This research has only fulfilled part of that goal, as only the PO format has been thoroughly covered. To make XLIFF a more attractive option in open source localisation, tools and filters to convert between XLIFF and other common OSS file formats are needed.

By adopting XLIFF as the common resource format in open source localisation, we have also managed to decouple technologies in the localisation and internationalisation process. Developers are no longer limited to using Gettext, and can investigate using other alternatives without disrupting the localisation process. Such decoupling brings forth a unique opportunity for further research in the area of localisation technologies in the development process.

During the course of this research, we have noticed an increasing interest from the open source community in regard to adopting XLIFF in localisation processes presently based on Gettext and the PO format. Developers from leading localisation projects (including Ubuntu's Rosetta, Gnome, KDE, Fedora and Pootle) have expressed an interest in XLIFF as a replacement for PO. It is, however, too early to determine if, and when, these new standards will be adopted in these projects, as much of the supporting infrastructure and tools (mainly XLIFF filters and localisation tools) do not yet exist. In addition, despite the deficiencies and limitations of Gettext and PO, current localisation practices are to a large extent successful — with projects like KDE and GNOME being localised to over 80 languages, and the benefits of switching to XLIFF once will have to be overwhelming to justify such a move.

In Section 5 we looked beyond XLIFF, discussing possible future opportunities for open source localisation processes, further extending the bridge between open source and proprietary standards. The discussion was built on the foundation laid in the previous section — XLIFF as the common resource format — and focused on three main areas in need of better solutions in open source: *translation reuse, terminology management, and service-based localisation workflows*.

Translation reuse in open source has up until now only been addressed in an ad hoc fashion, through PO Compendia and the KBabel PO editor, and there are no structures in place for sharing translation memory data. In addition, the data fed into these translation memories are not ideal, as current PO-based processes have no support for segmentation, alignment and abstraction of inline codes and markup. Adopting XLIFF and TMX in these processes provides a foundation for building quality translation memories and storing higher quality data, which can in turn increase translation efficiency in years to come. There are, however, no existing open source TMX-certified Translation Memory systems available, and further research and development is needed in this area.

Terminology management is arguably as important as translation reuse for quality localisation solutions. In current open source processes terminology is, at best, handled in an ad hoc fashion. Terms are not identified prior to the translation process, and if terminology management is undertaken at all, language teams simply use combined bilingual glossaries of words that have been previously identified in the translation process. The broader localisation industry has identified and acknowledged the importance of proper terminology management, and has agreed on a standard file format, TBX, for exchanging terminology. Further research is needed in defining workflows incorporating terminology in open source, and further, developing open source terminology management systems supporting these standards.

As with many other industries, the localisation industry is looking at ways of automating the business processes through service oriented architectures based around SOAP Web Services. This work is now being formalised through the OASIS Translation Web Services committee, developing a specification that provides interfaces for automating communication between localisation vendors and customers. As the commercial interest in open source continues to grow, Translation Web Services (TWS) can be of great benefit to open source software. Open source software vendors can use this channel to localise resources (now based on XLIFF) through commercial localisation service providers. There is, however, a need for service oriented architectures in the collaborative open source localisation environment, as this process is not covered by the TWS specification. In open source localisation, translators are the active part (the localisation provider), providing services to an open source localisation project (the client), and research is needed to define a service oriented architecture for these environments. As with industry standards in the field, the goal of this research is to provide an automated system eliminating many of the trivial and non-localisation related tasks relating to the process. In other words, the goal must be to let the translator concentrate on the translation and not the surrounding technicalities.

The standards we have covered in this paper, except for TMX, are still fairly youthful, and are still in a process of maturing. XLIFF 1.1 was approved as an OASIS Committee Specification in November 2003, but is still being finalised before submission for approval as an OASIS standard. Revision 1.1 of the standard limited the amount of free-form metadata allowed through deprecation of the <prop> element, and provided a larger set of pre-defined attribute values. As free-form data is being restricted in the XLIFF namespace, the specification now recommends using other namespaces within the document for storing additional information, further expanding the possible usage areas of the format. This has recently been demonstrated through the work on the XLIFF Representation Guide for HTML, where common (X)HTML attributes are added to the translation units, further enriching the set of available metadata.

