UNIVERSIDAD AUSTRAL DE CHILE
SEDE PUERTO MONTT
ESCUELA DE INGENIERIA EN COMPUTACION

SISTEMA COLABORATIVO DE TRADUCCION PARA APLICACIONES LINUX

Seminario de Titulación
para optar
al título de Ingeniero en Computación

PROFESOR PATROCINANTE:
Sr. Moisés Coronado

PROFESOR CO-PATROCINANTE:
Sra. Claudia Zil Bontes

Rolando Alberto Martínez Gallardo

PUERTO MONTT – CHILE
Octubre - 2008

Universidad Austral de Chile
Escuela de Ingeniería en Computación

Los Pinos s/n, Balneario Pelluco
Sede Puerto Montt
Casilla 1327 · Fono: 56 65 260990
Fax: 56 65 277156
Email: ecomputa@uach.cl
www.uach.cl

Puerto Montt, 26 de septiembre de 2008

COMUNICACIÓN INTERNA N° 204/08

DE : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA DE INGENIERIA EN COMPUTACION

A : Dr. Renato Westermeier H. – **VICERRECTOR SEDE PUERTO MONTT**
Mag. César Pino Soto – **COORDINADOR ACADEMICO SEDE PUERTO MONTT**
Sra. Cristina Barriga – **REGISTRO ACADEMICO**
Sra. Alba Vásquez - **ENCARGADA DE TITULACIÓN CAMPUS PUERTO MONTT**

C.c : Sr. Rolando Martínez Gallardo
Sr. Moisés Coronado Delgado
Sra. Claudia Zil Bontes
Sra. Viviana Alvarado Espinoza

MOTIVO:

Informar a usted, las calificaciones obtenidas por el alumno de Ingeniería en Computación **Sr. Rolando Alberto Martínez Gallardo** Rut 15.295.858-7, en su informe de Titulación "***Sistema Colaborativo de Traducción para Aplicaciones Linux***"

Prof. Moisés Coronado Delgado	6.0
Prof. Claudia Zil Bontes	6.2
Prof. Viviana Alvarado Espinoza	6.2
Promedio Seminario	**6.13**

Sin otro particular, le saluda atentamente,

UNIVERSIDAD AUSTRAL DE CHILE
ESCUELA DE INGENIERIA
EN COMPUTACION
PUERTO MONTT

SANDRA RUIZ AGUILAR
DIRECTORA

SRA/mva

PUERTO MONTT, 24-07-2008

De : Sr. Moisés Coronado Delgado
PROFESOR PATROCINANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***Rolando Alberto Martínez Gallardo*** en su Seminario de Titulación "**Sistema Colaborativo de Traducción para Aplicaciones Linux**":

NOTA: 6,0

JUSTIFICACION:

A falta de documentación que acompañe al proyecto como el proyecto

OTRAS OBSERVACIONES:

Sr. Moisés Coronado Delgado
PROFESOR PATROCINANTE

PUERTO MONTT, 24 de septiembre del 2008

De : Sra. Claudia Zil Bontes
PROFESORA CO-PATROCINANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***Rolando Alberto Martínez Gallardo*** en su Seminario de Titulación **"Sistema Colaborativo de Traducción para Aplicaciones Linux":**

NOTA: 6,2

JUSTIFICACION:

Aplicación adecuada de metodología y técnicas.
Uso de procedimientos y técnica de apoyo.
Documento útil.

OTRAS OBSERVACIONES:

Sra. Claudia Zil Bontes
PROFESORA CO-PATROCINANTE

PUERTO MONTT, 25 de septiembre 2008

De : Sra. Vivian Alvarado Espinoza
PROFESORA INFORMANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***Rolando Alberto Martínez Gallardo*** en su Seminario de Titulación **"Sistema Colaborativo de Traducción para Aplicaciones LINUX":**

NOTA: 6,2 (seis coma dos)

JUSTIFICACION:

Presenta un adecuado desarrollo del tema. Buen proyecto en relación a su futura aplicabilidad e implicancias.

OTRAS OBSERVACIONES:

Faltó detallar y realizar algunas actividades de la metodología seleccionada.
No se observan descripción de pruebas con los usuarios

Sra. Viviana Alvarado Espinoza
PROFESORA INFORMANTE

Dedicado a mi familia por todo su apoyo en estos años de estudio….

Agradecimientos

A mi familia, que sin su eterno apoyo no hubiese logrado lo que ahora tengo. Gracias hermano Alex por tu preocupación en estos largos años.

Gracias por todo tu apoyo, comprensión y amor en esos días donde mi genio no era el mejor; a ti Bandy, mi amor.

Un cordial saludo de agradecimiento a aquellos profesores que a lo largo de mi carrera me enseñaron aprender a aprender.

A todos los que aportaron para que este seminario de tesis llegue a buen puerto; profesores Moisés Coronado y Claudia Zil y a nuestra querida secretaría de la carrera de Ingeniería en Computación Maribel Villanueva.

Gracias a mis revisores: Bandy Troncoso y Luisa Hernandez.

INDICE

Tablas

Figuras

Diagramas

Códigos

SINTESIS

El presente proyecto nace con el fin de diseñar y construir una herramienta web, capaz de manipular y organizar archivos de localización que cumplan con el estándar Gnu-Gettext, todo esto con el objetivo de poder mejorar el proceso de localización de aplicaciones Gnu-Linux, de tal forma de entregar a los usuarios un lugar centralizado, desde donde puedan acceder a sus traducciones.

Para el desarrollo de este proyecto se utilizaron las principales prácticas dentro de la metodología “Agile Unified Proccess” (AUP) tales como uso de un repositorio, framework de pruebas de unidad, orientación a objetos, patrones de diseño, entre otros. Además de lo anterior se utilizó el motor de base de datos MySql para guardar toda la información relacionada con usuarios y archivos de localización administrados por la herramienta.

Otro punto importante en el desarrollo de la aplicación, fue el proyecto de código abierto Mono, el cual entregó toda la infraestructura necesaria (IDE, framework de clases base y run-time) para la correcta finalización de este proyecto.

Una vez finalizadas las etapas propuestas por la metodología se obtuvo un producto de software totalmente funcional, capaz de entregar

información relacionada con la traducción de cada archivo de localización alojado en la aplicación.

ABSTRACT

This project was created with the aims of design and constructs a web tool that is able to manipulate and organize localization files that fulfills the Gnu-Gettext standard. The objective is to improve the Gnu-Linux application localization process so it can contribute to give the user a unique location to have access to needed translations.

Activities such as source repository, unit testing framework, object orientation, design patterns, among others, where used to develop this project. They are related to the methodology “Agile Unified Process” (AUP). Besides from the previously mentioned DataBase Management System MySql was also used to store all the data related with the user and the localization files administered by the tool.

Other important thing to mention about the development of the application was the open source Mono project which gives the needed infrastructure (IDE, framework based classes, and run-time) for an appropriated ending to the project.

Once the proposal stages of the methodology have ended the result was a total functional service software capable of giving information related to the translation of every localization file within the application.

1. Introducción

En la era de la globalización, donde existe un verdadero esfuerzo de las empresas para que sus productos traspasen fronteras, forzosamente aparece la necesidad de proveer información en una variedad de idiomas. Lo anterior, es substancialmente cierto para la industria del software, donde el producto en si está conformado por información que debe ser comprendida por los usuarios pertenecientes a distintas localidades geográficas, desde donde está siendo utilizada la aplicación.

En el desarrollo de software, internacionalización y localización son procesos utilizados para adaptar una aplicación a distintos países y culturas. Es decir, una aplicación originalmente escrita en inglés podrá ser llevada al español siempre y cuando haya sido internacionalizada y localizada a este último idioma.

En Linux, desde 1995 se está entregando soporte para estos dos procesos gracias a Gnu – Gettext, el cual entrega a los programadores y traductores un conjunto de herramientas útiles y necesarias para llevar a buen término sus tareas.

Ejemplo de una herramienta que apoya el proceso de localización y que hace uso de Gnu-Gettext es Kbabel, que entregando una interfaz sencilla y fácil de usar permite a los traductores la manipulación de los archivos de localización, que es donde se encuentran los mensajes extraídos desde el código fuente de la aplicación.

No obstante, aunque existen herramientas que apoyan el proceso de localización de una aplicación, aún queda trabajo por hacer en relación a la coordinación que deba tener un grupo de traductores voluntarios.

Otra tarea pendiente en términos de localización de aplicaciones GNU-Linux, es la capacidad de trabajar “on-line”, es decir, dar la posibilidad al usuario de traducir sus aplicaciones a través de un sistema web, de tal forma que los usuarios voluntarios comprometidos con una aplicación específica, puedan fácilmente acceder al sistema, sin necesidad de tener instalado el entorno requerido para la localización de la aplicación.

Por las necesidades anteriormente comentadas en el ámbito de la localización de aplicaciones en entornos Gnu- Linux, es que el alumno tesista implementará un sistema web que sea de utilidad en la creación, mantención y/o organización de archivos de localización. El presente documento describe las actividades realizadas dentro del proyecto “Desarrollo de un Sistema Colaborativo de

Traducción para Aplicaciones Linux" y el cual se ha organizado en capítulos. A continuación se entrega un guía al lector.

El capítulo 2 entrega los objetivos del proyecto, tanto el objetivo general como los específicos.

En el capítulo 3 detalla el planteamiento del problema, esfuerzos anteriores y solución propuesta.

Dentro del capítulo 4 se describe la metodología utilizada para el desarrollo de este proyecto.

El capítulo 5 se exponen tantos los recursos software como hardware utilizado en este proyecto.

El capítulo 6 expone lo realizado en la etapa de inicio del proyecto. Inserto en este capítulo se expondrán temas de importancia tales como gnu-gettext, definición de requerimientos, consideraciones para el desarrollo, entre otros.

El capítulo 7 detalla temas de elaboración de diseños y especificación de arquitectura.

El capítulo 8, entrega detalles sobre la construcción del software. Dentro del capítulo se nombraran los siguientes temas: desarrollo dirigido por pruebas, ejecución de prueba de unidad utilizando Nunit en un entorno Linux, ejecución de una aplicación ASP.NET en Linux, entre otros.

El capítulo 9 describe las tareas de publicación y difusión del proyecto.

El capítulo 10 entrega una introducción al concepto de “integración continua” y cómo fue aplicada al proyecto a desarrollar.

El capítulo 11 hace entrega de una breve reseña al proyecto Mono. Aquí se entregará una introducción y algunos consejos en la portabilidad de aplicaciones ASP.NET desde Windows a Linux.

En el capítulo 12 se encuentran las conclusiones y recomendaciones.

El capítulo 13 y 14 contendrán la bibliografía y documentos anexos, respectivamente.

2. Objetivos

2.1. Objetivo general

El objetivo de este proyecto es desarrollar una herramienta que sea de utilidad en la creación, mantención y/o organización de archivos de localización.

2.2. Objetivos específicos

- Administrar de usuarios que corresponderán a usuarios traductores, coordinadores o revisores.
- Administrar de proyectos de traducción, donde cada proyecto agrupará un conjunto de archivos de localización dependientes de una única aplicación.
- Editar de archivos de localización mediante una interfaz web.
- Crear una interfaz de usuario, que permita poner nota a los mensajes ya traducidos y con esto mejorar la calidad de traducciones.
- Seguir en tiempo real el estado de las traducciones para cada aplicación.
- Asociar imágenes a los mensajes de texto que serán traducidos, de tal forma de mejorar la contextualización de lo que se quiere traducir.

- Instalación de un repositorio de código, siguiendo las prácticas encontradas en [Sink2004].
- Instalación de un sistema de seguimiento de tareas y bugs.

3. Planteamiento del problema

3.1. Antecedentes

3.1.1. Definición del problema

Actualmente, para poder llevar una aplicación Linux a múltiples lenguajes, se hacen necesarios los procesos de internacionalización y localización, ambos soportados por Gnu – Gettext [Gnu1995].

Por Internacionalización, también conocido como i18n, se entiende como el proceso de tomar una aplicación diseñada para una localidad y re-estructurarla para que pueda ser usada en diferentes localidades; también se define como el proceso de crear un programa lo suficientemente flexible para que se ejecute correctamente en cualquier localidad. En i18n se encuentran los programadores, los cuales deben seguir las normas establecidas para que su aplicación pueda ser llevada a otros idiomas. I18n corresponde a la abreviatura de “internacionalización” y se deriva la primera letra que es “I” más 18 caracteres que tiene la palabra, más “n” que es la última letra.

Por Localización, también conocido como l10n, se entiende como el proceso de agregar soporte para una nueva localidad hacia una aplicación

internacionalizada. En I10n están los traductores que se encargan de localizar los mensajes extraídos desde el código fuente. L10n corresponde a la abreviatura de “localización” y se deriva de la primera letra que es “l” más 10 caracteres que tiene la palabra más “n” que es la última letra.

GNU-Gettext, básicamente es un conjunto de librerías que entregan las convenciones de cómo un programa debe ser escrito y qué utilidades usar para poder extraer desde el código fuente, los mensajes que se pretende sean traducidos y que luego serán guardados en archivos de texto plano, comúnmente denominados “archivos de localización” o archivos PO.

De esta manera, cada archivo PO contendrá cada mensaje original de un determinado programa con su traducción a una lengua determinada, teniendo así tantos archivos como lenguas soportadas tenga la aplicación.

Teniendo en consideración lo antes comentado, en Internet existen grupos de voluntarios los cuales se encargan de apoyar el proceso de localización de aplicaciones tomando como base los archivos PO. Específicamente, el trabajo de los voluntarios consistirá en editar estos archivos, completando las traducciones para los mensajes extraídos desde el código fuente de la aplicación.

Para aclarar lo antes comentado, la figura 1 muestra el proceso de internacionalización y localización trabajando con Gnu – Gettext, el cual comienza desde el código fuente de la aplicación hasta llegar a un archivo binario, el cual será invocado por el programa desde donde se extrajeron las cadenas a localizar. El uso de esta librería utilizando el framework Mono, se encuentra disponible en el Anexo C “Uso de Gettext utilizando C#”.

Un resumen del proceso que muestra la figura 1, es el siguiente

- Teniendo el código fuente de la aplicación, se usa la utilidad xgettext, la cual trae todos los mensajes que se desean traducir y los almacena en un archivo de plantilla (archivo.pot). Este archivo de plantilla no contendrá los mensajes traducidos de ningún idioma, sino que servirá como base para cada uno de los archivo “po” que se quieran localizar.

- Se crea un archivo “po” (español.po) específico del idioma al que se quiere llevar la aplicación. Los mensajes a traducir serán tomados a partir del archivo de plantilla.

- Traducción de los mensajes editando el archivo “po” (español.po), utilizando algún editor de preferencia. Para la localización de aplicaciones existen editores especializados tales como kbabel y gtranslator.

- Utilizando la herramienta msgfmt, se convierte el archivo “po” a un archivo binario “mo”.

- Se instala el archivo binario “mo” en el directorio correspondiente.

- Cuando la aplicación lanza una nueva versión, quizás existan nuevas cadenas por traducir. Para realizar las traducciones de estas nuevas cadenas los traductores no necesitan realizar todo el trabajo nuevamente. Para esto existe la utilidad msgmerge, la cual realiza una combinación o “merge” entre el nuevo archivo de plantilla, generado con la aplicación xgettext, y el actual archivo (español.po).

Como muestra la figura 1, una de las partes participantes en este proceso es el editor de archivos PO, el cual será una herramienta de uso frecuente para las personas que participan en el proceso de localización.

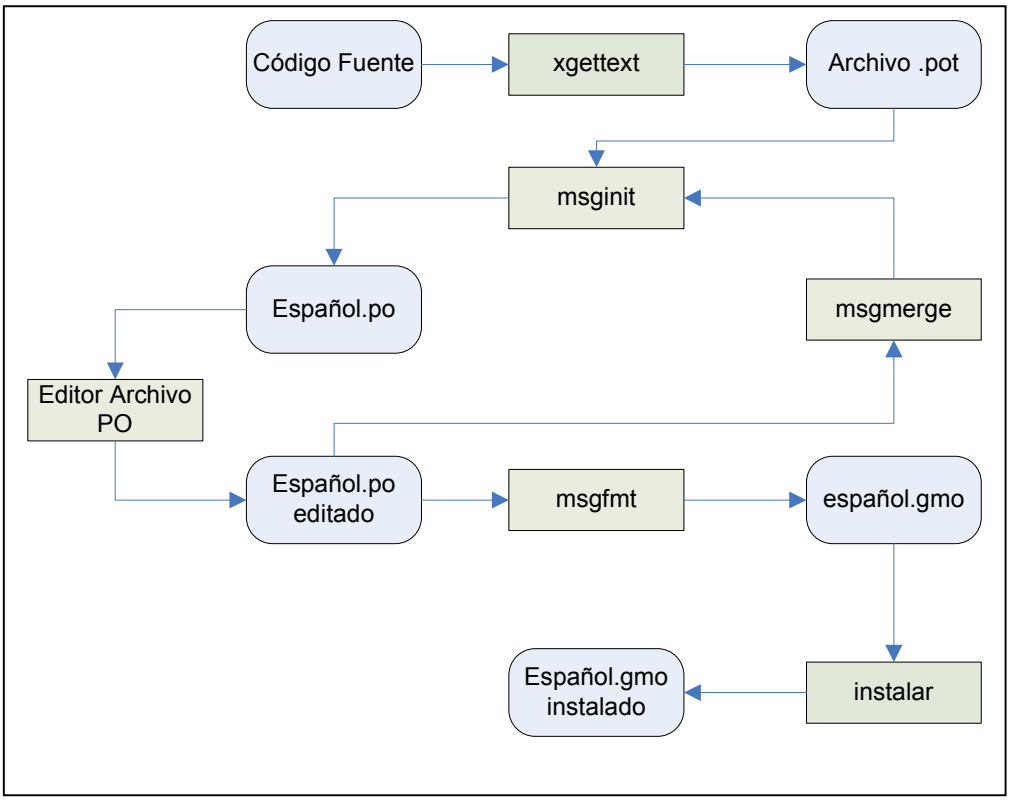


Figura 1. Proceso de Internacionalización y localización en Gnu - Gettext

Sin embargo, aún cuando el proceso de localización se encuentra normado por procedimientos establecidos por cada uno de los grupos de traducción, pueden surgir desventajas tales como:

- Desconocimiento del avance real en las traducciones, debido a que la edición de los archivos se realiza de manera offline.

- La designación de archivos por traducir para cada persona se encuentran en un archivo de asignaciones, el cual es completado y actualizado de manera manual por el coordinador del grupo.
- Posible descoordinación en la traducción de un archivo. Puede ocurrir que dos o más voluntarios estén trabajando en el mismo archivo.
- No existe un proceso de revisión. Es responsabilidad de cada traductor entregar un archivo sin faltas gramaticales y ortográficas.
- Contextualización de los mensajes del texto a traducir. Los mensajes ya traducidos no concuerdan de manera total con el contexto donde se encuentra inserto el mensaje.

3.1.2 Esfuerzos Anteriores

Actualmente existe un proyecto de software libre llamado Webtranslator correspondiente a una aplicación web que está destinada a la traducción de los manuales de PostgreSQL del inglés al castellano. Aunque esta aplicación se aproxima a los objetivos del proyecto de tesis, esta no cumple todas las funcionalidades que se quieren implementar, tal como la agrupación de los archivos de localización dentro de entidades llamadas “Proyectos de Localización”.

