



Facultade de Informática da Universidade de A Coruña
Tecnoloxías da Información e das Comunicacións

PROXECTO DE FIN DE CARREIRA
Enxeñería Informática

**GNOMECAT, un editor de ficheiros GNU Gettext para o
proxecto GNOME**

Alumno: Marcos Chavarría Teijeiro
Director: Fernando Bellas Permuy
Data: 15 de xuño de 2015

Resumen:

A internacionalización e localización de software son dous aspectos moi importantes nos que traballar en produtos software modernos. Que un programa estea localizado a un idioma que o usuario entenda é fundamental para que este usuario se sinta cómodo empregándoo.

O sistema de internacionalización e localización GNU Gettext é unha das solucións para estes dous problemas máis empregada actualmente. Para facer a localización dos programas os tradutores deben editar unha serie de ficheiros. Para facilitar dita tarefa existen ferramentas denominadas CAT (*Computer Assisted Translation*).

Neste proxecto preténdese crear unha nova ferramenta para a asistencia á tradución centrada na organización de software libre GNOME. Dita ferramenta editará os ficheiros de GNU Gettext e empregará unha interface construída coas ferramentas que aporta o stack de GNOME.

Para facilitar a participación de futuros desenvolvedores empregarase unha linguaxe de programación cunha sintaxe moi amigable en contraste coa empregada noutros proxectos de GNOME. En concreto empregarase a linguaxe de programación Vala.

Este proxecto foi elaborado durante dous veráns consecutivos como parte do programa Google Summer of Code.

Lista de palabras chave:

- GNOME
- Localización (L10N)
- Internacionalización (I18N)
- Computer Assisted Translation (CAT)
- GTK+
- Vala
- GNU Gettext
- Google Summer of Code

Agradecementos

Quero aproveitar este espazo para amosar o meu agradecemento a todas aquelas persoas que dunha maneira ou outra estiveron relacionadas coa elaboración deste proxecto. Gustaríame agradecer a axuda prestada polos mentores dos dous Summer of Code que fixen e a guía que me ofreceu Fernando Bellas para poder presentar a realización deste programa como o meu Proxecto de Fin de Carreira. Quero, ademais, facer facer unha especial mención a Daniel Mustieles, coordinador do equipo de tradutores de GNOME ao castelán, pai deste proxecto e a persoa que máis me axudou para realizalo. Tamén, expresar o meu agradecemento a todos os usuarios e desenvolvedores de GNOME que dende un chat de IRC, un correo ou un comentario dun blogue me axudaron a mellorar o programa.

Gustaríame agradecer tamén as horas de dedicación de todo o persoal docente que dende os tres anos que empecei a ir a escola, ata as últimas materias da carreira me estiveron formando. Sen todo o tempo que dedicaron a formarme non tería os coñecementos necesarios sobre a informática e sobre a vida para poder levar a cabo o presente proxecto.

Non me quero esquecer de todas o resto de persoas que me acompañan neste camiño que é a vida. Quero dar grazas a meus pais, tíos, avós e o resto da miña familia que sempre me apoiou canto puido e máis.

Teño a sorte de contar con moitos e bos amigos e amigas así que por último e non por elo menos importante gustaríame darlles a todos eles as grazas por eses anacos de tempo agora xa inmortais nos que xunto a eles tiveron a oportunidade de viaxar, saltar, bailar, estudar, rir, chorar, falar, cantar, berrar, comer, soñar, beber... en definitiva, de vivir.

A todos vós, moitas grazas!

Marcos Chavarría Teijeiro
A Coruña, 15 de xuño de 2015



Este traballo está licenciado baixo Creative Commons
Attribution-ShareAlike 4.0 International License.

Índice general

Índice de figuras	IX
1. Introducción	1
1.1. Internacionalización e Localización de Software	1
1.2. O Proxecto GNOME	1
1.2.1. Localización e Internacionalización no Proxecto GNOME	3
1.3. Google Summer of Code	3
1.4. Motivación	4
1.5. Estrutura da memoria	4
2. Estado do Arte	7
2.1. Internacionalización e localización con GNU Gettext	7
2.1.1. Ficheiros Gettext PO	11
2.2. Ferramentas CAT do mercado	14
2.2.1. GTranslator	14
2.2.2. Lokalize	15
2.2.3. Virtaal	16
2.2.4. OmegaT	17
2.2.5. Google Translation Toolkit	17
2.2.6. Transifex	18
2.2.7. Outras ferramentas	19
2.3. Características xenéricas das ferramentas CAT	20

2.3.1. Memoria de Tradución	20
2.3.2. Glosario	21
2.3.3. Previsualización	21
2.3.4. Tradución Directa	22
3. Fundamentos Tecnolóxicos	23
3.1. Ferramentas empregadas	23
3.2. Bibliotecas empregadas	25
4. Metodoloxía	27
4.1. eXtreme Programming	27
4.1.1. Valores de eXtreme Programming	27
4.1.2. Prácticas recomendadas por eXtreme Programming	29
4.2. Metodoloxía seguida	31
5. Planificación e Seguimento	33
5.1. Google Summer of Code 2013	33
5.1.1. Primeira Iteración: Análise, deseño xenérico e inicio da implementación	34
5.1.2. Segunda Iteración: Linguaxes e Interface	35
5.1.3. Terceira Iteración: Interface, iteradores e buscas	38
5.1.4. Cuarta Iteración: Ficheiros po, Autotools, Proxectos, barra de busca	40
5.1.5. Quinta Iteración: Preferencias, limpar código e documentación . . .	42
5.1.6. Estado ao fin do GSoC 2013	43
5.2. Primeiro cuatrimestre curso 2013/2014	44
5.2.1. Primeira iteración: cambios na interface	44
5.2.2. Segunda iteración: Refactorizar navegadores e melloras na interface .	45
5.2.3. Presentación para o GSoC 2014	47
5.3. Google Summer of Code 2014	47
5.3.1. Primeira iteración: Redeseño UI	48
5.3.2. Segunda Iteración: Bindings Gettext-PO e interface	50

5.3.3. Terceira Iteración: Perfiles de usuario e interface	51
5.3.4. Cuarta Iteración: Plugins e GUADEC	53
5.3.5. Quinta Iteración: Detalles finais e escribir documentación	55
6. Análise de Requisitos	57
6.1. Consultas cos tradutores	57
6.1.1. Resumo das peticións	58
6.2. Análise doutros aplicativos do mercado	60
6.3. Requisitos do aplicativo	60
7. Deseño e Implementación	63
7.1. Módulo de Ficheiros	63
7.1.1. Implementación Xenérica	63
7.1.2. Ficheiro PO	65
7.2. Módulo de Linguaxes	68
7.3. Interface Gráfica	69
7.3.1. Evolución	69
7.3.2. Paneis	74
7.4. Navegación e Busca a través do documento	81
7.5. Preferencias	83
7.5.1. Módulo de Perfiles	84
7.6. Plugins	85
8. Conclusións e Traballo futuro	87
8.1. Conclusións	87
8.2. Traballo Futuro	87
A. Escoller a licenza do programa	89
B. Instrucións de Compilación e Instalación	91
C. Bibliografía	93

Índice de figuras

1.1. GUADEC A Coruña 2012	2
2.1. Interface de GTranslator	15
2.2. Interface de Lokalize	16
2.3. Interface de Virtaal	17
2.4. Interface de OmegaT	18
2.5. Interface de Google Translation Toolkit	19
2.6. Interface de Transifex	20
2.7. Ferramentas CAT máis empregadas segundo [Lon06].	21
5.1. Diagrama de Gantt da primeira iteración do GSoC 2013.	36
5.2. Diagrama de Gantt da segunda iteración do GSoC 2013	37
5.3. GUADEC 2013 (Brno)	38
5.4. Diagrama de Gantt da terceira iteración do GSoC 2013.	39
5.5. Diagrama de Gantt da cuarta iteración do GSoC 2013.	41
5.6. Diagrama de Gantt da quinta iteración do GSoC 2013.	43
5.7. GUADEC Hispana 2013 (Madrid)	45
5.8. Diagrama de Gantt da primeira iteración do primeiro cuatrimestre do curso 2013/2014.	46
5.9. Diagrama de Gantt da segunda iteración do primeiro cuatrimestre do curso 2013/2014.	47
5.10. Diagrama de Gantt da primeira iteración do GSoC 2014.	49
5.11. Diagrama de Gantt da segunda iteración do GSoC 2014.	51

5.12. Diagrama de Gantt da terceira iteración do GSoC 2014.	52
5.13. GUADEC 2014 (Strasbourg)	53
5.14. Diagrama de Gantt da cuarta iteración do GSoC 2014.	54
5.15. Diagrama de Gantt da quinta iteración do GSoC 2014.	56
6.1. Diagrama UML de casos de uso do aplicativo	62
7.1. Diagrama de Clases do módulo ficheiros	64
7.2. Diagrama de Clases do ficheiro PO	66
7.3. Diagrama de Clases do módulo de Linguaxes	68
7.4. Primeira versión da interface de usuario	70
7.5. Diálogos da primeira versión da interface de usuario	70
7.6. Segunda versión da interface de usuario	72
7.7. Barra de busca	72
7.8. Diferentes Barras de Ferramentas	73
7.9. Barra de Notificacións	74
7.10. Panel de Benvida	75
7.11. Panel de Ficheiros Abertos	75
7.12. Panel de Abrir Ficheiro	76
7.13. Panel de Edición de Ficheiros	78
7.14. Selección dun consello	79
7.15. Cadro de edición de mensaxes con plurais	80
7.16. Partes do panel preferencias	81
7.17. Panel de perfil	82
7.18. Diagrama de Clases de Navegación e Busca	83

Capítulo 1

Introdución

Neste capítulo de introdución intentarase explicar os aspectos necesarios para entender en que consiste o proxecto, o que me fixo levalo a cabo e, por último, unha explicación da estrutura da presente memoria.

1.1. Internacionalización e Localización de Software

A internacionalización é o proceso de adaptación dun software para que este poida ser traducido a varios idiomas e ser usado en diferentes rexións sen modificar a súa enxeñería. A localización de software consiste na adaptación de dito software a unha rexión determinada. Isto, aínda que afecta fundamentalmente ó idioma, tamén ten outros elementos como as divisas, a forma de formatar as datas ou a utilización de determinados símbolos que nunhas áreas teñen un significado e noutras outro distinto.

Trátase dun aspecto moi importante do mundo do software pois se ben unha persoa que saiba inglés (o idioma orixe da maior parte dos programas) a internacionalización dos programas é un problema de comodidade, para o que non entenda a lingua de Shakespeare; trátase dun problema de usabilidade. Unha persoa que non sexa nativa dixital e que non entenda inglés terá serios problemas para entender calquera software moderno non localizado.

1.2. O Proxecto GNOME

GNOME é ambiente de escritorio, unha infraestrutura de desenvolvemento e unha comunidade de software libre.

Como ambiente de escritorio foi creado polos mexicanos Miguel de Icaza e Federico

Mena en 1997 como alternativa a KDE compatible coa licencias GPL¹. Trátase de crear un solución software para todo o mundo poñendo interese en aspectos como a accesibilidade, a internacionalización ou a usabilidade.

Como infraestrutura de desenvolvemento GNOME prové unha gran cantidade de aplicativos e bibliotecas para crear os programas tanto para a plataforma GNOME como para outras plataformas.



Figura 1.1: GUADEC A Coruña 2012

Como comunidade reúne a gran cantidade de persoas, tanto voluntarias como profesionais, que axudaron e axudan a que o proxecto siga adiante. Dentro desta comunidade poden encontrarse persoas de moi diversa procedencia e oficio sendo a maioría programadores, aínda que tamén existen persoas dedicadas a marketing, á administración, ó deseño ou á administración de sistemas, entre outras.

GNOME ademais pon especial interese en intentar sumar xente ao proxecto. Participa sempre no programa de iniciación ao desenvolvemento de software libre Summer of Code promovido por Google e é promotor da iniciativa Outreachy² que pretende acercar ás mulleres á contribución en software libre.

En Europa a comunidade de GNOME réunese na GUADEC, que é o acrónimo de **GNOME Users And Developers Conference**. Un evento que se fai anualmente e que no ano 2012 tivo lugar na Facultade de Informática da Coruña (Figura 1.1).

¹En aquel momento KDE empregaba unha licenza QPL que aínda que era libre non era compatible GPL.

²Inicialmente denominábase GNOME Outreach Program for Women

1.2.1. Localización e Internacionalización no Proxecto GNOME

Como xa dixemos un dos aspectos máis importante para o proxecto GNOME é o acercamento ás persoas e para lograr isto ponse pleno interese na internacionalización e na localización do ambiente de escritorio e das aplicacións de GNOME. GNOME está traducido a máis de 50 idiomas moitos dos cales teñen máis do 90 % das cadeas traducidas.

O proxecto GNOME emprega o sistema GNU Gettext para internacionalizar e localizar os seus programas e conta con unha plataforma web de nome Damned Lies para xestionar a localización de todo o proxecto. Este programa amosa as estatísticas do estado actual das traducións e axuda a xestionar o ciclo de traballo dos tradutores. Permite asignar módulos a tradutores, que os tradutores suban os ficheiros traducidos e que se faga unha revisión do traballo do tradutor.

Outros proxectos de software como Facebook, Twitter ou, dentro do software libre, Ubuntu e Fedora contan con plataformas online dende as que se pode facer directamente a tradución. GNOME opta por facer a tradución *offline* para que esta sexa dunha maior calidade.

Para que os tradutores poidan traballar de forma cómoda é necesario a existencia de ferramentas que lle faciliten o traballo. Estas ferramentas coñécense como CAT, que é o acrónimo de Computer Assisted Translation. Neste traballo preténdese realizar unha destas ferramentas centrada no proxecto GNOME.

1.3. Google Summer of Code

O Summer of Code (xeralmente abreviado SoC ou GSoC) é un programa da empresa estadounidense Google para o fomento do desenvolvemento de software libre entre estudantes de todo o mundo. Realízase unha vez ao ano durante o verán e os participantes deberán ser estudantes, maiores de 18 anos e non ter durante ese verán ningún traballo remunerado.

Existen diversas organizacións de software libre que se presentan para participar no programa aportando unha lista de ideas sobre a que os futuribles participantes deberán presentar un proxecto do que queren traballar durante o verán. Se os estudantes son elixidos, a organización asignará a cada estudante un ou máis mentores que deberán guiar ao participante a levar a cabo as tarefas asignadas. Existen avaliacións tanto para os estudantes como para os mentores. Se o estudante é avaliado con éxito recibirá 5500 dólares polo traballo realizado.

Trátase dun programa de moito éxito en todo o mundo e sobre todo en países como India, onde carecen de iniciativas locais similares. Na edición do verán do ano 2014, participaron

190 organizacións entre as cales se atopan principais axentes no mundo do software libre como GNOME, KDE, GNU ou Firefox e 1307 estudantes de todo o mundo.

1.4. Motivación

Durante a GUADEC Hispana³ 2012 que tivo lugar en A Coruña dentro da GUADEC do mesmo ano, asistín a unha charla impartida por Daniel Mustieles, o coordinador do equipo de tradutores ao castelán do proxecto GNOME. Nesta charla Daniel, resaltaba a necesidade de que o programa actual de asistencia á tradución fose mellorado.

GTranslator está escrito en C e emprega unha biblioteca chamada GObject para facer orientación a obxectos dentro de C. Isto fai que achegarse ó proxecto sexa complicado para un novato polo que falando con outros desenvolvedores da plataforma decidiuse que era mellor reescribir o programa noutra linguaxe e ao mesmo tempo arranxar algúns erros que tiña anteriormente o programa.

Durante o ano seguinte presentei un proxecto para o Google Summer of Code para escribir un novo programa CAT para a plataforma GNOME. Este programa sería unha re-escritura e re-deseño de GTranslator nunha linguaxe de programación moito máis amigable, Vala.

1.5. Estrutura da memoria

A memoria componse de oito capítulos nos que se expón os pasos que se deron para crear o programa GNOME CAT.

Capítulo 1. Introducción. Este capítulo. Nel intentamos explicar de forma resumida en que consiste este proxecto, as motivacións para levalo a cabo e a estrutura da memoria.

Capítulo 2. Estado do Arte. Explicaranse as tecnoloxías empregadas no ámbito da internacionalización e localización de software en xeral e software libre en particular. Farase ademais unha análise de diversas ferramentas do mercado.

Capítulo 3. Fundamentos Tecnolóxicos. Enumeraranse as ferramentas e bibliotecas empregadas durante a elaboración deste proxecto.

³A GUADEC Hispana é a versión para hispanofalantes da GUADEC

Capítulo 4. Metodoloxía. Detallarase o conxunto de prácticas e métodos seguidos durante a realización do programa e que están inspiradas fundamentalmente na metodoloxía eXtreme Programming.

Capítulo 5. Planificación e Seguimento. Explicarase a planificación e o seguimento de cada unha das iteracións que fixemos para elaborar o programa.

Capítulo 6. Análise de Requisitos. Explicarase como se levou a cabo o análise de requisitos para esta aplicación e detallarase cada un dos mesmos.

Capítulo 7. Deseño e implementación. Neste capítulo explicaremos detalles concretos do deseño e a implementación de certas partes do programa.

Capítulo 8. Conclusións e Traballo Futuro. Relataranse as conclusións despois da elaboración do presente proxecto e explicaranse posibles liñas de traballo futuro.

Capítulo 2

Estado do Arte

Neste capítulo explicarase como funciona a internacionalización e localización con GNU Gettext. Ademais analizaremos as diferentes alternativas que existen no mercado como ferramentas de asistencia á tradución e ás características que cada unha incorpora. Ó final faremos un resumo das características que empregan os programas CAT¹.

2.1. Internacionalización e localización con GNU Gettext

Gettext é un sistema para a internacionalización e localización amplamente usado en entornos UNIX. Conta con varias implementacións, sendo a primeira de Sun Microsystems no ano 1990. A implementación máis usada é a que GNU liberou no ano 1995. Pese ser unha solución antiga, é a día de hoxe a mellor que se pode atopar no mercado.

Para internacionalizar un programa con GNU Gettext non empregaremos as cadeas de texto directamente como podería ser no Fragmento de Código 2.1.

Fragmento de Código 2.1: helloworld.c (Sen Internacionalizar)

```
#include <stdio.h>

int
main ()
{
    printf ("Hello World!");
}
```

En lugar diso chamaremos a unha función especial que proporciona Gettext de nome `gettext()` pero que é máis empregada a través do seu alias `_()`. Ademais configuraremos

¹Computer Assisted Translation

o programa para que colla a tradución do idioma que queiramos. Desta forma o programa anterior quedaría como se pode ver no Fragmento de Código 2.2.

Fragmento de Código 2.2: helloworld.c

```
#include <stdio.h>
#include <locale.h>
#include <libintl.h>

#define _(str) gettext(str)

#define

int
main ()
{
    setlocale (LC_ALL, "");
    bindtextdomain ("helloworld", "/usr/local/share/locale");
    textdomain ("helloworld");

    printf (_("Hello World!"));
}
```

A función `gettext()` é a encargada de substituír a cadea orixinal pola tradución. Non obstante, vemos que debemos configurar algunhas cousas antes de poder chamar á función.

En primeiro lugar debemos establecer a linguaxe que queremos empregar no programa. Para iso usamos a función `setlocale()`. O primeiro argumento da función determina que parte do locale actual queremos modificar. Entre outras podemos atopar:

- **LC_ALL.** Queremos cambiar todo.
- **LC_ADDRESS.** Queremos cambiar a forma de formatar os enderezos.
- **LC_MESSAGES.** Os mensaxes do programa.
- **LC_NUMERIC.** O formatado das cantidades non monetarias
- **LC_TIME.** O formatado de datas e horas.

Por último especificamos o código de idioma ou no caso de empregar a cadea baleira empregamos os valores das variables de entorno.

Os códigos de idioma empregan a normativa ISO 639 polo que son da forma

language[_territory][.codeset][@modifier]

Por exemplo o código do galego empregando codificación UTF-8 é `gl_ES.UTF-8`.

Ademais debemos indicarlle ó programa onde ten que atopar as traducións. Para iso empregamos a función `bindtextdomain()` que liga un nome de dominio a unha ruta dentro do sistema e a función `textdomain()` que lle indica o programa cal é o nome de dominio que debe empregar. Un dominio é un conxunto de cadeas que se empregan nunha parte determinada dun programa. Cada dominio debe ter un nome de dominio único dentro dun programa.

Con estes parámetros Gettext xa é capaz de atopar as traducións que no caso do programa anterior atoparíanse en `/usr/local/share/locale/gl/LC_MESSAGES/helloworld.mo`.

Unha vez que internacionalizamos o noso programa debemos traducir as cadeas. Pero para traducir as cadeas debemos extraelas antes do código fonte. Para iso empregaremos a utilidade *xgettext*. Empregando as opcións adecuadas obtemos no ficheiro que se amosa no Fragmento de Código 2.3.

Fragmento de Código 2.3: helloworld.pot

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2014-11-12 19:24+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: helloworld.c:14
#, c-format
msgid "Hello World!"
msgstr ""
```

Os ficheiros Gettext coa extensión *POT* trátanse de plantillas xenéricas para todos os idiomas. Para obter o arquivo específico para o noso idioma debemos empregar a ferramenta *msginit* coa que obteremos un ficheiro como o que se pode ver no Fragmento de

Código 2.4.

Fragmento de Código 2.4: helloworld.po (Sen Traducir)

```
# Galician translations for HELLOWORLD package.
# Copyright (C) 2014 THE HELLOWORLD COPYRIGHT HOLDER
# This file is distributed under the same license as the ch package.
# Marcos Chavarría Teijeiro <chavarria1991@gmail.com>, 2014.
#
msgid ""
msgstr ""
"Project-Id-Version: ch 01\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2014-11-12 19:24+0100\n"
"PO-Revision-Date: 2014-11-12 19:54+0100\n"
"Last-Translator: Marcos Chavarría Teijeiro <chavarria1991@gmail.com>\n"
"Language-Team: Galician\n"
"Language: gl_ES\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=ISO-8859-1\n"
"Content-Transfer-Encoding: 8bit\n"

#: helloworld.c:14
#, c-format
msgid "Hello World!"
msgstr ""
```

Desta forma obtemos un ficheiro Gettext PO que é o ficheiro que temos que editar. Traducindo o arquivo obtemos algo como o Fragmento de Código 2.5.

Fragmento de Código 2.5: helloworld.po (Traducido)

```
# Galician translations for HELLOWORLD package.
# Copyright (C) 2014 THE HELLOWORLD COPYRIGHT HOLDER
# This file is distributed under the same license as the HELLOWORLD package
.
# Marcos Chavarría Teijeiro <chavarria1991@gmail.com>, 2014.
#
msgid ""
msgstr ""
"Project-Id-Version: HELLOWORLD 1.0\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2014-11-12 19:24+0100\n"
"PO-Revision-Date: 2014-11-12 19:54+0100\n"
"Last-Translator: Marcos Chavarría Teijeiro <chavarria1991@gmail.com>\n"
"Language-Team: Galician\n"
"Language: gl_ES\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=ISO-8859-1\n"
```

```
"Content-Transfer-Encoding: 8bit\n"

#: helloworld.c:14
#, c-format
msgid "Hello World!"
msgstr "Ola Mundo!"
```

Antes de poder empregar o ficheiro no noso programa temos que compilalo. Para iso empregamos a utilidade `msgfmt` coa que obtemos o ficheiro `helloworld.mo`. Se movemos o ficheiro o directorio adecuado (o que especificamos en `textdomain()`) o noso programa xa estará localizado.

2.1.1. Ficheiros Gettext PO

Como xa dixemos antes, os ficheiros PO son os ficheiros que temos que editar para localizar o noso programa. Primeiro dicir que se trata ficheiros de texto plano e que polo tanto podemos editar con calquera editor de ficheiros de texto plano. Non obstante, o ideal é empregar algunha ferramenta que nos facilite a tarefa como pode ser unha ferramenta CAT.

As súas principais características son:

Soporte de plurais Algo que pode parecer trivial como o soporte de plurais deixa de selo cando consideramos que non todos as linguaxes do mundo empregan dous plurais. A lingua eslovaca, por exemplo, conta con tres formas de plural de forma que o plural faise diferente para 1, 3 e 5 elementos.

Gettext representa a forma de plural de cada linguaxe con unha cadea da seguinte forma:

$$nplurals = n; plural = exp;$$

Onde n representa o número de plurais da linguaxe e exp a expresión para calcular cando debemos empregar cada forma. Por exemplo a forma plural do galego representase como $nplurals = 2; plural = (n! = 1);$. Isto é que temos 2 plurais e que so se emprega a forma singular cando o número de elementos é igual a 1.

No código fonte para que GetText escolla a tradución adecuada temos que empregar a función `ngettext`. Esta función recibe como parámetros a cadea orixinal en singular, a cadea orixinal en plural e o número de elementos. No Fragmento de Código 2.6 temos un exemplo:

Fragmento de Código 2.6: Plurais en GetText (Código Fonte).

```
[...]
printf (gettext ("We have %d car.", "We have %d cars.", n), n);
[...]
```

A cadea do ficheiro PO (Fragmento de Código 2.7) correspondente ó código anterior pódese ver no seguinte fragmento de código. Vemos como temos unha entrada `msgstr` por cada plural. Desta forma o plural número 0 corresponde ós singular e o plural número 1 correspóndese coa primeira forma do plural.

Fragmento de Código 2.7: Plurais en GetText (Ficheiro PO).

```
#: helloworld.c:19
#, c-format
msgid "We have %d car."
msgid_plural "We have %d cars."
msgstr[0] "Temos %d coche."
msgstr[1] "Temos %d coches."
```

Marcado de traducións difusas Permítese marcar certas traducións como difusas de forma que o tradutor indica que non está seguro de que dita tradución sexa correcta. Se marcásemos a tradución *"Hello World!"* como difusa o ficheiro PO tería o aspecto que se pode ver no Fragmento de Código 2.8.

Fragmento de Código 2.8: Ficheiro POT con comentario.