As the localisation industry adopts service-based architectures and Translation Web Services, we believe there will be a strong focus on the linkage between standards-based XML file formats. The beginnings of this movement can already be seen, through the inclusion of elements from the XLIFF namespace in the TWS specification, and it is reasonable to believe that there will also be a stronger linkage between XLIFF and TBX in future revisions of the standard, allowing linking of terms identified in the XLIFF document with definitions in the TBX document.

Adoption by the open source community of the standards-based file formats discussed in this paper will be of considerable benefit to OSS localisation processes, and provide an open, fertile environment within which these standards may evolve toward technical maturity. This will not only benefit the open source communities, but also the commercial localisation industry. Up until now, most implementations taking advantage of these standards have been proprietary solutions, only tested by a single vendor. Successful adoption of these standards in open source can in turn provide a playing field in which these standards can evolve and mature independently of commercial interests.

As a conclusion to this paper, the XLIFF resource format is mature and rich enough to provide a valuable replacement for the Gettext PO format. But for this to happen, development of localisation tools supporting XLIFF is needed, and, further, development of XLIFF filters for other file types that currently use PO in the localisation process. Open source localisation can also benefit from standards beyond XLIFF and, with further research, the localisation process can be enriched through standardised management of translation reuse and terminology. To fully take advantage of this, structured localisation workflows need to be developed — and these can benefit from the development in the area of Translation Web Services, providing a way of automating the localisation process.

Acknowledgements

The authors would like to acknowledge the assistance and support of Paul Gampe and the team at Red Hat in Brisbane, and all those who have contributed to this work through the XLIFF Tools project: Josep Condal, Bart Cornelis, Fredrik Corneliusson, Karl Eichwalder, Tim Foster, David Fraser, Bruno Haible, Rodolfo M. Raya and Yves Savourel.

References

Berglund, E. and Priestly, M. (2001) Open-source documentation: in search of user-driven, just-in-time writing. In *19th annual international conference on Computer documentation,* October 21-24 2001, pp. 132-141. Sante Fe, New Mexico, USA.

Costin, C. and Kiefer, M. (2004) *The KBabel Handbook.* http://docs.kde.org/en/3.1/kdesdk/kbabel/ (accessed 1/10/2004).

Diehl, T., Schutte, F. and Zeini, A. (2002) *The KDE Translation HOWTO.* http://i18n.kde.org/sitedoc.html (accessed 29/8/2004).

Foster, T. (2004a) *Open source translation tools, where are they.* http://blogs.sun.com/roller/page/timf/20040823#open_source_translation_tools (accessed 1/10/2004).

Foster, T. (2004b) *Using Translation Technology at Sun Microsystems.* http://developers.sun.com/dev/gadc/technicalpublications/whitepapers/translation_technology_sun.pdf (accessed 29/8/2004).

Free Software Foundation (2002) *GNU gettext Utilities.* http://www.gnu.org/software/gettext/manual/gettext.html (accessed 29/8/2004).

Free Software Foundation (2004) *GNU Coding Standards.* http://www.gnu.org/prep/standards/standards.html (accessed 8/6/2005).

Free Standards Group (2003) *OpenI18N 1.3 Globalization Specification.* http://www.openi18n.org/docs/pdf/OpenI18N1.3.pdf (accessed 2/10/2004).

Gwynne, T. and Harries, D. (2004) *Localising GNOME applications.* http://developer.gnome.org/projects/gtp/l10n-guide/ (accessed 29/8/2004).

Hars, A. and Ou, S. (2001) Working for free? Motivations of participating in open source projects. In *34th Annual Hawaii International Conference on System Sciences,* 3-6 Jan. 2001, pp. 9. Hawai, USA.

Iverson, S. (2003) 'Working with Translation Memory' *Multilingual Computing & Technology,* 14(7), 35-37.