3.1.3 Solución propuesta

De acuerdo a los antecedentes y esfuerzos comentados anteriormente, se busca un software que permita la localización de aplicaciones Linux, cumpliendo las labores básicas tales como edición de archivos de localización, corrección ortográfica utilizando un sistema colaborativo de revisión y asociación de imágenes a los mensajes a traducir.

La solución está conformada por lo siguiente:

- Sistema web utilizando el entorno de ejecución proveído por el proyecto Mono [Mono2008].
- Un repositorio de archivos de localización.
- Una base de datos relacional, cuyo diseño está apoyado por [Connolly2005] y [Ambler2003]. Básicamente la base de datos guardará información de usuario registrados en el sistema junto con el contenido de los archivos de localización.

3.2 Justificación

- **Situación sin proyecto:**

La situación sin proyecto, implicaría que los traductores seguirán trabajando de manera local en cada uno de sus equipos, no teniendo en tiempo real la verdadera situación en la que se encuentra la traducción de una aplicación o paquete.

Además, la responsabilidad de una buena ortografía quedará en manos de cada traductor, no dejando la posibilidad de que el proceso de revisión sea tomado por uno o más revisores.

Algunas traducciones hechas por los voluntarios se realizarán fuera del contexto, es decir, existirán mensajes traducidos que tengan poca o ninguna relación con la interfaz gráfica desde donde es lanzado el mensaje en cuestión.

- **Situación con proyecto:**

Con la publicación de este proyecto, los traductores obtendrán una herramienta sin costos con la cual podrán localizar una aplicación o un grupo de aplicaciones, según ellos requieran.

De esta manera, los traductores dispondrán de una herramienta online con la cual podrán subir archivos de localización que luego serán editados utilizando este sistema web.

Otra ventaja presente en la herramienta, es la edición conjunta de un archivo de localización, es decir, uno o más voluntarios tendrán derecho a editar los mensajes pertenecientes a un archivo de localización, dejando de lado la limitación que entrega la situación sin proyecto.

3.3 Equipo de trabajo y Contribución del Alumno

Para cubrir las necesidades de este proyecto, el alumno tesista estudió y puso en práctica una metodología ágil de desarrollo de software llamada “Agile Unified Proccess” (AUP) disponible en [Ambler2005] y explicada con mayor detalle en la sección “metodología” de este documento.

El equipo de trabajo constó de las siguientes personas:

- Moisés Coronado. Patrocinador y experto del dominio.
- Rolando Martínez Gallardo (Alumno tesista).

Las contribuciones a este proyecto son las siguientes:

- Extender el uso de prácticas y herramientas utilizadas por las metodologías ágiles de desarrollo de software (Source Control, Test Driven Development, bug tracking systems, entre otros).
- Ayudar en la localización de aplicaciones Gnu-Linux. En primera instancia el software sería utilizado por la distribución Famelix para la localización de sus aplicaciones.

3.4. Delimitación

- El sistema no contempla un revisor automático de ortografía incrustado dentro de la aplicación web.
- La portabilidad del sistema web desde un servidor Linux a otro Windows no será tratado durante el desarrollo de este proyecto.
- A diferencia del software tradicional, este proyecto no va dirigido a un único cliente, es por esto que, actividades tales como las pruebas de aceptación serán validadas de acuerdo a criterios establecidos entre el experto del dominio y el alumno tesista.

- La localización de la aplicación desarrollada se realizará de acuerdo a los estándares y/o limitaciones impuestas por la plataforma de desarrollo.
- El contenido de los archivos de localización provendrán únicamente desde archivos de código fuente.

4. Metodología

Para este seminario de titulación, la metodología elegida es “Agile Unified Proccess” (AUP), la cual fue fundada y dada a conocer por Scott W. Ambler en Septiembre del 2005, y que el lector puede consultar en [Ambler2005]. Esta metodología es un enfoque simplificado de RUP [Kruchten2003] y describe una simple y fácil manera de entender el desarrollo de software utilizando técnicas y conceptos tales como Test Driven Development, Agile Modeling y refactorización de base de datos.

Básicamente el ciclo de vida de esta metodología consiste en un proceso secuencial de fases, las cuales se van realizando de manera incremental por medio de disciplinas. La figura 2 muestra un esquema general de la metodología.

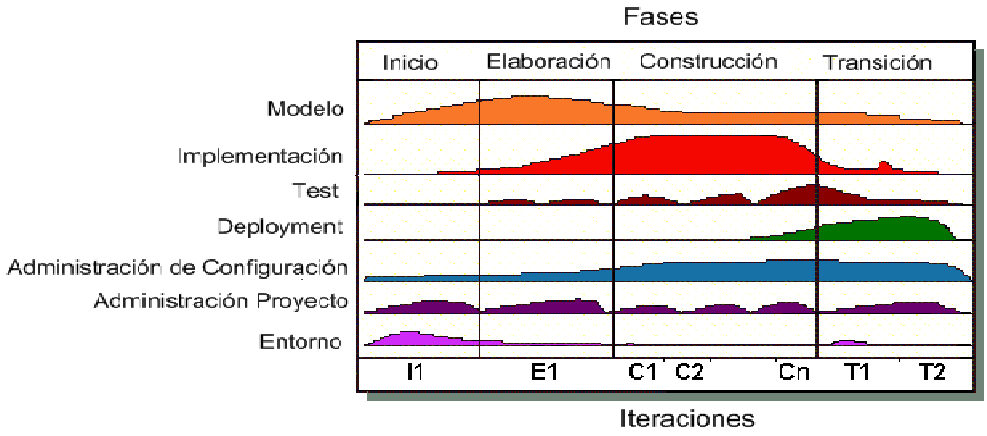


Figura 2. Ciclo de vida de AUP

La decisión de adoptar AUP como metodología, se basa en las nuevas características que a corto plazo pueda tener el software, es decir, la necesidad de adaptar el software a nuevos requerimientos, lo cual concuerda con el desarrollo de software incremental que esta metodología sugiere. Otra de las razones para escoger esta metodología, es la experiencia que el alumno tiene en las prácticas que esta metodología sugiere, tales como construcción dirigido por pruebas, uso de un repositorio de código e integración continua.

El desarrollo de este seminario de tesis se hará pasando por cada una de las fases y disciplinas que la metodología indica, aunque sólo se tomarán las actividades que realmente tengan importancia para el término exitoso del seminario.

La fase de construcción estará dividida en incrementos dependientes de la cantidad de requerimientos funcionales descubiertos en la fase de inicio.

A continuación se entrega una descripción de cada una de las fases de la metodología.

4.1 Inicio

Se identificará el ámbito del proyecto y se definirá en términos generales lo que el sistema hará y dejará de hacer. Debido a que este proyecto no está orientado a un cliente corporativo, las actividades de estimación de costos, definición de riesgos y estudio de viabilidad, no serán realizadas.

La mayor parte de las tareas corresponderán a la disciplina de ambiente y modelado.

4.2 Elaboración

Las actividades con mayor importancia en esta fase corresponden a la elaboración de distintos diagramas que apoyen la adquisición de manera

detallada de los requerimientos. Algunos de ellos serán diagramas de caso de uso, de clases, interacción, entre otros.

Basándose en el nivel de complejidad de la aplicación, se entregará un prototipo de arquitectura, el cual será validado y probado durante esta fase. Para la traza de los distintos artefactos de salida de esta fase se utilizará un repositorio de código.

4.3 Construcción

Para esta fase se tienen contemplado posibles cambios en los modelos concebidos en las fases de inicio y elaboración. Las disciplinas que toman mayor importancia son la de modelamiento, implementación y administración de la configuración.

La mayor cantidad de tiempo se tomará en la implementación la cual abordará las siguientes actividades:

- Codificación basada en TDD [NewKirk2004].
- Integración continua.
- Desarrollo del modelo de la interfaz de usuario.
- Desarrollo del esquema de datos.

De acuerdo a AUP, el desarrollo de la fase de construcción se hará de manera incremental, donde cada interacción actuará sobre uno o varios requerimientos.

Al término de la construcción, se obtendrá como resultado un software que cumpla con los requerimientos obtenidos en las fases tempranas de esta metodología.

4.4 Transición

Tomando como entrada el sistema software ya construido. Se pasará a la fase de transición, consistente en la corrección de errores, publicación del software y creación de documentos de ayuda para el manejo de la aplicación.

5. Recursos

5.1 Hardware

Básicamente, el hardware necesario para este proyecto se dividió de acuerdo a la fase de desarrollo en que se encuentre el proyecto. De esta manera, la etapa pura de desarrollo del software dispondrá de 3 equipos, mientras que la etapa de producción contendrá un único equipo, en el cual finalmente se alojará la aplicación.

El siguiente hardware dará soporte a todo el ciclo de vida de la aplicación.

Tabla Nº 1. Descripción hardware utilizado.

Equipo	***Descripción***	***Función***
HP Pavilion dv5000	AMD Turion64, 1 GB Ram, 80 GB HD.	Contendrá el software para el desarrollo de la aplicación.
Equipo de escritorio armado.	Pentium IV, 2 GB Ram, 160 HD	Equipo que alojará la aplicación, repositorio de código y el sistema de seguimiento de proyecto.
Dell PowerEdge 830 Server	Intel Pentium D 3.2 GHz, 1 GB Ram DDR2, 250 GB HDD	Equipo donde se alojará el sistema una vez construido.

5.2 Software

Dentro de las herramientas que apoyan la fase de construcción, tenemos la siguiente lista.

Tabla Nº 2. Descripción software utilizado.

Software	***Función***
Mono	Framework de desarrollo que permite crear y correr aplicaciones en distintos lenguajes (C#, Visual Basic) y sobre distintos sistemas operativos (Linux, Windows, Unix). Básicamente, Mono contiene un set de herramientas como compiladores, una máquina virtual y una librería de clases
MonoDevelop	Entorno de desarrollo Integrado que trabaja sobre el sistema operativo Gnu – Linux.
Nunit	Framework de pruebas de unidad.
Tortoise	Aplicación windows útil para comunicación con el repositorio de código.
C#	Lenguaje de programación orientado a objetos.
Windows XP Professional Edition	Sistema Operativo que permite el uso de las herramientas de desarrollo necesarias.
Visual Studio 2005	Entorno de desarrollo integrado útil para la creación de componentes que podrán ser ejecutados utilizando la máquina virtual de

	mono.
Módulo de mono para Apache	Un módulo que permite al servidor web Apache poder servir aplicaciones escritas para ASP.Net
XSP	Servidor web escrito en C#
Ubuntu Gnu-Linux	Distribución Linux que alojará la aplicación durante su etapa de desarrollo.
Apache	Servidor Web utilizado para ejecutar la aplicación.
Subversión	Repositorio de código.
MagicDraw Uml	Aplicaciones para creación de modelos (casos de uso, clases, interacción, entre otros).
Power Designer	Utilizado para el diseño de la base de datos.
MySql	Sistema gestor de base de datos.
MySqlConnector	Proveedor de ADO.Net para MySql
Navicat Lite for MySQL	Administrador de base de datos para MySql.
Debian Gnu-Linux	Distribución Linux destinado a ser el servidor de aplicaciones. Una vez finalizado el proyecto, éste alojará la aplicación Web en cuestión.

Los criterios de elección de las herramientas arriba comentadas han sido la experiencia en el uso, junto con la preferencia por la utilización de software libre.

6. Inicio del Proyecto

El proyecto surge como una necesidad de poder llevar la traducción de aplicaciones Linux a la web. Esta necesidad cumple con entregarles a los usuarios traductores, un sistema en donde puedan editar sus archivos de localización de tal manera de poder llevar una aplicación específica a múltiples idiomas.

6.1 Investigación previa

6.1.1 Localización de aplicaciones y conceptos relacionados

En el último tiempo, la localización ha pasado de ser un esfuerzo realizado por algunas empresas proveedoras de software, a convertirse en una necesidad y en muchos casos un factor clave para el éxito y la aceptación internacional de productos software.

Muchas veces el término localización puede llevar a múltiples definiciones pero en términos generales, localización es la traducción y adaptación de un producto software, el cual incluye la aplicación software en sí misma y todo lo relacionado con la documentación. El término "localización" se deriva de la palabra "locale", que tradicionalmente significa una pequeña zona. Hoy en día, “locale” se asocia o es usada en un contexto más técnico, donde éste

representa una combinación específica de lenguaje, región y juego de caracteres.

Diferencias típicas entre localización y traducción es el hecho que traducción es típicamente una actividad hecha después que el documento origen haya finalizado, mientras que los proyectos de localización realizan las tareas de desarrollo y traducción de forma paralela de tal forma de permitir la salida del software en múltiples lenguajes.

6.1.2 Internacionalización

LISA [Lisa2008], acrónimo del inglés *the localization industry standards association,* define internacionalización de la siguiente manera:

"Internacionalización es el proceso de generalizar un producto de tal manera de manejar múltiples lenguajes y convenciones culturales sin necesidad de rediseño. Internacionalización toma lugar a nivel del diseño del programa"

En general un programa software es internacionalizado durante el ciclo de desarrollo del programa y corresponde a un antecesor al proceso de localización. Un aspecto importante de la internacionalización es la separación del texto desde el código fuente de la aplicación, es decir, mover todas aquellas cadenas de caracteres a un archivo de recursos. Con este tipo de

procedimientos se previene corrupción del código fuente, porque los archivos de recursos sólo contienen cadenas traducibles y no código fuente.

6.1.3 Localización

LISA [Lisa2008], define localización de la siguiente manera:

"Localización implica tomar un producto y hacerlo cultural y lingüísticamente apropiado al locale destino (ciudad / región y lenguaje) donde se va a usar".

6.1.4 Globalización

LISA [Lisa2008], define globalización de la siguiente manera

"Globalización se refiere a los asuntos asociados con tomar un producto global. Dentro de la globalización de productos de alta tecnología, esto implica integrar localización dentro de la compañía, después de una apropiada internacionalización y diseño del producto, marketing, ventas y soporte dentro del mercado mundial".

6.1.5 Traducción

Traducción es el proceso de convertir texto escrito a otro lenguaje. Esto requiere que el significado completo del texto origen sea llevado con precisión al lenguaje destino. Todo esto con especial atención a los matices culturales y

de estilo. De esta manera las diferencias entre traducción y localización puede ser definida como:

“La traducción es sólo una de las actividades de localización, además de la traducción, una proyecto de localización incluye muchas otras tareas como la administración de proyectos, ingeniería de software, pruebas y publicación”

6.2 La librería Gnu-Gettext

Gnu gettext es un conjunto de herramientas y/o librerías que apoyan el proceso de internacionalización y localización de GNU. Estas herramientas incluyen una serie de convenciones acerca de cómo los programas deben ser escritos y como deben ser llamados y organizados los ficheros y los directorios.

Dentro del proceso de Gnu – gettext se utilizan 3 tipos de archivos:

- Los archivos POT, que hacen referencia a Portable Object Template, que son los archivos base para la generación de archivos PO.
- Los archivos PO, que hace referencia a Portable Object y que son entendibles y editables por humanos. La tarea de estos archivos es asociar cada cadena original y traducible, que provenga del código fuente, con su traducción a una lengua determinada. Es decir un archivo PO individual está dedicado a la traducción de un único lenguaje.

- Los archivos MO, que hacen referencia a Machine Object, y que son archivos binarios que son leídos por el programa ejecutable.

6.2.1 Archivos POT

Cuando el archivo PO se genera utilizando las herramientas gettext, existe exactamente un espacio en blanco entre cada entrada. Con respecto a los comentarios, existen de dos tipos: aquellos en los que el símbolo # está seguido inmediatamente de un espacio en blanco (creados y mantenidos exclusivamente por el traductor), y aquellos en los que el símbolo # está seguido de algún carácter que no sea un espacio en blanco, que son creados y mantenidos automáticamente por *gettext*.

Después tendremos dos cadenas, correspondientes la primera a la cadena original que aparece en el código fuente, y la segunda a la traducción de esta cadena. La cadena original es la que sigue a *msgid*, mientras que la traducción es la que sigue a *msgstr*. Las primeras son creadas y mantenidas por gettext, y las segundas son de las que se tiene que hacer cargo el traductor. Los archivos con esta extensión serán tomados como archivos base y a partir de estos mismos se construirán los archivos PO correspondientes a cada lenguaje a los que se quiera llevar una aplicación.

6.2.2 Archivos PO

El contenido de un archivo PO será un reflejo de su archivo base o archivo POT. Los archivos con esta extensión serán los que finalmente contendrán las cadenas traducidas para un determinado leguaje (ejemplos: message-es.po, message-en.po).

6.2.3 Archivos MO

Los archivos MO, serán los responsables de entregar al programa ejecutable la traducción de una determinada cadena de caracteres.

De esta manera y siguiendo el ejemplo anterior, dentro de los proyectos de software libre existen comunidades de traductores que de manera desinteresada se unen para poder llevar una aplicación a múltiples idiomas. La responsabilidad de quien traduce el o los archivo(s) es de unas pocas personas conocidas como coordinadores. Junto con lo anterior cada traductor utilizará el editor que más le acomode para realizar su trabajo. Entre los traductores más conocidos se encuentran gtranslator y kbabel.

6.3 Localización utilizando una herramienta Web

Visto un poco de qué se trata la internacionalización y localización en GNU, existen aún posibilidades no exploradas a lo que respecta del entorno utilizado para llevar la localización de aplicaciones. Es así, como el alumno, al finalizar este seminario entregará una herramienta web de fácil uso para administración de archivos de localización.

6.3.1 La administración de Proyectos de Localización

Como cada aplicación tiene sus propios archivos de localización, esta herramienta organizará los mismos en entidades llamadas proyectos de localización, donde cada proyecto será un contenedor de archivos “pot”, ”po” y “mo”. La siguiente figura muestra lo antes explicado.

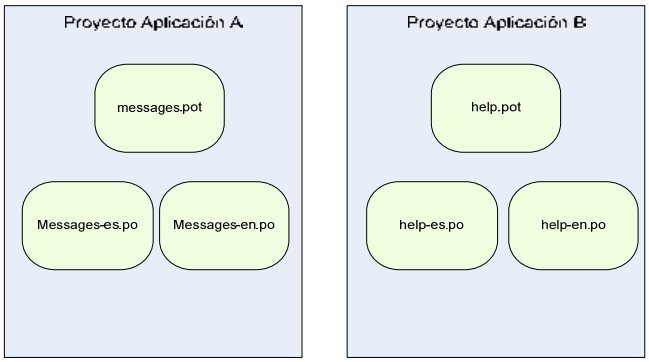


Figura Nº 3. Organización de archivos de localización.

6.4 Definición de Requerimientos

- Se requiere que permita administrar proyecto de traducción; crearlos, modificarlos y eliminarlos.
- Se requiere que permita administrar usuarios y sus respectivos permisos sobre cada proyecto de traducción.
- Se requiere poder administrar los archivos de localización asociados a cada proyecto registrado en el sistema. Crear, editar y eliminar.
- Se requiere entregar información relacionada con el avance en la localización de un proyecto.