```
[...]
#: helloworld.c:15
#, c-format
#, fuzzy
msgid "Hello World!"
msgstr ""
[...]
```

Formato das traducións Os ficheiros PO permiten indicar se as cadeas a traducir teñen un formato determinado. Por exemplo, a cadea do exemplo no Fragmento de Código 2.5 pódese ver que ten o flag `c-format` debido a que é parte dunha sentenzia `printf` e podería levar indicadores de formato da forma `%s`.

Cabeceira con metadatos Existe unha cadea especial nos documentos Gettext PO. Trátase da cadea baleira que serve para almacenar metadatos do ficheiro. No fragmento de código 2.5 podemos ver estes metadatos. Algúns dos metadatos existentes son:

- **Project-Id-Version.** Nome único para o proxecto deste arquivo de tradución.
- **Report-Msgid-Bugs-To.** Ligazón onde reportar erros nas cadeas orixinais ou para pedir contexto para facer a tradución.
- **POT-Creation-Date.** Data de creación do ficheiro POT.
- **PO-Revision-Date.** Data da última actualización das traducións.
- **Last-Translator.** Nome e enderezo de correo electrónico do último tradutor.
- **Language-Team.** Enderezo de correo electrónico do equipo de tradutores.
- **Language.** Linguaxe do ficheiro expresada coa codificación ISO 639.
- **Content-Type.** Tipo MIME do ficheiro, que será sempre `text/plain` e codificación dos caracteres.
- **Plural-Forms.** Expresión da forma plural empregada.

Ademais destes campos, nos comentarios, gárdanse os nomes de todas as persoas que contribuíron a esta tradución.

Gardado dos orixes das cadeas Gettext almacena para cada cadea en que lugares do código aparece esta. O cal pode ser moi interesante para implantar a previsualización das traducións. Por exemplo no Fragmento de Código 2.5 vemos como a cadea *"Hello World!"* pode atoparse na liña 14 do ficheiro `helloworld.c`.

Comentarios dos programadores É unha función moi importante xa que en moitas ocasións nas linguaxes a mesma palabra empregase como verbo ou como nome polo que en ocasións é importante incorporar un contexto para esa tradución. Para facer isto é necesario simplemente poñer un comentario no programa antes de empregar a cadea. Por exemplo no Fragmento de Código 2.9, estamos engadindo un comentario a cadea *"Hello World!"*.

Fragmento de Código 2.9: Tradución con comentario.

```
[...]
    // Translators: We are just waving the world.
    printf (_("Hello World!"));
[...]
```

O ficheiro POT resultado tería a forma que se pode ver no Fragmento de Código 2.10.

Fragmento de Código 2.10: Ficheiro POT con comentario.

```
[...]
#. Translators: We are just waving the world.
#: helloworld.c:15
#, c-format
msgid "Hello World!"
msgstr ""
[...]
```

Comentarios dos tradutores A biblioteca permite que os tradutores comenten as cadeas. No Fragmento de Código 2.11 podemos ver o aspecto que tería o ficheiro PO se engadimos un comentario á cadea *"Hello World!"*.

Fragmento de Código 2.11: Ficheiro PO con comentario.