Jewtushenko, T. (2004) An Introduction to XLIFF. In *IV International LRC Localisation Summer School 2004,* 2/6/2004, pp. N/A. LRC, University of Limerick, Ireland.

KDE.org Team (2004) *What is Scripty?* http://developer.kde.org/documentation/misc/whatisscripty.php (accessed 10/10/2004).

Levitt, G. (2003) 'Internet-based Sharing of Translation Memory' *Multilingual Computing & Technology,* 14(5), 38-41.

Localisation Industry Standards Association (2004) *Segmentation Rule eXchange (SRX) Specification.* http://www.lisa.org/oscar/seg/srx.htm (accessed 8/6/2005).

Palas, P. and Karásek, M. (2003) *Globalization of Enterprise Data Using Web Services.* http://moravia-it.com/Download/Globalization_of_Enterprise_Data_Using_Web_Services.pdf (accessed 3/6/2005).

Raya, R. M. (2004a) *Re: Translation tools for Fedora Project.* https://listman.redhat.com/archives/fedora-trans-list/2004-January/msg00077.html (accessed 1/10/04).

Raya, R. M. (2004b) *XML in localisation: A practical analysis.* http://www-106.ibm.com/developerworks/xml/library/x-localis/ (accessed 1/11/2004).

Savourel, Y. (2001) *XML Internationalization and Localization,* Sams Publishing, Indianapolis, Indiana USA.

Savourel, Y. (2003) 'An Introduction to Using XLIFF' *Multilingual Computing & Technology,* 14(2), 28-34.

Schaudin Software Localization Solutions (2005) *RC-WinTrans X8 Project Manager Key Features.* http://www.wernerschaudin.de/54.0.html (accessed 26/5/2005).

The Translation Web Services Technical Committee (2005) Translation Web Services (Draft Committee Specification). http://www.oasis-open.org/committees/download.php/13001/trans-ws-spec.html (accessed 8/6/2005).

The XLIFF Technical Committee (2003a) *A white paper on version 1.1 of the XML Localisation Interchange File Format (XLIFF).* http://www.oasis-open.org/committees/download.php/3110/XLIFF-core-whitepaper_1.1-cs.pdf (accessed 29/8/2004).

The XLIFF Technical Committee (2003b) XLIFF 1.1 Specification. http://www.oasis-open.org/committees/xliff/documents/xliff-specification.htm (accessed 1/11/2004).

Van Schouwen, R. (2004) GNOME *Localization and Usability.* http://www.cs.vu.nl/~reinout/pdf/GNOME_localization_and_usability.pdf (accessed 29/8/2004).

Viswanadha, R. and Scherer, M. (2004) Localizing with XLIFF & ICU. In *26th Internationalization and Unicode Conference,* San Jose, California, USA.

Wang, S. (2004) *Fedora Translation Project FAQ.* http://fedora.redhat.com/participate/translation-faq/ (accessed 29/9/2004).

Warburton, K. (2005a) *Sloppy Terminology: The Sky Won't Fall, Will It?* http://www.lisa.org/term/ (accessed 8/6/2005).

Warburton, K. (2005b) *Terminology management: Globalizing your e-business.* http://www-306.ibm.com/software/globalization/topics/terminology/index.jsp (accessed 7/6/2005).

Zetzsche, J. (2003) 'TMX Implementation in Major Translation Tools' *Multilingual Computing & Technology,* 14(2), 23-27.

Asgeir Frimannsson recently completed an honours degree in Software Engineering at the Queensland University of Technology. His thesis on XML standards in OSS Localisation — a work on which much of this paper is based — was awarded the LRC Best Thesis Award in 2005. He will shortly continue as a PhD student in a related area, and can be reached at asgeirf@gmail.com.

James M. Hogan is a Senior Lecturer in the Faculty of Information Technology at the Queensland University of Technology, where he leads research and teaching in Software Internationalisation. His interests include XML standards and context-based matching and retrieval. He may be contacted at j.hogan@qut.edu.au.