Al investigar más sobre las utilidades contenidas en la libreria Gnu-Gettext, se encuentra la lista contenida en la tabla 3, la cual entrega una descripción sobre qué trabajo realizan y cómo pueden ser útiles para este proyecto.

Tabla Nº 3. Comandos utilizados por la herramienta.

Comando	Descripción
Msginit	Crea un nuevo archivo po.
Msgmerge	Crea un nuevo archivo po a partir de la unión de dos archivos po.
Msgfmt	Genera un archivo binario de extensión mo. También conocido como archivo mo.

6.4.1 Consideraciones para el desarrollo

- Los proyectos de localización contendrán archivos que cumplan con el estándar Gnu-gettext.
- La aplicación se comunicará con la biblioteca Gnu-Gettext para poder ejecutar comandos útiles para el manejo de archivos de localización.

6.5 Entorno de desarrollo

Luego de expuestos los antecedentes relacionados con la localización de aplicaciones utilizando la librería Gnu-Gettext, se hizo una búsqueda de los utilitarios necesarios para la construcción de la herramienta.

6.5.1 Framework de Desarrollo de Aplicaciones

Todo el código fuente escrito para la herramienta en cuestión será construido sobre el Framework 2.0, pero finalmente será ejecutado sobre el “RunTime” o entorno de ejecución de Mono. Una excepción a la regla de codificación, será el uso de las librerías de Mono, para la compilación y creación de este componente que tenga directa comunicación con las librerías Gnu- Gettext.

6.5.2 Framework de Pruebas de Unidad

Como el desarrollo de este proyecto se basó en TDD, se debió buscar un framework de pruebas de unidad que cumpla con la portabilidad de los tests desde una estación de desarrollo con Windows XP hacia una estación Gnu – Linux; esto con el fin de revisar y poder corroborar la correcta ejecución del código escrito, debido a posible faltas de implementaciones en el Run-Time de Mono.

Dentro de las alternativas de frameworks de pruebas de unidad se encontraron NUnit y Mbunit, los cuales se probaron en distintas plataformas (Windows y Linux). Finalmente la elección recayó en NUnit, debido al soporte multiplataforma que éste ofrece.

A continuación una breve descripción de la herramienta escogida: “NUnit es un framework de pruebas de unidad para todos los lenguajes .Net. Inicialmente portado desde JUnit, su actual versión de producción es la 2.4. Esta escrita enteramente en C# y ha sido completamente rediseñado para tomar ventaja de muchas de las características de los lenguajes de .Net”.

6.5.3 Repositorio de Código

Aunque, el siguiente proyecto será desarrollado por una única persona, se espera que en futuras versiones sean involucrados más desarrolladores. Es por esto que durante el desarrollo de la herramienta se utilizará el sistema de control de código “subversión”.

Una breve introducción de Subversión:

“Subversión es un sistema de control de versiones Open Source. Subversión administra archivos y directorios, y los cambios hechos sobre ellos todo el tiempo. Esto permite recuperar viejas versiones de tu información, o examinar la historia de cómo tu información ha cambiado”

La estructura de directorios para esta herramienta se entregará en capítulos posteriores del proyecto.

Como el fin de este documento no es dar una explicación de fondo relacionada con la herramienta Subversión, el lector puede profundizar en el tema de “Sistema de control de versiones” dirigiéndose directamente a su sitio oficial [Subversion2008].

6.5.4 Entorno de Desarrollo Integrado y Complementos

Para las primeras etapas del desarrollo, se utilizó Visual Studio 2005 en su versión Express Edition. Luego para la programación de componentes que interactúan con librerías de GNU-Gettext se utilizó el IDE MonoDevelop en su versión 1.0.

6.5.5 Servidor Web

El servidor web utilizado para servir la aplicación fue Apache en su versión 2.2.8. Junto con este servidor se requirió el módulo de apache para mono, el cual se hace necesario para ejecutar la herramienta en cuestión. Para más información relacionada con la ejecución de aplicaciones ASP.Net por favor visitar el sitio oficial del proyecto Mono [Mono2008].

6.5.6 Sistema Gestor de Base de datos

El sistema gestor de Base de datos escogido fue Mysql en su versión 5.0. Algunas de las razones para la elección del mismo fueron:

- Open Source, con una comunidad que entrega soporte en temas como instalación, mantención y desarrollo.

- Existen más de un cliente gráfico para la administración de las base de datos contenidas en el servidor.
- Compatible para la instalación en sistemas operativos del tipo Linux.
- Documentación accesible.

Herramientas anexas y necesarias para el desarrollo sobre este motor fueron las siguientes:

- MySqlConnector: Proveedor ADO .Net para Mysql.
- Navicat Lite for MySql: Administrador de base de datos.

6.5.7 Sistemas Operativos

La etapa de construcción de la aplicación en su gran parte fue realizada en el ambiente Windows. Para los casos de componentes de interacción con librerías Gnu-Gettext se utilizó Linux Ubuntu en su versión 7.10.

Para la ejecución de la aplicación, es decir, para el entorno de ejecución, se utilizó la versión Sarge de Debian.

6.6 Modelado Inicial

La disciplina de modelado para la fase de Inicio dentro de la metodología correspondió a un modelo de dominio de la aplicación el cual es representado utilizando el lenguaje unificado de modelado. Con esto se consiguió capturar las principales entidades involucradas en el sistema. El diagrama 1 muestra un modelo de clases desde la perspectiva de “Conceptual” descrita en [Fowler1997].

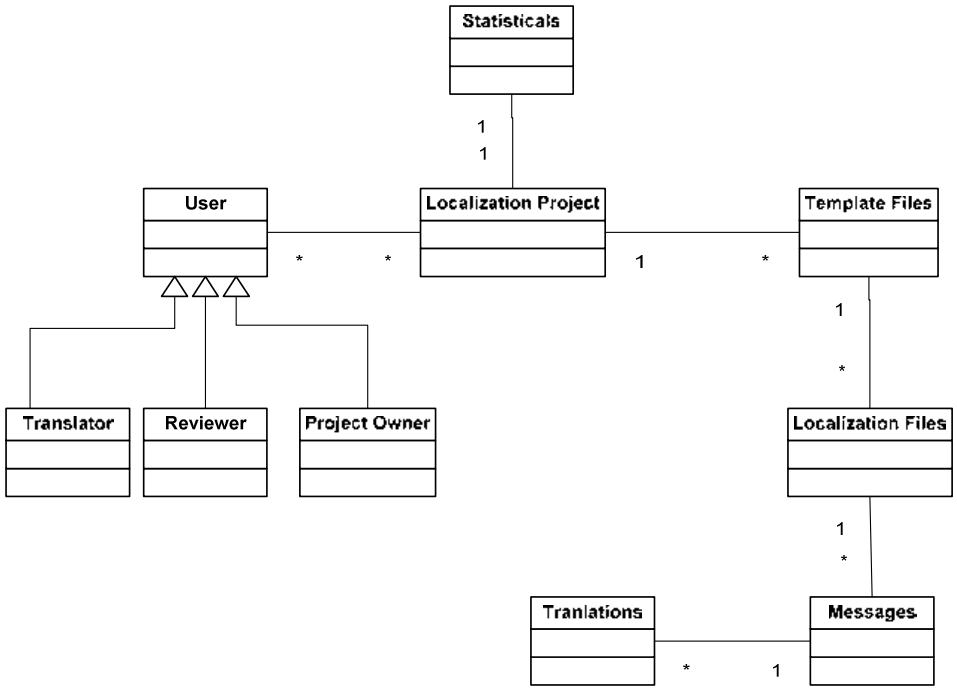


Diagrama Nº 1. Modelo de Dominio

Además del diagrama anterior, se utilizaron diagramas de casos de usos para adquirir una lista de funcionalidades de la herramienta.

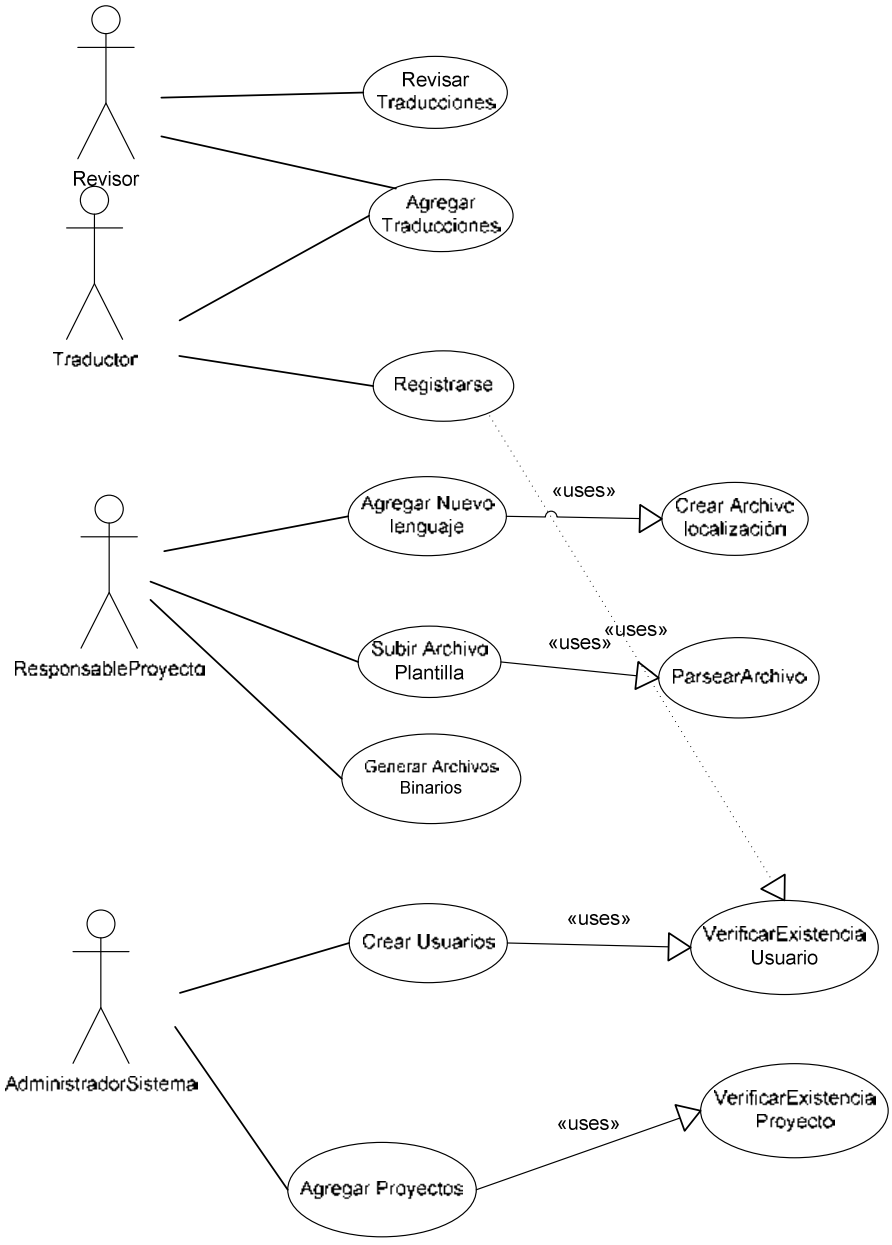


Diagrama Nº 2. Casos de Uso.

6.7 Administración de la Configuración

Una vez que se definió la herramienta Subversión, se necesitó crear una estructura de archivos y directorios, con el fin de poder llevar un orden de la información contenida en este proyecto. A continuación, se entrega una descripción de cada carpeta.

Tabla Nº 4. Descripción de Carpetas

Nombre Directorio	**Descripción**
Database	Se guardarán los diagramas para la base de datos (modelo conceptual, modelo físico).
Deployment	Scripts para la instalación y ejecución de pruebas sobre el sistema.
Docs	Documentación relacionada con el proyecto: diagramas de clases, casos de uso, entre otros.
SW	Código fuente de la aplicación.

A continuación se entrega un diagrama con la estructura de directorios.

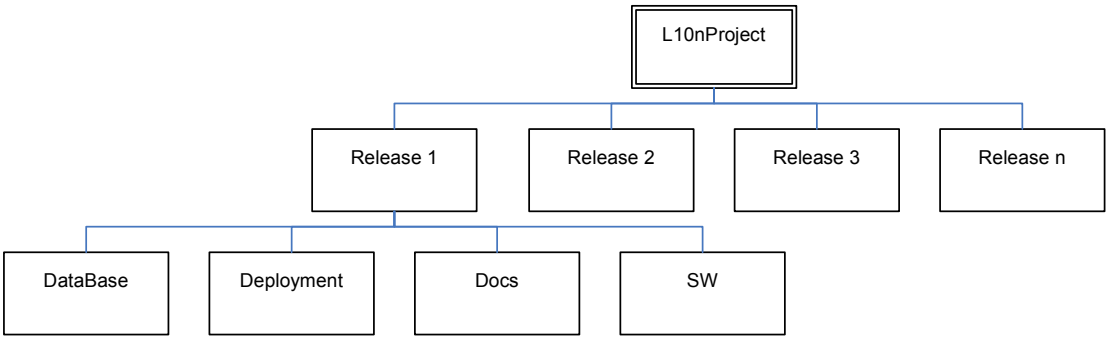


Figura Nº 4. Estructura de directorios en el repositorio de código

Dependiendo de la cantidad de versiones que la herramienta pueda tener, se creó un directorio padre el cual contendrá carpetas con el prefijo "*Release*"; donde cada directorio *Release N* representará una versión. La figura muestra esto con más detalles.

La vista de la carpeta *"SW"* dentro del repositorio de código sería como muestra la figura 5.

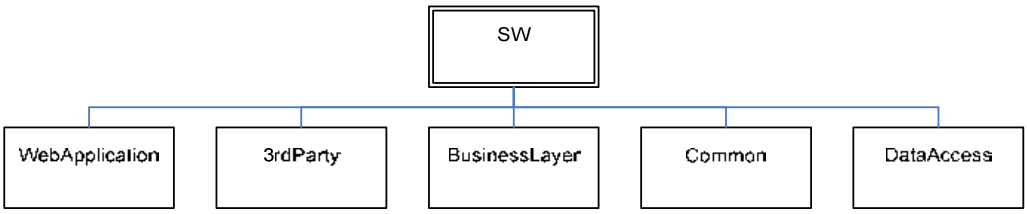


Figura Nº 5. Carpeta *"SW"*

7. Elaboración

7.1 Modelado

Una vez identificado entidades dentro del modelo de clases concebido en la fase anterior, se crea un diseño más elaborado del mismo para así obtener atributos, multiplicidad y navegabilidad.

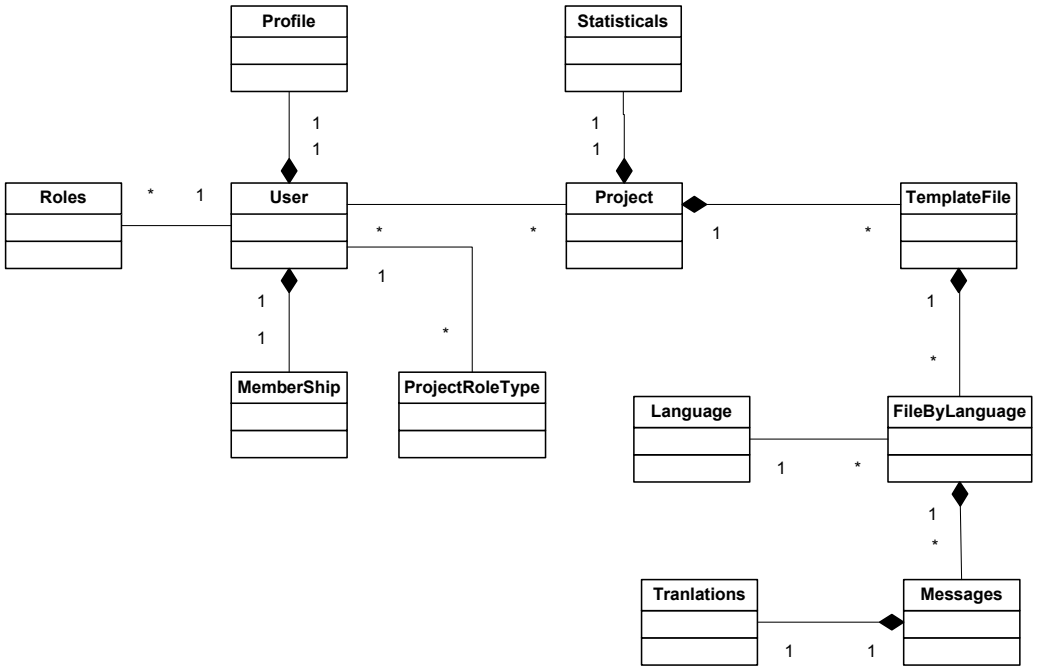


Diagrama Nº 3. Modelo de dominio completo

Para no perder la legibilidad del modelo, la descripción de cada clase y sus atributos se entrega en el siguiente apartado.

7.1.1 Especificación de Clases y Relaciones

En las siguientes tablas se entrega una descripción de cada una de las clases que conforman el modelo de dominio.

Tabla Nº 5. Descripción de clases

Nombre Clase	**Descripción**
Project	Es la entidad contenedora de archivos de localización POT y PO.
User	Corresponde a los usuarios que acceden al sistema.
Roles	Corresponde a los posible roles que un usuario pueda tener dentro del sistema (administrador de sitio o usuario común).
MemberShip	Contendrá información relacionada con el login de cada usuario.
ProjectRoleType	Corresponde a los posibles roles que un usuario pueda tener en un proyecto (Administrador de proyecto, revisor o

	traductor)
Profile	Contendrá datos del usuario registrado.
TemplateFile	Representa un archivo de localización POT.
Statisticals	Contendrá las estadísticas de un proyecto.
Language	Representa los lenguajes a los que puede ser llevado una aplicación.
FileByLanguage	Representa un archivo de localización PO.
Messages	Representa una lista de mensajes que corresponden a las cadenas de caracteres traducibles.
Translations	Contendrá una lista de traducciones posibles para cada mensaje.

Tabla Nº 6. Asociaciones del modelo

Nombre asociación	Descripción	Multiplicidad
User - Project	Indica que un usuario puede navegar o estar asociado a una o más instancias de Project y viceversa.	*..*
FileByLanguage - Language	Indica que una instancia de FileByLanguage puede estar asociada a una única instancia de Language.	1..*
User –Roles	Una instancia de User se asociará a una única instancia de Roles y una instancia de la clase Roles se podrá asociar con más de una instancia de User.	*..1
User - ProjectRoleType	Una instancia de User	*..*

	contendrá una o más instancias de ProjectRoleType y viceversa.	

Las navegabilidades para las asociaciones descritas en la tabla anterior corresponden a un enfoque bidireccional.