```
[...]
# This is a note from translators.
#: helloworld.c:15
#, c-format
msgid "Hello World!"
msgstr ""
[...]
```

2.2. Ferramentas CAT do mercado

Nesta sección analizaremos algunhas das ferramentas de asistencia á tradución existentes. Veremos as características que incorporan estes programas así como estudar a súa interface de usuario.

2.2.1. GTranslator

GTranslator é a aplicación oficial do proxecto GNOME para a asistencia á tradución. Este aplicativo só permite a tradución de arquivos GNU Gettext. As características máis destacables deste programa son a posibilidade de abrir varios ficheiros en diferentes lapelas, soporte de memorias de tradución, perfís para diferentes tradutores, edición dos comentarios dos ficheiros .po e un sistema de plugins que permite estender a ferramenta.

En canto a interface, como podemos ver na Figura 2.1, a parte máis importante do programa é a lista de mensaxes. Abaixo desta lista temos un panel onde se pode editar cada mensaxe e á súa dereita a memoria de tradución. A disposición dos elementos desta

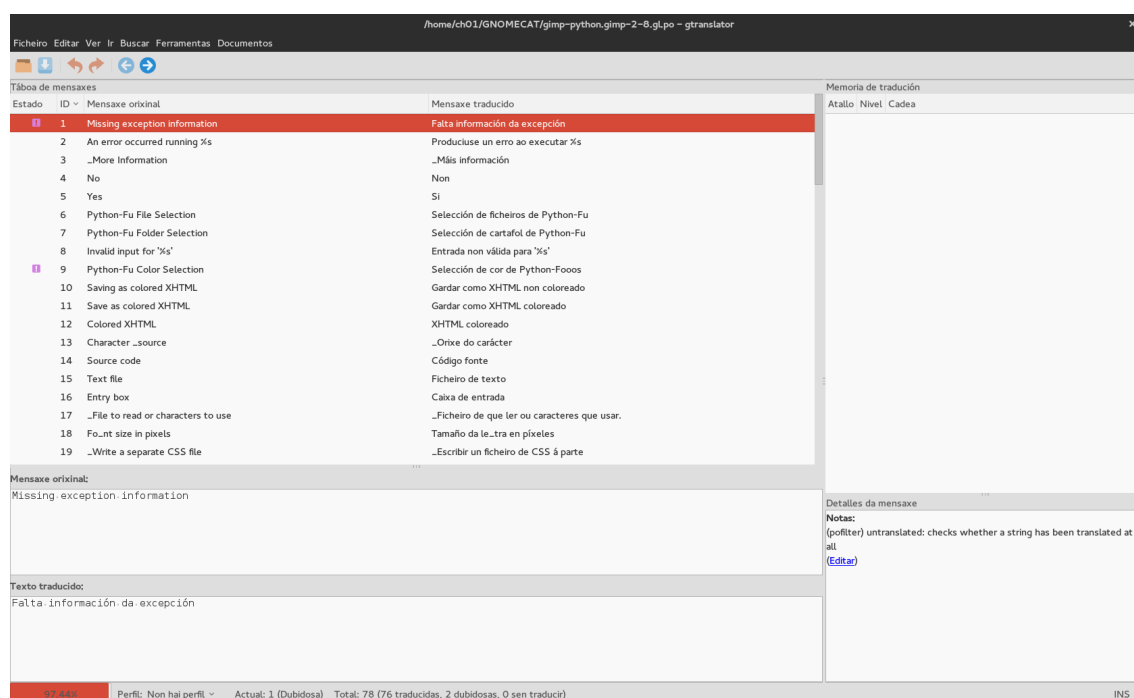


Figura 2.1: Interface de GTranslator

interface é configurable xa que permite mover e ampliar cada un dos módulos. O programa tamén incorpora atallos de teclado que permiten moverse polo documento e seleccionar cada elemento da memoria de tradución.

Este programa pese a ser o aplicativo oficial de GNOME é moi pouco usado. As razóns disto son a ausencia dunha característica chave que o diferencie doutras ferramentas do mercado, a presenza de fallas importantes que afectan a usabilidade e a ausencia dun mantedor que resolva estes problemas.

2.2.2. Lokalize

Lokalize é o programa oficial para o soporte á tradución en KDE. Foi escrita dende cero empregando a tecnoloxía de KDE Platform 4 e baseándose no código de KBabel. As súas características máis destacables son: o soporte para ficheiros GNU Gettext e o formato QT TS, entre outros; a xestión de proxectos incorporando unha vista que permite ver un resumo de cada ficheiro dentro do proxecto; uso de memorias de tradución e de glosarios; comprobación da ortografía e vista previa das traducións a partir de scripts feitos polo usuario.

En canto á interface, o programa permite abrir varios ficheiros cada un na súa lapela. Como se pode ver na Figura 2.2, a vista de tradución está centrada no panel de edición, onde aparecen tanto a cadea orixinal como a cadea para traducir. Na columna da esquerda

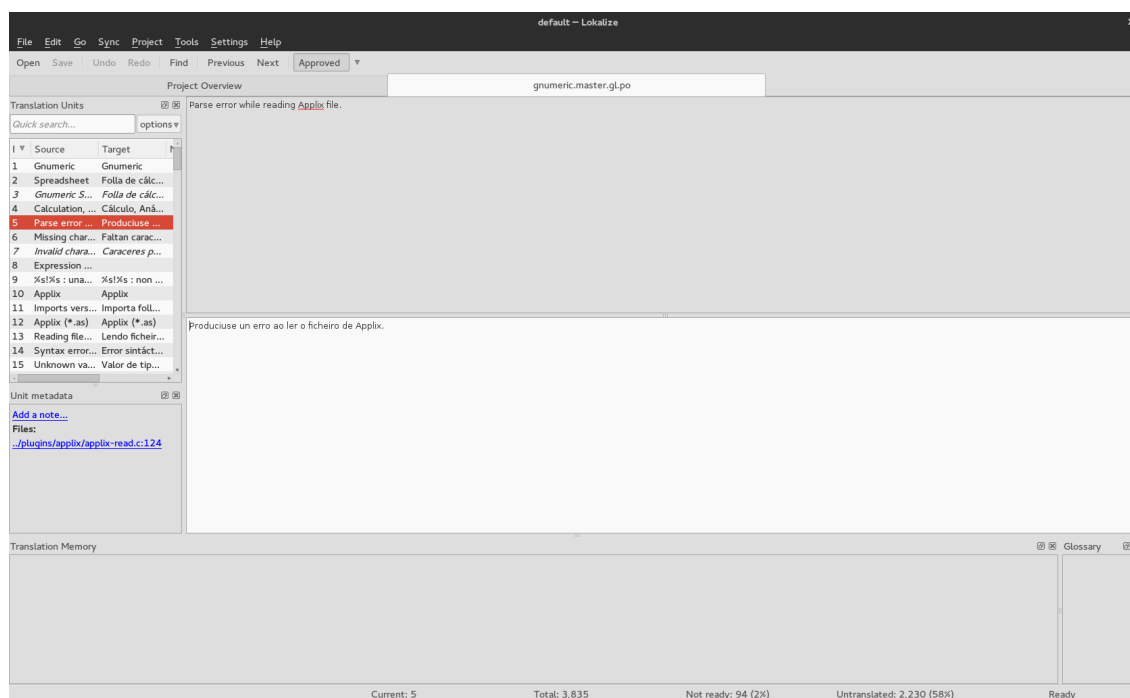


Figura 2.2: Interface de Lokalise

podemos ver a lista de cadeas, onde se amosan os primeiros caracteres da cadea orixinal e da tradución e o estado desta tradución. Ademais, tamén se pode ver o contexto da tradución, engadir un novo comentario e ver en que ficheiros estaba dita tradución. Por último, tamén podemos ver na parte inferior a memoria de tradución e o glosario.

2.2.3. Virtaal

Virtaal é unha ferramenta CAT[Hou] creada por Translate House². As características máis destacables son a incorporación de suxestión a tradución, comprobación da calidade das tradución e, sobre todo, a capacidade de abrir unha gran variedade de formatos a través da biblioteca Translate Toolkit.

Como se pode ver na Figura 2.3, a interface é minimalista e moi centrada na tradución. A diferenza dos casos anteriores, trátase dunha interface fixa e que integra a lista das mensaxes coa edición da propia mensaxe. Tamén se indican posibles fallos que poida ter a tradución, como falta de puntos ó final, ausencia de marcadores de formato, etc.

²Compañía que surxiu a partir dunha comunidade de tradutores de Sudáfrica e que está especializada na creación de ferramentas e bibliotecas para axudar a tradución.



Figura 2.3: Interface de Virtaal

2.2.4. OmegaT

Ferramenta CAT[Pro] lanzada no ano 2001 e pensada fundamentalmente para tradutores profesionais. Ten soporte para o uso de memoria de tradución, glosario, tradución directa entre outras cousas. Destaca a gran cantidade de formatos que pode empregar.

A interface de OmegaT dista bastante do resto de programas. Como amosa a Figura 2.4, non existe o concepto de lista de mensaxes para traducir. O documento amósase como unha sucesión de cadea e se facemos clic encima dunha, permitíranos traducila. Trátase dun programa moi usado para a tradución tanto profesional como amateur.

2.2.5. Google Translation Toolkit

É a ferramenta CAT desenvolvida por Google e lanzada no ano 2008. A diferenza dos aplicativos analizados anteriormente, esta trátase unha solución puramente web. Entre as súas principais características, encóntrase a posibilidade de facer tradución automática empregando Google Translator, o uso de memorias de tradución compartidas, glosarios, soporte de etiquetas HTML, entre outros, e atallos de teclado. Ten soporte para varios formatos, como ficheiros PO, documentos de Microsoft Word, de LibreOffice ou mesmo

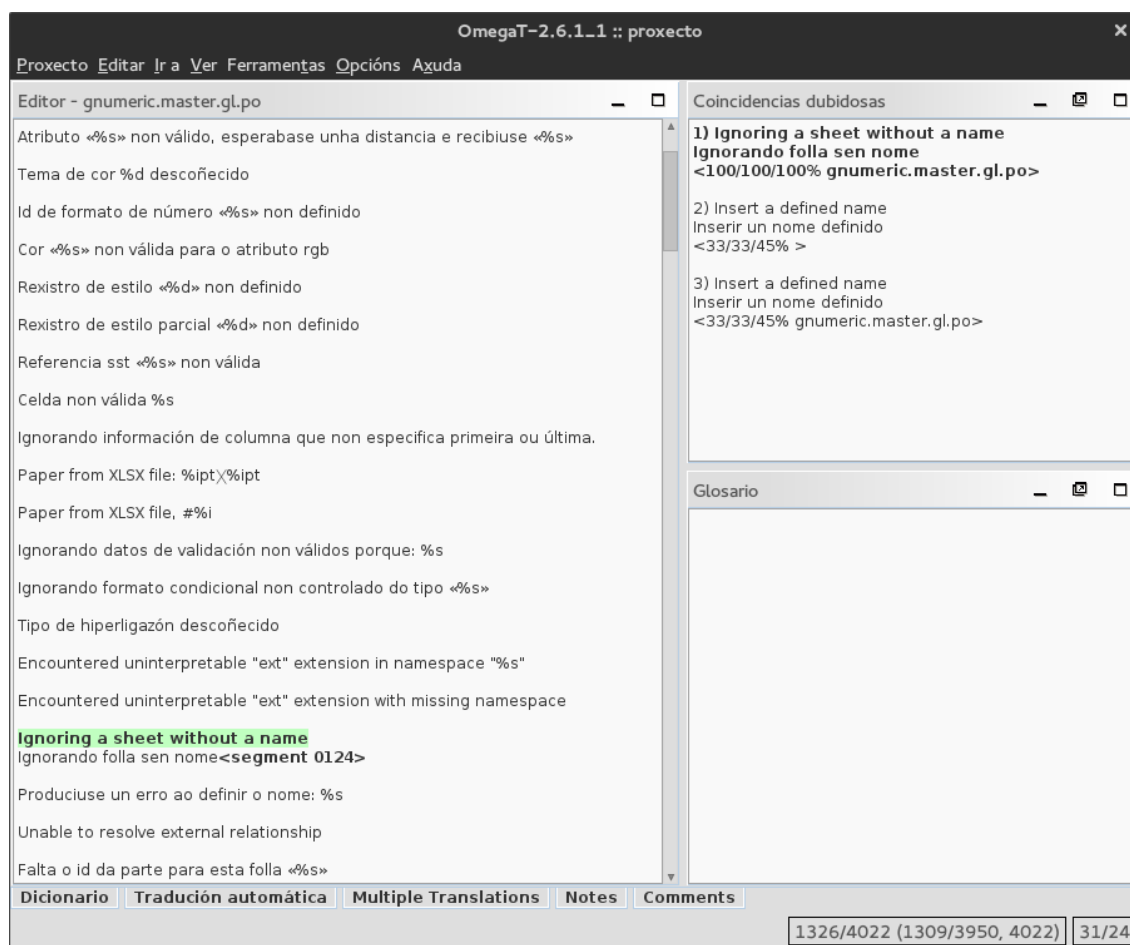


Figura 2.4: Interface de OmegaT

artigos da Wikipedia.

Como se pode ver na Figura 2.5, na interface mestúrase a lista de cadeas co seu cadro de edición. Ademais, emprega unha interface semellante á do resto de ferramentas ofimáticas de Google, temos botóns para autocompletar etiquetas e para avanzar a seguinte tradución. Aínda que foi pensado para a tradución colaborativa de documentos de ONGs e artigos da Wikipedia, na actualidade emprégase maioritariamente para a tradución de proxectos comerciais.

2.2.6. Transifex

Trátase dunha plataforma que xurdiu a partir dun proxecto do Google Summer of Code do ano 2007 que pretendía crear unha plataforma online máis amigable que o Damned Lies de GNOME que naquel momento tamén empregaba Fedora unha distribución de GNU/Linux. Trátase dunha solución de pago con plans que van dende os 19 ós 300 dólares.

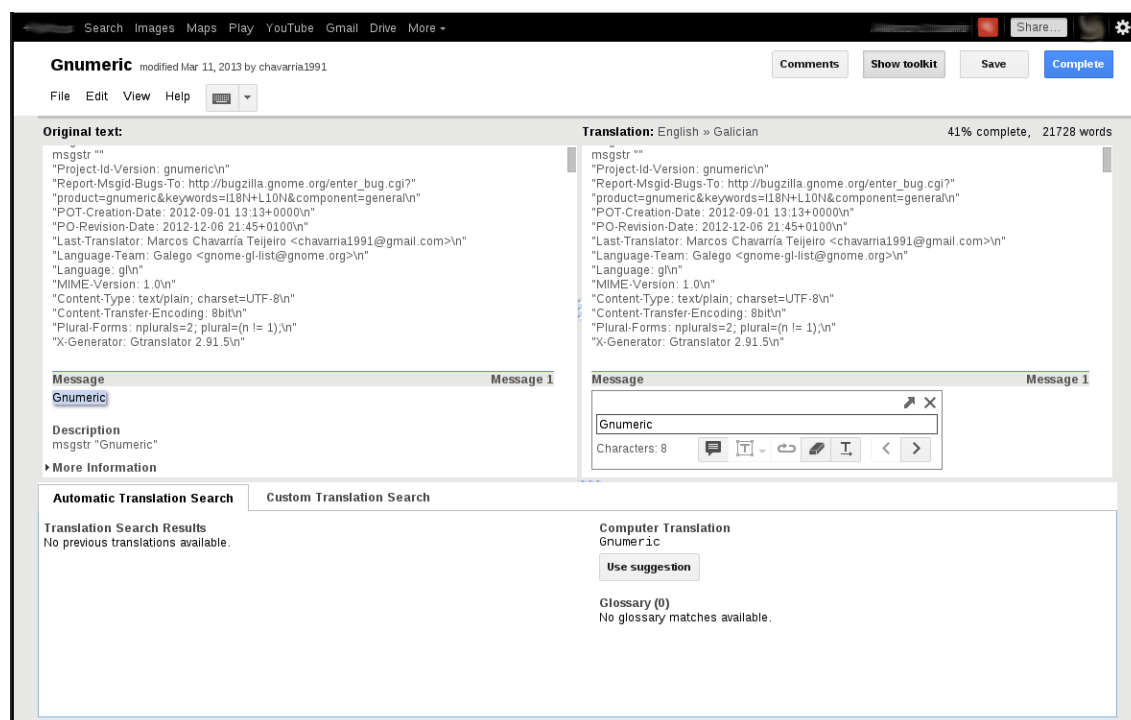


Figura 2.5: Interface de Google Translation Toolkit

Non obstante, os proxectos de código aberto poden usar o servizo de forma gratuíta e dispón dun período de mostra 30 días. As súas principais características son a posibilidade de descargar o documento e volvelo subir para poder traducilo con outra ferramenta CAT, editor online, memoria de tradución e unha API que permite integralo con outros servizos. Ademais, tamén soporta unha gran variedade de formatos, entre os que se atopan os ficheiros PO, DTD de Mozilla ou XML, entre outros.

A interface de Transifex, como se pode ver na Figura 2.6, separa a lista de cadeas do cadro de edición. No cadro de edición temos a posibilidade de consultar a memoria de tradución ou o glosario. O programa tamén incorpora resaltado de sintaxe.

2.2.7. Outras ferramentas

Existen moitas máis ferramentas CAT no mercado. De feito, segundo unha enquisa [Lon06] elaborada polo Imperial College London a cerca de 900 tradutores profesionais de 54 países diferentes, OmegaT é o único programa que aparece na enquisa (cun 7 % de usuarios) de todos os programas anteriormente citados. As ferramentas máis usadas son ferramentas para Microsoft Windows e ferramentas con licenzas privativas e usualmente moi caras. Algunhas destas ferramentas son TRADOS, Wordfast, DejaVu SDLX ou STAR Transit. Na Figura 2.7 podemos ver unha gráfica coas ferramentas máis empregadas segundo este estudo.

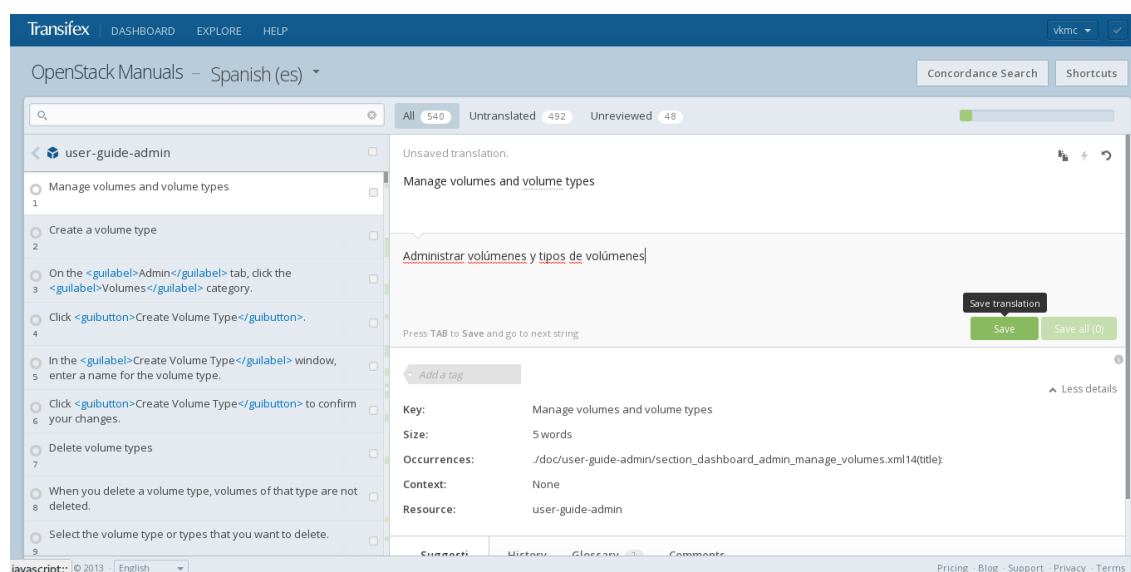


Figura 2.6: Interface de Transifex

Hai que ter tamén en conta que se trata dun estudo bastante antigo, polo que algunhas das ferramentas analizadas aínda non existían. Por exemplo, a enquisa cita a ferramenta KBabel, que é a ferramenta de KDE na que está baseada Lokalizе, pero con menos dun 2 % de usuarios.

2.3. Características xenéricas das ferramentas CAT

Algunhas das características que aparecen de forma recorrente en todas as ferramentas analizadas son as seguintes:

2.3.1. Memoria de Tradución

Unha memoria de tradución é unha base de datos composta de textos orixinais acompañados das súas traducións. Estes textos almacénanse en segmentos onde a separación entre segmentos vén dada por signos de puntuación ou o cambio de parágrafo, sendo esta última forma a máis frecuente.

A principal función dunha memoria de tradución é a extracción de coincidencias totais ou parciais. Os programas que teñen esta característica buscan na base de datos un segmento que coincida de forma exacta ou parcial coa cadea que se está a traducir e móstrase este segmento como suxestión. Xunto coa suxestión tamén se amosa o grao de proximidade entre a cadea a traducir e para cadea da memoria de tradución.

Existe un formato estándar de compartición de memorias de tradución de nome Trans-

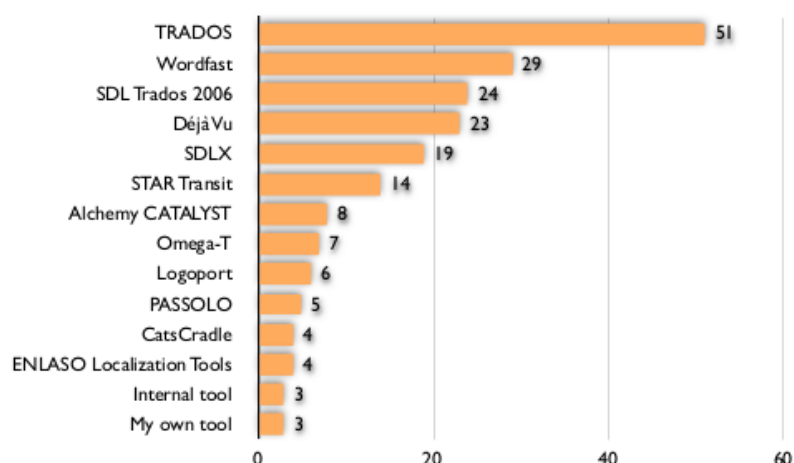


Figura 2.7: Ferramentas CAT máis empregadas segundo [Lon06].

lation Memory eXchange (TMX) e plataformas online que almacenan gran cantidade de cadeas e, polo tanto, hai máis posibilidade de obter unha mellor coincidencia. O software Amagama creado por Translate House, os creadores de Virtaal, é un exemplo de memoria de tradución online.

2.3.2. Glosario

Un glosario é unha base de datos de termos xunto cunha ou varias traducións aceptadas. Diferénciase da memoria de tradución en que só se proporciona a tradución a termos e non a cadeas completas. De igual forma que no caso das memorias de tradución, existe un formato estándar de para a compartición de glosarios de nome TermBase eXchange (TBX) e plataformas online para almacenar os glosarios.

2.3.3. Previsualización

As funcións de previsualización permítenlle ó tradutor ver como vai quedar a cadea traducida no programa final.

Os programadores e deseñadores fan as interfaces tendo en conta a lingua orixinal e non ningunha das traducións, polo que se unha tradución é moito máis longa cá orixinal pode verse mal no programa final.

Para conseguir esta característica pódense empregar varias técnicas:

- **Programa orixinal.** Esta técnica que se pode empregar en calquera cadea consiste en compilar o ficheiro PO e executar o programa final con ese ficheiro PO. Desta poderemos ver como queda a nosa tradución para o usuario final. A desvantaxe deste

método consiste en que teremos que saber en que parte do programa se emprega a cadea que queremos previsualizar. Este método é empregado por Lokalize que permite a definición de scripts para a previsualización de cadeas.

- **Renderizado de interfaces de usuario en XML.** Nas bibliotecas de interfaces modernas existe a posibilidade de definir interfaces en ficheiros XML e despois renderizalas. Neste caso as cadeas para traducir van nestes arquivos e sería posible renderizar esta interface coa tradución que estamos a realizar. Este método, aínda que si que amosaría a pantalla onde aparece a cadea actual, só é válido para as cadeas que proveñen destes ficheiros XML e non as definidas da forma tradicional. Un exemplo deste método pódese ver na rede no aplicativo Deckard³ que permite ver as traducións de aplicativos de GNOME.

2.3.4. Tradución Directa

A tradución de cadeas de forma automática empregando algoritmos deseñados para tal proceso e que empregan grandes bases de datos aloxadas, xeralmente en internet. Tanto Google como Microsoft teñen os seus produtos corporativos que fan traducións e existen alternativas libres como OpenTrad ou Apertium. Estas traducións, aínda que válidas, acostuman ser de baixa calidade, polo que necesitan unha revisión.

³deckard.malizor.org

Capítulo 3

Fundamentos Tecnolóxicos

Neste capítulo detallaremos os compoñentes tecnolóxicos que empregamos para levar a cabo este proxecto. Primeiro falaremos das ferramentas empregadas e despois das diferentes bibliotecas que usamos no programa.

3.1. Ferramentas empregadas

Vala É unha linguaxe de programación orientada a obxectos creada por GNOME para acercar a linguaxe de programación C as características das linguaxes de programación modernas. Emprega C xunto coa biblioteca GObject como linguaxe intermedia. Esta linguaxe cunha sintaxe moi parecida a C# incorpora características como o uso e funcións anónimas, bucles foreach, xestión automática de memoria ou o manexo de excepcións.

Autotools Tamén coñecidas como *GNU build system* [MTDL⁺15] son un conxunto de ferramentas que permiten permitir a configuración e compilación de programas portables a varias plataformas UNIX.

Git É un sistema de control de versións distribuído escrito por Linus Torbards e usado, entre outros moitos proxectos, para xestionar o desenvolvemento de Linux (o kernel). Trátase dunha das solucións de control de versións máis empregada na actualidade pois é moito máis rápida que as outras existentes e, ao ser distribuída, permite traballar sen conexión a internet.

GitHub Unha plataforma de almacenamento de proxectos empregando o sistema de control de versións Git. Permite subir proxectos Open Source de forma gratuíta ós seus

servidores. Conta ademais con ferramentas para a creación de Wikis relacionadas cos proxectos e incorpora un xestor de fallos.

Listas de Correo As listas de correo son un medio de comunicación asíncrona moi usadas no mundo do desenvolvemento software. Os usuarios escriben un correo electrónico a un enderezo especial que reenvia dito correo a todos os usuarios subscritos ás listas. Ademais, os correos enviados a lista son almacenados nun historial para a súa posterior consulta.

IRC (Internet Relay Chat) É un protocolo de comunicación entre usuarios creado no ano 1988. É amplamente usado no mundo do desenvolvemento software como unha ferramenta rápida para consultar dúbidas entre desenvolvedores. Existen múltiples clientes dispoñibles para case calquera plataforma. En concreto empregouse o programa XChat.

JHBuild Unha serie de scripts [[GNOD](#)] que permiten a descarga automática do código fonte dun proxecto e das súas dependencias e a instalación destes proxectos dentro dun entorno separado do resto do sistema. Desta forma os desenvolvedores poden traballar coas últimas versións das bibliotecas que tenden a ser inestables sen ter que usalas en todo o seu sistema.

Glade Unha ferramenta para a creación de interfaces de usuario con GTK+ as interfaces creadas expórtanse en ficheiros con formato XML.

Dia É un programa para a creación de diagramas de todo tipo. Ten un paquete para facer diagramas UML.

JSON É un formato para o intercambio de datos. Acrónimo de JavaScript Object Notation, trátase dun subconxunto da notación para definir obxectos en JavaScript. Para poder analizar este formato no noso programa empregaremos a biblioteca JSON-GLib.

GDB GNU Project debugger. É o depurador estándar para o proxecto GNU. Permite a execución paso a paso, monitorizar e alterar o valor das variables entre outras moitas características.

L^AT_EX É un sistema de composición de textos. Creado por Leslie Lamport como un gran conxunto de macros de T_EX para facilitar o seu uso, produce documentos de gran calidade

con pouco esforzo. Permite incluír dentro dos textos formulas matemáticas, fragmentos de códigos ou figuras, entre outros.

3.2. Bibliotecas empregadas

GLib É unha biblioteca de propósito xeral creada por GNOME [GNOa] a partir de 5 librerías, GObject, GLib, GModule, GThread e GIO.

GObject incorpora características da orientación a obxectos a linguaxe de programación C, como pode ser a creación de clases, herdanza ou as propiedades entre outras. *GLib* constitúe un conxunto de tipos básicos empregados no resto de bibliotecas. *GModule* permite a carga de módulos ou extensións de forma dinámica. Por último, *GThread* é unha biblioteca de xestión de fíos de execución e *GIO* é unha biblioteca pensada para facilitar a entrada e saída nos programas.

Esta biblioteca constitúe unha dependencia básica en todos os aplicativos feitos co *stack* de GNOME e, de feito, é unha dependencia de case todas as demais bibliotecas que empregamos.

GTK+ Biblioteca multiplataforma para a creación de interfaces gráficas de usuario desenvolvida por GNOME. Incorpora unha serie de widgets para a creación de programas tanto grandes como pequenos. Está escrita en C, pero existen bindings para moitas linguaxes.

As interfaces gráficas pódense crear de forma programática ou cargando un ficheiro XML. Aínda que pode parecer moito máis traballoso a creación dun ficheiro XML con ese propósito, a existencia de ferramentas como Glade¹.

LibGee Trátase dunha biblioteca que fornece a GObject de estruturas de datos comunmente usadas como listas, táboas hash ou conxuntos, entre outros.

LibPeas LibPeas é un motor de plugins para GObject. Escrito orixinalmente para GEdit, permítelle ás aplicacións estenderse a través dun sistema de plugins.

GettextPo Trátase dunha biblioteca que analiza ficheiros GNU Gettext PO. Esta biblioteca está implementada en C e non existen bindings para Vala, polo que, teremos que facelos nós.

¹glade.gnome.org

GTKSourceView Esta biblioteca estende o *widget* de GTK+ GtkTextView, que permite a visualización de textos, para incorporarlle certas características avanzadas como a xestión de cambios permitindo desfacer e refacer ou o resaltado da sintaxe.

JSON-GLib É unha biblioteca para a análise de cadeas de texto en formato JSON.

Capítulo 4

Metodoloxía

Nesta sección descríbese a metodoloxía levada a cabo para a realización deste proxecto. Unha metodoloxía é un conxunto de métodos ou prácticas empregadas para a realización dunha tarefa, neste caso a análise, deseño e implementación dunha nova ferramenta CAT para o proxecto GNOME. Escolleuse unha metodoloxía áxil de ciclo incremental. Moitas das prácticas que se tomaron son collidas da metodoloxía eXtreme Programming.

4.1. eXtreme Programming

Foi creada por Kent Beck no ano 1999[Bec00], e trátase dun dos máis destacados métodos de desenvolvemento áxil. eXtreme Programming (a partir de agora XP) avoga por ciclos de desenvolvemento moi curtos e elimina os roles clásicos de analista, deseñador e programador. Todo equipo participa en todas as partes do desenvolvemento. Desta forma, Beck define uns valores fundamentais da metodoloxía e unhas prácticas que axudan a adoptar estes valores.

4.1.1. Valores de eXtreme Programming

4.1.1.1. Comunicación

É o primeiro valor de XP. Segundo esta metodoloxía, os problemas nos proxectos poden ser traducidos a alguén que non falou con outra persoa sobre algo importante do proxecto. Esta mala comunicación non sucede por casualidade e é debido frecuentemente ás malas prácticas. Para solucionar isto, XP inclúe prácticas nas que é necesaria a comunicación para levalas a cabo. Ademais a figura do *coach* serve para mellorar a comunicación daquelas persoas que non o están facendo ben.

4.1.1.2. Simplicidade

A simplicidade non é unha tarefa sinxela. XP fai unha aposta e invita ao programador a pensar no que o proxecto necesita hoxe e non no que vai necesitar nun futuro. Para manter esta simplicidade ao longo do tempo, é frecuente a súa refactorización. A simplificación do deseño e da implementación axiliza tanto o desenvolvemento como o mantemento. A simplicidade require **comunicación**, pois canto máis comunicación teñamos por parte do cliente máis sinxelo poderemos facer o sistema, e canto máis simple sexa o sistema, menos comunicación será necesaria para explicar o sistema.

4.1.1.3. Retroalimentación

A retroalimentación ou feedback é fundamental nesta metodoloxía. Pode actuar en varias escalas de tempo. Obtemos feedback en cuestión de minutos ou días de parte dos test do sistema, das peticións dos clientes ou do director do proxecto. Tamén obtemos feedback ó longo dos meses cando o usuario pode analizar as características que implementamos. Neste sentido XP aposta por unha posta rápida en produción, de forma que teñamos sistemas en desenvolvemento e en produción de forma paralela. Con isto melloramos o sistema, xa que imos obtendo as opinións dos usuarios das decisións que xa tomamos e os erros cometidos non se volven repetir.

4.1.1.4. Coraxe

O coraxe é unha parte inherente á metodoloxía. É necesario para tirar o traballo de varios días e volver empezar debido a cambios nos requisitos ou aparición de fallos estruturais. É necesario para ser persistente coa resolución dun problema, as cousas que non se dan resolto un día en horas, pódense resolver ó día seguinte en cuestión de minutos. O coraxe non é útil sen os tres primeiros valores. Cunha boa comunicación existe a posibilidade de facer experimentos con máis risco. A simplicidade permítelle ó programador coñecer mellor o código e, polo tanto, ser máis valente á hora de facer cambios. A retroalimentación axuda a que alguén se sinta máis seguro ao facer un cambio.

4.1.1.5. Respecto

Por último, é necesario respecto. É necesario que os integrantes do equipo se preocupen polo resto de membros e polo que están facendo. Ademais o equipo debe preocuparse polo propio proxecto. Para que XP funcione os programadores débense sentir parte do proxecto e ter un feedback positivo ao respecto.

4.1.2. Prácticas recomendadas por eXtreme Programming

4.1.2.1. O Xogo da Planificación

A planificación é un diálogo entre a xente do negocio e o equipo técnico. Mentres a parte de negocio decide a importancia dun problema, a prioridade da implementación dunha característica ou outra, a composición das entregas ou as datas das mesmas, o equipo técnico é capaz de estimar canto tempo leva implementar unha característica, ten a capacidade de explicar as consecuencias de certa decisión, sabe como organizarse para levar a cabo unha tarefa e pode facer unha planificación máis detallada.

4.1.2.2. Entregas Pequenas

As entregas ou *releases* deben ser o máis pequenas posibles e conter os requirimentos máis valiosos. Aínda así, cada release debe ser autocontida e non ter características implementadas a medias solo para facer o ciclo de entregas máis curto.

4.1.2.3. Metáfora

Cada proxecto feito con XP ten unha metáfora. Unha metáfora é un símil sinxelo de como está composto o sistema. É útil para que os membros do equipo teñan unha visión global do que están facendo e que membros non técnicos do equipo poidan entendelo.

Outras metodoloxías chámanlle a isto **arquitectura**. O problema con empregar o termino arquitectura é que unha arquitectura non ten necesariamente un sentido de cohesión.

4.1.2.4. Deseño simple

Todos os deseños deben, executar todos os tests, non ter código duplicado, ter o menor número de clases e métodos e todas as partes son importantes para os programadores. Con esta filosofía XP intenta facer un deseño simple para as necesidades actuais do programa. Desta forma, a implementación será máis rápida e o tempo de aprendizaxe para os outros membros do equipo será máis curto.

4.1.2.5. Testing

Calquera característica incorporada que non inclúa un test, simplemente non existe. Os programadores inclúen test de unidade para as novas funcionalidades e os clientes crean test funcionales de como esperan que o programa funciona. Ambos test forman parte do

código do programa. Non é necesario escribir test para cada método pero si para cada método que se expoña.

4.1.2.6. Refactorización

Cando é necesario implementar unha nova característica no programa, os programadores pregúntanse se existe unha forma simple de implementala e implementana. Despois analizan o código para ver se existe unha forma de facelo de forma máis simple e que siga executando correctamente todos os tests. Isto chamase refactorizar.

É obvio que traballando desta forma emprégase moito máis tempo do necesario para a implementación de cada característica, pero desta forma poderemos engadir a seguinte característica nunha cantidade razoable de tempo.

4.1.2.7. Programación por parellas

Todo o código en produción é escrito por parellas de programadores con diferentes roles. Por un lado un dos programadores pensara de forma específica como implementar un certo método, mentres o outro pensará dun xeito máis global e estratéxico. Estas parellas cambian continuamente.

4.1.2.8. Pertenza Colectiva

En XP o código pertence a todo o equipo e se unha persoa ten oportunidade de engadir algo de valor a algún fragmento de código ten que facelo nalgún momento. Desta forma todo o mundo ten responsabilidade sobre todo o sistema e, aínda que non todo o mundo coñece cada parte de forma igual, todo o mundo coñece algo de cada parte de forma que son capaces de facer modificacións satisfactorias.