Tabla Nº 7. Composiciones del modelo

Nombre composición	Descripción	Multiplicidad
User - Profile	Indica que una instancia de User puede contener una única instancia de Profile.	1..1
Project - Statisticals	Indica que una instancia de Project puede contener una única instancia de Statisticals.	1..1
TemplateFile -	Indica que una instancia	1..*

FileByLanguage	de TemplateFile contendrá una o más instancias de FileByLanguage.	
FileByLanguage - Messages	Indica que una instancia de FileByLanguage contendrá una o más instancias de Messages.	1..*
Messages - Translations	Indica que una instancia de Messages contendrá una o más instancias de Translations.	1..*
User - MemberShip	Una instancia de User contendrá una única instancia de la clase Membership.	1..1

7.1.2 Especificación de atributos

A continuación se entregan los atributos identificados en cada una de las clases participantes del modelo de dominio.

Tabla Nº 8. Atributos clase "Project"

Nombre Atributo	Tipo	Descripción
ProjectName	Alfanumérico	Nombre del Proyecto
Description	Alfanumérico	Una descripción del proyecto.

Tabla Nº 9. Atributos clase "User"

Nombre Atributo	Tipo	Descripción
Name	Alfanumérico	Nombre de Usuario
isAnonymous	Booleano	Indica si es anónimo
LasActivityDate	Fecha	Fecha de última actividad

Tabla Nº 10. Atributos clase “Roles”

Nombre Atributo	Tipo	Descripción
Name	Alfanumérico	Nombre del Rol

Tabla Nº 11. Atributos clase “MemberShip”

Nombre Atributo	Tipo	Descripción
Email	Alfanumérico	Correo electrónico del usuario
Comment	Alfanumérico	comentario
Password	Alfanumérico	Clave de ingreso
PasswordFormat	entero	Formato de la clave.
PasswordQuestion	Alfanumérico	Pregunta de seguridad en caso de pérdida de password.
PasswordAnswer	Alfanumérico	Respuesta a pregunta seguridad. Útil en la pérdida u olvido de password por parte del usuario.

IsApproved	Booleano	Es aprobado.
LastActivityDate	Fecha	Fecha de última actividad.
LastLoginDate	Fecha	Fecha de último ingreso al sistema.
LastPasswordChangedDate	Fecha	Ultima fecha en la que se cambió el password
CreationDate	Fecha	Fecha en la que se creó el usuario
IsLockedOut	Booleano	Indica si el usuario está bloqueado
LastLockedOutDate	Fecha	Fecha del último bloqueo realizado al usuario

Tabla Nº 12. Atributos clase "ProjectRoleType"

Nombre Atributo	Tipo	Descripción
Description	Alfanumérico	Descripción del tipo de rol válido para un usuario dentro de un proyecto (Administrador, revisor, traductor)

Tabla Nº 13. Atributos clase "Profile"

Nombre Atributo	Tipo	Descripción
FullName	Alfanumérico	Nombre Completo.
Gender	Gender	Género del usuario.
Country	Alfanumérico	País del usuario.
NativeLanguage	Language	Lenguaje nativo del usuario.
Theme	Alfanumérico	Diseño de tema escogido para trabajar con la herramienta.

Tabla Nº 14. Atributos clase "TemplateFile"

Nombre Atributo	Tipo	Descripción
Name	Alfanumérico	Nombre del archivo del plantilla.
ParentProject	Project	Proyecto padre al que pertenece el archivo.
Path	Alfanumérico	Ruta de acceso al archivo.
IsOld	Booleano	Indica si este archivo está actualizado o no.

Tabla Nº 15. Atributos clase "Statisticals"

Nombre Atributo	Tipo	Descripción
NumberOfFiles	Entero	Número de archivos para el proyecto.
CompletedPercentage	Entero	Porcentaje de traducciones completadas.
NumberOfTraslators	Entero	Número de traductores

		activos en el proyecto.

Tabla Nº 16. Atributos clase “Language”

Nombre Atributo	Tipo	Descripción
Description	Alfanumérico	Nombre del lenguaje
IsSystem	Booleano	Indica si el lenguaje es utilizado por el sistema.
LanguageCode	Alfanumérico	Nombre del código de lenguaje.

Tabla Nº 17. Atributos clase “FileByLanguage”

Nombre Atributo	Tipo	Descripción
Name	Alfanumérico	Nombre del archivo de localización PO.

Tabla Nº 18. Atributos clase “Messages”

Nombre Atributo	Tipo	Descripción
Msgid	Alfanumérico	Corresponde a la clave extraída desde el archivo de localización PO.
Msgstr	Alfanumérico	Corresponde a la traducción del mensaje.
Comment	Alfanumérico	Comentario.
ImagePath	Alfanumérico	Ruta de la imagen asociada a la cadena traducible.

Tabla Nº 19. Atributos clase “Translations”

Nombre Atributo	Tipo	Descripción
AlternativeMessage	Alfanumérico	Corresponde a una alternativa traducción al mensaje extraído desde el archivo de localización po.

Score	Entero	Puntuación de la traducción.

7.2 Diseño de Interfaz de Usuario

Otro punto importante dentro del diseño de la herramienta, fue la creación de una interfaz de usuario amigable y que cumpla con cada una de las reglas de consistencias definidas en [Pressman2005] y que nombran a continuación:

- Toda la información visual es organizada acorde a un estándar de diseño consistente que es mantenido en todas las pantallas de visualización.
- Mecanismos para navegar de una tarea a otra son consistentemente definidos e implementados.
- Los mecanismos de entrada están restringidos a un conjunto limitado que son usados consistentemente a través de la aplicación.

Las interfaces de usuario que correspondieron a páginas web, fueron construidas utilizando los lenguajes y términos incluidos en la tecnología ASP.NET 2.0, HTML, CSS, Skins, Themes y Master Pages.

La figura a continuación muestra una visión general de la interfaz de usuario.

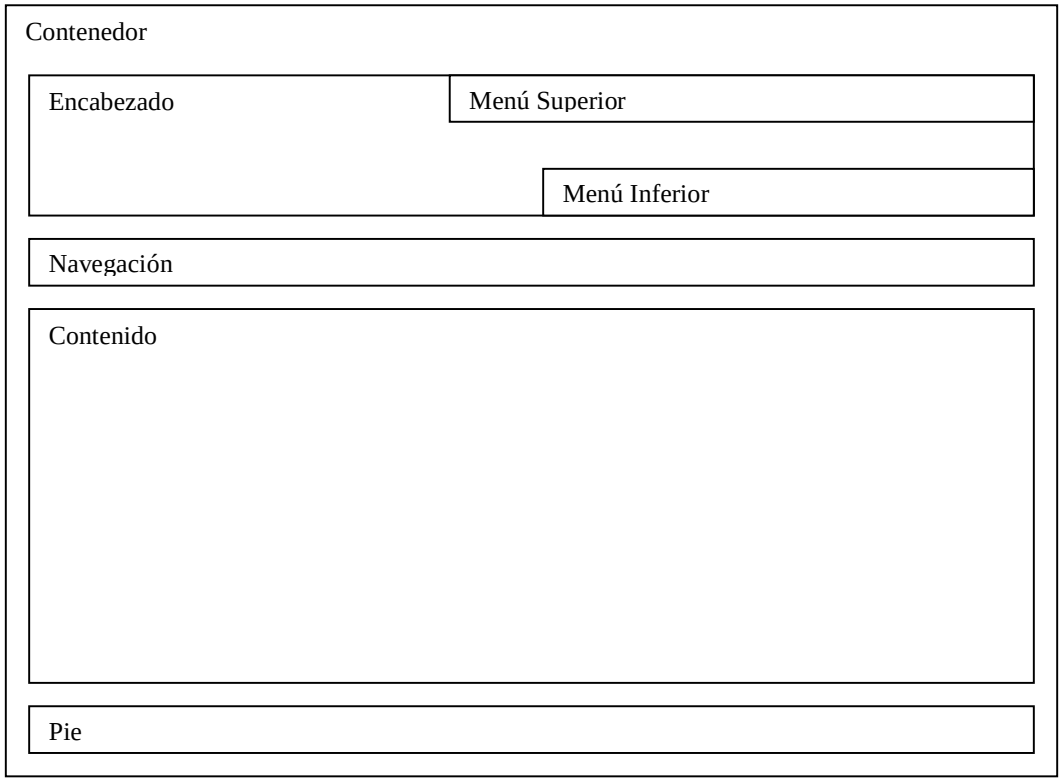


Figura Nº 6. Diseño de interfaz de usuario

La figura anterior se compone de las siguientes capas:

- La capa contenedor incluye las demás capas que conforma en layout de la página.
- La capa encabezado despliega el logo y nombre de la herramienta.

- La capa menú superior despliega un menú informativo útil para usuarios registrados y no registrados.
- La capa menú inferior despliega un menú que cambia de acuerdo a los privilegios de los usuarios registrados.
- La capa navegación despliega la ubicación dentro del árbol de navegación del sistema.
- La capa contenido muestra las páginas web de la aplicación.
- La capa pie muestra contenido informativo sobre herramientas utilizadas durante el diseño de la aplicación web.

Algunas consideraciones en el diseño de la interfaz de usuario:

- La estructura de las páginas fueron implementadas sobre una página maestra o *Master Page*, la nueva característica ofrecida por ASP.NET en su versión 2.0. Una página maestra es una página web que contiene marcas y controles que serán compartidos por todas las páginas de la aplicación.
- El diseño de la página, con lo que respecta a estilos, puede ser fácilmente cambiada utilizando la característica de temas en ASP.NET 2.0, donde un tema es una colección de imágenes, hojas de estilo y skins.

7.2.1. Diseño de pantallas

En las siguientes figuras se muestran el diseño de algunas pantallas importantes de la aplicación.

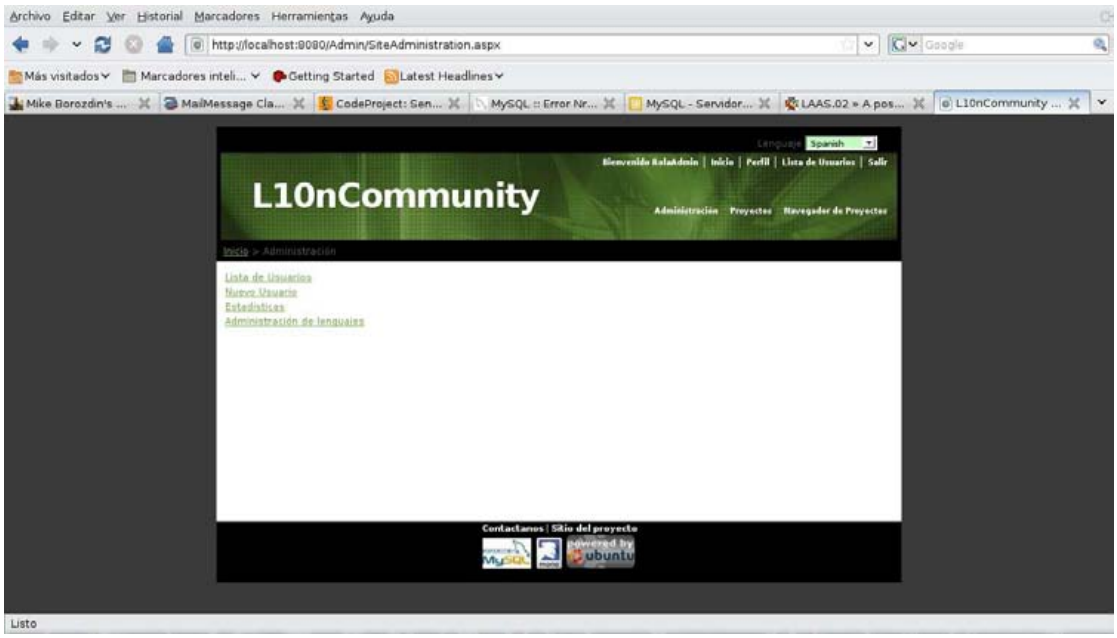


Figura Nº 7. Pantalla de Administrador de sistema

En la figura 7 se muestra la pantalla principal del administrador de sistemas. A partir de esta pantalla se navega a las diferentes páginas destinadas a proveer las distintas funcionalidades. Como se puede notar, en la esquina superior derecha, se encuentra ubicado un control de selección de lenguaje de la

aplicación, de esta forma el usuario puede seleccionar en qué lenguaje quiere ver la aplicación.

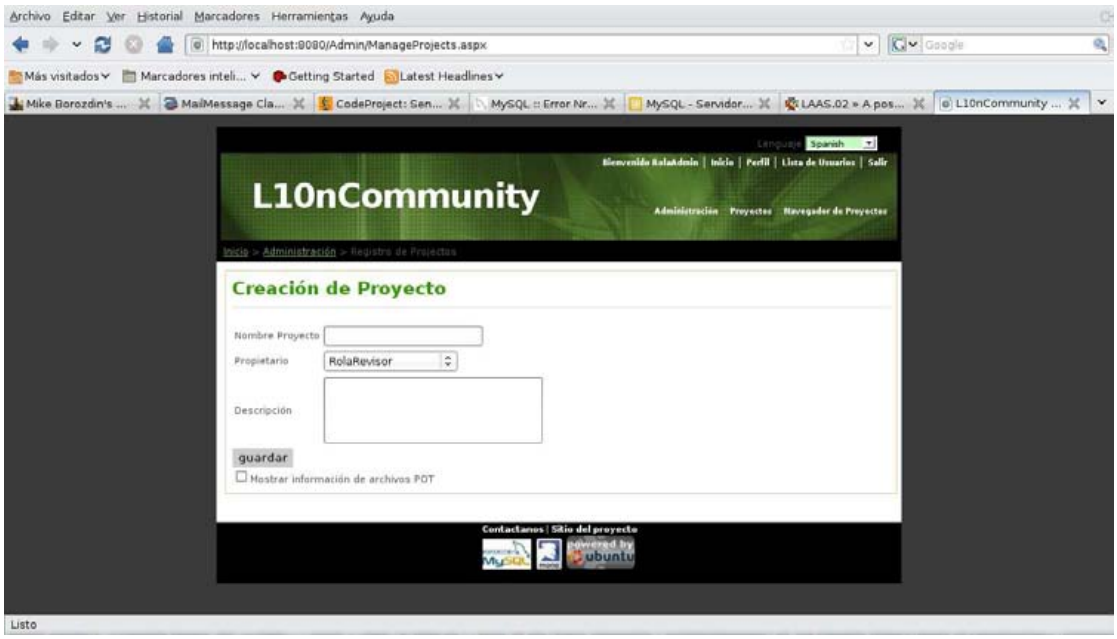


Figura Nº 8. Pantalla de creación de nuevo proyecto.

La figura 8 muestra la pantalla de creación de proyectos, la cual permite asignar un nombre, descripción y un dueño al proyecto en si. Junto con lo anterior esta pantalla permitirá subir archivos de plantilla POT.

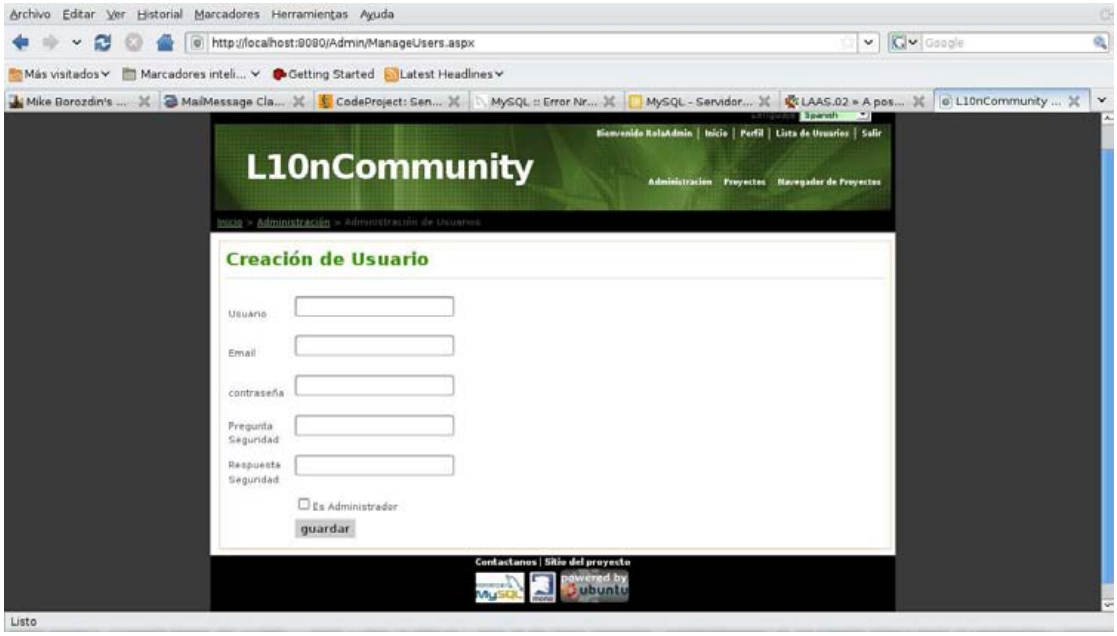


Figura Nº 9. Pantalla de creación de nuevo usuario

La figura 9 muestra la pantalla de creación de usuario por parte del administrador del sistema. El administrador del sitio puede decidir si el usuario a crear es o no un administrador del sistema al igual que él.

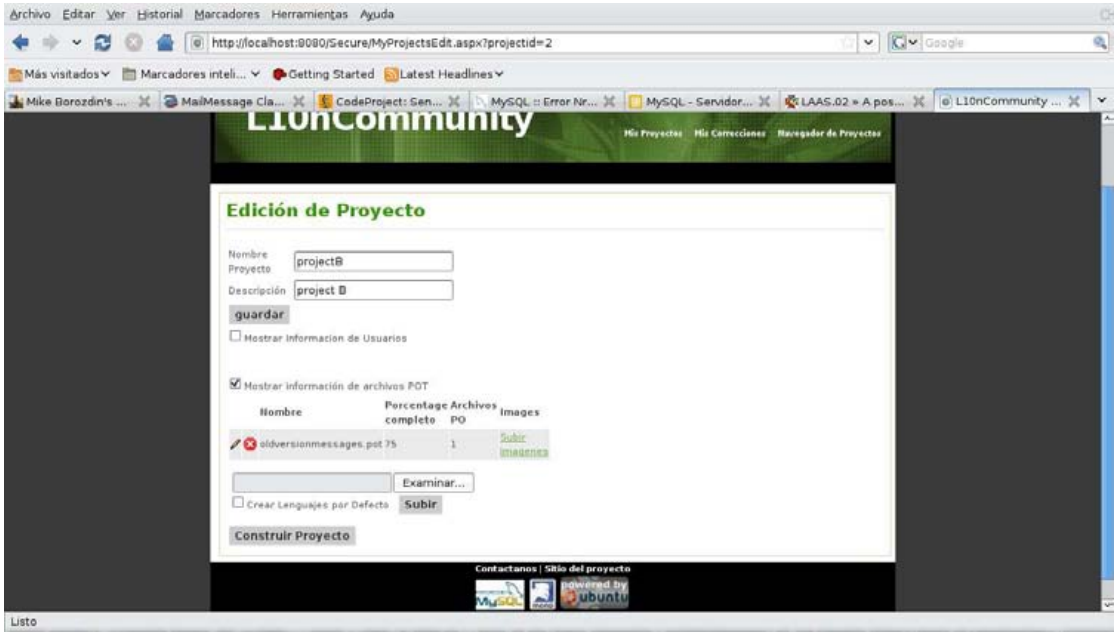


Figura Nº 10. Edición de proyecto

La figura 10 muestra la edición de un proyecto donde se pueden asignar nuevos usuarios revisores y agregar nuevos archivos POT y PO.