Isto contrasta coas prácticas doutras metodoloxías onde o código escrito por unha persoa só pertence a esa persoa e para engadir nova funcionalidade é necesario facer unha petición a dito programador. Esta práctica pode facer máis lento o desenvolvemento e diminúe o factor camiión¹.

4.1.2.9. Integración Continua

Os test son executados con cada cambio e, soamente se todos os test son executados correctamente, sóbense os cambios ao produto final. É importante executar os test a cada

¹**Truck Factor:** O numero de membros dun equipo dentro dun proxecto, que no caso de seren atropellados por un camiión, o proxecto non podería completarse.

cambio xa que así saberemos a que se debe o fallo e quen ten que corrixilo. Se para implementar unha característica os seus desenvolvedores non son capaces de que todos os tests funcionen, probablemente necesiten volver a empezar pois non tiñan os coñecementos necesarios para implemementala.

4.1.2.10. Semana de 40 horas

XP establece unha xornada laboral de 8 horas e 5 días á semana. Para esta metodoloxía é importante que os programadores estean frescos e inspirados cada mañá e con xornadas largas de traballo dita tarefa é imposible. O descanso é algo fundamental para poder ter boas ideas.

4.1.2.11. Cliente no sitio

Un cliente do proxecto, é dicir, unha persoa que realmente vaia usalo cando estea en produción, debe sentarse xunto ó equipo e poder responder preguntas e resolver disputas entre membros do equipo.

4.1.2.12. Estándares de programación

Cando se traballa cambiando de parellas cada pouco tempo e facendo refactorizacións continuas, cómpre que todo o equipo siga uns estándares de programación. Isto é, un mesmo estilo de código e unha mesma forma de facer certas cousas.

4.2. Metodoloxía seguida

Dentro das prácticas e valores suxeridos por eXtreme Programming seguimos un sub-conxunto do mesmo. Moitas das prácticas non se puideron levar a cabo debido a que se tratada dun proxecto dunha persoa.

Pertenza Colectiva O programa que estamos facendo é software libre e polo tanto o seu código está dispoñible a todo aquel que queira melloralo. En concreto, o noso programa está subido á plataforma GitHub. Calquera persoa pode enviar cambios que poden ser integrados no programa ou reportar fallos.

Estándares de Programación Na elaboración do programa intentamos seguir un estilo de código fixo en todos os ficheiros. Desta forma facilitamos aos novos desenvolvedores a lectura do código do noso programa.

Semana de 40 horas Durante a elaboración do programa as horas diarias dedicadas ao programa foron aproximadamente cinco. Pensamos que desta forma o programador tería a mente máis descansada para poder programar.

Deseño Simple Intentamos solucionar os problemas empregando un deseño simple, tendo en algunhas ocasións que refactorizar gran parte do código.

Cliente no sitio Os clientes do programa, é dicir, os tradutores de GNOME, tiveron sempre a posibilidade de instalar o programa e comentar, ben a través do blogue, no xestor de fallos que incorpora GitHub, ou vía email as melloras que lles gustaría incorporar ao programa.

Capítulo 5

Planificación e Seguimento

Neste capítulo detallaremos a planificación de cada unha das iteracións que fixemos para realizar o programa e tamén o seguimento realizado. A elaboración deste proxecto levouse a cabo durante tres períodos de tempo diferenciados e separados no tempo:

- O verán do ano 2013 como parte do programa GSoC.
- O primeiro cuatrimestre do curso 2013/2014.
- O verán do ano 2014 novamente como parte do GSoC.

Neste capítulo trataremos a planificación do proxecto e como se levou a cabo neses tres períodos.

5.1. Google Summer of Code 2013

A duración do período do traballo programando do Google Summer of Code son aproximadamente 4 meses, un total de 15 semanas. Durante a edición de 2013 planifícase que se traballe 13 semanas. As dúas semanas restantes corresponden á asistencia á GUADEC e ó comezo do ano lectivo universitario 2013/2014. Cóntase traballar aproximadamente 5 horas diarias o que dá un total de 325 horas.

O programa GSoC pide ós participantes reportes periódicos en forma de artigos en blogs, así que estas serán as nosas iteracións a través das cales iremos recibindo feedback por parte dos futuros usuarios do aplicativo. En función deste feedback iremos modificando o programa. Neste caso empregárase un blogue persoal creado con anterioridade e de nome Aquelando.info. As publicacións deste blogue así como as de moitos desenvolvedores do proxecto GNOME están ligadas con Planet GNOME, que é un agregador de blogues, polo

que a súa difusión é moi alta. A periodicidade das publicacións e polo tanto das iteracións variará dunha a outra pero é de entre dúas e tres semanas.

Durante este GSoC fixéronse 5 iteracións que explicamos a continuación.

5.1.1. Primeira Iteración: Análise, deseño xenérico e inicio da implementación

A primeira iteración iníciase o 13 de Xuño e rematará o 30 de Xuño. O obxectivo desta primeira iteración é facer unha análise das necesidades dos tradutores e realizar un deseño xenérico da estrutura do programa.

Análise e deseño Para a análise de requisitos, instalaranse e estudaranse as características dalgúns dos programas do mercado. Ademais, contactaremos con diversos equipos de tradución a través das súas listas de correo. Unha vez que se obteña certo feedback por parte dos equipos comezarase a realizar un deseño xenérico da estrutura do programa.

Implementación da clase abstracta ficheiro Empezarase a implementar parte do núcleo do sistema en concreto unha clase abstracta ficheiro e unha clase *DemoFile* que servirá como *mock* para poder implementar a interface. Para a implementación desta clase, terase en conta o concepto de estensibilidade xa que este programa, aínda que se centrará na edición de ficheiros PO por ser estes os “oficiais” de GNOME, tamén queremos que sexa útil para a comunidade de tradutores en xeral.

5.1.1.1. Reporte e feedback

Escribíronse un total de 3 reportes. Un¹ facendo referencia ás peticións obtidas por parte dos equipos de tradución e dous^{2,3} con algúns detalles que se tiveron en conta durante o deseño do programa. Houbo bastantes comentarios no blogue e entre outras cousas os usuarios destacaron:

- A importancia de non usar o patrón Singleton.
- Non facer os widgets dependentes da aplicación para que poidan ser reutilizados en outras aplicacións como Anjuta.
- Usar gettext-po para implementar os ficheiros po.

¹Starting the GSoC Project!!

²ValaCAT. Some design aspects

³ValaCAT. Some design aspects (Part 2)

- Empregar unha interface máis semellante a anterior de GTranslator.
- Eliminar a columna de ID pois non se considera útil.
- Auto-expandir o tamaño dos campos cando as cadeas sexan grandes.

Con respecto a estas ideas modificouse a interface que se ía facer para buscar un deseño moi parecido ao de GTranslator empregando incluso a mesma biblioteca GNOME Docking Library que permite modificar os bloques da interface. Ademais eliminouse por completo o ID da cadea da interface. En canto á biblioteca gettext-po xa tiñamos en mente empregala.

5.1.1.2. Tarefas e seguimento

A descomposición de tarefas desta iteración é a seguinte:

WBS 1.1 Análise de Requisitos

WBS 1.1.1 Estudo de ferramentas existentes.

WBS 1.1.2 Enviar correos a diferentes equipos de tradutores.

WBS 1.1.3 Analizar respostas dos equipos.

WBS 1.2 Deseño xenérico do aplicativo.

WBS 1.3 Deseño e implementación do ficheiro xenérico.

WBS 1.4 Escribir primeiros reportes.

Ó igual que sucederá en todo este proxecto, a planificación destas tarefas é lineal pois só dispoñemos dun recurso. Na Figura 5.1 pódese ver o diagrama de Gantt que corresponde a esta iteración.

Planificáronse 70 horas para completar todas as tarefas e puidéronse completar con éxito todas as tarefas nese tempo.

5.1.2. Segunda Iteración: Linguaxes e Interface

A segunda iteración durou dende o 1 de Xullo ata o 18 de Xullo. Nesta iteración profundizarase no núcleo do sistema e empezaremos a traballar na interface de usuario.

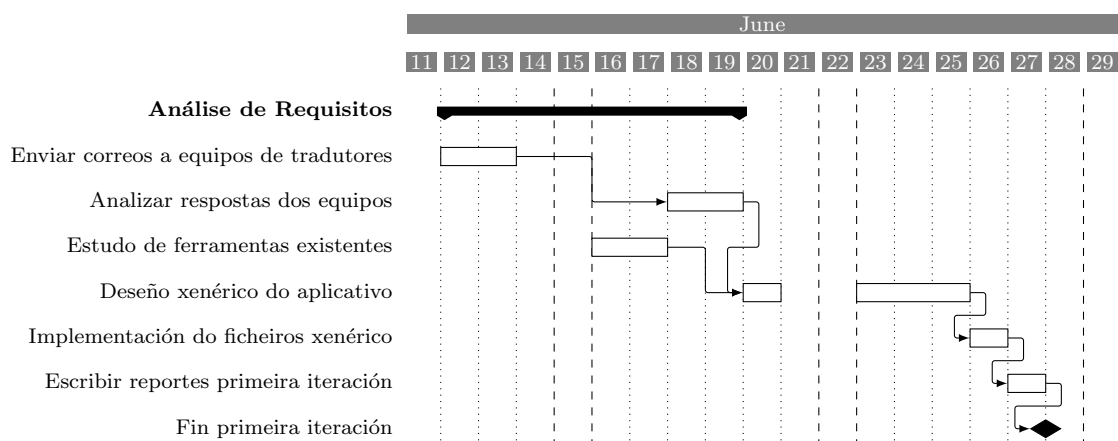


Figura 5.1: Diagrama de Gantt da primeira iteración do GSoC 2013.

Linguaxes Seguirase implementando o núcleo do sistema. En concreto implementarase o módulo de linguaxes que contará coas clases *Linguaxe* e *Forma Plural*. As linguaxes empregadas almacenaranse nun par de ficheiros. A idea detrás destes ficheiros é engadir información adicional a cada linguaxe e a cada forma plural. Entre esta información inclúense etiquetas que explicarán a que identificador de forma plural corresponde cada plural. Por exemplo, para os plurais do galego, a forma plural 0 tería unha etiqueta “1 elemento” e a forma plural tería a forma “0 ou máis de 1 elementos”.

Interface de usuario Empezarase a traballar na interface de usuario. Concretamente nesta iteración implementarase a estrutura xeral e os *widgets* de lista e mensaxes, de edición de mensaxes e para amosar o contexto da mensaxe.

5.1.2.1. Reporte e feedback

Escribiuse un⁴ reporte onde se puxo un vídeo no que se amosaba o estado do programa e todas as características implementadas. Recibimos un comentario preguntando para qué serven certas partes da aplicación que aparecen no vídeo e respóndese explicando as ideas empregadas e reforzándoo con outro vídeo centrado nesas partes.

5.1.2.2. Tarefas e Seguimento

As tarefas que se realizarán durante esta iteración son as seguintes:

WBS 2.1 Deseño e Implementación do módulo de Linguaxes.

⁴[ValaCAT application current status](#)

WBS 2.1.1 Creación dos ficheiros JSON coa lista de formas plurais e de linguaxes.

WBS 2.2 Interface de usuario.

WBS 2.2.1 Implementación da lista de mensaxes.

WBS 2.2.2 Implementación do editor de mensaxes.

WBS 2.2.3 Implementación da barra de estado.

WBS 2.2.4 Implementación da lapela xenérica.

WBS 2.2.5 Implementación da lapela para ficheiros.

WBS 2.3 Creación do Makefile

WBS 2.4 Creación de filtros para as cadeas.

WBS 2.5 Escribir reportes.

Na Figura 5.2 pódese ver o diagrama de Gantt que corresponde a esta iteración.

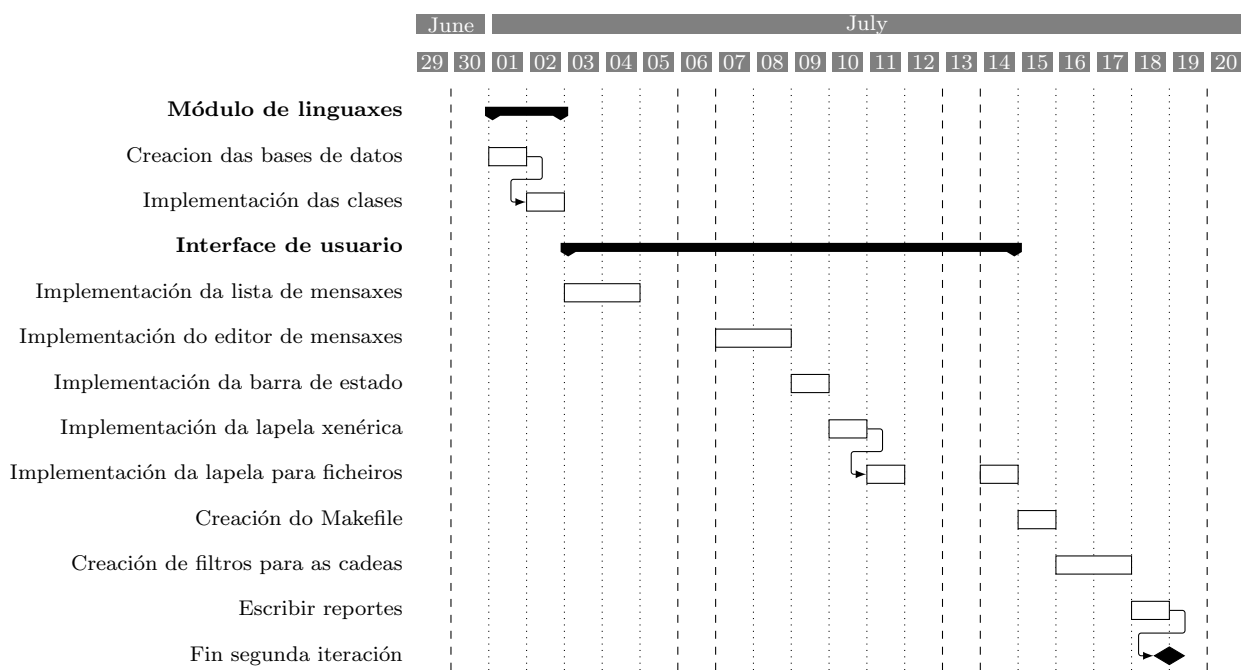


Figura 5.2: Diagrama de Gantt da segunda iteración do GSoC 2013

Planificáronse un total de 70 horas para esta iteración. As dificultades a hora de implementar o editor de mensaxes fixeron que esta tarefa levara 5 horas máis do previsto. Desta forma as tarefas completáronse en 75 horas. Estas horas fixéronse traballando máis horas ao día.

5.1.3. Terceira Iteración: Interface, iteradores e buscas

A terceira iteración dura entre o 19 de Xullo e o 5 de de Agosto. Nesta iteración continuaremos traballando na interface, e implementaranse o sistema de navegación a través do documento e as buscas.

Iteradores e Buscas Os iteradores son o sistema que se empregará para navegar a través do documento. Nesta iteración comezarse a súa implementación. Estes iteradores construíranse de forma que o usuario poida navegar a través de todas as cadeas, das cadeas sen traducir ou das cadeas con tradución difusa. Outra parte importante desta iteración é a implementación dun módulo para buscar texto no documento.

Interface de usuario Continuarase o traballo na interface. Traballaremos no widget de edición para que inclúa resaltado de sintaxe e de espazos en branco empregando a biblioteca GTKSourceview. Ademais empezaremos a implementar a clase Aplicación. Por último implementárase a parte da interface que permite usar as buscas.

5.1.3.1. Presentación GUADEC 2013 (Brno)

A GUADEC é a xuntanza europea de desenvolvedores e usuarios de GNOME. No ano 2013 esta xuntanza ten lugar na cidade de Brno (Figura 5.3). Os participantes no GSoC están invitados a ir a dita reunión e expoñer o seu traballo nunha charla lóstrego dun máximo de 3 minutos.



Figura 5.3: GUADEC 2013 (Brno)

Ademais de presentar o proxecto, nesta xuntanza participei no evento como voluntario o que me permitiu coñecer a moita xente converténdose nunha experiencia moi enriquecedora tanto profesional como persoalmente.

5.1.3.2. Reportes e Feedback

A Fundación GNOME ponse en contacto comigo a través do email para pedirme que, xa que esta vai ser o aplicativo oficial de GNOME, ten que ter a palabra GNOME no seu nome.

Escribíronse dous reportes, un⁵ falando da miña experiencia na GUADEC, e outro⁶ falando dos avances no programa e pedindo ideas para o novo nome do aplicativo. Daniel Mustieles responde que “GNOME Translator” ou “GNOME Translation Tool” serían boas opcións.

5.1.3.3. Tarefas e seguimento

A descomposición en tarefas desta iteración é a seguinte:

WBS 3.1 Deseño e implementación dos iteradores.

WBS 3.2 Deseño e implementación do sistema de busca.

WBS 3.3 Interface de usuario.

WBS 2.3.1 Resaltado do texto ao facer click nos consellos.

WBS 2.3.2 Modificar editor de mensaxes

WBS 3.4 Escribir reportes.

Na Figura 5.4 pódese ver o diagrama de Gantt correspondente as tarefas desta iteración.

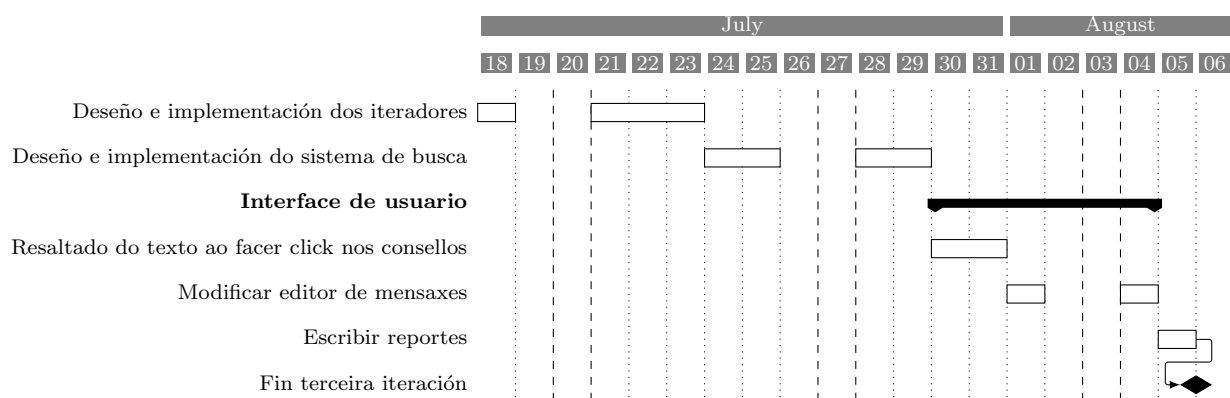


Figura 5.4: Diagrama de Gantt da terceira iteración do GSoC 2013.

Esta iteración planificouse para un total de 65 horas. Nesta ocasión non houbo ningún problema e puidéronse executar as tarefas no tempo previsto.

⁵[GUADEC 2013](#)

⁶[Searching...](#)

5.1.4. Cuarta Iteración: Ficheiros po, Autotools, Proxectos, barra de busca

Esta iteración comeza o 6 de Agosto e remata o 30 do mesmo mes. O obxectivo desta iteración é engadir soporte para ficheiros po, proxectos, autotools e engadir unha barra de busca a interface.

Ficheiros PO O obxectivo consiste en implementar o soporte para ficheiros PO pois ata ese momento estíbbase empregando un mock. Crearanse bindings para a biblioteca gettext-po e implementárase unha subclase da clase abstracta File.

Proxectos Durante esta iteración tamén se implementarán os proxectos, definindo proxecto como un conxunto de ficheiros que están na mesma carpeta.

Autotools Implementárase tamén o soporte para autotools do proxecto pois ata o momento víñase a empregar un simple ficheiro Makefile.

Interface de usuario Crearanse accións para facer e desfacer cambios, navegar a través do documento entre outras cousas. Estas accións poderán ser activadas a través de botóns na interface de usuario ou mediante atallos de teclado que se crearán posteriormente. Ademais engadirase o soporte para abrir ficheiros dende a propia interface e a posibilidade de ver os ficheiros recentes.

5.1.4.1. Reportes e feedback

Falando co anterior *maintainer* de GTranslator a través de IRC, este aporta bastantes consellos sobre o programa como o uso de Autotools ou varios detalles da interface de usuario.

Realizáronse un total de dous reportes⁷⁸, pero ninguén escribiu ningún comentario no blogue.

5.1.4.2. Tarefas e Seguimento

As tarefas que se realizaron durante esta iteración foron as seguintes:

WBS 4.1 Implementación dos ficheiros po.

⁷GSoC application status report

⁸Po files, projects, navigation and other stuff I have been doing

WBS 4.2 Implementación de proxectos

WBS 4.3 Implementación de accións

WBS 4.3.1 Accións desfacer-refacer

WBS 4.3.2 Accións de navegar polo documento.

WBS 4.4 Engadir barra de busca.

WBS 4.5 Eliminar barra de estado.

WBS 4.6 Substituir Makefile por Autools.

WBS 4.7 Internacionalización do programa.

WBS 4.8 Escribir reportes.

Na Figura 5.5 pódese ver o diagrama de Gantt correspondente a esta iteración.

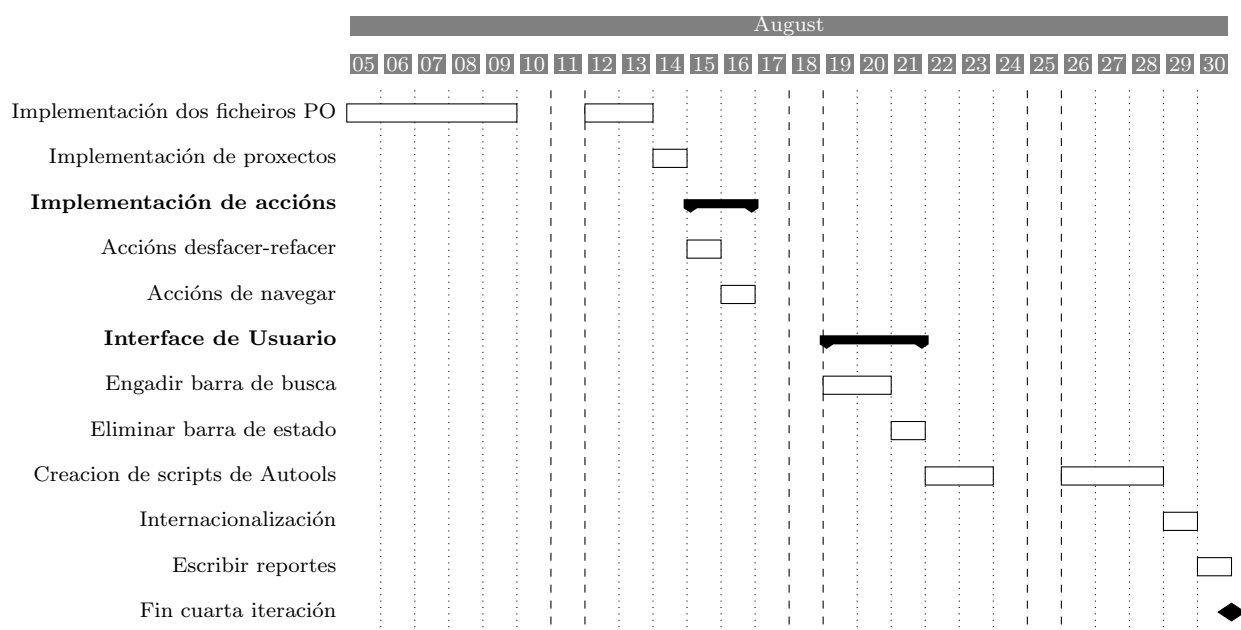


Figura 5.5: Diagrama de Gantt da cuarta iteración do GSoC 2013.

Planificáronse un total 100 horas para esta iteración. Tardamos 5 horas máis das previstas en implementar o soporte para Autotools e outras 5 horas adicionais por problemas encontrados na implementación dos bindings da biblioteca de GetText. De igual forma que nos casos anteriores estes problemas resolvéronse ampliando a xornada laboral.

5.1.5. Quinta Iteración: Preferencias, limpiar código e documentación

Esta iteración comeza o 1 de setembro e remata o 23 do mesmo mes. Durante ela seguirase traballando na interface, limparase o código e crearase algo de documentación de cara á entrega final do Google Summer of Code.

Preferencias Empezaranse a implementar as preferencias no programa pois, ata o momento, moitas das opcións estaban *hardcoded*, e dicir escritas directamente no código. Para facer isto farase uso do compoñente de GLib GSettings que permite o almacenamento sinxelo de configuración das aplicacións. Ademais implementarase un diálogo semellante ao empregado en GTranslator para poder modificar estas preferencias.

Mellorar a calidade do código e documentación Revisión completa do código para manter un mesmo estilo ao longo de todo o código. Ademais actualizáronse os diagramas UML creados na primeira iteración.

5.1.5.1. Reportes e Feedback

Escribiuse un⁹ reporte onde se conta o estado da aplicación demostrándoo con un vídeo. Ó tratarse do último post desa edición do GSoC, agradezo a oportunidade que me deu Google para poder estar ese verán traballando en software libre e describo a miña experiencia.

5.1.5.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron as seguintes:

WBS 5.1 Deseño e implementación das preferencias.

WBS 5.2 Corrixir Clase ficheiro.

WBS 5.3 Corrixir erros de estilos no código.

WBS 5.4 Actualizar diagramas UML.

WBS 5.5 Escribir reportes.

Hai que ter en conta que nesta iteración a cantidade de tempo traballada por día é inferior nalgúns momentos, pois ten que compatibilizarse co horario lectivo do curso

⁹[GSoC Final Report](#)

2013/2014. Na Figura 5.6 pódese ver o diagrama de Gantt que corresponde a esta iteración.

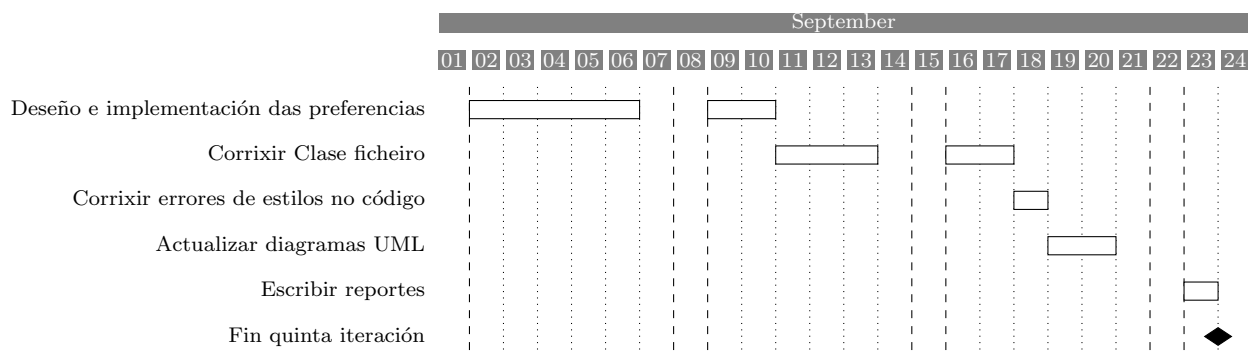


Figura 5.6: Diagrama de Gantt da quinta iteración do GSoC 2013.

Planificáronse un total de 60 horas para resolver as tarefas e non houbo ningún problema en completalas con éxito.

5.1.6. Estado ao fin do GSoC 2013

O finalizar esta iteración o mentor do GSoC avalía correctamente o proxecto presentado polo que o programa é completado con éxito. O programa entregado ten entre outras moitas as seguintes características:

- Posibilidade de abrir ficheiros po.
- Navegación a través do documento.
- Posibilidade de buscar.
- Editor con resaltado de sintaxe e de espazos en branco.
- Preferencias.

Pero tamén presenta algúns fallos:

- Lentitude ao cargar ficheiros moi grandes.
- Fallo ao gardar un ficheiro.
- O aspecto da interface non é satisfactorio.
- Problemas ao buscar.

Estas limitacións serán eliminadas durante os seguintes períodos.

5.2. Primeiro cuatrimestre curso 2013/2014

Durante este período a implementación do programa faise en paralelo coa vida universitaria. Planifícanse dúas iteracións, a primeira durante o que falta do mes de setembro e o mes de outubro e a segunda durante o final do mes de novembro. Nestes dous períodos de tempo a carga de traballo das tarefas das materias do curso 2013/2014 é menor e permítenos continuar traballando coa aplicación.

5.2.1. Primeira iteración: cambios na interface

A primeira iteración comprende os últimos días de setembro e o mes de outubro. Durante este tempo farase un redeseño da interface gráfica e impleméntanse as pistas e os comprobadores:

Interface Gráfica Preténdese conseguir un mellor resultado do actual. Empregaremos os deseños iniciais que teñen un parecido máis importante a aplicación Virtaal. Reutilízase os widgets creados combinándoos para conseguir o efecto desexado.

Pistas e Comprobadores As pistas (*hints*) son posibles traducións para unha cadea determinada. Nesta iteración crearase tanto o panel da interface que permite ver estas pistas como a clase que lle prove as pistas a dita interface. Os comprobadores (*checkers*), por outro lado, apórtanlle pistas de cada tradución feita.

5.2.1.1. Presentación GUADEC Hispana 2013 (Madrid)

A GUADEC Hispana (Figura 5.7) é unha reunión de usuarios e desenvolvedores de GNOME de fala castelá e que serve tamén para a reunión anual (como obriga a lei) da organización GNOME Hispano. Ademais nesta reunión fanse charlas sobre GNOME e outros temas relacionados.

Entre esas charlas tiven a oportunidade de presentar o proxecto polo que os asistentes amosaron o seu interese por que se seguira desenrolando.

5.2.1.2. Reportes e feedback

Escribiuse un reporte¹⁰ mais ningún escribiu ningún comentario. Non obstante durante a GUADEC-es houbo xente que se interesou polo programa.

¹⁰[Changing the tool UI](#)



Figura 5.7: GUADEC Hispana 2013 (Madrid)

5.2.1.3. Tarefas e seguimento

A descomposición en tarefas para esta iteración é a seguinte:

WBS 1.1 Implementar Pistas e Proveedor de Pistas.

WBS 1.2 Implementar Comprobador.

WBS 1.3 Interface de Usuario.

WBS 1.3.1 Eliminar biblioteca GDL.

WBS 1.3.2 Engadir widget de Pistas.

WBS 1.3.3 Misturar lista de mensaxes con editor de mensaxes.

Hai que ter en conta que ao ter que compatibilizar a realización destas tarefas co horario académico o número de horas traballadas cada día é menor que no período anterior e non é regular. Na Figura 5.8 pódese ver o diagrama de Gantt que corresponde a esta iteración.

Para a realización destas tarefas planificáronse 50 horas que se completaron con éxito.

5.2.2. Segunda iteración: Refactorizar navegadores e melloras na interface

Esta iteración sucede no final do mes de novembro. Durante este tempo cambiaráselle o nome ao aplicativo, refactorízase a API para navegar a través do documento e realízanse pequenos cambios na interface de usuario.

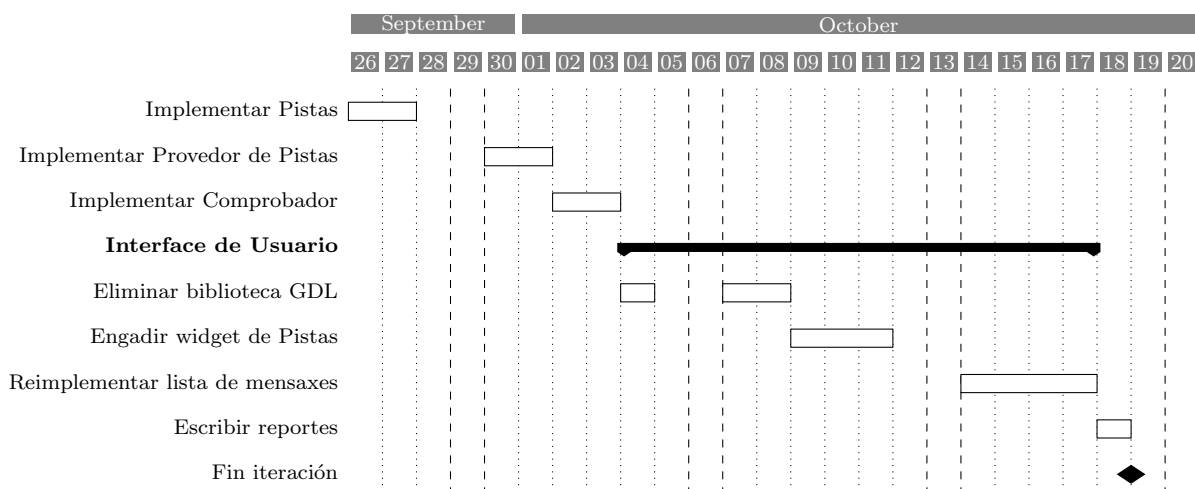


Figura 5.8: Diagrama de Gantt da primeira iteración do primeiro cuatrimestre do curso 2013/2014.

Refactorización dos navegadores Crearase unha API a nivel de aplicación para movernos a través do documento. O obxectivo é implementar operacións para seleccionar e tamén deseleccionar cadeas e fragmentos destas cadeas a través de todo o documento.

Cambio de nome Reutilizarase un cambio de nome do aplicativo cambiando o anterior ValaCAT por GNOME CAT. Este cambio é producido polo requirimento por parte da fundación GNOME de que o programa tiña que ter a marca GNOME no seu nome.

Interface de usuario Farase un redeseño á opción de buscar avanzado para eliminar o diálogo existente e engadir as opcións existentes á propia barra de busca.

5.2.2.1. Reportes e Feedback

Escríbese un¹¹ contando os últimos avances do aplicativo e o cambio de nome. Recibimos un comentario dicindo que GNOME escribese con letras maiúsculas polo que temos que corrixir o programa. Outra xente interesase polo significado de CAT.

5.2.2.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron as seguintes:

WBS 2.1 Cambiar nome do aplicativo.

¹¹Welcome GnomeCAT

WBS 2.2 Refactorizar navegadores.

WBS 2.3 Interface de usuario.

WBS 2.3.1 Redeseño da opción buscar avanzado.

De igual forma que no caso anterior o feito de estar dentro do período lectivo, causa que o número de horas traballadas sexa menor do normal. Na Figura 5.9 pódese ver o diagrama de Gantt que corresponde a esta iteración.

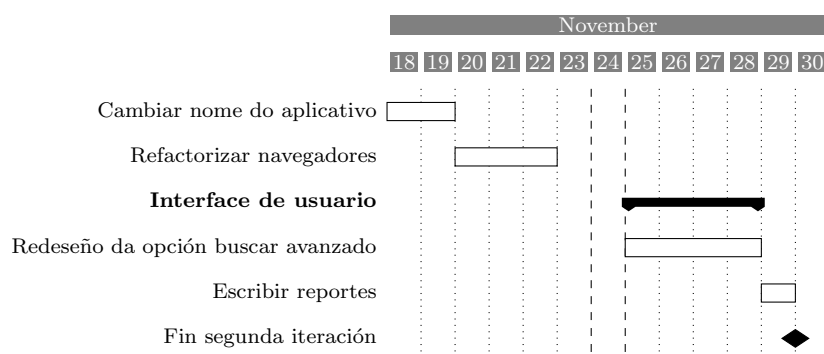


Figura 5.9: Diagrama de Gantt da segunda iteración do primeiro cuatrimestre do curso 2013/2014.

Para a realización das tarefas descritas planifícanse 35 horas que son suficientes para completalas nos días estipulados.

5.2.3. Presentación para o GSoC 2014

Debido a evidente falta de tempo para completar o programa decídese presentar o programa de novo ó programa Google Summer of Code. Falase coa coordinadora dos programas de iniciación en GNOME para saber se isto é posible e respóndenos afirmativamente. Como mentor falamos co *maintainer* de GTranslator que nos di que non ten ningún problema en facer de mentor. Presentamos o proxecto e este sae aceptado.

5.3. Google Summer of Code 2014

O Google Summer of Code durante o 2014 dura dende o 19 de maio ata o 11 de agosto un total de 15 semanas. Na planificación realizada cóntanse 13 semanas pois nunha das semanas o curso lectivo 2013/2014 aínda non acabou e na outra realizase o viaxe a GUADEC en Strasbourg. Da mesma forma que no caso anterior planéanse traballar 5 horas diarias o que dá un total de 325 horas.

A maior experiencia no programa que estamos a facer permítenos facer unha planificación máis exacta. Faranse un total de cinco iteracións dunha duración aproximada de dúas semanas.

5.3.1. Primeira iteración: Redeseño UI

Esta iteración dura dende o 23 de maio ata o 15 xuño. Durante ela fundamentalmente traballaremos nun redeseño da interface.

Interface Gráfica O obxectivo desta iteración é un novo redeseño da interface. Contactarase cos deseñadores de GNOME a través de IRC para obter algún feedback pola súa parte. Unha vez conseguido, implementarase a nova interface.

Bindings Gettext-PO: Orixes das mensaxes A biblioteca gettext-po ten soporte para conseguir as orixes das mensaxes. Nesta iteración tamén se mellorarán os seus bindings para recoller esta característica e tamén a implementaremos na clase PoFile engadindo a propiedade origins que permite obter en que ficheiros e liñas aparece dita cadea.

5.3.1.1. Reportes e Feedback

Escribíronse dous¹²¹³ reportes. En esta iteración hai bastante resposta por parte do usuario. Por unha parte reportase un erro na compilación do programa respondémoslle ao usuario dándolle un *workaround* mentres buscamos o motivo real do problema.

Ademais, o deseñador que fixo os mockups empregados agradece que os esteamos usando e sinala que sería boa idea substituír o menú de selección de idioma actual por algo máis vistoso. Tamén comenta que combinar os botóns de gardar e de volver sería boa idea pois así obrigáramos ó usuario a gardar o documento antes de volver á lista de ficheiros e desta forma evitar que se perdan os datos. Anotamos estas dúas ideas en GitHub para implementalas máis tarde.

5.3.1.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron:

WBS 1.1 Creación do ficheiro .desktop.

WBS 1.2 Mellora de Autotools: xeneración automática das cadeas a traducir.

¹²[GNOMECAT. Progress report](#)

¹³[Implementing the Editing Panel](#)

WBS 1.3 Interface de Usuario.**WBS 1.3.1** Nova estrutura xeral.**WBS 1.3.2** Creación da barra de ferramentas.**WBS 1.3.3** Panel de edición.**WBS 1.3.4** Panel de preferencias.**WBS 1.3.5** Panel de perfil.**WBS 1.3.6** Panel de benvida.**WBS 1.3.7** Menú de aplicativo.**WBS 1.4** Bindings Gettext-PO: implementación das orixes dos mensaxes.**WBS 1.5** Refactorización das buscas.**WBS 1.6** Refactorización das formas plurais.

Na Figura 5.10 pódese ver o diagrama de Gantt que corresponde a esta iteración.

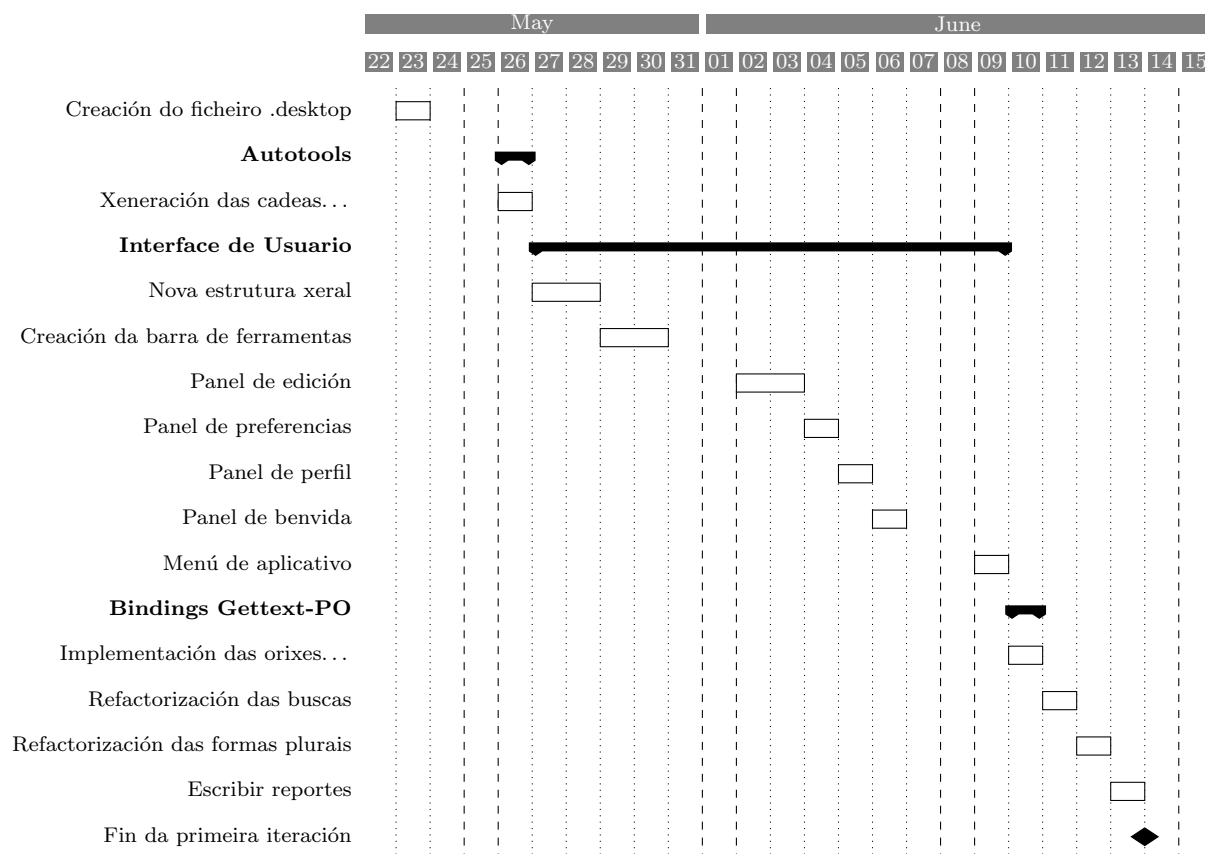


Figura 5.10: Diagrama de Gantt da primeira iteración do GSoC 2014.

Para facer as tarefas anteriores planifícanse un total de 80 horas. Non son necesarias horas adicionais nesta ocasión.

5.3.2. Segunda Iteración: Bindings Gettext-PO e interface

Esta iteración dura dende o 15 de xuño ata o 6 de xuño. Continuamos mellorando a interface de usuario, implementamos atallos de teclado e engadimos funcionalidade aos bindings de Gettext-PO.

Interface de usuario Traballárase en implementar atallos de teclado e mellorar o panel de editar perfís de usuarios dividindo o panel en dous subpaneis. Por último engadírase na interface a opción de obter os orixes de cada mensaxe.

Bindings Gettext-PO Implementación da opción de modificar as cabeceiras dos ficheiros PO aparte de solucionar algúns erros detectados.

5.3.2.1. Reportes e Feedback

Escíbese un¹⁴ reporte. Só recibimos un comentario sinalando que sería interesante que durante a GUADEC falase con un dos desenvolvedores de GNOME.

5.3.2.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron:

WBS 2.1 Interface de Usuario.

WBS 2.1.1 Atallos de teclado.

WBS 2.1.2 Widget de Pistas.

WBS 2.1.3 Orixes das cadeas.

WBS 2.1.4 Mellorar Panel de Perfil.

WBS 2.1.5 Mellorar Panel Abrir Ficheiro.

WBS 2.2 Arranxar Buscas.

WBS 2.3 Bindings Gettext-PO.

WBS 2.3.1 Xestión de cabeceiras.

WBS 2.3.2 Gardado de ficheiros.

WBS 2.4 Escribir reportes.

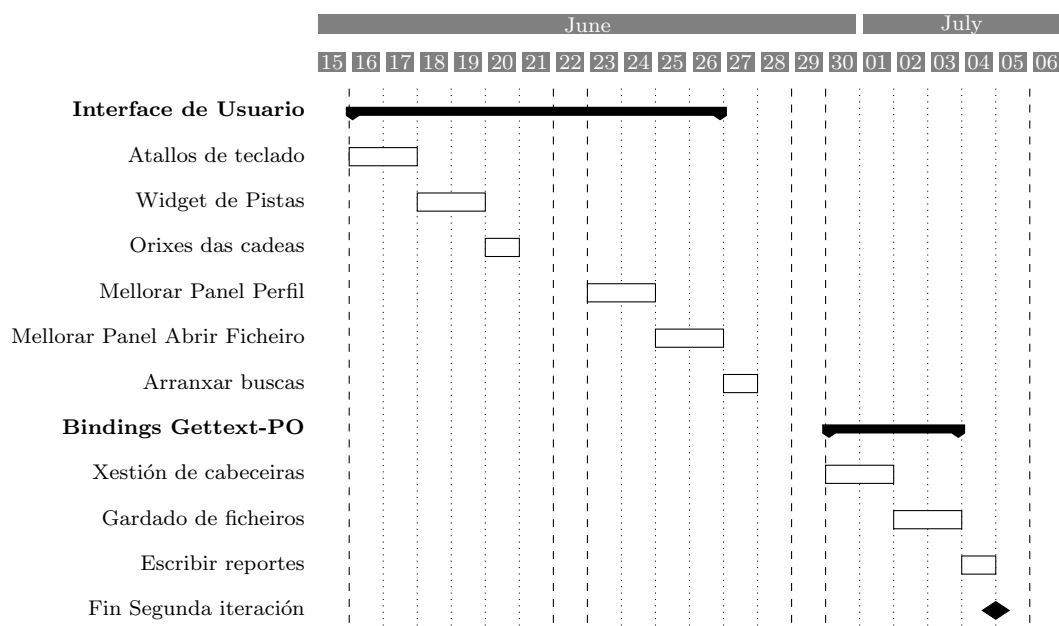


Figura 5.11: Diagrama de Gantt da segunda iteración do GSoC 2014.

Na Figura 5.11 pódese ver o diagrama de Gantt que corresponde a esta iteración.

Planificamos un total de 75 horas para esta iteración. Necesitamos 3 horas adicionais para resolver algúns problemas encontrados coa implementación dos bindings para a biblioteca Gettext-PO.

5.3.3. Terceira Iteración: Perfiles de usuario e interface

Esta iteración dura dende o 7 de Xullo ao 16 do mesmo mes. Melloramos algo a interface de usuario e implementamos algún detalle que faltaba no sistema de perfiles.

Interface de usuario: Barra de notificación Implementarase un sistema de notificacións para alertar ó usuario de certos eventos no programa como, por exemplo, cando os parámetros de busca non xeran ningún resultado. Para iso empregaremos o widget de GTK GtkInfoBar que amosaremos durante 3 segundos.

Perfiles de usuario Implementarase a funcionalidade de activar perfiles e engadirase un botón a interface para permitir borrar e activar perfiles.

¹⁴[Keep working on GNOME CAT](#)

5.3.3.1. Reportes e Feedback

Escribese un¹⁵ reporte pero, non recibimos feedback por parte dos tradutores nesta iteración.

5.3.3.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron:

WBS 3.1 Interface de usuario: barra de notificación.

WBS 3.2 Perfiles de usuario.

WBS 3.21 Función borrar perfil.

WBS 3.22 Función activar perfil.

WBS 3.3 Modificar ficheiro das formas plurais.

Na Figura 5.12 pódese ver o diagrama de Gantt que corresponde a esta iteración.

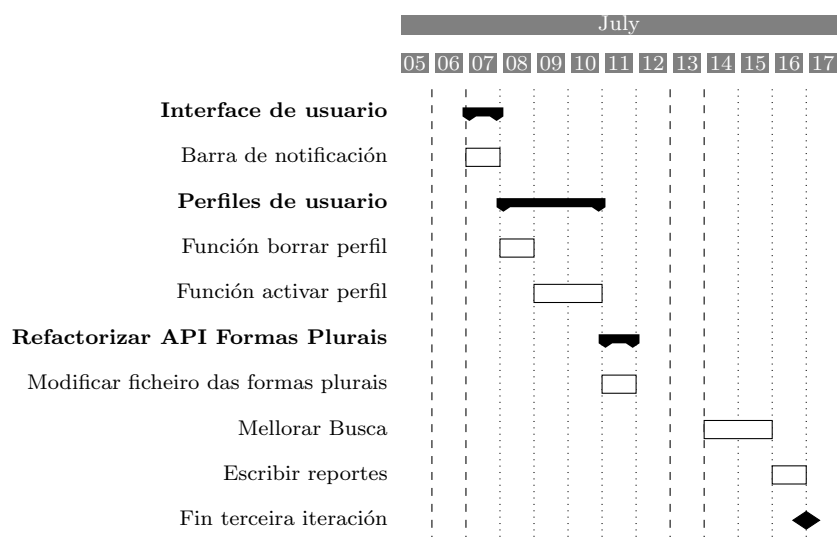


Figura 5.12: Diagrama de Gantt da terceira iteración do GSoC 2014.

Para completar as tarefas anteriores planificamos 40 horas que foron suficientes para conseguir rematar as tarefas con éxito.

¹⁵[Details and more details](#)

5.3.4. Cuarta Iteración: Plugins e GUADEC

Esta iteración dura dende o 16 de xullo ata o 5 de Agosto tendo a GUADEC polo medio. Consequimos implementar o motor de plugins e creamos algúns plugins de exemplo.

Plugins Engadiremos o soporte de plugins ó programa. Por unha parte incorporaremos un motor de plugins e refactorizaremos o código para que certas partes do mesmo deixen de depender da interface de usuario. Ademais, modificaremos a configuración de Autotools para que compile os plugins.

5.3.4.1. Presentación GUADEC 2014 (Strasbourg)

Da mesma forma que na edición anterior, acudimos á GUADEC, a reunión de programadores e usuarios de GNOME en Europa. Esta vez celébrase en Strasbourg (Figura 5.13).



Figura 5.13: GUADEC 2014 (Strasbourg)

Tamén temos a oportunidade de presentar o proxecto ante a comunidade GNOME nunha charla lóstrego de 3 minutos onde explicaremos os novos cambios na interface, a implementación e plugins e as novas características implementadas.

5.3.4.2. Reportes e Feedback

Escríbese un¹⁶ reporte. Ninguén fai comentarios no blogue sobre o programa, pero durante a GUADEC fálase con algún desenvolvedor que nos dá consellos sobre algún dos problemas que temos. Marina Zhurakhinskaya recoméndanos falar con Allan Day que é o deseñador líder de GNOME para que nos diga se hai algo que debemos mellorar no deseño do aplicativo.

5.3.4.3. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron:

WBS 4.1 Melloras en Autools

WBS 4.2 Interface de usuario: *scrolling* nas buscas.

WBS 4.3 Implementación de plugins.

WBS 4.4 Refactorizar clase File: evitar dependencias coa interface.

Na Figura 5.14 pódese ver o diagrama de Gantt que corresponde a esta iteración.

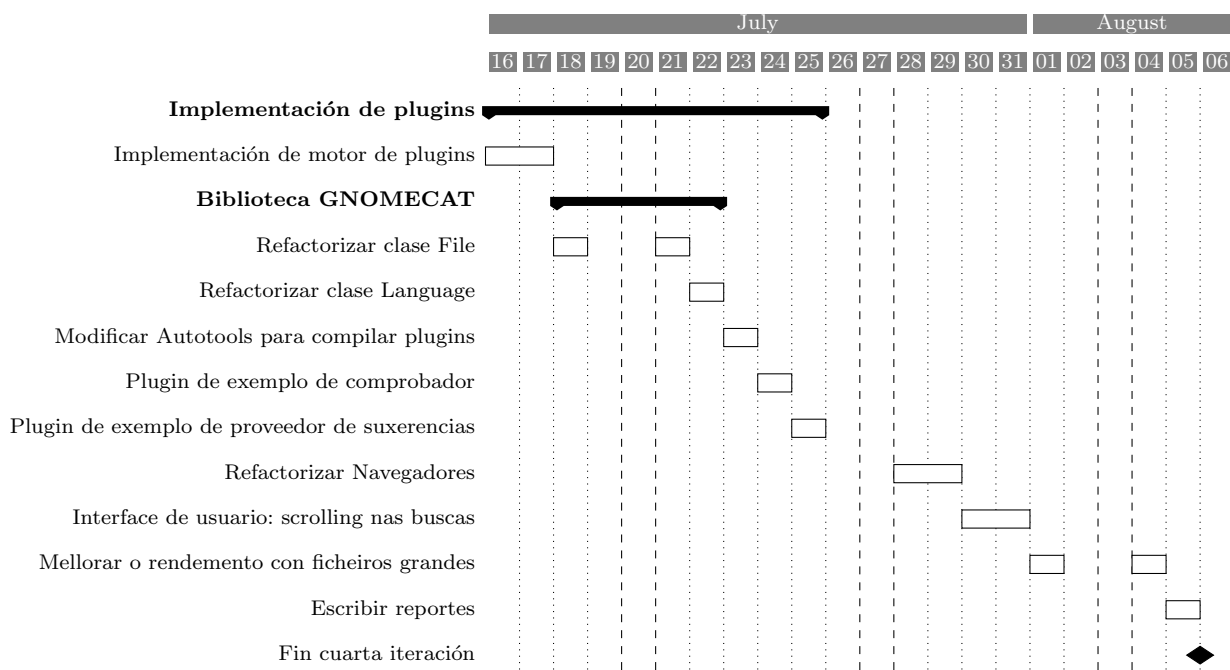


Figura 5.14: Diagrama de Gantt da cuarta iteración do GSoC 2014.

¹⁶API, GUADEC and big files

Planificamos 75 horas para as tarefas desta iteración. Precisamos 5 horas adicionais para modificar os scripts de Autotools que permiten a compilación dos plugins. Para non incrementar o número de días desta iteración aumentamos o número de horas que traballamos determinados días.

5.3.5. Quinta Iteración: Detalles finais e escribir documentación

Esta iteración dura dende o 5 de Agosto ata o 15 do mesmo mes. Nela puliremos algún detalle da interface e melloraremos a calidade do código e a documentación.

Documentación e limpar código O obxectivo é facer unha revisión do código para que este manteña un mesmo estilo. Ademais actualizarase a documentación existente e crearse algunha nova.

Interface: Lista de mensaxes Intentarase mellorar o rendemento da lista de mensaxes que detectamos que non funciona correctamente con ficheiros grandes. Este é un problema que temos dende a primeira edición do GSoC. Ademais engadiremos a función de filtrar e ordenar mensaxes.

5.3.5.1. Reportes e Feedback

Nesta ocasión non se escriben reportes mais si que recibimos feedback de algún tradutor a través do correo electrónico. As cousas que estes tradutores consideran que se deben mellorar son as seguintes:

- Mellora dos atallos de teclado.

Ademais falamos con Allan Day que nos di que cando poida fará unha revisión da interface de usuario do programa.

5.3.5.2. Tarefas e seguimento

As tarefas realizadas durante esta iteración foron:

WBS 5.1 Documentación.

WBS 5.2 Limpar código.

WBS 5.3 Interface de usuario: Refactorizar lista de mensaxes.

WBS 5.3.1 Implementar novo widget de lista de mensaxes.

WBS 5.3.2 Implementar filtros para lista de mensaxes.

WBS 5.3.3 Implementar a función de ordenar para mensaxes.

WBS 5.4 Melloras en Autotools.

Na Figura 5.15 pódese ver o diagrama de Gantt que corresponde a esta iteración.

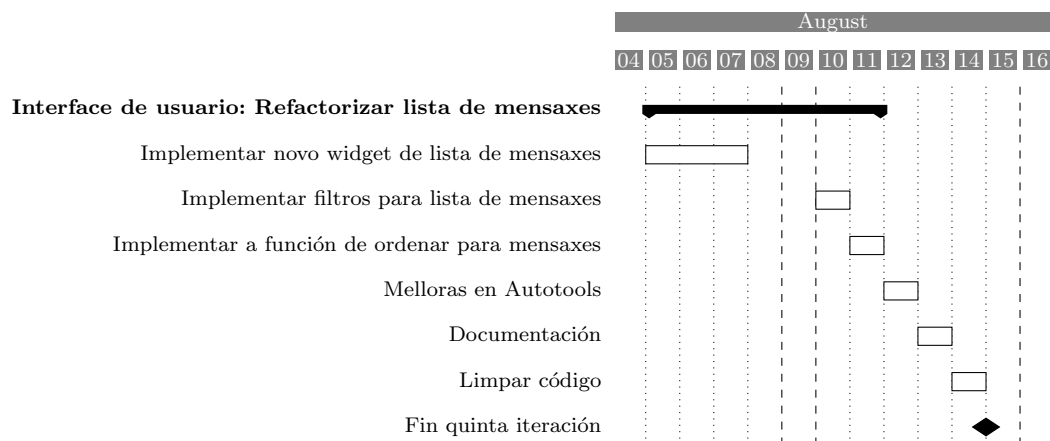


Figura 5.15: Diagrama de Gantt da quinta iteración do GSoC 2014.

Para resolver as tarefas desta iteración necesitaremos 40 horas. Non se necesitan horas adicionais para completar o traballo.

Capítulo 6

Análise de Requisitos globais

Neste capítulo explicaremos o proceso de análise de requisitos levado a cabo para a construción da ferramenta.

6.1. Consultas cos tradutores

Para facer unha nova ferramenta de asistencia á tradución consultamos os usuarios principais da ferramenta. Para iso enviamos correos a unhas cantas listas de correo de tradutores de diferentes proxectos de software libre como os seguintes:

GNOME Dentro do proxecto GNOME hai un grupo específico para a internacionalización dos programas da plataforma. Existen listas de correo para cada linguaxe e unha a nivel internacional. En concreto as listas ás que enviamos un correo preguntando por ideas para o novo programa, foron a lista *internacional*, a lista de tradutores ó *galego* e a lista de tradutores ó *castelán*.

Proxecto Trasno É unha comunidade de tradutores de proxectos de software libre ao Galego. Serve de punto de encontro para os tradutores galegos e realiza periodicamente reunións para a discusión da terminoloxía a usar e a presentación de novas ferramentas.

OpenSuse e Fedora Estas dúas distribucións de Linux contan cos seus respectivos equipos de tradutores. Aínda que a maior parte dos programas destas distribucións son traducidos por outros proxectos, como pasa no caso do entorno de escritorio GNOME, si que hai partes do sistema que necesitan ser traducidas polos propios tradutores de cada distribución.

OpenOffice e LibreOffice Tanto OpenOffice como o seu *fork*, LibreOffice, contan con equipos internacionalización aos que se lles consultou para recoller ideas para o novo programa.

Mozilla O proxecto Mozilla, autores do navegador Firefox e do xestor de correo electrónico Thunderbird, conta cun equipo de tradutores propio. Neste caso non usan ficheiros de tipo Gettext PO, senón tamén ficheiros XML.

Case en todos os correos enviados houbo xente que respondeu dando ideas sobre o que lles gustaría que se incorporase ó novo programa. Na seguinte sección podemos ver a maioría destas utilidades explicadas.

6.1.1. Resumo das peticións

Abrir e gardar ficheiros en diferentes formatos. O programa debería permitir abrir outros formatos de ficheiros aparte do formato Gettext po. Esta petición foi feita sobre todo por membros de equipos de tradutores que non son de GNOME. Existe unha biblioteca de nome *translate-toolkit* que facilita a conversión entre formatos.

Xestión de cabeceiras Que o programa modifique automaticamente as cabeceiras con metainformación que tanto o formato Gettext PO como outros tipos de formatos teñen.

Perfiles de usuarios Permitir ter diversos perfiles de usuarios para diferentes proxectos de tradución ou para diferentes linguaxes.

Vista de proxecto Os tradutores consideran moi interesante agrupar os ficheiros relacionados en proxectos e poder ter estatísticas de proxectos.

Buscar e buscar e reemplazar O programa debe permitir buscar e reemplazar palabras dentro do documento.

Dividir/Misturar ficheiros Os tradutores consideran que é interesante que en ficheiros grandes exista a posibilidade de dividir e volver a unir ficheiros de tradución para que así sexa posible que máis dunha persoa traballe no mesmo ficheiro a vez.

Medidas Económicas É interesante incorporar unha ferramenta que permita calcular o custe da tradución efectuada por un tradutor. Isto é especialmente útil cando falamos de tradutores profesionais e non amateur.

Resaltado da sintaxe O programa debe resaltar aqueles elementos da cadea que non son traducibles e pertencen o dominio das linguaxes de programación. Por exemplo na seguinte cadea "*Temos %i coches.*" o elemento *%i* debería estar resaltado xa que so parte do formatado da cadea por parte do programa.

Comunicación con servidor web Os tradutores consideran a posibilidade de que o programa permita certa comunicación con xestores de traducións en internet. No caso de GNOME o programa Damned Lies xestiona os ficheiros .po para todos os programas de GNOME e os tradutores deben baixar e subir os ficheiros dende esa plataforma.

Navegación dentro do documento A posibilidade de navegar a través das cadeas. Engadir a posibilidade de ir á seguinte cadea traducida, sen traducir ou con unha tradución difusa.

Memoria de tradución Engadir unha memoria de tradución que permita exportar e importar ficheiros en diferentes formatos. Ter varias memorias de tradución con prioridade entre elas, engadir a posibilidade de editar a memoria de tradución e acceder a diversos servidores que xestionan memorias de tradución online como poden ser Trobador, amaGama, Open-tran ou Transvision.

Glosario Engadir a posibilidade de consultar a tradución de termos contra ficheiros unha base de datos local ou contra un servidor de glosario como pode ser Terminator.

Busca de termos Que o programa permita a busca nun dicionario tanto local como en internet. Cítanse dicionarios en internet como o que proporciona Wikipedia ou a Universidade de Santiago de Compostela para o galego.

Previsualización das traducións Moitas das interfaces que se fan actualmente empregan ferramentas de cuarta xeración que permitirían xerar a interface coas traducións. No caso de GNOME, Glade é o programa que permite a creación de interfaces e xa existe unha ferramenta online que permite a previsualización de traducións de nome Deckard. O que se pide é que o programa incorpore esa posibilidade dende a súa interface.

Programa controlable totalmente a través do teclado O programa en xeral, pero sobre todo a interface de edición de ficheiros, debe ser manexable totalmente a través do teclado para mellorar a produtividade dos tradutores. Os tradutores piden tamén a posibilidade de que se permita personalizar os atallos de teclado.

Tradución doutras linguaxes Hai linguas como o galego e o portugués que se parecen moito polo que, as veces, resulta moi útil poder, en vez de partir de cero, coller unha tradución dun idioma semellante e editala.

Comprobacións Os tradutores resaltaron a utilidade de que a ferramenta comprobe certos parámetros para comprobar a calidade da tradución. Entre outros:

- **Ortografía e Gramática.** O programa avisará se a cadea traducida ten erros tanto ortográficos como gramaticais.
- **Coherencia terminolóxica.** O programa avisará ó usuario cando este empregue unha tradución dun termo diferente o que se vén empregando no resto do ficheiro.
- **Etiquetado XML e marcas de formato.** O programa avisará ó usuario se faltan ou están mal escritos as diferentes etiquetas XML ou marcas de formato.

Multiplataforma A aplicación debe estar dispoñible para varios sistemas operativos (BSD, Windows, MAC OS, etc.).

Tradución automática O aplicativo debe incorporar mecanismos de tradución automática empregando ferramentas como Google Translator, Bing Translator, Opentrad ou Apertium.

Estatísticas Débese amosar estatísticas tanto a nivel de ficheiro como de proxecto do número de cadeas ou palabras traducidas, sen traducir ou difusas.

6.2. Análise doutros aplicativos do mercado

Para facer o novo produto software para a tradución tamén consultamos outras ferramentas similares existentes no mercado para intentar imitar as súas vantaxes e evitar as súas deficiencias. O resultado desta análise pódese consultar na Sección 2.2.

6.3. Requisitos do aplicativo

Con toda a información obtida realizamos unha lista de casos de uso que debe cumprir o programa.

- **Abrir ficheiro.** O programa debe ser capaz de abrir ficheiros PO pero tamén ter un deseño extensible que permita abrir outros formatos.
- **Gardar ficheiro.** Debemos poder gardar ficheiros PO pero tamén ter un deseño extensible que permite gardar noutros formatos.
- **Amosar ficheiro.** Debemos amosar o contido dos ficheiros e dicir, as cadeas e as estatísticas deste ficheiro.
- **Editar ficheiro.** Debemos permitir editar o contido dos ficheiros.
- **Buscar cadeas.** A aplicación permitirá buscar entre as cadeas do ficheiro.
- **Navegar polas cadeas.** Permitiremos navegar polas cadeas do ficheiro podendo avanzar entre as cadeas traducidas, sen traducir, etc.
- **Xestionar perfiles.** Permitiremos xestionar diferentes perfiles para os tradutores.
- **Obter pistas.** Amosaremos ao usuario pistas que lle indiquen que está a facer algo mal.
- **Obter suxerencias.** Debemos amosar ao tradutor diferentes alternativas de traducións.

Na figura 6.1 podemos ver o Diagrama UML de casos de uso para este programa.

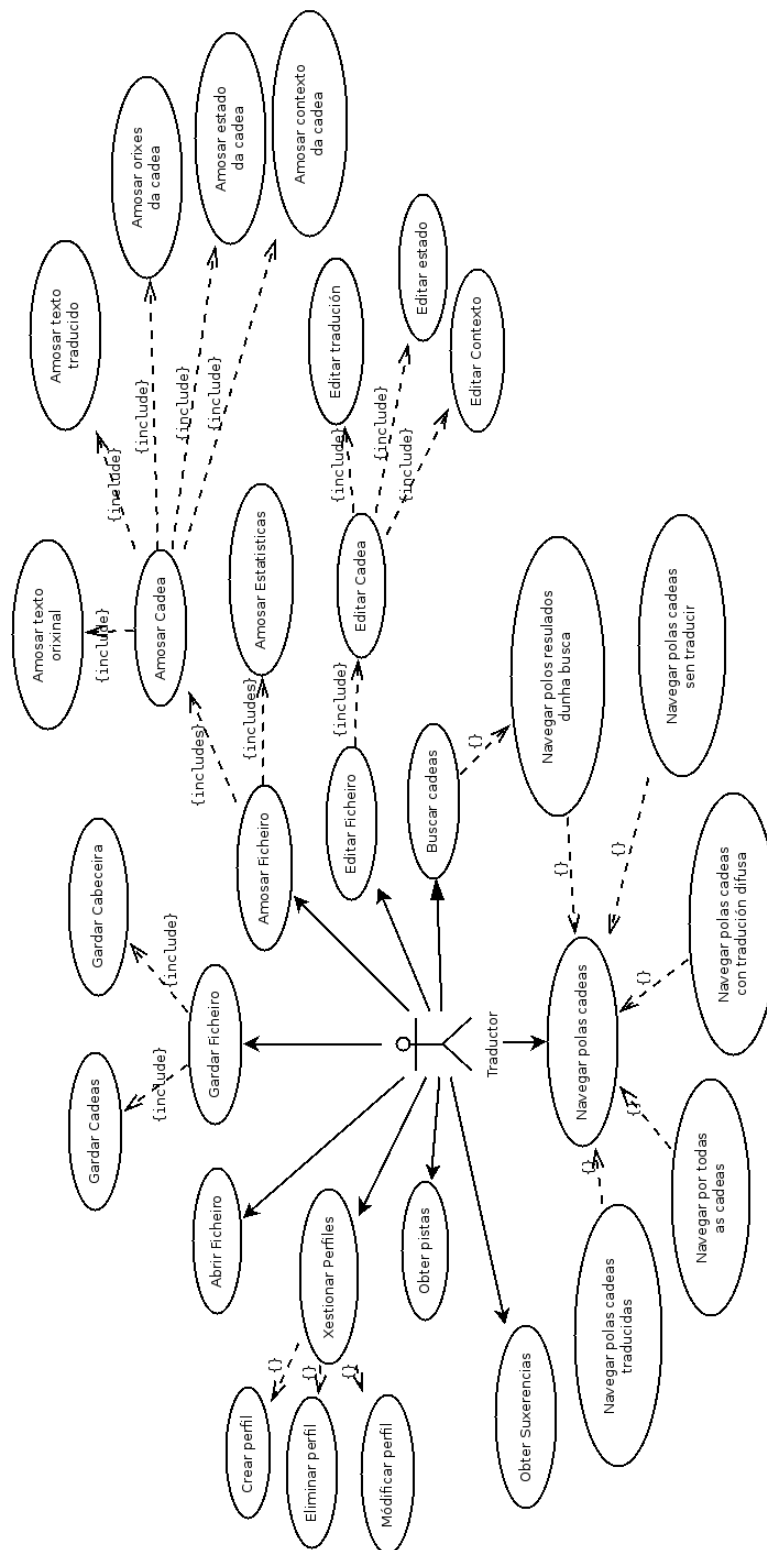


Figura 6.1: Diagrama UML de casos de uso do aplicativo

Capítulo 7

Deseño e Implementación

Neste capítulo veremos detalles do deseño e implementación de diferentes partes do programa.

7.1. Módulo de Ficheiros

O módulo de xestión de ficheiros é un do máis importantes e complexos de todo o programa, o cal é lóxico tendo en conta que o programa é un editor de ficheiros.

En canto ao deseño, ó principal obxectivo é a estensibilidade do mesmo pois, aínda que o programa está centrado na edición de ficheiros PO e son estes os únicos soportados na actualidade, preténdese que o programa sexa capaz de soportar varios tipos de ficheiros nun futuro. Na figura 7.1 pódese ver o diagrama de clases deste módulo.

7.1.1. Implementación Xenérica

A implementación da clase `File` contén propiedades para conseguir información sobre o nome e *path* onde está dito ficheiro, ademais garda estatísticas sobre o número de mensaxes traducidos, sen traducir ou con tradución difusa. Estas estatísticas actualízanse cada vez que se engade ou elimina unha cadea ou cada vez que esta se modifica. Tamén contén un valor booleano que permite saber se o ficheiro foi modificado. Aparte de métodos para engadir e eliminar cadeas, e para conseguir e modificar os metadatos do ficheiro esta clase ten un método para gardar e recuperar o ficheiro. O método para gardar (*save*) emprega o patrón *Template Method* [GHJV95] o cal nos permite actualizar o estado do ficheiro a non modificado.

Un ficheiro contén instancias de mensaxes (*Message*). As mensaxes teñen como propiedades un estado, orixes, e consellos. A API prové métodos para conseguir e modificar

tanto as cadeas orixinais como as traducións, na súa forma en singular ou nalgunha das formas plurais. A implementación do método para modificar unha tradución tamén emprega o patrón Template Method para actualizar o estado da mensaxe. Esta clase tamén ten métodos para engadir e eliminar consellos e para obter o contexto da mensaxe.

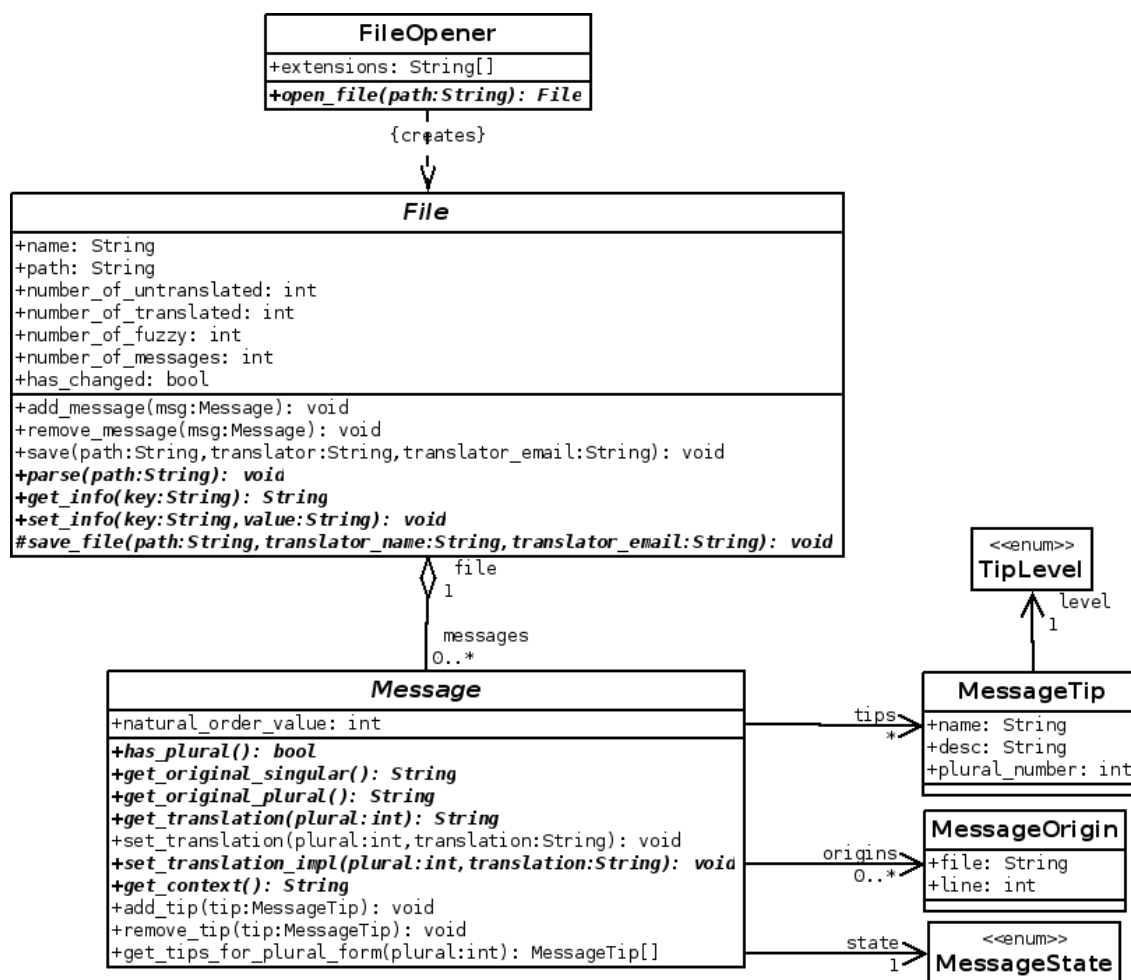


Figura 7.1: Diagrama de Clases do módulo ficheiros

Ademais a clase FileOpener recolle un método para crear ficheiros e un conxunto de extensións que se poden abrir con este FileOpener.

7.1.1.1. Consellos (Tips)

Os consellos son a solución que damos para aportar información ao usuario sobre a tradución que está a realizar. Algunhas das cousas que poden axudar a indicar esta característica son se a tradución está a ser demasiado longa, se hai algunha palabra que está mal escrita na tradución ou se non emprega a terminoloxía adecuada.

Cada consello ten un nome que debe ser xenérico a cada clase de consello, unha descrición

que ten que explicar o problema con detalle, un nivel que indica a gravidade do consello e unha referencia a forma plural a que corresponde dito consello. Ademais, pode conter unha ou máis referencias á localización exacta do problema que permitirá destacala na interface gráfica.

A creación de consellos farase ó modificarse unha cadea e farase a través de plugins.

7.1.1.2. Pistas (Hints)

As pistas amosaranlle ó usuario posibles traducións ou aproximacións ás traducións. Estas pistas poden ser obtidas de memorias de traducións, de traducións do mesmo ficheiro noutra linguaxe, ou da tradución directa, por exemplo.

Cada instancia dunha pista (`Hint`) contén a tradución suxerida, unha cadea que identifica a orixe de dita pista e un valor que indica a precisión de dita suxerencia.

De igual forma que no caso dos consellos a creación de pistas correrá ó cargo de plugins creados a tal propósito. Neste caso actualizaranse as pistas de cada mensaxe ao seleccionalo mesmo na interface.

7.1.2. Ficheiro PO

A implementación específica para ficheiros PO (Figura 7.2) estende as clases `File`, `FileOpener` e `Message` abstractas para implementar os métodos e permitir empregar ficheiros PO.

Para a análise e a actualización dos ficheiros PO empregaremos a biblioteca `gettext-po`. No momento no que se iniciou a implementación deste módulo, non existía implementación desta librería en Vala polo que tivemos que crear uns *bindings* para poder empregala.

Tanto a clase `PoFile` como a clase `PoMessage` delegan a maior parte dos seus métodos nas instancias das clases dos bindings da biblioteca `GettextPo` `File` e `Message` respectivamente. Desta forma faise un claro uso do patrón *Adapter* [GHJV95].

De forma adicional, a clase `PoFile` contén unha instancia da clase `PoHeader`. Esta clase que estende a `PoMessage` ten os metadatos que se atopan nos ficheiros PO na tradución da cadea baleira. Ademais, esta, dispón de métodos para obter e modificar metadatos e para actualizar os datos dos autores das traducións de dito ficheiro. A clase `PoFile` delega nesta clase á hora de conseguir e modificar metadatos e tamén cando garda un ficheiro.

A clase `PoFileOpener` só é capaz de abrir ficheiros con extensión PO e simplemente emprega o método `parse` da clase `ficheiro` para crear unha nova instancia.

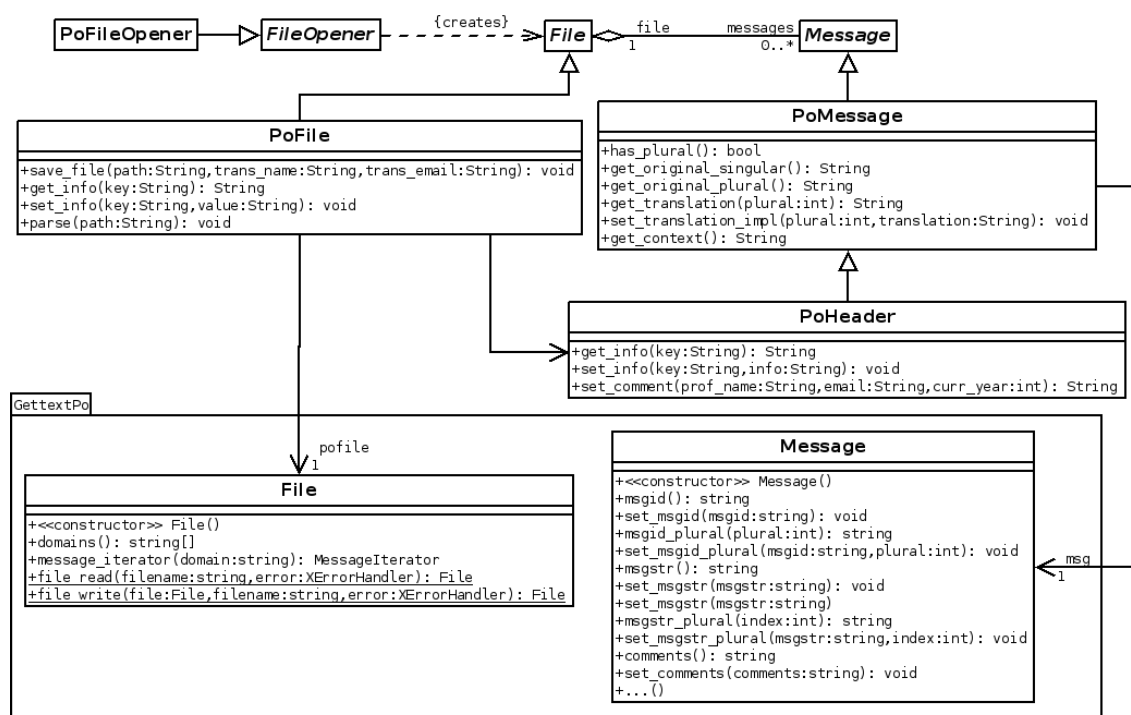


Figura 7.2: Diagrama de Clases do ficheiro PO

7.1.2.1. Implementación dos bindings

Como xa mencionamos vímonos na obriga de crear nós os bindings para a biblioteca gettext-po. Vala é unha linguaxe deseñada para permitir o fácil acceso a outras bibliotecas escritas en C, especialmente se se trata de bibliotecas baseadas en GObject. Despois de todo Vala emprega C como linguaxe intermedio. A biblioteca gettext-po non é unha biblioteca baseada en GObject polo que a creación destes bindings é un pouco máis complicado.

Para poder empregar unha biblioteca escrita en C no noso programa escrito en Vala só temos que crear un ficheiro con extensión VAPI que conteña en sintaxe Vala as clases da biblioteca engadindo unhas etiquetas que permitan a súa tradución ó código C correcto. No Fragmento de código 7.1 podemos ver parte dos bindings creados para a biblioteca gettextpo.

Fragmento de Código 7.1: Bindings da biblioteca GettextPo