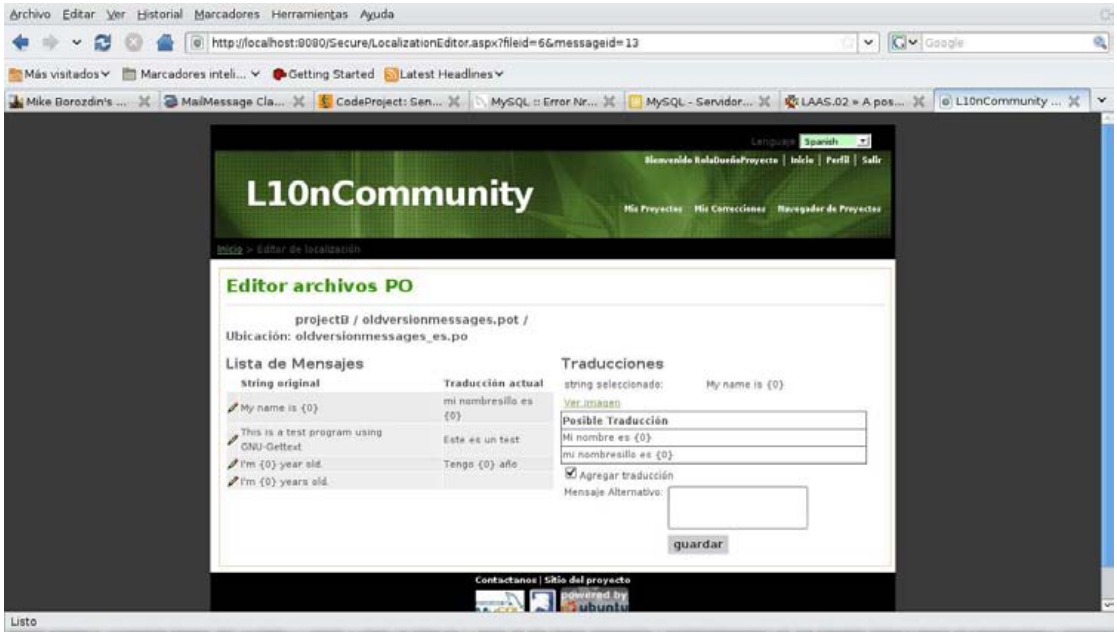


Figura Nº 11. Pantalla de localización de archivos de traducción

La figura 11 muestra la pantalla de traducción para las cadenas extraídas desde un archivo de localización. Se puede apreciar en la imagen dos grillas “Lista de Mensajes” y “Traducciones”. La primera grilla visualiza los mensajes extraídos desde un archivo de localización, la segunda grilla corresponde a las posibles traducciones que una cadena pueda tener. Para agregar una nueva traducción asociada a un mensaje, bastará que el usuario presione sobre “agregar traducción”.

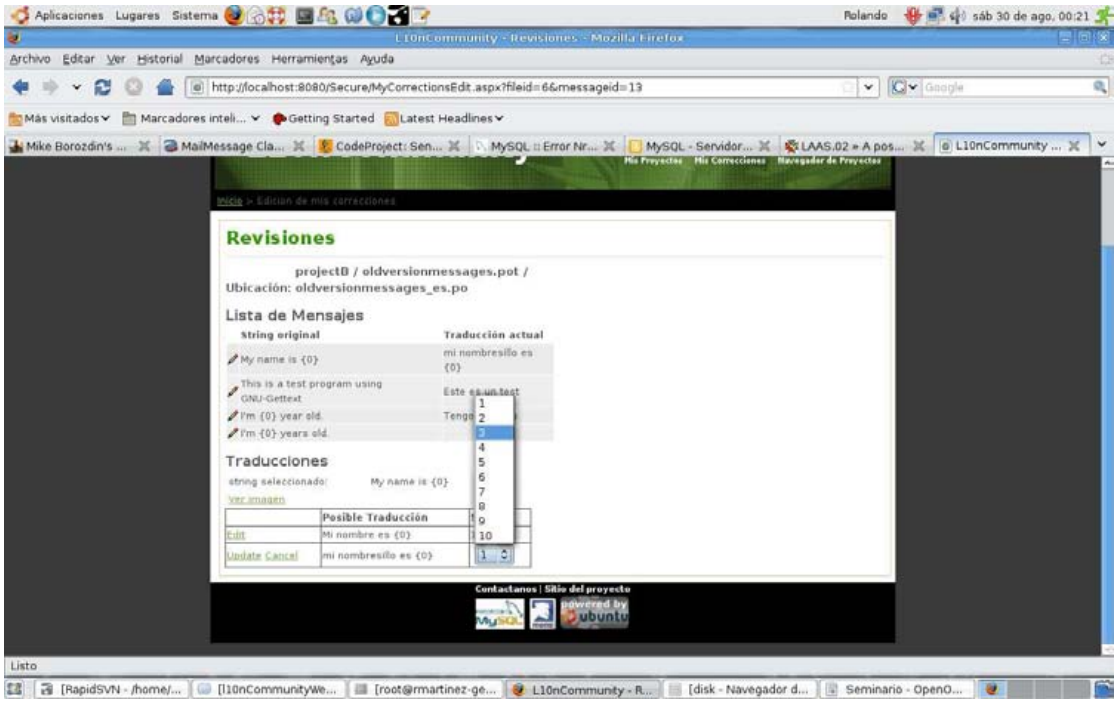


Figura Nº 12. Pantalla de revisión de traducciones.

La figura 12 muestra la pantalla de revisión de las cadenas traducidas, donde el usuario revisor puede asignar una puntuación a cada una de las posibles traducciones asociadas a un mensaje en particular.

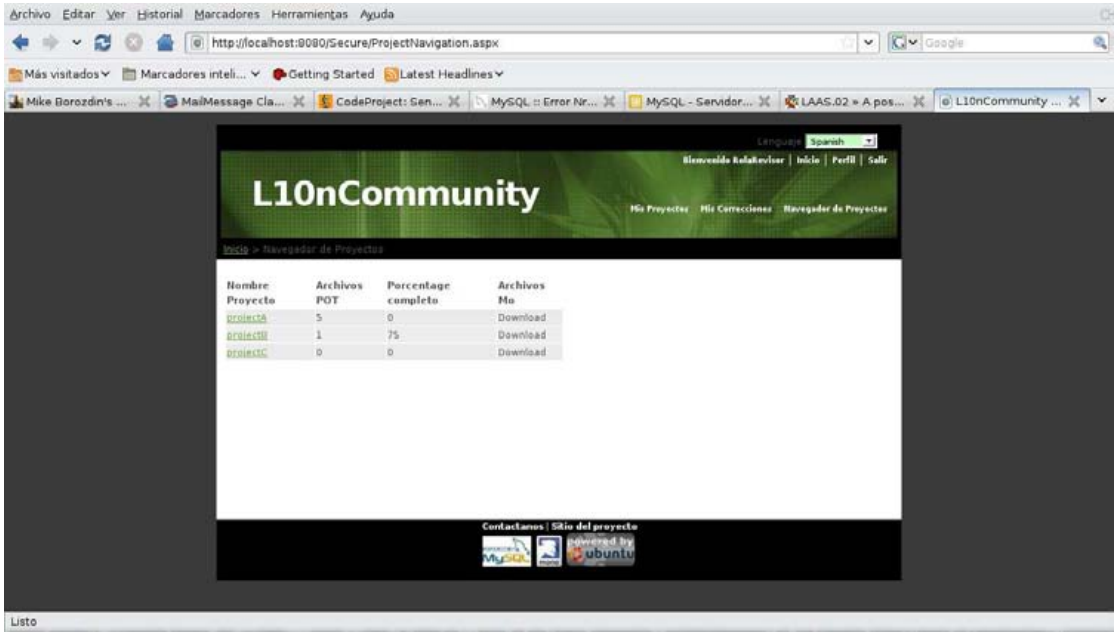


Figura Nº 13. Pantalla Navegación de proyectos.

La figura 13 muestra la navegación de proyectos, la cual se hace necesaria para poder ingresar a la pantalla de edición de un archivo de localización en particular. Las figuras 14 y 15 muestran las pantallas necesarias para poder agregar traducciones a las cadenas asociadas al archivo “messages_es.po”.

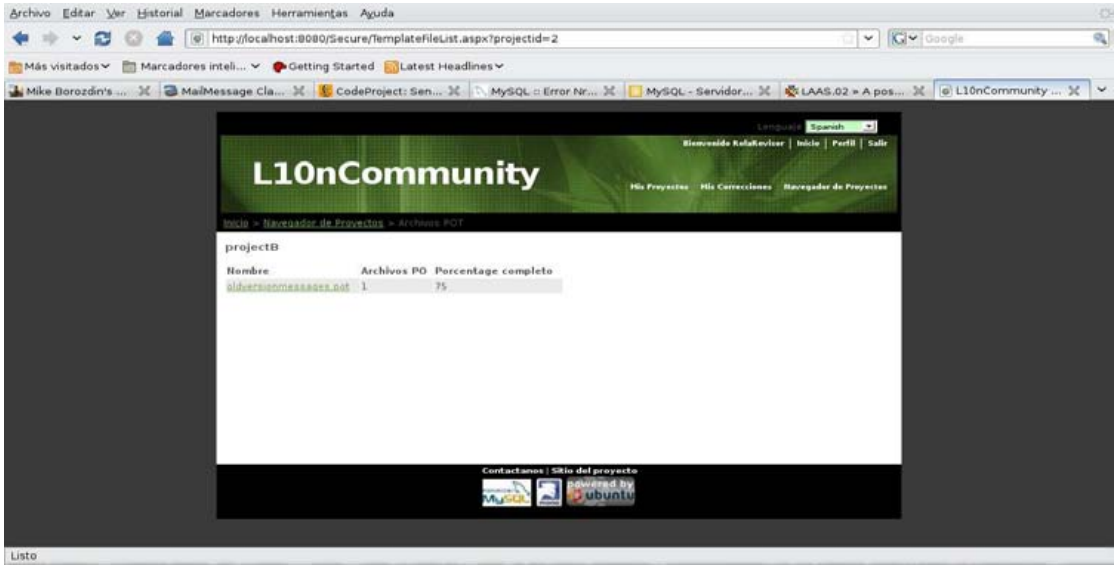


Figura Nº 14. Lista de archivos POT asociados a un proyecto

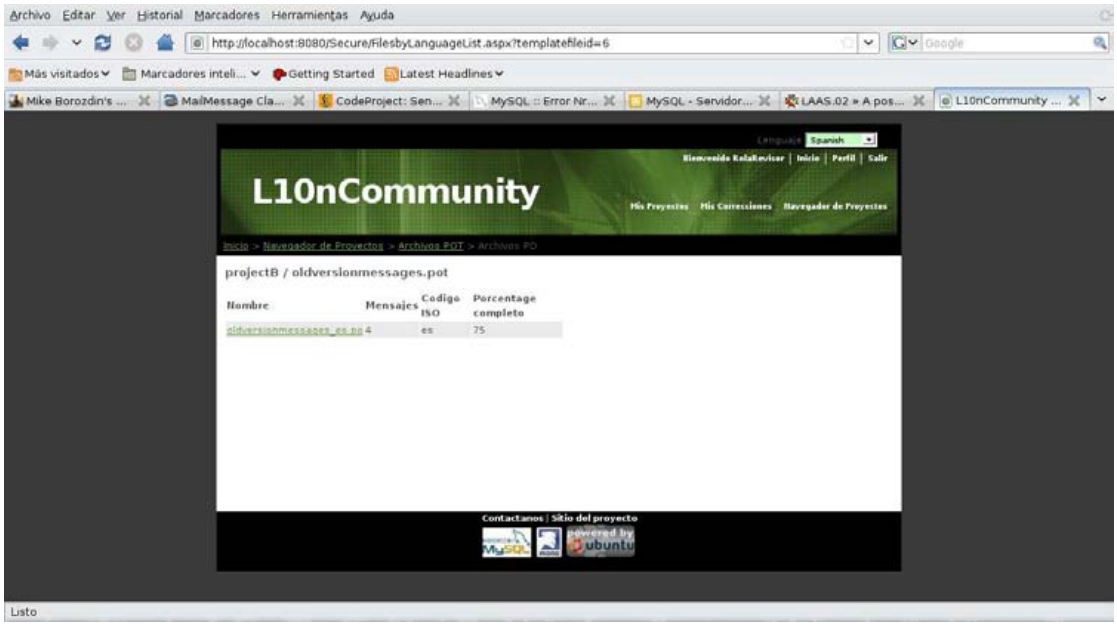


Figura Nº 15. Lista de archivos PO asociados a un archivo POT

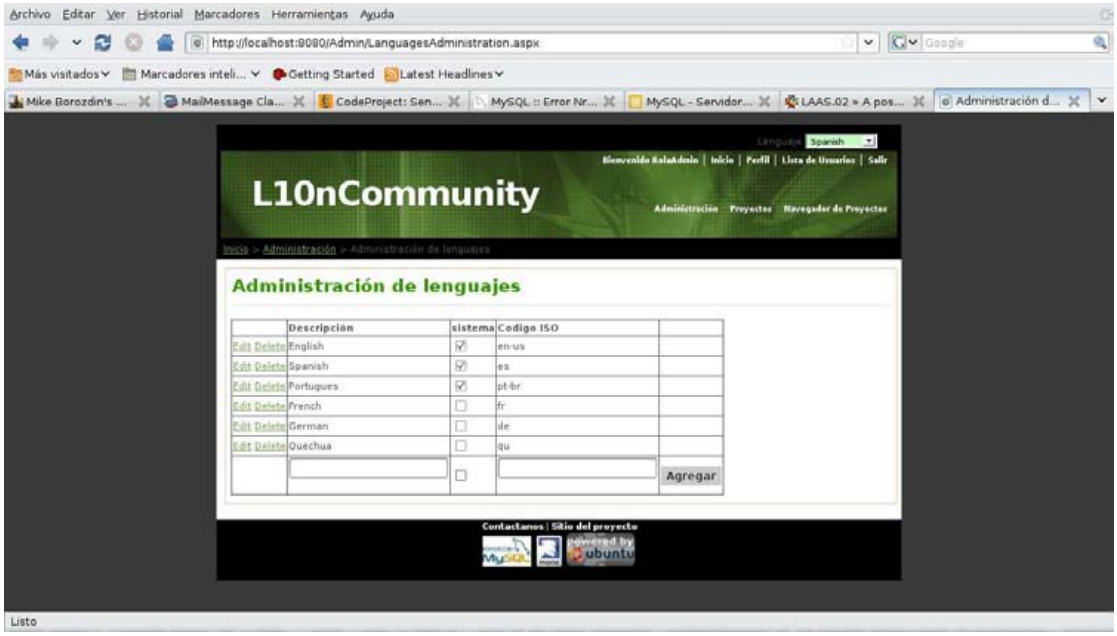


Figura Nº 16. Pantalla de agregación de nuevos lenguajes

En la figura 16 se muestra la pantalla de administración de lenguajes, que permite agregar, eliminar y editar los lenguajes soportados por la aplicación.

7.2.2. Mapa de Navegación de la aplicación

A continuación se entregan dos mapas de navegación, donde se muestran las estructuras organizativas de la aplicación y el camino que debe seguir cada usuario para obtener la funcionalidad requerida.

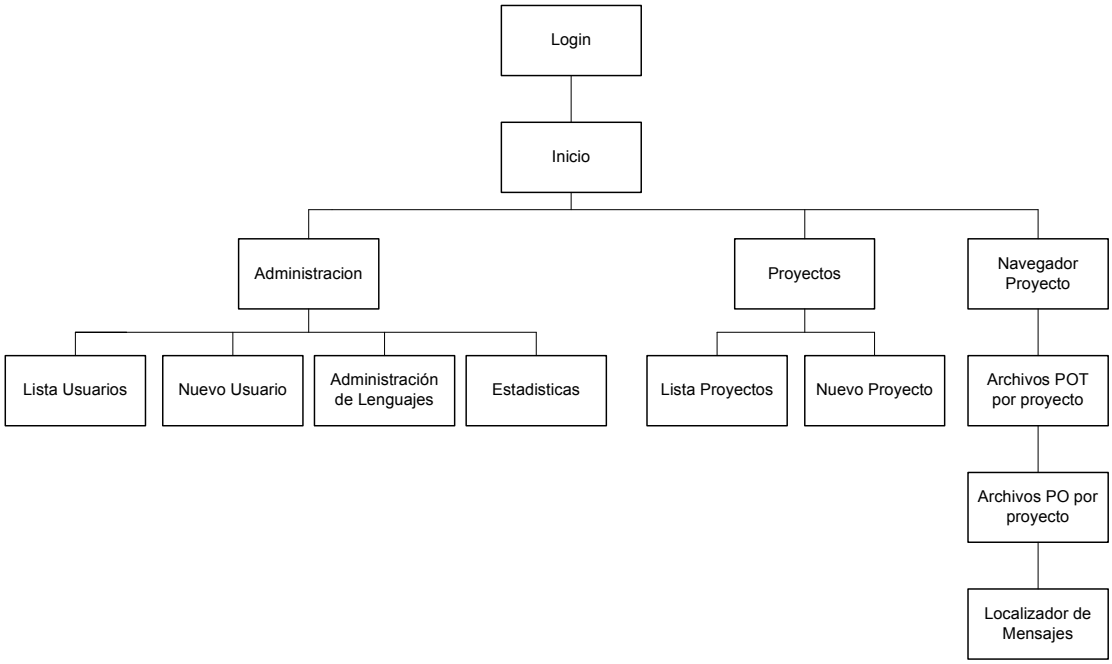


Figura Nº 17. Estructura organizativa para administrador de sistema.

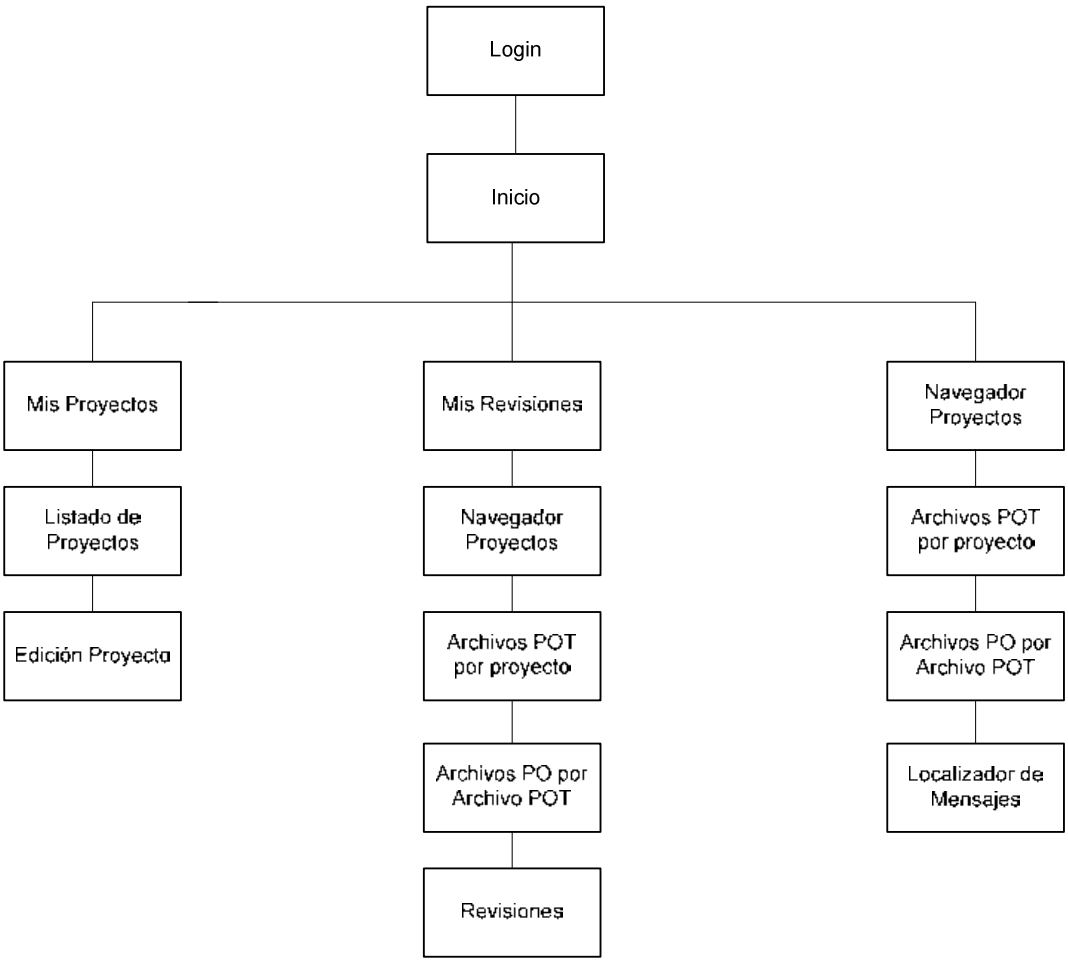


Figura Nº 18. Estructura organizativa para usuario del sistema

7.3. Identificación de Riesgos Técnicos

7.3.1 Parsing de Archivos

El desarrollo de esta actividad contempló la creación o adaptación de un componente capaz de leer archivos de texto plano, que sigan el estándar de archivos pot Gnu-Gettext y luego poder extraer de los mismos las cadenas que serán localizadas. La figura 19 muestra una visión temprana del componente en cuestión.

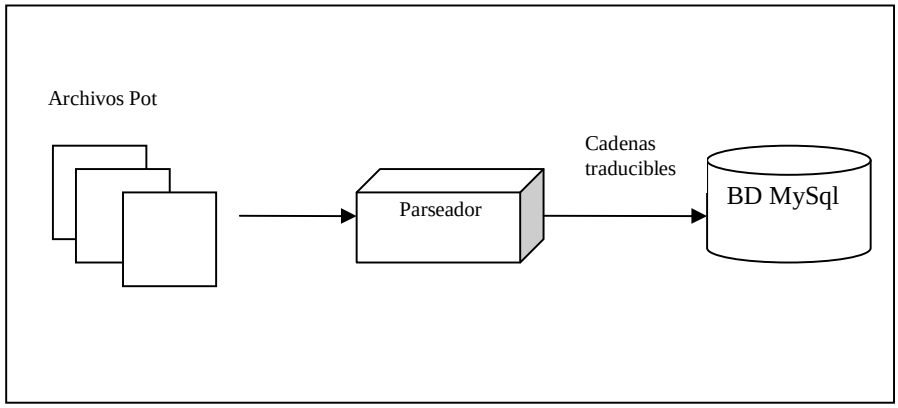


Figura Nº 19. Parseador de Archivos de localización.

7.3.2 Métodos o funciones no implementadas en Mono

Para algunos casos, código fuente escrito en ambiente Windows no podrá ser ejecutado utilizando el entorno de ejecución de Mono, esto debido a la no implementación de algunos tipos (clases) dentro de este Framework. Para evitar problemas relacionados con la ejecución de la herramienta dentro del entorno Linux, se dispuso de revisiones periódicas del código fuente compilado, también conocido como assemblie dentro de la plataforma Microsoft.Net y Mono. El software necesario para estas revisiones fue el analizador de migración de mono (Moma, Mono migration analyzer), el cual ayuda a identificar "casos" que se podrían tener cuando se porta una aplicación desde Windows a Linux.

Para dar solución a los "casos" encontrados utilizando esta herramienta, se re-implementó el fragmento de código con problemas.

7.4 Arquitectura Lógica

La arquitectura del sistema se basa en el ya conocido enfoque de 3 capas, explicado en [Lhotka2006]. Básicamente este tipo de arquitectura permite dividir la aplicación en componentes, entregándole a cada uno responsabilidades distintas. De forma tradicional, en el diseño de la herramienta

se divisaron 3 componentes centrales: acceso a datos, lógica de negocios e interfaz de usuario.

De esta manera la utilización de este tipo de arquitectura en el diseño de la aplicación traerá las siguientes ventajas:

- Código organizado.
- Fácil mantención del código.
- Mejor reutilización de código.
- Una mejor experiencia para el equipo que desarrolla y mantiene la aplicación.

La siguiente figura 20 muestra el modelo de la aplicación con sus respectivas capas.

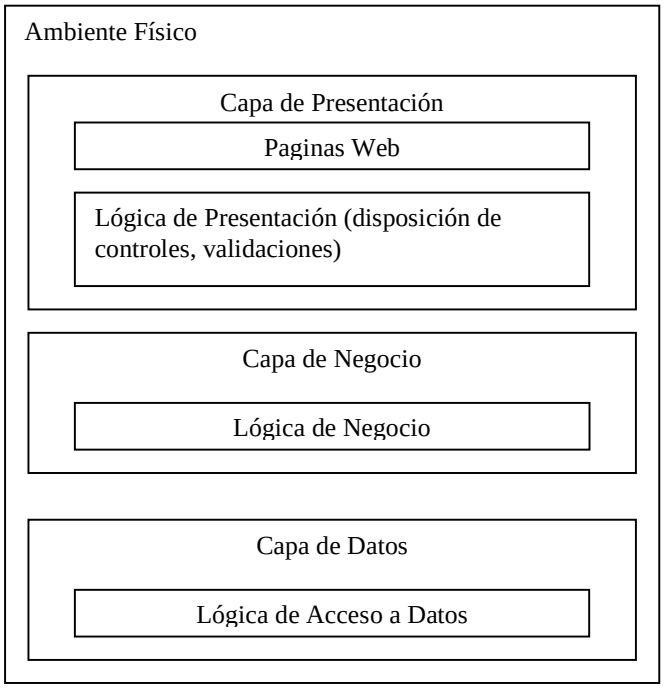


Figura Nº 20. Modelo de la Aplicación

7.4.1 Capa Acceso a Datos y Patrón de Registro Activo

Para la persistencia de las entidades identificadas en el modelo de dominio, se utilizó el enfoque mapeo objeto relacional, ORM, acrónimo del ingles object relational mapping y que se encuentra definido ampliamente en [Ambler2005]. Adicionalmente para implementación de este enfoque se utilizó el patrón empresarial de registro activo o active record, en inglés, descrito en [Fowler2002] y en el anexo A “Patrones de Diseño usados”.

Durante la construcción de esta pieza de software, se pensó en la extensibilidad que la herramienta pueda tener en términos de adición de nuevas tablas a la base de datos, es por esto que se diseñó un mapeador que permita ahorrar el máximo número de líneas de código fuente a lo que se refiere a interacción con la base de datos (creación de objectos DbConnection, DbCommand, entre otros).

El modelo de clases que acompaña al componente de acceso a datos se describe a continuación.

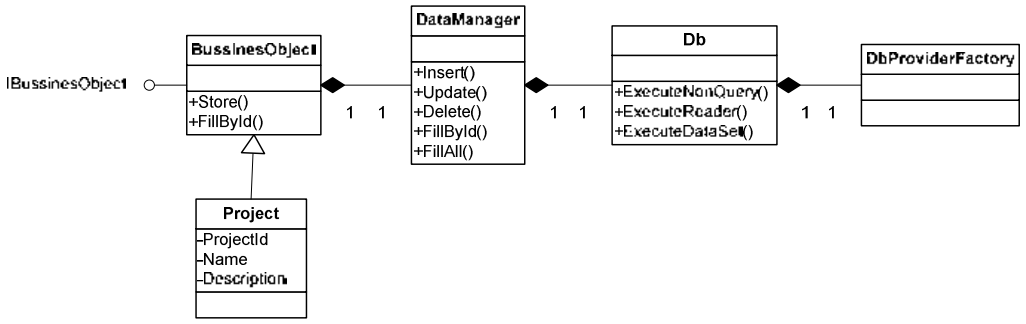


Diagrama Nº 4. Diagrama de clases para componente acceso a datos.

Dentro del modelo anterior las principales clases son las que siguen:

- Db. Una instancia de Db es capaz de ejecutar comandos sobre una base de datos MySql utilizando como miembro una instancia del tipo abstracto DbProviderFactory, pero que es inicializada con una instancia de MySqlClientFactory. Con lo anterior se logra un bajo acoplamiento, teniendo así dentro del sistema sólo una clase que inicializa objetos particulares con la librería MySqlConnector.
- DataManager. Una instancia de esta clase permite ejecutar comandos como insert, update y delete sobre la base de datos Mysql. Para lograr este objetivo, sus métodos aceptan objetos del tipo BusinessObject. Para obtener los valores de las propiedades de los subtipos de BussinessObject, la instancia de DataManager hace un recorrido de las propiedades de los objetos utilizando reflection.
- BussinesObject. Corresponde a la clase padre, desde donde heredarán todas las clases identificadas dentro del modelo de dominio de la aplicación. Cada instancia de la clase BussinesObject, tendrá una instancia (posiblemente estática o compartida) del tipo DataManager la cual manejará la impedancia de objetos en el motor de base de datos.
- Project. Dentro del modelo de clase se muestra la clase Project para ejemplificar el uso del componente de acceso a datos.

Por último, a lo que se refiere a la comunicación con el motor de base de datos se hizo necesario el uso del proveedor ADO.Net para Mysql llamado MySql Connector.

7.4.2 Capa de lógica de Negocios

El componente de lógica de negocios representará el modelo de dominio de la herramienta. Junto con lo anterior este componente contendrá dos referencias a: componente de acceso a datos y biblioteca Gnu-Gettext. Algunos alcances en el diseño de este componente:

- Para la persistencia de los objetos del dominio en el motor de base de datos, se hará necesario heredar el objeto BussinessObject incluido en el componente de acceso a datos.
- Las validaciones serán realizadas en cada objeto del dominio.

7.4.3 Componente Gnu - Gettext

Para las tareas relacionadas con la ejecución de comandos y parsing de archivos, se creó un componente que permita entregar servicios a los módulos que conforman la aplicación de una manera centralizada y con un bajo acoplamiento. Cabe destacar que este componente se baso en código Open Source proveniente del proyecto MonoDevelop, específicamente del complemento de traducción de archivos PO que viene en la versión 1.0.

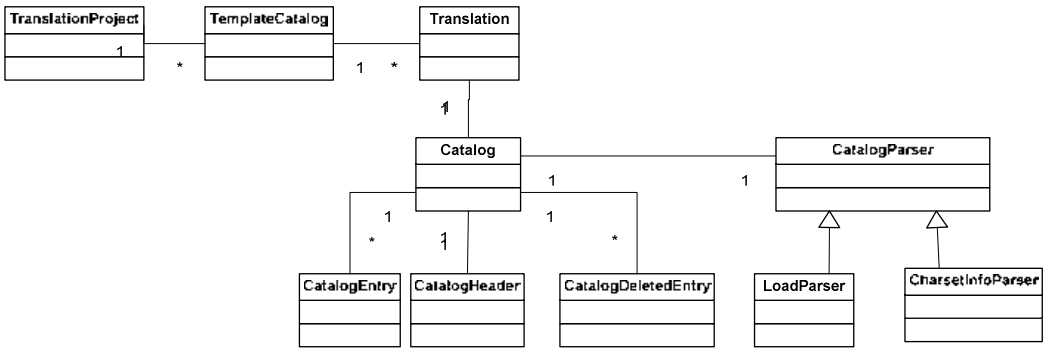


Diagrama Nº 5. Diagrama de Clases Gnu-Gettext

La comunicación con los demás componentes se realizará a través de una única clase (clase fachada, inspirada en el patrón facade), la cual estará a cargo de delegar responsabilidades dentro del modelo de clases provisto en el diagrama 5.

La construcción de este componente se basó en el código ya existente dentro del proyecto Mono-Develop. La descripción de cada clase en la siguiente tabla.

Tabla Nº 20. Descripción de clases Gettext

Nombre	**Descripción**
TranslationProject	Se encarga de la construcción de un directorio para el proyecto dentro del sistema de archivo donde se aloje la herramienta.
TemplateCatalog	Corresponde a un archivo de plantilla POT dentro del sistema de archivos.
Translation	Representa una traducción a un idioma.
Catalog	Representa un archivo de localización PO dentro del sistema de archivos. Es capaz de leer y guardar cambios ocurridos en el archivo.
CatalogEntry	Representa una cadena traducible dentro del archivo de localización.
CatalogHeader	Representa la información de

	cabecera contenida dentro de un archivo de localización.
CatalogDeletedEntry	Corresponde a un mensaje traducible obsoleto dentro del archivo de localización.
CatalogParser	Clase base para los parseadores de la información.
LoadParser	Parseador del contenido del archivo de localización.
CharsetInfoFinder	Parseador de la cabecera del archivo de la localización.

7.4.4 Capa de presentación

Inserto dentro de la capa de presentación se encuentra un proyecto web el cual contiene las páginas web que permiten la interacción con el usuario de la herramienta. Algunas de las tecnologías, herramientas y conceptos aplicados en la construcción de este componente se listan a continuación.

- Construcción de controles web personalizados.
- Localización de aplicaciones ASP.NET 2.0 utilizando los conceptos de archivos de localización globales.

- Utilización de hojas de estilo CSS. Una excelente guía para el entendimiento de la mismas la ofrece [Eguíluz2008].

7.4.5 Administración de Usuarios y Patrón Estrategia.

Inserto dentro de ASP.NET se encuentra el modelo proveedor, derivado del patrón estrategia. Este modelo permite desarrollar componentes de administración de usuarios que serán implementados derivando de un modelo de clases abstractas.

Para el caso particular de esta herramienta se tomará código existente proveniente del conector de MySql para .Net, y que luego será modificado y adaptado a las necesidades de la herramienta en sí. El diagrama siguiente muestra el modelo de clases implementado para la administración de usuarios utilizando una base de datos MySql.

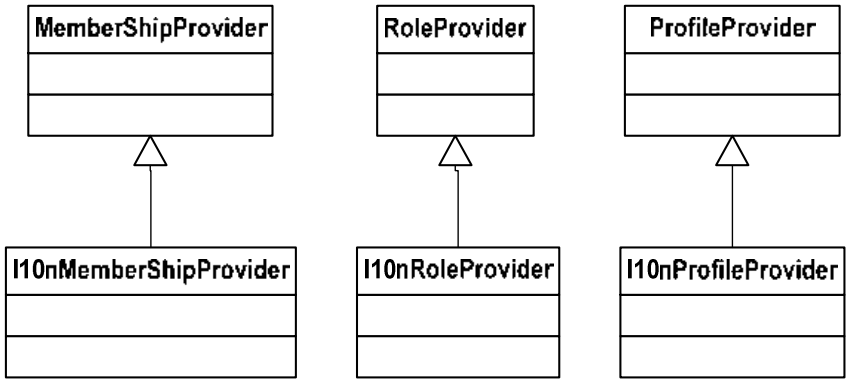


Diagrama Nº 6. Modelo de proveedores

La ventaja de utilizar el modelo de proveedores de ASP.NET 2.0 es la integración que posee con los controles de autenticación y modelo de seguridad de las aplicaciones web. Para más información relacionada con el modelo proveedor junto con la API que exponen, se puede encontrar en [Esposito2006].

7.5 Arquitectura Física

La disposición de los componentes desarrollados para esta herramienta serán ubicados en una única máquina, aunque ésta entrega la posibilidad de separar el motor de base de datos de la aplicación.

7.6 Pruebas de Componentes

Las pruebas de los componentes del sistema se realizaron ocupando el framework de pruebas de unidad NUnit, entregando la ventaja de ser multiplataforma. Para la creación de los test de unidad se siguieron las siguientes reglas.

- Cada Proyecto de código fuente fue acompañado por un proyecto donde se alojaron sus test correspondientes.
- En la creación de los test no se ocuparon conceptos de stub ni tampoco de mocking.
- La ejecución de los test en Linux se realizaron utilizando la aplicación NAnt que a su vez llamaban a la ejecución de la consola de NUnit.

8. Construcción

8.1. Modelado

8.1.1. Base de datos

Al inicio de la fase de construcción comienza la etapa de diagramación de un modelo conceptual de datos que soporte cada requerimiento de la herramienta con respecto a los datos que está deba almacenar. Este modelo que finalmente será convertido a un esquema de datos se diseñó utilizando las restricciones descritas y explicadas en el modelo de dominio ubicado en la fase de diseño. Los pasos seguidos para la implementación de la base de datos se basaron en la metodología de Thomas Connolly y Carolyn Begg [Connolly2005], la cual propone una serie de etapas:

- Diseño conceptual
- Diseño lógico
- Diseño físico

8.1.1.2. Diseño Conceptual

Para la creación de este diseño conceptual se tomaron en cuenta los siguientes componentes:

- Entidades
- Relaciones
- Atributos

8.1.1.2.1 Identificación de entidades

Del análisis obtenido a través de los requerimientos y la investigación realizada sobre la localización de aplicaciones utilizando el estándar Gnu-Gettext se desprenden las siguientes entidades.

Nombre Entidad	Project
Descripción	Almacena información que identifica un proyecto de localización en particular
Aliases	PROJECT
Ocurrencia	Ninguno, uno o muchos proyectos de localización.

Nombre Entidad	User
Descripción	Almacena información relacionada con los usuarios registrados en el sistema.
Aliases	USER
Ocurrencia	Ninguno, uno o muchos usuarios.

Nombre Entidad	TemplateFile
Descripción	Representa un archivo de plantilla POT.
Aliases	TEMPLATEFILE
Ocurrencia	Ninguno, uno o muchos archivos de plantilla por proyecto de localización.

Nombre Entidad	FileByLanguage
Descripción	Representa un archivo de localización PO.
Aliases	FILEBYLANGUAGE
Ocurrencia	Ninguno, uno o muchos archivos de localización por proyecto.

Nombre Entidad	Message
Descripción	Representa una cadena extraída desde un archivo de localización PO o POT
Aliases	MESSAGE
Ocurrencia	Uno o muchos mensajes por archivo de localización.

Nombre Entidad	Translations
Descripción	Representa una traducción tentativa para un mensaje extraído desde un archivo de localización.
Aliases	TRANSLATION
Ocurrencia	Ninguna, una o muchas traducciones para un único mensaje.

Nombre Entidad	Language
Descripción	Almacena información relacionada con un lenguaje del sistema.
Aliases	LANGUAGE
Ocurrencia	Ninguna, uno o muchos lenguajes

	registrados en el sistema.