```

[CCode (cprefix = "Po", lower_case_cprefix = "po_")]
namespace GettextPo {

    [CCode(cheader_filename = "gettext-po.h", cname="struct po_file",
        free_function="po_file_free")]
    [Compact]

```

```

public class File {

    [CCode (CCode = "po_file_create")]
    public File();

    [CCode (array_length = false, array_null_terminated = true, cname="
        po_file_domains")]
    public unowned string[] domains ();

    [CCode (cname="po_file_write")]
    public static unowned GettextPo.File file_write (GettextPo.File
        file,

                                string filename,
                                XErrorHandler handler);

    [...]
}

[CCode(cheader_filename = "gettext-po.h", cname="struct po_message")]
[Compact]
public class Message {

    [CCode (cname="po_message_create")]
    public Message();
    public unowned string msgid ();
    public unowned string? msgid_plural ();
    [...]
}

```

Como podemos ver, o traballo practicamente límtase a establecer o parámetro `cname` en cada clase e método. Hai que ter en conta que non sempre é necesario especificar este parámetro pois as bibliotecas empregan case sempre unhas normas de nomeado que fan que usen primeiro o nome da biblioteca, despois o nome da clase e logo o nome do método separado por barras baixas. Desta forma no método da biblioteca `po_message_msgid()`, `po_` corresponde o nome da biblioteca, `message_` ao nome da clase e `msgid` ao nome do método. Os bingings de Vala empregan estas normas para nomear os métodos polo que, en ocasións, non é necesario especificar o parámetro `cname`. Isto sucede, como se pode ver no Fragmento de código 7.1, no caso da clase `Mensaxe` (`Message`).

Ademais, é necesario especificar o nome do ficheiro cabeceira que se emprega e no caso de que o tipo de retorno sexa un array temos que especificar máis parámetros, como acontece no método `domains()` da clase `File`. Por último, é importe especificar a pertenza de cada valor e os métodos correctos para liberar as instancias, desta forma evitaremos que haxa perdas de memoria e fallos de segmentación.

7.2. Módulo de Linguaxes

O módulo de linguaxes xestiona os linguaxes e formas plurais existentes. Contén dúas clases, a clase `Language` e a clase `PluralForm`. Estas dúas clases provén un método estático para obter as instancias existentes. Estas instancias créanse a primeira vez que se carga a clase consultando unhas bases de datos que consisten nuns ficheiros JSON¹.

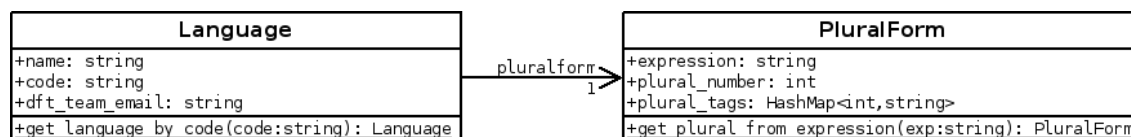


Figura 7.3: Diagrama de Clases do módulo de Linguaxes

Como podemos ver na Figura 7.3, en cada instancia dunha linguaxe contén o nome da linguaxe, o seu código empregando a norma ISO 639-1, a súa forma plural e o email do equipo de tradución por defecto. Engadimos o idioma do equipo de tradución por defecto para autocompletar este campo no perfil do usuario. No Fragmento de Código 7.2 podemos ver un fragmento do ficheiro JSON contendo as linguaxes.

Fragmento de Código 7.2: Fragmento da Base de Datos de Linguaxes

```

{
  "languages" : [
    [...]
    {
      "code" : "es",
      "name" : "Spanish; Castilian",
      "pluralform" : "nplurals=2; plural=(n != 1);",
      "default-team-email": "gnome-es-list@gnome.org"
    },
    [...]
  ]
}
  
```

En canto á clase `PluralForm`, esta clase inclúe o número de plurais, a expresión desa forma plural e un conxunto de etiquetas. Estas etiquetas pretenden facilitar ó usuario a identificación de que número corresponde a cada forma plural. No Fragmento de Código 7.3 podemos ver unha parte da base de datos de formas plurais.

Fragmento de Código 7.3: Fragmento da Base de Datos de Plurais