Nombre Entidad	ProjectRoleType
Descripción	Representa los tipos de rol que un usuario puede tener en un proyecto de localización.
Aliases	PROYECTROLETYPE
Ocurrencia	Ninguno o uno por cada usuario asociado a un proyecto.

Nombre Entidad	Profile
Descripción	Contiene información personal del usuario.
Aliases	PROFILE
Ocurrencia	Ninguno o un perfil por cada usuario registrado en el sistema.

Nombre Entidad	Gender
Descripción	Contiene información los géneros en el sistema.
Aliases	GENDER
Ocurrencia	Un género por cada perfil de usuario.

8.1.1.2.2 Identificación de relaciones

Una vez identificadas las entidades se buscan las relaciones entre ellas.

Tipo Entidad	TemplateFile
Tipo de Relación	HasTemplateFile
Descripción	TemplateFile puede pertenecer un proyecto y un proyecto con n TemplateFile.
Tipo Entidad	Project
Cardinalidad	1,n

Tipo Entidad	FileByLanguage
Tipo de Relación	IsTranslatedTo

Descripción	Un archivo de localización (FileByLanguage) puede estar asociado a un archivos de plantilla (TemplateFile) y un TemplateFile a varios archivos de localización
Tipo Entidad	TemplateFile
Cardinalidad	1,n

Tipo Entidad	Message
Tipo de Relación	Contains
Descripción	Un mensaje puede estar asociado a un único archivo de localización y un archivo de localización puede tener muchos mensajes
Tipo Entidad	FileByLanguage
Cardinalidad	1,n
Tipo Entidad	Translations
Tipo de Relación	TranslationByMessage
Descripción	Un mensaje puede tener varias traducciones (translations) y una

	traducción puede pertenecer a un mensaje.
Tipo Entidad	Message
Cardinalidad	1,n

Tipo Entidad	Translations
Tipo de Relación	TranslationByMessage
Descripción	Un mensaje puede tener varias traducciones (translations) y una traducción puede pertenecer a un mensaje.
Tipo Entidad	Message
Cardinalidad	1,n

Tipo Entidad	Translations
Tipo de Relación	TranslationByUser
Descripción	Un usuario puede realizar varias traducciones y una traducción puede pertenecer a un único usuario.
Tipo Entidad	User
Cardinalidad	1,n

Tipo Entidad	User
Tipo de Relación	hasprofile
Descripción	Un usuario puede contener un único perfil y un perfil pertenecer a un único usuario.
Tipo Entidad	Profile
Cardinalidad	1,1

Tipo Entidad	Profile
Tipo de Relación	hasgender
Descripción	Un perfil de usuario puede tener un único género y un género puede ser encontrado en uno o más perfiles de usuario.
Tipo Entidad	Gender
Cardinalidad	1,n
Tipo Entidad	User
Tipo de Relación	HasRevisions
Descripción	Un usuario puede una o más revisiones sobre una traducción y una

	traducción puede ser revisada por uno o más usuarios.
Tipo Entidad	Translation
Cardinalidad	n,n

Tipo Entidad	User
Tipo de Relación	ProjectRole
Descripción	Un usuario puede tener un tipo de rol para un proyecto y un proyecto puede tener uno o más tipos de roles.
Tipo Entidad	Project
Cardinalidad	n,n

Tipo Entidad	ScoreDetails
Tipo de Relación	hasDetails
Descripción	Una traducción puede tener más de una puntuación (score) y una puntuación pertenece a una traducción.
Tipo Entidad	Translations
Cardinalidad	1,n

8.1.1.2.3. Identificación de atributos en entidades y relaciones

Para el término del modelo conceptual se identificaron los atributos de cada entidad.

Entidad Project

Nombre	Tipo de dato	Nulo	Descripción
ProjectName	varchar(50)	NO	Nombre del proyecto de localización.
Description	Text	Si	Descripción.

Entidad TemplateFile

Nombre	Tipo de dato	Nulo	Descripción
Name	varchar(50)	NO	Nombre del archivo POT.
Path	varchar(70)	NO	Ruta completa del archivo.

Entidad FileByLanguage

Nombre	Tipo de dato	Nulo	Descripción

Name	varchar(50)	NO	Nombre del archivo PO.

Entidad Message

Nombre	Tipo de dato	Nulo	Descripción
MsgId	Text	NO	Cadena original extraída desde el archivo PO.
Msgstr	Text	SI	Traducción de la cadena.
Comment	varchar(50)	SI	Comentario acerca de la traducción
ImagePath	Text	SI	Ruta de la imagen asociada a la cadena original.

Entidad Translation

Nombre	Tipo de dato	Nulo	Descripción
AlternativeMessage	Text	NO	Posible traducción

			para un mensaje.
Score	Integer	SI	Puntaje total del mensaje.

Entidad ProyectRoletype

Nombre	Tipo de dato	Nulo	Descripción
Description	Varchar(50)	NO	Descripción del tipo de rol.

Entidad Profile

Nombre	Tipo de dato	Nulo	Descripción
FullName	Varchar(50)	NO	Nombre completo del usuario.
Country	Varchar(40)	NO	País del usuario.
Theme	Varchar(40)	NO	Tema de interfaz gráfica usada por el usuario.

Entidad Language

Nombre	Tipo de dato	Nulo	Descripción
Description	Varchar(50)	NO	Descripción del lenguaje.
Code	Varchar(30)	NO	Código de lenguaje
IsSystem	Bool	NO	Indica si el lenguaje es utilizado por el sistema para su propia localización.

Entidad Gender

Nombre	Tipo de dato	Nulo	Descripción
Description	Varchar(10)	NO	Nombre de género.

Entidad ScoreDetail

Nombre	Tipo de dato	Nulo	Descripción
Score	Integer	NO	Puntuación entrega por un usuario a una traducción.

Entidad ProyectRoleType

Nombre	Tipo de dato	Nulo	Descripción
Description	Varchar(50)	NO	Nombre de rol posible para un usuario dentro de un proyecto.

8.1.1.2.4. Diagrama Modelo conceptual

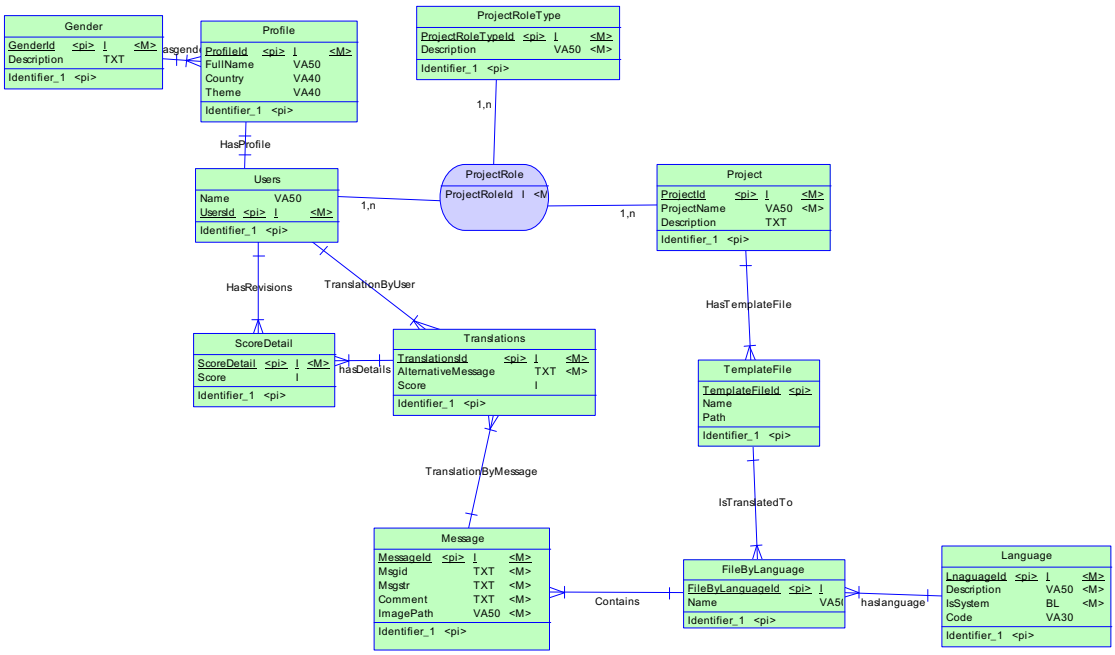


Diagrama Nº 7. Modelo Conceptual de datos

8.1.1.3. Modelo Lógico – Físico de base de datos

Una vez obtenido el diagrama conceptual de datos, se pasa a la segunda y tercera etapa, consistente en redefinir algunas estructuras y así evitar inconvenientes en la implementación final de la base de datos.

8.1.1.3.1. Relaciones complejas en el modelo

Durante la concepción del modelo conceptual de datos, no se encontraron relaciones muchos a muchos, aunque si se encontró la relación “projectrole”, la cual involucra tres entidades: users, proyectroletype y proyect.

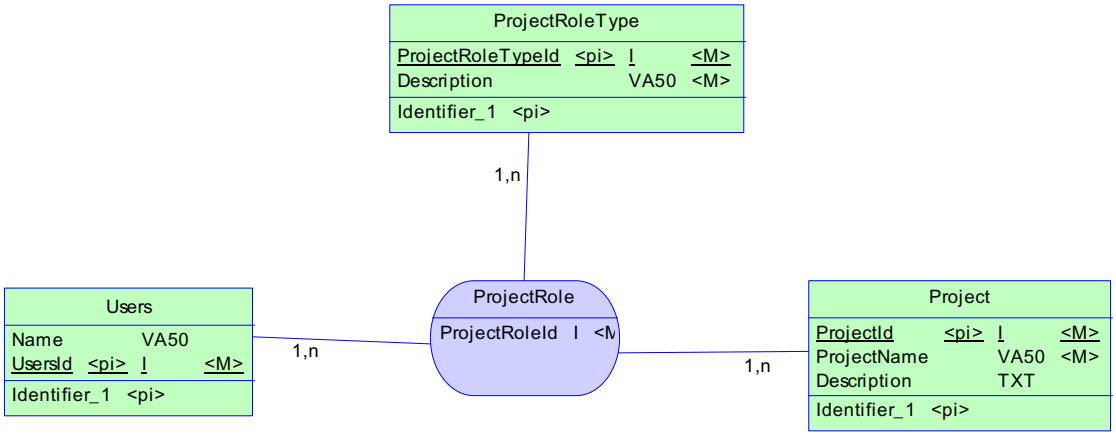


Diagrama Nº 8. Relación proyectorole.

8.1.1.3.2. Claves primarias

Para el esquema final de datos, se utilizaron claves primarias autogeneradas, que para el caso del motor MySql corresponden a una secuencia. Sin embargo, en algunas tablas del modelo físico se consideran identificadores naturales para cuidar la integridad de los datos. La elección de identificadores autogenerados se basa en la facilidad en la programación que esta aproximación entrega.

8.1.1.3.4. Tablas adjuntas al modelo físico de datos

Para la implementación final del modelo físico de datos se adjuntaron tablas útiles en la administración de usuarios y que siguen un esquema definido de acuerdo al modelo de seguridad de ASP.NET. Las tablas adjuntas al modelo en cuestión son:

- MemberShip
- Roles
- UsersInRoles
- Users. Aunque esta tabla ya se encontraba en el modelo de datos conceptual, solo se debió agregar más campos al modelo físico.

8.1.1.3.5. Modelo físico de datos

El siguiente diagrama muestra el modelo final de datos para la aplicación, el cual fue llevado al DBMS MySql utilizando la herramienta PowerDesigner.

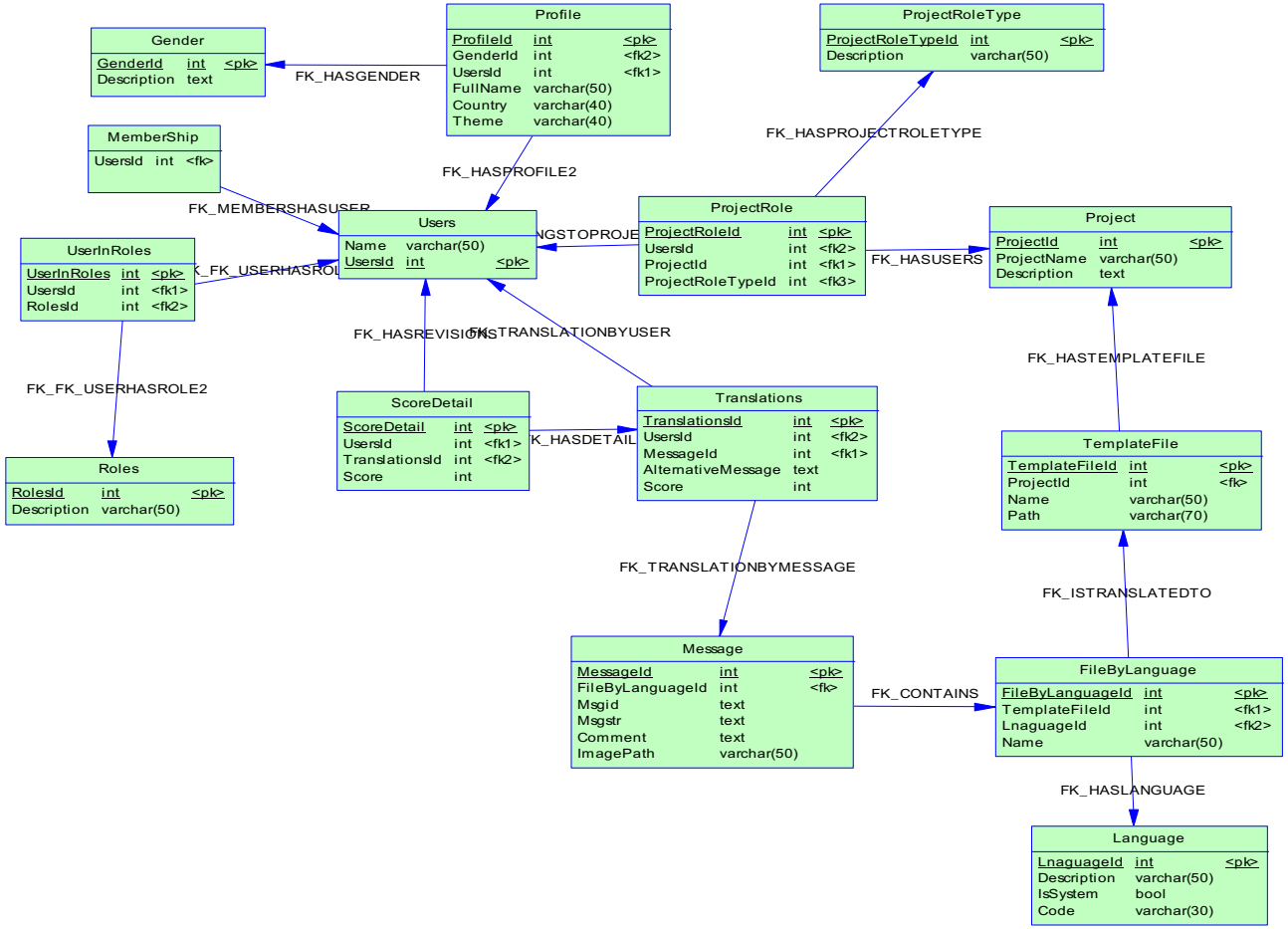


Diagrama Nº 9. Diagrama físico de datos

8.1.2. Diagrama de Paquetes del Producto

El diagrama a continuación muestra los paquetes de software, con sus dependencias en líneas discontinuas. Los paquetes de color blanco corresponden a librerías desarrolladas durante el proyecto y las de color gris corresponden a librerías de terceras partes. Las librerías de clases que contienen los tests fueron excluidas del diagrama para clarificar el mismo. Junto con lo anterior se excluyeron las bibliotecas standard que el framework Mono implementa.

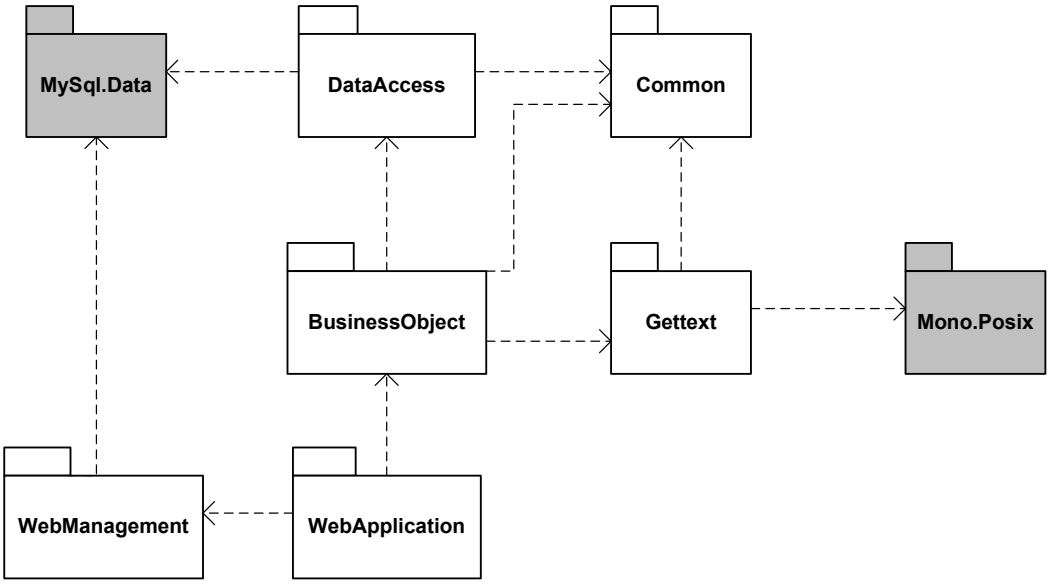


Diagrama Nº 10: Paquetes del producto

A continuación una descripción de cada uno de los paquetes:

- WebApplication corresponde a una aplicación web y agrupa todas las páginas web y clases necesarias para la correcta presentación de la información.
- WebManagement corresponde a una biblioteca de clases que implementa la seguridad dentro del sitio. Inserto dentro del paquete se encuentran las implementaciones a las clases abstractas membershipprovider, roleprovider y profileprovider.
- DataAccess, biblioteca que contiene las clases para la comunicación con el motor de base de datos y clases base útiles para la implementación del patrón de registro activo.
- BusinessObjects, biblioteca que contiene las clases involucradas en el dominio de la herramienta.
- Gettext, es una biblioteca de clases responsable de la creación y parsing de archivos de localización junto con la ejecución de comandos linux.
- Common, contiene enumeraciones y configuraciones comunes a todos los demás paquetes. Una configuración típica es la cadena de conexión hacia el motor de base de datos, el que puede ser usado como se muestra a continuación.

```
private static readonly string m_connectionString = SystemSettings.ConnectionString;
```

Evitando así llamadas repetitivas de este estilo en distintas secciones del código:

```
m_ConnectionString = ConfigurationManager.ConnectionStrings["LocalMySqlServer"].ConnectionString;
```

- MySql.Data, proveedor de acceso a datos para el motor de base de datos MySql.
- Mono.Posix, biblioteca que provee funcionalidades para acceder a características de sistemas Unix.