```

{
  "forms" : [
    {
  
```

¹JSON


```
"expression" : "nplurals=2; plural=(n > 1);",
"number_of_plurals" : 2,
"tags" : [
  {
    "number" : 0,
    "tag" : "Equal to 0 or 1"
  },
  {
    "number" : 1,
    "tag" : "Greater than 1"
  }
]
},
[...]
```

7.3. Interface Gráfica

A interface gráfica é unha das partes á que máis tempo lle dedicamos neste proxecto. O obxectivo dende o principio foi construír algo simple pero potente que permitira ó usuario editar os ficheiros PO de forma sinxela, pero que fose capaz de aportarlle moita información que lle axudase a facer a tradución.

7.3.1. Evolución

Durante a execución do proxecto probamos diferentes formas da interface gráfica ata chegar ó resultado actual. Estas versións pretenden imitar programas existentes e obedecen xeralmente a conversacións con tradutores.

7.3.1.1. Primeira Versión: moi semellante a GTranslator

Para comezar a traballar, e tras amosar varios mockups nos reportes recibindo feedback sobre eles, decidimos facer unha interface de usuario moi parecida a de GTranslator. Esta interface inclúe bloques para a lista de mensaxes, editar ditos mensaxes e amosar o contexto. Estes bloques pódense mover por toda a interface e incluso separar da mesma xa que estamos empregando a biblioteca GNOME Docking Library. Na Figura 7.4 podemos ver, o aspecto desta primeira versión da interface.

Como podemos ver a interface contén un barra de ferramentas que permite abrir ficheiros, gardalos, desfacer e refacer cambios, buscar no documento e ver as preferencias.

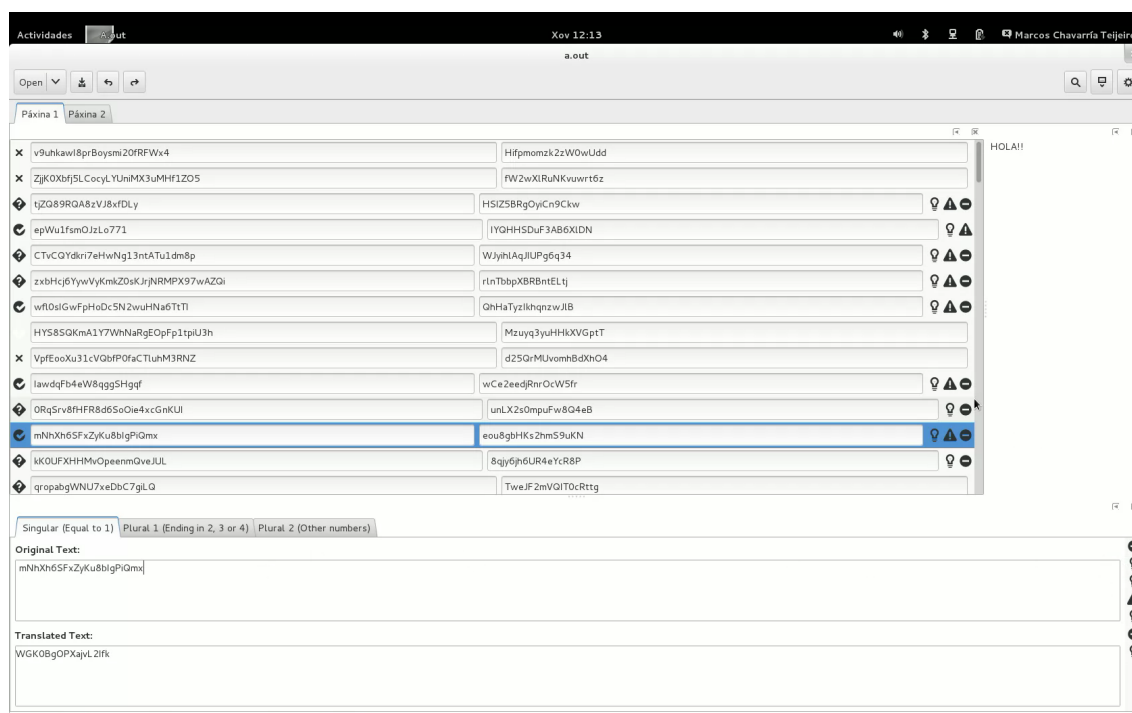


Figura 7.4: Primeira versión da interface de usuario

Permite abrir diversos ficheiros en varias lapelas. En cada unha das lapelas podemos ver a lista de mensaxes e o widget de edición.

En cada mensaxe da lista, aparte das cadeas orixinal e traducida, podemos ver o estado da cadea e se está ten consellos (Tips) activos e de o nivel de estes. Por outro lado, no widget de edición podemos ver unha lapela por cada forma plural que podemos editar e unha lista vertical con iconas cos consellos. Ao pasar o rato polo consello veremos a súa descrición.

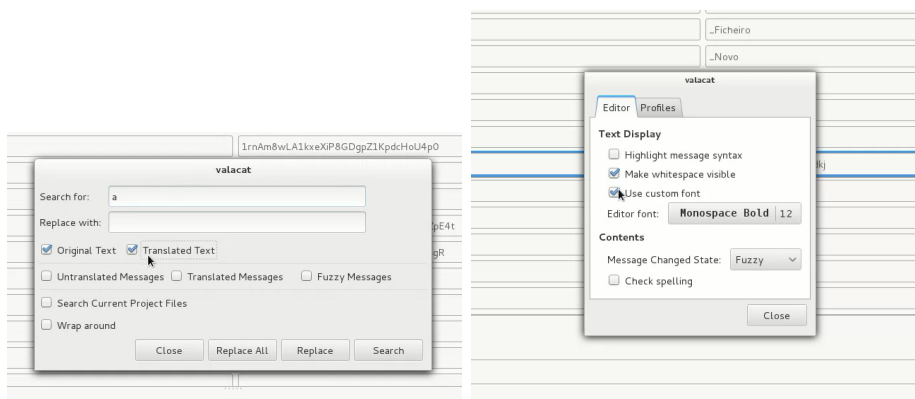


Figura 7.5: Diálogos da primeira versión da interface de usuario

Como podemos ver na Figura 7.5 tanto á hora de xerar unha busca como para ver as

preferencias, esta interface empregará uns diálogos non modais.

En conversacións en IRC con tradutores e anteriores mantedores de GTranslator chegamos á conclusión de que o uso de GNOME Docking Library era unha mala idea, pois significaba un erro de deseño xa que, se o usuario tiña que modificar o aspecto da interface, era debido a que o deseño non era correcto. Esta biblioteca, ademais, ten bastantes fallos polo que non era recomendable usala.

Ademais, durante a GUADEC, comentáronos da existencia dun widget moito mellor para xestionar a buscas, consistente nunha barra horizontal que se desplegaba cando a busca estaba activa.

Por outro lado, o resultado deste modelo de interface tampouco era demasiado satisfactorio polo que intentamos probar cunha versión máis próxima ó programa Virtaal.

7.3.1.2. Segunda Versión: semellante a Virtaal

Para lograr un aspecto máis parecido ó da aplicación Virtaal fusionamos o widget de edición e o widget de lista de cadeas.

A aplicación segue mantendo as lapelas que permiten abrir máis dun ficheiro ó mesmo tempo, pero eliminamos a posibilidade de personalizar a interface. En lugar diso crearemos unha interface con dúas columnas, na primeira teremos o widget para listar e editar as cadeas e na segunda un widget para ver o contexto e outro para ver as pistas.

Modificamos a barra de ferramentas que converteremos nun `GTK.HeaderBar` que permite fusionar esta barra de ferramentas co marco da ventá. O uso de `HeaderBars` son un patrón de deseño recomendado por GNOME na súa Guía de Interfaces Humanas (*HIG*)[\[GNOb\]](#). Na Figura 7.6 podemos ver como quedou a nova inteface.

Otro dos aspectos da interface no que se traballou foi na eliminación do diálogo de nova busca. Como se pode ver na Figura 7.7, ó activar a busca, desplégase unha barra horizontal que permite introducir termos para buscar.

Para construír esta barra empregamos o widget `GTK.SearchBar`. Creamos un modo de busca normal e un modo avanzado. O modo normal amosa unha caixa de texto para introducir a busca e frechas para avanzar na busca, e o modo avanzado engade unha segunda entrada de texto e un botón para a opción de buscar e remplazar e permite seleccionar que cadeas queremos incluír na busca.

Esta interface, aínda que máis pulida cá anterior, presentaba problemas á hora de editar longas cadeas de texto. Ademais, non deixaba demasiado espazo para engadir información sobre as cadeas. Debido a isto escribimos un artigo con algúns mockups baseados en algúns deseños feitos para GTranslator. Durante o GSoC 2014 creamos a que está pensada como

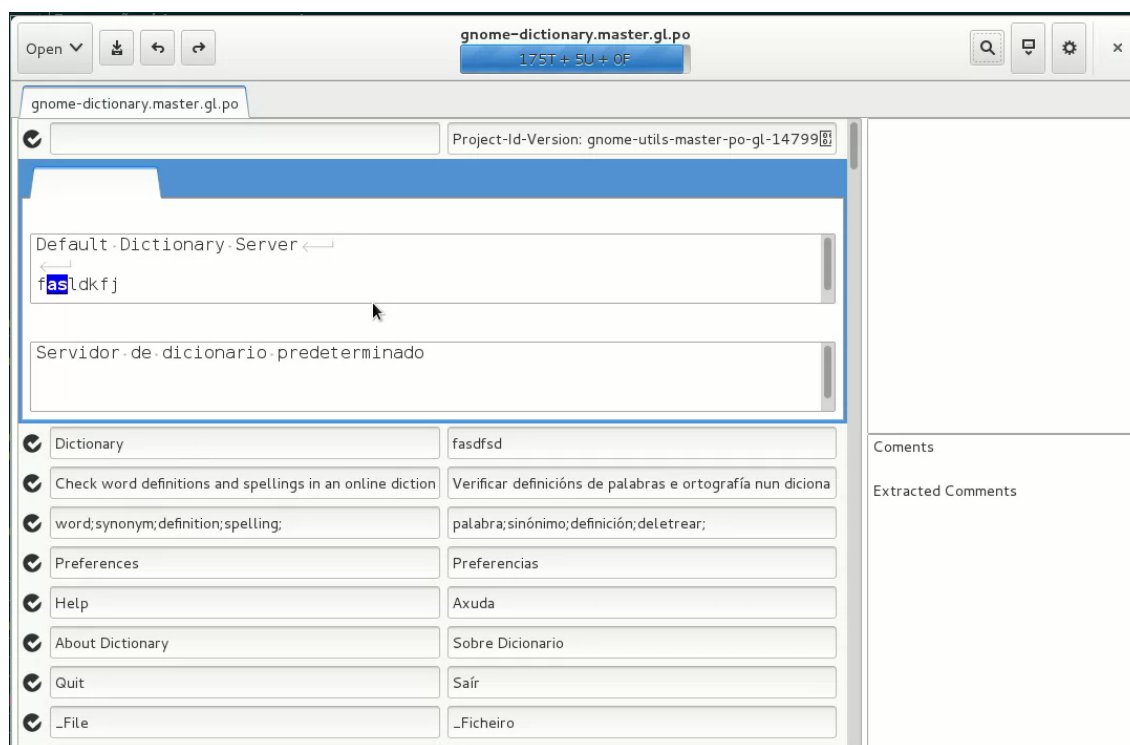


Figura 7.6: Segunda versión da interface de usuario

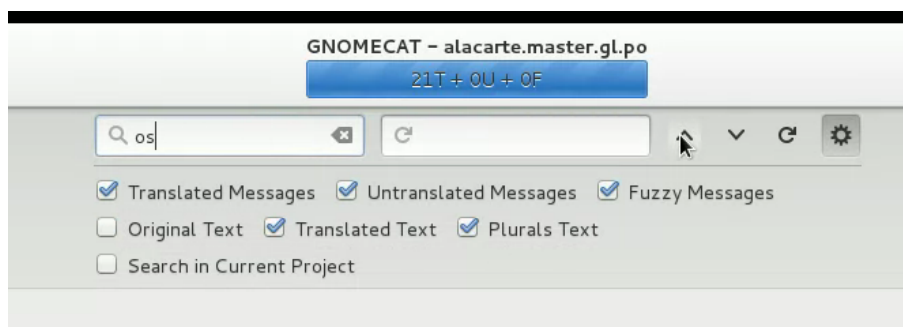


Figura 7.7: Barra de busca

a interface definitiva do programa.

7.3.1.3. Terceira Versión

Para a realización desta terceira versión e definitiva intentamos poñernos en contacto cos deseñadores de GNOME para que nos axuden co deseño da nova interface. Ante a baixa participación neste sentido por parte dos deseñadores, usamos uns deseños feitos por Daniel Korostil para un redeseño de GTranslator que nunca chegou a implementarse. Baseándonos nestes mockups e en ideas propias creamos un novo concepto de interface que ten como idea principal a de intentar eliminar onde sexa posible calquera diálogo externo

ó programa.

A ventá pasa a compoñerse dunha barra de ferramentas integrada nun `GTK.HeaderBar`, unha barra de notificacións e un conxunto de paneis.

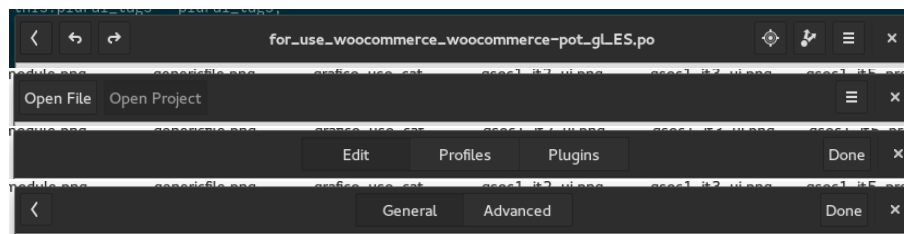


Figura 7.8: Diferentes Barras de Ferramentas

A barra de ferramentas é dinámica e os botóns que amosa dependen de que pantalla se estea mostrando no programa. Esta é unha das suxerencias que a Guía de Interfaces Humanas de GNOME fai.

Para lograr isto dentro do `GTK.HeaderBar` introducimos un widget denominado `GTK.Notebook`. Este widget é o mesmo que se empregaba en versións anterior da interface para ter diferentes ficheiros abertos en outras tantas lapelas, pero empregando a opción que este incorpora de ocultar as lapelas. Desta forma, cada páxina do `GTK.Notebook` é unha das opcións da barra de ferramentas. Na Figura 7.8 podemos ver algúnnhas das diferentes formas da barra de ferramentas.

Os botóns da barra de tarefas xeran accións de GTK que serán manexadas pola ventá. A ventá manexará estas accións delegando no panel que esté activo nese momento facendo uso do patrón *State*[GHJV95]. No Fragmento de Código 7.4 pódese ver un exemplo da implementación dun dos manexadores.

Fragmento de Código 7.4: Implemenación do manexador da acción gardar

```
private void on_edit_save ()
{
    (window_panels.get_nth_page (window_panels.page) as Panel).on_edit_save
    (this);
}
```

Tamén empregamos un `GTK.Notebook` para cambiar entre paneis. Os paneis son as diferentes pantallas que se poden ver no programa. Desta forma, temos un panel para abrir ficheiros, un panel para as preferencias e un panel para editar os ficheiros entre outros. Na implementación cada panel ten que implementar a interface `Panel`. Esta interface pide que se defina un tipo de barra de tarefas e dá unha implementación xenérica ós manexadores das accións da ventá. Cada panel será libre de aportar unha implementación específica destas accións. Na seguinte sección trataranse en detalle cada un dos paneis da interface.

En canto á barra de notificación, é a resposta que damos á necesidade de avisar ó usuario de certos eventos do programa. Por exemplo, avisamos ó usuario que está introducindo unha busca, de que non existe ningunha cadea que cumpra o criterio ou, se este está avanzando polas cadeas, que xa chegou á última. Para implementar esta parte fixemos uso do widget `GTK.InfoBar`. Amosamos a notificación durante unha certa cantidade de tempo e despois ocultámolo. Na Figura 7.9 vemos unha captura da barra de notificación.

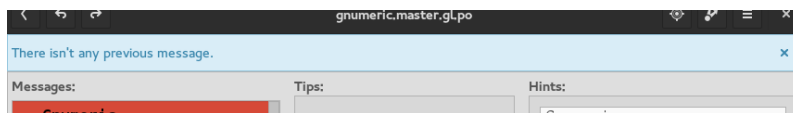


Figura 7.9: Barra de Notificacións

Por último, tamén engadimos un menú de aplicación que permite acceder ó panel de preferencias, pechar o programa e iniciar un dialogo para obter información sobre quen fixo GNOME CAT e a súa licenza, entre outros. O uso de menús de aplicación en GNOME é unha recomendación da súa Guía de Interfaces Humanas. Pódese acceder ó menú de aplicación dende GNOME facendo click na icona da aplicación na barra superior do entorno de ventás.

7.3.2. Paneis

Os paneis empregados no programa son os seguintes:

7.3.2.1. Panel de Benvida

O panel de benvida amósaselle ó usuario cando non hai ningún perfil dispoñible no programa. Isto sucede se é a primeira vez que se inicia o programa ou se de forma externa se borrou a información dos perfís.

Como podemos ver na Figura 7.10, amósaselle unha mensaxe de información ó usuario permíndolle acceder ó panel de creación do primeiro perfil.

A barra de ferramentas non inclúe ningún botón xa que a interface só permite avanzar cara ó panel de creación do primeiro perfil ou pechar o programa.

7.3.2.2. Panel de ficheiros abertos

Este panel, como o seu nome indica, amosa unha lista de ficheiros abertos. É o que primeiro se amosa cando abrimos o programa e xa temos perfil creado. Ademais, permite abrir novos ficheiros.

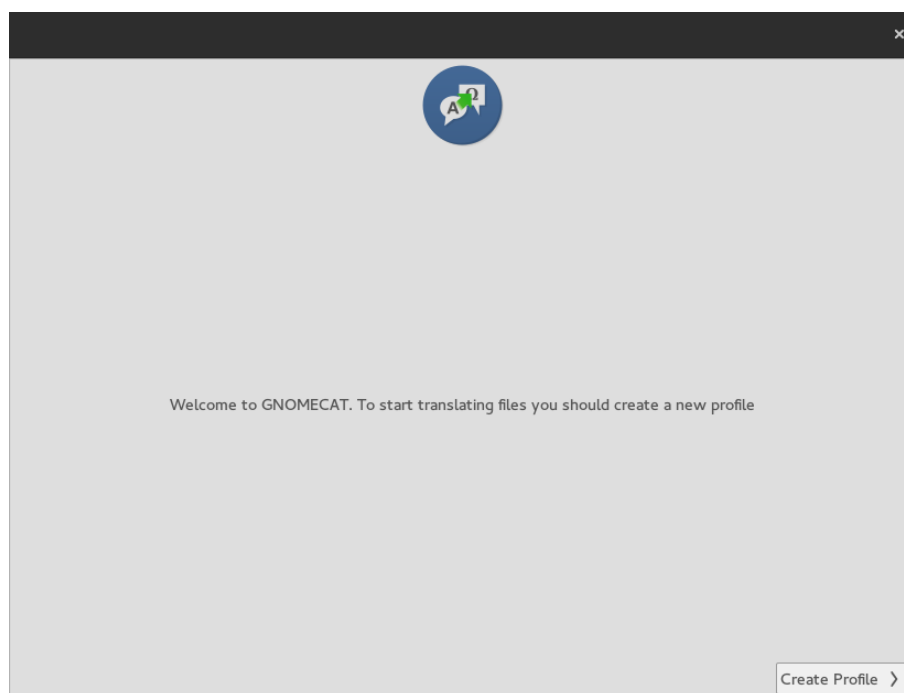


Figura 7.10: Panel de Benvida

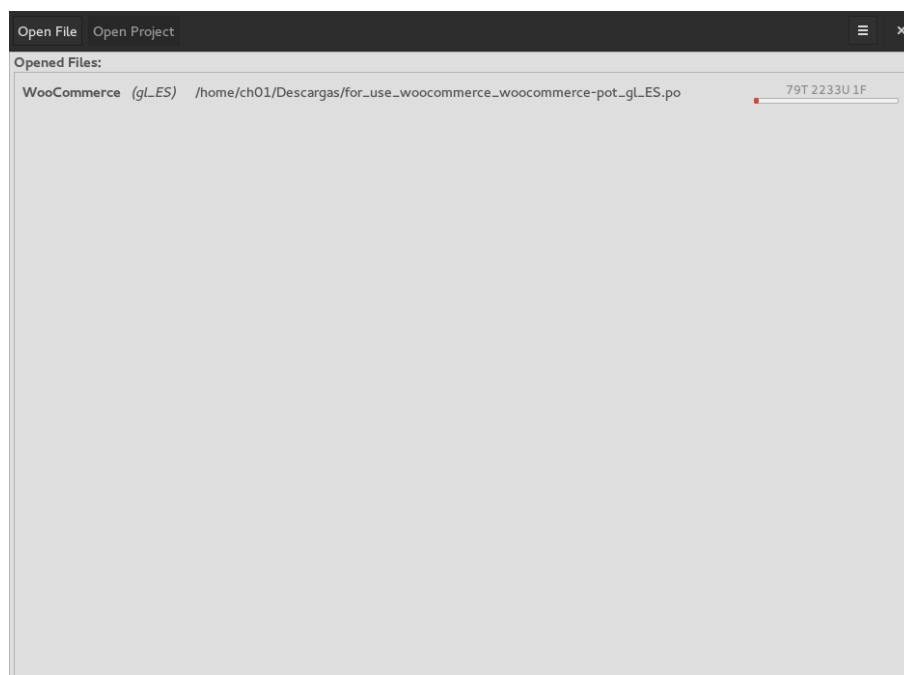


Figura 7.11: Panel de Ficheiros Abertos

Na Figura 7.11 pódese ver este panel. Para cada un dos ficheiros abertos amosamos o nome do proxecto segundo aparece nos metadatos do ficheiro PO; a linguaxe deste ficheiro, o path absoluto onde este se atopa e as estatísticas de tradución.

Ademais de permitir abrir un novo ficheiro, a barra de ferramentas tamén permite acceder ó panel de preferencias.

7.3.2.3. Panel de abrir ficheiro

O panel de abrir ficheiros amosa os ficheiros abertos de forma recente e permite abrir novos ficheiros.

Para poder implementar a lista de ficheiros recentes tivemos que crear un widget personalizado. O novo widget emprega a clase `GTK.RecentManager`. Esta clase, que funciona empregando o patrón *Singleton* [GHJV95], controla os ficheiros que se abren en todo o sistema. O widget creado conéctase á instancia de `RecentManager` para modificar a lista de ficheiros recentes cada vez que un ficheiro compatible é aberto.

A hora de amosar a información sobre os ficheiros amosamos os mesmos campos que no caso do panel de ficheiros abertos. Isto é debido a que nos dous casos empregamos un `GTK.ListBox` ao que lle introducimos un widget `PoFileRow` por cada un dos ficheiros.

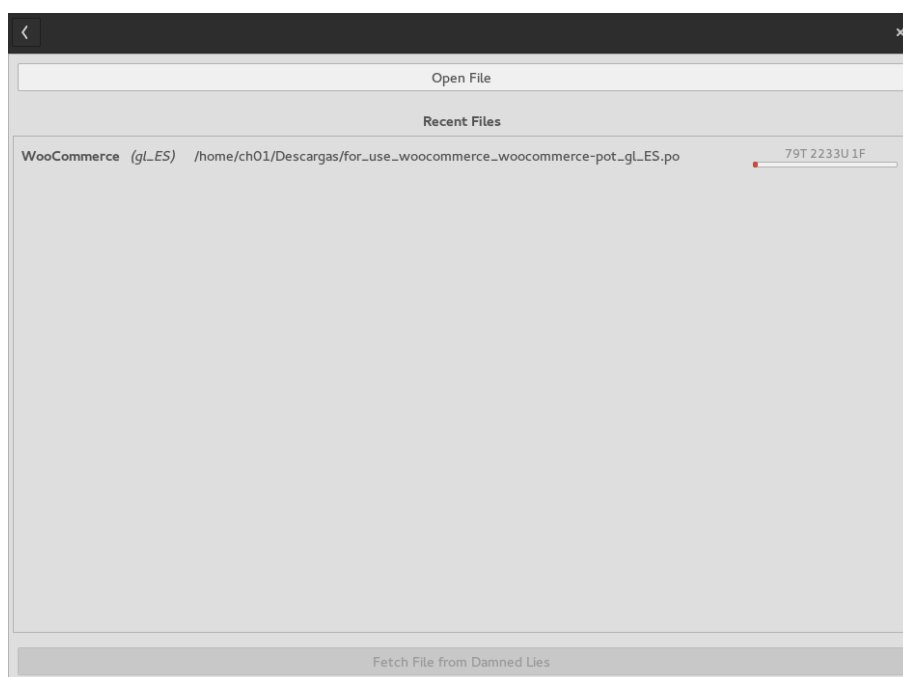


Figura 7.12: Panel de Abrir Ficheiro

Como podemos ver na Figura 7.12 a barra de ferramentas permite volver a último panel que neste caso sempre será o panel de ficheiros abertos.

7.3.2.4. Panel de edición

O panel de edición é o máis importante deste programa e ao que lle dedicamos máis tempo. Como os seu nome indica permite editar os ficheiros. Está composto de tres columnas, unha lista de mensaxes, unha lista de pistas e unha columna central que contén os consellos, entradas para editar as cadeas e o contexto. As columnas que teñen a lista de mensaxes e de pistas teñen un ancho fixo e a columna central aproveitará todo o espazo restante.

Como se pode comprobar nesta versión da interface, as caixas de edición de cadeas son considerablemente máis grandes que nas outras versións. Na Figura 7.13 podemos ver unha captura deste panel.

A parte que máis traballo nos supuxo foi implementar a lista de mensaxes. Inicialmente deseñamos este *widget* como un `GTK.ListBox` coas súas columnas. Este é un widget engadido na versión 3.10 de GTK+ e que empregamos moito no noso programa. A vantaxe de empregalo consiste en que é moi sinxelo personalizar cada columna e aínda que non o usamos no noso programa existiría a posibilidade de engadir columnas de diferentes tipos. O problema que atopamos cando construímos o widget de listar mensaxes con un `ListBox` é que este funcionaba realmente mal cando o ficheiro tiña moitas cadeas. Primeiro pensamos que a lentitude debíase ao alto consumo de memoria cando cargabamos un destes ficheiros pero máis tarde falando con algún desenvolvedor de GTK+, démonos conta de que este widget non estaba deseñado para soportar tantas columnas e que debíamos empregar unha alternativa.

A alternativa escollida foi a de empregar un `GTK.TreeView`. Este widget si que está deseñado para soportar gran cantidade de columnas pero a personalización de cada columna é moito máis complicada. Para facelo tivemos que implementar un `GTK.CellRenderer` para o renderizado das columnas. Nesta clase especificamos a man onde se debuxa cada elemento da columna e que tamaño ten.

A clase `TreeView` incorpora métodos que permiten ordenar e filtrar de forma sinxela as columnas que se amosan neste widget. Isto permitiunos engadir estas opcións a nosa interface. Como se pode ver na figura anterior debaixo da lista de mensaxe temos unha barra coas estatísticas do documento e dous botóns un permite filtrar os resultados ocultando ou amosando as cadeas traducidas, sen traducir ou con tradución difusa e o outro permite ordenar as mensaxes amosando primeiro as mensaxes sen traducir, despois as mensaxes con tradución difusa e por último as mensaxes traducidas.

En canto ao widget de lista de pistas, neste caso si que empregamos un `GTK.ListBox` xa que o número de columnas nunca vai ser demasiado alto. En cada unha das columnas amosamos a información que temos de cada pista (`Hint`). Isto é, a cadea suxerida, a

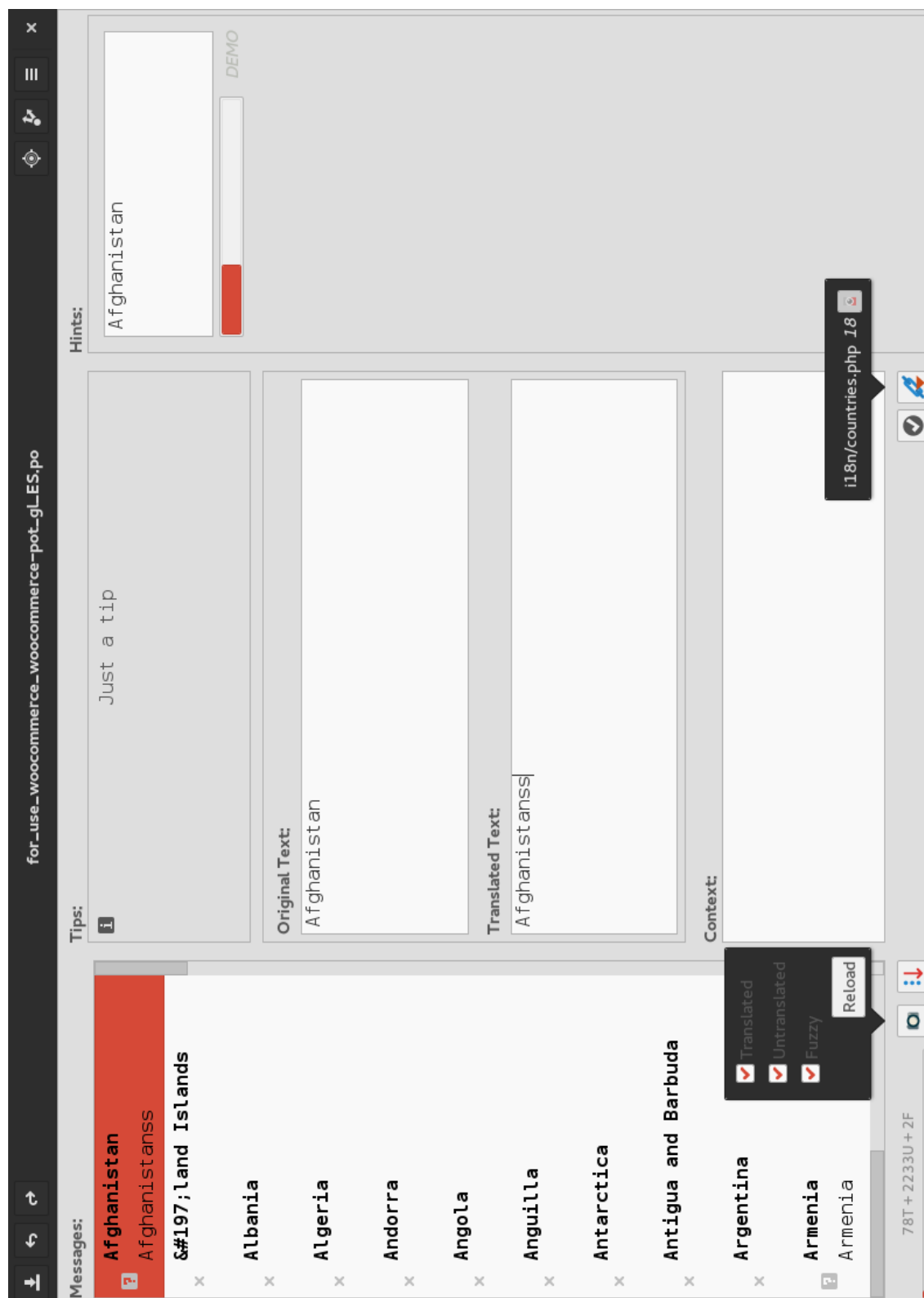


Figura 7.13: Panel de Edición de Ficheiros

orixe desta pista e a súa precisión. Para amosar a precisión empregamos unha barra de progreso. Ao facer dobre click nos Hints o texto a traducir substituirase polo pista. Ademais incorporamos atallos de teclado para as 4 primeiras pista de forma que se pulsamos as teclas Ctrl e un dos catro primeiros números activarase a correspondente pista.

Dentro da columna central a lista de consellos (Tip) atopase na parte superior. Esta localización esta pensada para que o tradutor non teña que apartar demasiado a mirada do cadros de edición de mensaxes.

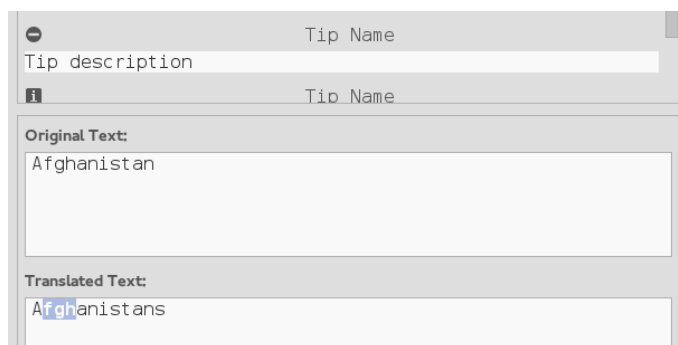


Figura 7.14: Selección dun consello

Desta forma cada columna amosará o nome do Tip e o seu nivel de gravidade. O nome do consello debe ser suficiente para que o usuario se de conta do erro ao que se refire o consello. En caso de non ser suficiente o usuario pode premer encima do consello nese caso amosarase a descrición do consello e resaltarase a parte do texto á que refire o mesmo. Na Figura 7.14 podes mover ver unha captura desta situación.

Os cadros que permiten ver a cadea orixinal están construídos co widget `GTK.SourceView` este widget que non é parte da librería GTK estende o widget `GTK.TextView` e permite facer de forma sinxela, desfacer e refacer accións, resaltado de sintaxe e resaltado de espacios en branco entre outros. Na Figura 7.15 podemos ver unha cadea con resaltado de sintaxe e de espazos en branco.

No caso de que a mensaxes a traducir teña plurais amosaremos cada plural nunha lapela. Como se pode ver na Figura 7.15, no texto que amosa a lapela, aparte do número de plural amosaremos unha etiqueta explicando a que números se refire ese plural.

Debaixo do cadro de texto que amosa o contexto, temos botóns para ver as orixes da mensaxe actual e para cambiar o estado da mensaxe entre traducida e difusa. Para ver as orixes tamén se usa un widget `GTK.Popover`.

A barra de tarefas ten botóns para volver a lista de ficheiros, gardar o ficheiro actual, facer e desfacer as accións, buscar, navegar a través do documento e ver as preferencias. Os botóns de volver a lista de ficheiros e de gardar son intercambiabiles de forma que so un deles se amosa en cada momento. Se o ficheiro foi modificado o botón que se amosa e o

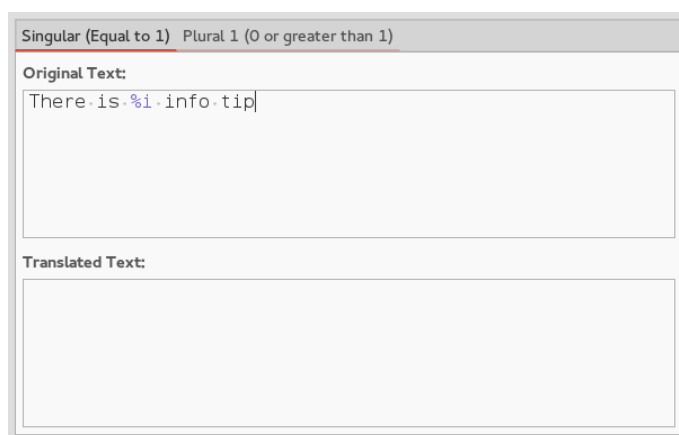


Figura 7.15: Cadro de edición de mensaxes con plurais

de gardar. Desta forma para ir a lista de ficheiros gardados deberemos salvar os cambios.

7.3.2.5. Panel de Preferencias

O panel de preferencias permite modificar como se algunhas opcións do programa, manexar os perfíles e activar ou desactivar plugins. Como podemos ver nas capturas da Figura 7.16 temos tres pantallas para modificar as preferencias do programa.

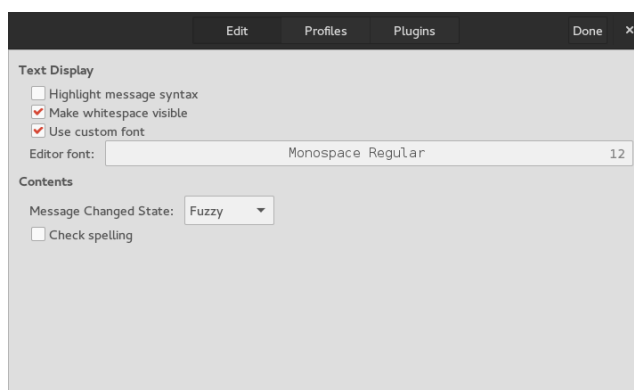
Navegamos entre estas tres pantallas empregando un widget `GTK.StackSwitcher` incrustado na barra de ferramentas.

A pantalla de edición permite modificar o tipo de letra de algunhas partes do programa, o uso de resaltado de sintaxe ou o de espazos en branco. Ademais permite establecer cal é o estado ao que pasa unha cadea cando se traduce. O panel de perfil permite ver os perfíles existentes crear novos perfíles a través do panel de perfil, modificalos, eliminalos e activalos. Por último ca pantalla de plugins podemos ver a lista de plugins existentes e activalos ou desactivalos.

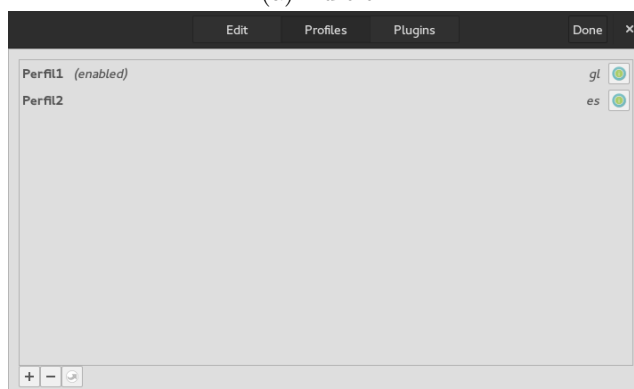
7.3.2.6. Panel de Perfil

O panel perfil permite a creación e modificación de perfíles. Esta dividido en dous subpaneis, un cos datos básicos do perfil e outro con datos avanzados. Estes dous paneis pódense ver na Figura 7.17.

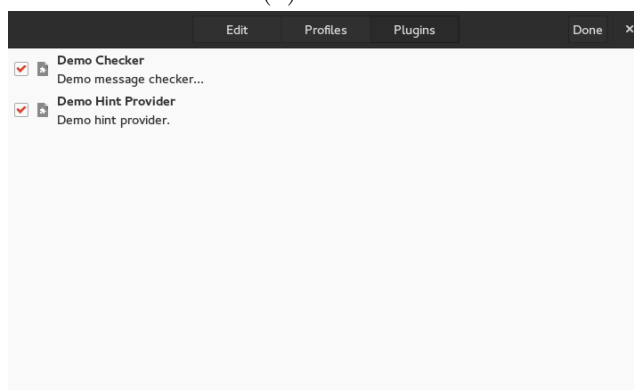
Esta división intenta mellorar a experiencia de usuario pois os valores avanzados auto-completanse cos valores por defecto no momento no que se selecciona a linguaxe.



(a) Edición



(b) Perfiles

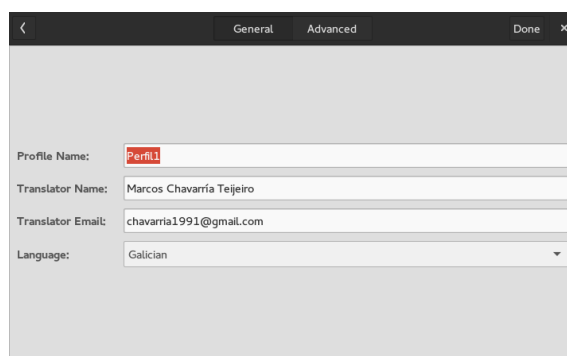


(c) Plugins

Figura 7.16: Partes do panel preferencias

7.4. Navegación e Busca a través do documento

Unha das partes importantes para facilitar a usabilidade da aplicación é que esta permita que o tradutor navegue polo ficheiro e busque termos de forma sinxela e rápida. Para facer isto creamos a clase abstracta `Navigator`. Esta clase ten métodos sen argumentos que permiten avanzar ao seguinte, ao anterior, ao primeiro ou ao último elemento. O valor de volta destes métodos é un valor booleano que indica se esta operación foi posible.

The screenshot shows a mobile application interface with a dark header bar containing a back arrow, the tabs 'General' and 'Advanced', and a 'Done' button with a close icon. The 'General' tab is selected. Below the header, there are four input fields: 'Profile Name' with a red 'Perfil.1' label, 'Translator Name' with the text 'Marcos Chavarria Tejeiro', 'Translator Email' with 'chavarria1991@gmail.com', and 'Language' with a dropdown menu showing 'Galician'.

(a) Opcións básicas

The screenshot shows the same mobile application interface but with the 'Advanced' tab selected. It displays three input fields: 'Plural Form' with a dropdown menu showing 'nplurals=2; plural=(n != 1);', 'Encoding' with a dropdown menu showing 'UTF-8', and 'Team Email' with the text 'gnome-gl-list@gnome.org'.

(b) Opcións Avanzadas

Figura 7.17: Panel de perfil

Para a implementación a busca dentro do documento creamos a clase `Search`. Para implementar os métodos para avanzar a través dos resultados da busca empregamos dous iteradores, un que itera dentro dos mensaxes que cumpren os criterios e outro que itera dentro dos fragmentos das mensaxes.

Como podemos ver na Figura 7.18 os iteradores teñen métodos para conseguir o elemento actual, o primeiro, o último, o seguinte e o anterior. Ademais incorporan metodos para comprobar se o elemento actual é o último ou o primeiro.

As buscas permiten especificar o estado das mensaxes que queremos buscar e se o texto de busca está na cadea orixinal ou na traducida e se está na forma singular ou na plural. esta forma cando se inicia unha busqueda crease un iterador de ficheiro que itera entre os mensaxes que cumpren os requisitos e por cada mensaxe que cumpra os requisitos creamos un iterador de mensaxe que iterará devolvendo os fragmentos de mensaxe que cumpran os requisitos. Un `MessageFragment` inclúe unha mensaxe, a forma plural do fragmento, se o fragmento está na cadea orixinal ou na tradución e o índice e lonxitude do fragmento.

Para seleccionar un fragmento da busca empregamos o método `select()` do panel de edición.

En canto a implementación da navegación entre ficheiros a clase `FileNavigator` empre-

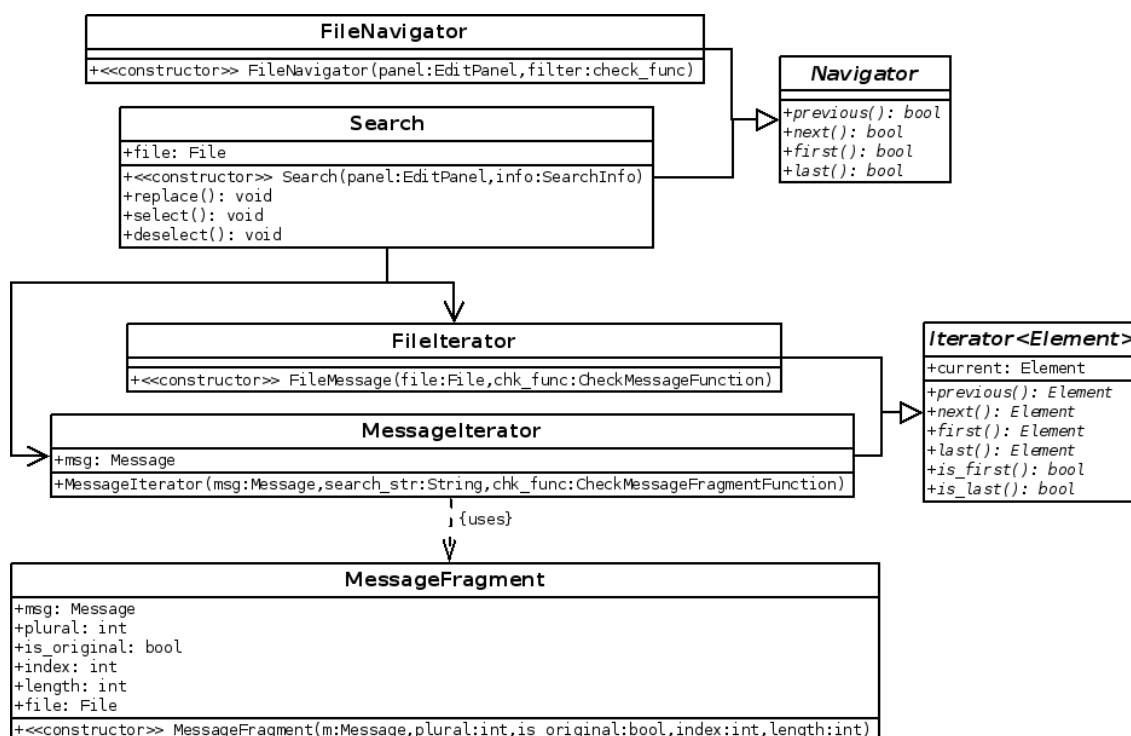


Figura 7.18: Diagrama de Clases de Navegación e Busca

gouse os métodos propios do widget TreeView de GTK. Créanse catro navegadores deste tipo un para as cadeas traducidas, outro para as cadeas sen traducir, outro para as cadeas con tradución difusa e outro para calquera cadea.

7.5. Preferencias

Necesitamos almacenar certas opcións do programa como pode ser o tipo e tamaño da letra a empregar ou se facemos resaltado e sintaxe. Para gardar toda esta información e que este dispoñible entre diferentes sesións do programa empregamos GSettings. GSettings é unha API para a xestión de configuración que forma parte do modulo GIO (GNOME Input/Output) dentro de GLib.

GSettings prové de métodos para obter e almacenar valores a partir dunha clave. Para almacenar estes valores soporta varios sistemas. Nós en concreto usamos o estándar en GNOME, *dconf*. Este sistema está altamente optimizado para facer lecturas (o caso de uso máis habitual) de valores de configuración. Para usar GSettings debemos crear un ficheiro coas claves e tipos dos valores que imos empregar. No Fragmento de Código 7.5 podemos ver unha parte deste ficheiro.

Fragmento de Código 7.5: Fragmento da dos esquemas creados de GSettings

```
<?xml version="1.0" encoding="UTF-8"?>
<schemalist>
  <schema path="/org/gnome/gnomecat/editor/" id="org.gnome.gnomecat.Editor"
    >
    <key name="custom-font" type="b">
      <default>true</default>
      <summary>Use custom font.</summary>
      <description></description>
    </key>
    <key name="message-changed-state" type="s">
      <choices>
        <choice value='fuzzy' />
        <choice value='translated' />
      </choices>
      <default>'fuzzy'</default>
      <summary>Message Changed State</summary>
      <description>The state to set when a message is translated.</
        description>
    </key>
    [...]
  </schema>
  [...]
</schemalist>
```

Este ficheiro temos que compilalo e instalalo cas ferramentas que nos dá o módulo. Unha vez instalado obtemos unha instancia da clase GSettings que emprega o patrón Singleton e usamos esta instancia para obter e almacenar os valores.

Unha das vantaxes de GSettings é que permite enlazar unha propiedade dun obxecto de GObject con un valor de GSettings de forma que se modificamos o valor dun dos dous o outro vese reflexado. Empregamos esta característica para implementar a interface de preferencias na nosa aplicación de forma sinxela.

7.5.1. Módulo de Perfiles

Como vimos anteriormente ao falar da interface gráfica, os perfiles permiten configurar o nome e email co que se gardan os ficheiros e a linguaxe actual. Aínda que poden existir varios perfiles gardados so un estará activo en cada momento. Para almacenar estes perfiles tamén empregaremos GSettings.

7.6. Plugins

Os plugins permiten estender o programa engadíndolle funcionalidade. Dous dos puntos máis sinxelos para engadir funcionalidade son as pistas e os consellos. Estes dous elementos que están pensados para axudar os tradutores son aportados polos plugins. Existen dúas sinais de nomes `check_message` e `provide_hints` que son usadas para crear instancias de pistas e de consellos. Estas sinais son chamadas dende os plugins.

Para implementar estes plugins empregamos a biblioteca *LibPeas*. Esta biblioteca foi creada para poder incluír extensións en GEdit, o editor de texto de GNOME. Trátase dun motor de plugins baseado en GObject. Permite a carga de plugins en varios linguaxes de programación como Vala, C, Python ou JavaScript.

Os plugins son unha unidade de compilación diferente ao programa principal. Para que desde esta unidade independente de compilación coñeza parte da API creada para ficheiros, perfís e linguaxes debemos incluír estes ficheiros tamén nesta unidade de compilación. Creamos ademais unha pequena interface que inclúe as dúas sinais que citamos anteriormente. Esta interface é implementada pola clase aplicación.

A biblioteca LibPeas emprega o patrón Singleton para darnos unha instancia dun motor de plugins. A este motor temos que indicarlle a ruta do sistema onde ten que buscar os plugins e cargar os plugins que atopen. Ademais cargamos un obxecto de clase

Para implementar un plugin temos que crear dous ficheiros. O primeiro ficheiro contén no nome do plugin información sobre o plugin como a descrición, o autor ou autores e o Copyright. No Fragmento de Código 7.6 pódese ver un exemplo deste ficheiro.

Fragmento de Código 7.6: Ficheiro de metadatos do plugin

```
[Plugin]
Module=demochecker
Name=Demo Checker
Description=Demo message checker...
Authors=Marcos Chavarría Teijeiro <chavarria1991@gmail.com>
Copyright=Copyright (C) 2014 Marcos Chavarría Teijeiro
```

O outro ficheiro é a implementación do propio plugin. Debemos implementar unha ou máis extensións e unha función onde as rexistraremos. A función ten que ter o nome `peas_register_types()` e usará a etiqueta `ModuleInit` de forma que se executará cando se inicialice o módulo.

Fragmento de Código 7.7: Fragmento da implementación do plugin DemoChecker

```
public class DemoChecker : Peas.ExtensionBase, Peas.Activatable
{
```

```

    public Object object { owned get; construct; }

    public void activate ()
    {
        (object as GNOMECAT.API).check_message.connect (on_check_message);
    }

    public void deactivate ()
    {
        (object as GNOMECAT.API).check_message.disconnect (on_check_message);
    }

    public void update_state ()
    {
    }

    [...]
}

[ModuleInit]
public void peas_register_types (GLib.TypeModule module)
{
    var objmodule = module as Peas.ObjectModule;
    objmodule.register_extension_type (typeof (Peas.Activable),
                                      typeof (DemoPlugins.DemoChecker));
}

```

Para implementar unha extensión temos que implementar unha clase que implemente a interface `Activable` e herede da clase `ExtensionBase`. A interface `Activable` inclúe os métodos que se executarán cando se active e desactive o plugin. No Fragmento de Código 7.7 amosase parte do código empregado para o plugin de exemplo *DemoPlugin*. Podemos ver como nos métodos para activar e desactivar conectamos e desconectamos a sinal `check_message` para poder xerar os Tips de exemplo.

Capítulo 8

Conclusións e Traballo futuro

Neste último capítulo falaremos das conclusións que sacamos en claro da realización do presente proxecto e falaremos de posibles liñas de traballo futuro.

8.1. Conclusións

O software libre está deseñado para garantir a liberdade dos usuarios. O obxectivo é que calquera persoa que necesite facer algo cun ordenador poida empregar unha alternativa libre de forma sinxela. Neste contexto, a accesibilidade e a localización dos programas é fundamental.

A pesares disto, as alternativas libres para programas de tradución son moi limitadas e teñen en moitas ocasións baixa calidade. Neste proxecto de fin de carreira traballamos para sentar as bases dunha ferramenta que sexa robusta pero extensible ao mesmo tempo.

O aplicativo que resultou deste traballo, a pesar de que aínda necesita ser probado por tradutores, é unha ferramenta creada de forma que sexa facilmente estensible e escrita nunha linguaxe que permitirá que moitos desenvolvedores participen no proxecto. Isto unido a implementación dalgunhas características como as que se citan na próxima sección farán que esta ferramenta poida ser empregada amplamente por usuarios de GNOME e doutros proxectos de software libre.

8.2. Traballo Futuro

Como calquera proxecto de software libre este programa é un proxecto inconcluso. Por unha parte, algúns dos requisitos establecidos ao inicio do programa foron elaborados con pouca profundidade ou presentan fallos polo que solucionarlos é unha clara liña de traballo.

Ademais, aínda que neste proxecto a interface gráfica supuxo un esforzo importante, hai algunhas partes que poden ser mellorables.

Por outro lado, para que un proxecto destas características teña futuro é necesario crear unha comunidade arredor del. Polo que unha das liñas de traballo ten que ser buscar contribuidores para o proxecto e integralo dentro da infraestrutura de GNOME.

Se falamos de liñas de traballo en canto á implementación de novas características, existen varias liñas de traballo moi interesantes:

Implementación de unha Memoria de Tradución. Empregando a API que creamos para prover ao usuario de pistas a través de plugins, implementar un plugin que sexa unha Memoria de Tradución. Esta memoria deberá aceptar ficheiros po con tradución para incorporalos a unha base de datos que logo analizaremos co obxectivo de buscar posibles traducións das cadeas.

Integración con Damned Lies. Damned Lies é a plataforma oficial de GNOME para xestionar as traducións. Nela pódense baixar os ficheiros para traducir, subir os ficheiros traducidos, revisar os mesmos, e ver as estatísticas xerais de tradución para cada linguaxe. Sería realmente útil que todas estas tarefas se puidesen facer dende GNOME CAT. Para isto non só bastaría con implementar novas vistas en GNOME CAT, senón tamén a creación dunha API pública para a plataforma web Damned Lies que está escrita en Python empregando o framework Django.

Glosario. Implementación dun glosario de termos na interface de edición de GNOME CAT. Este glosario debería permitir importar e exportar diversas bases de datos tanto online como offline.

Previsualización das traducións. Como xa se comentou anteriormente esta é unha característica moi interesante para unha ferramenta CAT pois permite que os tradutores vexan como pode quedar a tradución no programa final. Aínda que existen varias alternativas para implementar esta funcionalidade, semella que a máis axeitada para tecnoloxías GNOME é empregar o motor de renderizado Glade como se fixo na aplicación web Deckard.

Apéndice A

Escoller a licenza do programa

Á hora de escoller unha licenza para o novo programa temos que ter en conta as licenzas das bibliotecas empregadas no noso programa. Na seguinte lista podense ver as bibliotecas empregadas para o programa así como a licenza de cada unha delas.

- **GLib** GNU Lesser General Public License v2.1
- **GTK+** GNU Lesser General Public License v2.1
- **LibGee** GNU Lesser General Public License v2.1
- **LibPeas** GNU Lesser General Public License v2.1
- **GettextPo** GNU General Public License v2
- **GTKSourceView** GNU Lesser General Public License v2.1
- **JSON-GLib** GNU Lesser General Public License v2.1

Como podemos ver, excepto a biblioteca GettextPO, que emprega unha licenza GNU GPLv2, o resto de bibliotecas empregadas usan unha licenza GNU LGPL v2.1. A licenza GNU Lesser GPL permite empregar calquera licenza en produtos derivados. Por outra parte a licenza GNU GPL v.2 obriga a empregar a licenza GNU GPL v2 ou versións posteriores da mesma licenza.

No noso caso eliximos a licenza GNU General Public Licence v3 que é unha versión máis actualizada da licenza para programas de GNU.

Apéndice B

Instrucións de Compilación e Instalación

GNOMECAT é Software Libre e o seu código fonte está dispoñible nun repositorio de GitHub de nome [chavaone/gnomecat](https://github.com/chavaone/gnomecat).

Para poder probar o aplicativo antes de nada debemos instalar as súas dependencias tanto para compilalo como para executalo. O programa depende das seguintes bibliotecas:

- glib-2.0
- gtk+-3.0
- gtksourceview-3.0
- gee-0.8
- json-glib-1.0
- libpeas-1.0
- gettext-po

Ademais, para descargar e compilar o programa necesitamos as seguintes ferramentas:

- vala
- automake e autoconf
- gettext e intltool
- git

O Fragmento de código B.1 amosa o comando que temos que executar para instalar as ferramentas e bibliotecas necesarias no caso de que empreguemos o xestor de paquetes *yum* propio de distribucións como Fedora, Red Hat ou CentOS. Se empregamos outros xestores de paquetes o comando sería similar.

Fragmento de Código B.1: Comando para instalar utilidades e dependencias

```
$> sudo yum install git automake autoconf gettext gettext-libs gettext-  
devel glib2 glib2-devel gtk+-devel gtksourceview3 gtksourceview3-devel  
libgee libgee-devel json-glib json-glib-devel libpeas libpeas-devel  
intltool vala
```

Unha vez instalada as dependencias so temos que descarga, compilar e instalar o programa. Como empregamos o sistema de control de versións git, para descargar o repositorio temos que facer uso do seu comando `clone`.

Para compilar a aplicación empregaremos os pasos típicos para intalar calquera aplicación que empregue Autotools. No Fragmento de código B.2 podemos ver os comandos que temos que exectuar para realizar estas tarefas:

Fragmento de Código B.2: Comando para instalar utilidades e dependencias

```
$> git clone https://github.com/chavaone/gnomecat.git  
$> cd gnomecat  
$> ./autogen.sh  
$> ./configure  
$> make  
$> sudo make install
```


Apéndice C

Bibliografía

- [Bec00] Kent Beck. *Extreme Programming explained*. 2000.
- [DMPH13] Ulrich Drepper, Jim Meyering, François Pinard, and Bruno Haible. *GNU Gettext Tools*, 0.18.2 edition, 2013.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. 1995.
- [Git] Git. Git Documentation. git-scm.com/documentation.
- [GNOa] GNOME. GLib Reference Manual. developer.gnome.org/glib.
- [GNOb] GNOME. GNOME Human Interface Guidelines. developer.gnome.org/hig.
- [GNOc] GNOME. GTK+ 3 Reference Manual. developer.gnome.org/gtk3.
- [GNOd] GNOME. JHBuild Manual. developer.gnome.org/jhbuild.
- [GNOe] GNOME. Vala Documenatation. valadoc.org.
- [GNOf] GNOME. Vala Reference Manual. wiki.gnome.org/Projects/Vala/Manual.
- [GNOg] GNOME. Writing a VAPI without using GObject introspection. wiki.gnome.org/Projects/Vala/LegacyBindings.
- [Hou] Translate House. Virtaal website. virtaal.translatehouse.org.
- [Lon06] Imperial College London. Imperial college london translation memories survey. 2006.
- [MTDL⁺15] David MacKenzie, Tom Tromey, Alexandre Duret-Lutz, Ralf Wildenhues, and Stefano Lattarini. *GNU Automake*, 2015.

- [OPHS14] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schleg. *The Not So Short Introduction to L^AT_EX2 ϵ* . 2014.
- [Pro] OmegaT Project. OmegaT Website. omegat.org.
- [RJB06] James Rumbaugh, Ivar Jacobson, and Grady Booch. *El lenguaje unificado de modelado: manual de referencia*. 2006.