8.2. Implementación

8.2.1. Paquete Gettext

Este paquete se conforma de prácticamente cinco clases principales.

8.2.1.1 Clase catalog

Representa un archivo de localización y dentro de sus principales métodos se encuentran “load” y “save”; el primero carga y realiza una extracción de todas las cadenas traducibles dentro del archivo de localización, mientras que el segundo guarda los cambios que se hayan producido en las cadenas extraídas.

Para mejorar la legibilidad del documento no se expondrá el código completo de la clase. El código 1 muestra el cuerpo del método “Load” de la clase “Catalog”, el cual recibe como parámetro la ruta completa del archivo de localización. En el código 2 se puede ver el uso de la clase haciendo llamada a los métodos “load”, “save” y “translate”.

```
//Metodo que carga un archivo de localizacion y

//extrae las cadenas traducibles

public bool Load ( string poFile)

{
        Clear ();
        isOk = false;
        fileName = poFile;

        bool finished = false;

        CharsetInfoFinder charsetFinder = new
        CharsetInfoFinder (poFile);

        charsetFinder.Parse ();
        headers.Charset = charsetFinder.Charset;
        originalNewLine = charsetFinder.NewLine;
        finished = true;
        if (! finished)
                return false;

        LoadParser parser = new LoadParser (this,
        poFile, Catalog.GetEncoding
        (this.Headers.Charset));
        if (!parser.Parse())
        {
                return false;
        }

        isOk = true;
        isDirty = false;
        return true;
}
```

Código N°1. Método Load

Un ejemplo de cómo usar esta clase a continuación.

```
//se entrega la ruta del archivo
string newPotFile = "./TestPofiles/newmessages.po";
//se crea una instancia y se entrega como parametro //al metodo load la
ruta del archivo de localizacion
Catalog MyCatalog= new Catalog();

bool isOk=MyCatalog.Load(poFile);
//se obtiene la primera cadena traducible dentro del //archivo
string key=MyCatalog[0].String;
//se traduce la cadena
bool Translated =MyCatalog.Translate(key,"translated");
//se guardan los cambios al archivo
MyCatalog.Save(poFile);
```

Código N°2. Ejemplo de uso de clase catalog

8.2.1.2 Clase TranslationProject

Representa un proyecto que engloba cierto número de archivos de plantillas POT y archivos de localización PO. Cada instancia de esta clase representa un directorio físico dentro del sistema de archivos. El código 3 muestra los métodos más importante de la clase “TranslationProject”.

```
public class TranslationProject: IEnumerable<TemplateCatalog>
	{

		//corresponde a una lista TemplateCatalog que representa
		//los archivo POT
		List<TemplateCatalog> m_templates = new
		List<TemplateCatalog> ();
		private string m_NameProject= string.Empty;
		private string m_DirectoryForProject= string.Empty;

		/// <summary>
		/// inicia un proyecto creando su directorio de trabajo
		/// </summary>
		public TranslationProject(string nameProject)

		{
			m_NameProject=nameProject;

	m_DirectoryForProject=BuildDirectory(nameProject);

		}

		public string DirectoryForProject {
			get
			{
				return m_DirectoryForProject;
			}
		}
		//agrega un archivo POT identificado por el atributo
		//templatename
		public TemplateCatalog AddNewTemplate(string
		templateNane)
		{
			CheckAddTemplate(templateNane);

			TemplateCatalog Tc =new
			TemplateCatalog(this,templateNane);

			m_templates.Add(Tc);
			return Tc;
		}
```

```
        public TemplateCatalog GetTemplate (string templateNane)
        {
                foreach (TemplateCatalog Template in this.m_templates)
                        {
                                if (Template.TemplateName ==templateNane)
                                        return Template;
                        }
                        return null;
                }
        }

        //remueve un objeto templatecatalog de la collecion de
        //m_templates
        public void RemoveTemplate(string templateName)
        {
                TemplateCatalog Template= GetTemplate(templateName);

                        if (Template != null)
                                m_templates.Remove(Template);

        }

        //entrega la cantidad de objetos templatecatalog dentro de la
        //collection m_templates
        public int Count
        {
                get { return m_templates.Count; }
        }

        //contruye un directorio que alojara el proyecto con sus
        //respectivos archivo
        protected string BuildDirectory(string nameProject)
        {
                string DirectoryForProject=
                Path.Combine(FileUtils.ProjectBaseDirectory,nameProject);

                if (!Directory.Exists(DirectoryForProject))

                        Directory.CreateDirectory (DirectoryForProject);

                return DirectoryForProject;
        }
}
```

Código Nº 3. Clase TranslationProject

8.2.1.3 Clase TemplateCatalog

Una instancia de esta clase representa un directorio que contiene un archivo de plantilla POT con sus correspondientes archivos de localización PO. El código 4 muestra el cuerpo de la clase “TemplateCatalog” junto a sus métodos más importantes.

```
public class
TemplateCatalog:TranlationsRepository,IEnumerable<Translation>
{

        private List<Translation> m_translations = new List<Translation>
        ();
        private string m_TemplateName=string.Empty;
        private string m_TemplateDirectory=string.Empty;
        private TranslationProject m_TranslationProjectParent;


        public TemplateCatalog(TranslationProject
        translationProjectParent , string templateName)
        {
                m_TemplateName=templateName;


                m_TranslationProjectParent=translationProjectParent;

                //en caso que no exista se construye el direcotorio de
                //trabajo
                m_TemplateDirectory=BuildTemplateDirectory();

                //copia el archivo desde el directorio temporal al
                //directorio de trabajo del archivo
                RetrieveFromTemporalFolder(m_TemplateDirectory,FileName);


        }

        // Directorio donde se aloja el archivo POT
        public string TemplateDirectory
        {
                get {
                        return m_TemplateDirectory;
                        }
        }

        //nombre del archivo sin la extension POT
        public string TemplateName
        {
                get {
                                return  m_TemplateName;
                    }
        }
```

```
        //retorna el nombre del archivo POT
        public string FileName
        {
                Get
                {
                        return (m_TemplateName + ".pot");
                }
        }

        //Obtiene la ruta completa del archivo POT
        private string GetPotFileName ()
        {
                return Path.Combine(TemplateDirectory, FileName);
        }


        //retorna una lista de traducciones donde cada translation /
        /object representa un archivo PO
        public  List<Translation> Translations
        {
                get
                {
                        return  m_translations;
                }
        }

        //agrega una nueva traducción de acuerdo al IsoCode
        public Translation AddNewTranslation(string isoCode)
        {
                Translation tr = new Translation (this, isoCode);
                m_translations.Add(tr);
                string TemplateFileName= GetPotFileName();
                string TranslationFileName=  tr.GetPoFile();
                File.Copy (TemplateFileName, TranslationFileName,true);

                return tr;
        }

        //busca un objeto translation de acuerdo al isocode
        public Translation GetTranslation (string isoCode)
        {
                foreach (Translation translation in this.m_translations)
                {
                        if (translation.IsoCode == isoCode)
                                        return translation;
                }
                return null;
        }

        //recorre la collecion de traducciones (po files) y los
        transforma a archivos binarios MO
        public bool BuildMoFiles()
        {
                bool status=false;
                foreach (Translation translation in this.m_translations)
        {
                status=BuilMoFile(translation);
        }
        return status;
}
```

```
// Tomando como parametro un objeto del tipo translation
// hace una llamda a la librería GNU-Gettext para convertir el archivo
//PO a un archivo MO. La llamda a gnu-gettext se hace utilizando la
//clase gettextcommand
        public bool BuilMoFile(Translation translation)
        {
                string outputDirectory=GetOutPutDirectory();

                string moDirectory=
                Path.Combine (Path.Combine (outputDirectory,
                translation.IsoCode), "LC_MESSAGES");
                if (!Directory.Exists (moDirectory))
                        Directory.CreateDirectory (moDirectory);

                string fromPoFileName= Path.Combine(TemplateDirectory,
                translation.PoFile);
                string toPoFileName=Path.Combine(moDirectory,
                translation.PoFile);

                File.Copy(fromPoFileName,toPoFileName);

                string moFileName=  Path.Combine( moDirectory,
                translation.MoFile);
                bool buildOk=
                GettextCommands.BuildMoFile(toPoFileName,moFileName);
                File.Delete(toPoFileName);

                return buildOk;
        }

}
```

Código Nº 4. Clase TemplateCatalog

Dentro de los métodos expuestos en el código 4, se encuentra el método “BuildMoFile”, el cual permite crear los archivos MO para cada uno de los archivos de localización que pertenezcan al proyecto.

8.2.1.4 Clase Translation

Una instancia de esta clase representa un archivo de localización PO.

```
public class Translation: TranlationsRepository
{

        TemplateCatalog m_TemplateParent;
        string m_isoCode;
        string m_poFileName=string.Empty;

        //se entrega al constructor el codigo iso y una instancia de
        //templatecatalog el cual representa su archivo POT padre
        public Translation(TemplateCatalog templateParent,string
        isoCode)
        {
                m_TemplateParent=templateParent;
                m_isoCode=isoCode;
        }


        public Translation(TemplateCatalog templateParent,string isoCode
        ,string poFileName)
        {
                m_TemplateParent=templateParent;
                m_isoCode=isoCode;
                m_poFileName=poFileName;

                //se trae el archivo desde el directorio temporal desde
                donde se suben los archivos

                RetrieveFromTemporalFolder(m_TemplateParent.TemplateDirec
                tory,PoFile);
        }


        public Translation(TemplateCatalog templateParent)
        {
                m_TemplateParent=templateParent;
        }

        //El directorio donde esta el archivo PO
        public string TranslationDirectory
        {
                get
                {
                        return m_TemplateParent.TemplateDirectory;
                }
        }
```

```
        //El codigo iso que representa el archivo PO
        public string IsoCode
        {
                get { return m_isoCode; }
                set { m_isoCode = value; }
        }

        //representa el archivo POT padre
        public TemplateCatalog ParentProject
        {
                get {return m_TemplateParent; }
                set {m_TemplateParent = value;}
        }


        //nombre del archivo po
        public string PoFile
        {
                get
                {
                if (m_poFileName == string.Empty)
                        return m_TemplateParent.TemplateName + "_" +
                        IsoCode + ".po";
                else
                        return m_poFileName  + ".po";
                }
                set
                {
                        m_poFileName=value;
                }
        }

        //representa el nombre del archivo mo binario
        public string MoFile
        {
                get
                {
                if (m_poFileName == string.Empty)
                        return m_TemplateParent.TemplateName + "_" +
                        IsoCode + ".mo";
                else
                        return m_poFileName +".mo";

                }
        }

        //representa el nombre completo del archivo PO.
        public string GetPoFile()
        {
                return Path.Combine(TranslationDirectory,PoFile);
        }
```

Código Nº 5. Clase Translation

8.2.1.5 Clase Gettextcommands

Contiene las sentencias necesarias para la ejecución de comandos de la librería Gnu-Gettext. Un ejemplo es el método “buildmofile”, que convierte un archivo de localización PO a un archivo MO.

```
public class GettextCommands
{

	//Actualiza un archivo po con una nueva version de un archivo de
	//plantilla POT. Utiliza el comando msgmerge de GNU-Gettext
	public static bool Merge(string poFile, string potFile)
	{
		System.Diagnostics.Process process = new
		System.Diagnostics.Process ();
		process.StartInfo.FileName = "msgmerge";
		process.StartInfo.Arguments = " -U " + poFile + " -v " +
		potFile;
		process.Start ();
		process.WaitForExit ();

		return (process.ExitCode == 0);

	}

	//Utilizando un archivo PO con cadenas ya traducidas, lo
	//convierte a un archivo binario MO. Utiliza el comando msgfmt
	//de GNU-Gettext
	public static bool BuildMoFile(string poFile,string moFile)
	{
		System.Diagnostics.Process process = new
		System.Diagnostics.Process ();
		process.StartInfo.FileName = "msgfmt";
		process.StartInfo.Arguments = poFile + " -o " + moFile;
		process.Start ();
		process.WaitForExit ();

		return (process.ExitCode == 0);
	}
```

Código Nº 6. Clase GettextCommands

8.2.1.6. Uso del Paquete Gettext

Ejemplos de uso de este paquete se encuentran en la clase "LocalGettextService" del paquete businessobject. El método a continuación crea un proyecto, agrega un nuevo archivo de plantilla y nuevos archivos de localización.

```
public void AddNewPotFileWithDefaultLanguages(string projectName,string
templateName)
{
       string templatefileNamewithoutExtension=templateName;
       TranslationProject pr= new TranslationProject(projectName);
       TemplateCatalog
       tc=pr.AddNewTemplate(templatefileNamewithoutExtension);

       //crea tres nuevas traducciones para un el archivo POT
       //con nombre templateName
       string SpanishIsoCode="es";
       string EnglishIsoCode="en";
       string PortugueseIsoCode="pr";

       tc.AddNewTranslation(SpanishIsoCode);

       tc.AddNewTranslation(EnglishIsoCode);
       tc.AddNewTranslation(PortugueseIsoCode);

}
```

Código Nº 7. Creación de un proyecto de localización

A continuación el método "BuildProject" encargado de construir los archivos binarios MO a partir de los archivos PO.

```
//toma como parametro un objeto de del tipo project
public void BuildProject(Project projectDB)
{
       TranslationProject tp = new
       TranslationProject(projectDB.ProjectName);
       string templatefileNamewithoutExtension;

       //por cada archivo POT en el proyecto
       foreach(TemplateFile potfileDB in projectDB.PotFiles)
       {

              templatefileNamewithoutExtension=
              potfileDB.Name.Substring(0,potfileDB.Name.Length -4);

              TemplateCatalog potfile =
              tp.AddNewTemplate(templatefileNamewithoutExtension);

              //por cada archvo PO asociado a un archivo POT
              foreach (FileByLanguage poFileDB in potfileDB.PoFiles)
              {
                     string
                     pofileWithoutExtension=poFileDB.Name.Substring(0,
                     poFileDB.Name.Length-3);


                     potfile.AddNewTranslation(poFileDB.IsoCode,
                     pofileWithoutExtension);
              }
              //construye los archivo MO para archivo PO asociado al
              archivo POT
              potfile.BuildMoFiles();
       }
}
```

Código Nº 8. Creación de archivos MO

8.2.2. Patrón de Registro Activo

La tarea de guardar información relacionada con los proyectos y archivos de localización dentro de una base de datos, se cumplió utilizando el patrón de registro activo descrito en [Fowler2002] o bien visitando el anexo A “Patrones de Diseño usados en la solución”.

Para el caso particular de esta herramienta, active record se implementó utilizando la clase base “BusinessObject”: A partir de ésta última, las clases involucradas en el dominio heredarán las siguientes propiedades y métodos.

Tabla Nº 21. Propiedades clase businessobject

Nombre	**Descripción**
ObjectName	Entrega el nombre del tipo de dato del objeto que se instancia (ej.- Project, Message).
Id	Entrega el identificador autogenerado desde de la base de datos MySql.
GetType	Devuelve un tipo de datos Type.
GetState	Entrega el estado en el que se encuentra un objeto dado (nuevo, agregado a la base de datos, eliminado desde la base de datos).

Tabla Nº 22. Métodos clase BusinessObject

Nombre	**Descripción**
Dataprovider	Devuelve un objeto del tipo DataManager, el cual finalmente se comunica con el motor de base de datos.
Store	De acuerdo al estado del objeto, se inserta, actualiza o borra la "fila" que éste representa en la base de datos.
Delete	Utilizando el id autogenerado, borra una tupla en la base de datos.
FillById	Utilizando el id autogenerado, llena las propiedades de un objeto con sus respectivos valores.

A continuación la clase base "BusinessObject", desde la cual todo objeto que tenga interacción con la base de datos debe heredar.

```
public class BusinessObject: IBusinessObject
    {
        protected string m_ObjectName;
        protected Nullable<Int32> m_Id;
        protected ObjectState m_objectState =
ObjectState.NotAddedToDataStorage;

       //nombre del objeto en la base de datos
        public string ObjectName
        {
            get {return  m_ObjectName; }
            set { m_ObjectName=value; }
        }

       //Identificador unico del objeto, corresponde al
       //identificador autogenerado en la Base de datos
        public Nullable <Int32>  Id
        {
            set { m_Id=value; }
            get { return m_Id; }
        }

       //entrega el tipo del objeto
        public new  Type GetType
        {
            get {  return base.GetType(); }
        }

       //proveedor de datos para el objeto
        public static DataManager DataProvider()
        {
            return DataManager.Instance();
        }

       //metodo de interaccion con la base de datos,
       //de acuerdo al estado del objeto, este podra ser
       //insertado, actualizado o borrado de la base de datos
        public void Store()
        {
            switch(GetState)
            {
                case ObjectState.NotAddedToDataStorage:
                    DataProvider().Insert(this);
                    this.m_objectState =
ObjectState.LoadedFromDataStorage;
                    break;
                case ObjectState.LoadedFromDataStorage:
                    DataProvider().Update(this);
                    this.m_objectState =
ObjectState.LoadedFromDataStorage;
                    break;
                case ObjectState.MarkedForDeletion:
                    DataProvider().Delete(this);
                    this.m_objectState =
ObjectState.DeletedFromDataStorage;
                    break;
            }
        }
```

```
        //marka al objeto como borrado y luego llama al procedimiento
        //almacenado de borrado asociado al objeto
         public void Delete()
         {
             this.GetState = ObjectState.MarkedForDeletion;
             this.Store();
         }

        //basandose en el Id, permite rescatar de la base de datos
        // los atributos de un objeto
         public void FillById(Int32?id)
         {
             DataProvider().FillById(this,id);
             this.m_objectState = ObjectState.LoadedFromDataStorage;
         }


        //obtiene el estado de un objeto, borrado, cargado de la base de
        //datos o no "nuevo"
         public ObjectState GetState
         {
             set{ m_objectState = value;}
             get { return m_objectState; }
         }

    }
```

Código Nº 9. Clase businessobject

Como resumen para el uso del patrón de directorio activo en la arquitectura presentada, se deben seguir los siguientes pasos:

- Crear una nueva clase que represente las filas de una tabla en la base de datos.
- Crear los procedimientos almacenados que acompañen a esta clase. Los nombres de estos procedimientos deberán seguir el orden: Nombre base de datos + nombre tabla + operación (ej.- I10nCommunity_Project_Insert)

A continuación la clase Project

```
public class Project : BusinessObject
    {

        public Project()
        {
            m_ObjectName = "Project";
        }

        public Project(Int32? id)
        {
            m_ObjectName = "Project";
            FillById(id);
        }

        private string m_ProjectName;
        private string m_Description;
        protected string m_State;

       //setea la propiedad Id la cual es heredada desde el tipo
       // BusinessObject
        public Nullable<Int32> ProjectId
        {
            get { return Id; }
            set { Id = value; }
        }

         //nombre del proyecto, El atributo FieldMapping es utilizado
       //sobre la propiedad “projectname” para indica que es un
       //atributo mapeable.
        [FieldMapping(PersistToDataStorage = true)]
        public string ProjectName
        {
            get { return m_ProjectName; }
            set { m_ProjectName = value; }
        }

        [FieldMapping(PersistToDataStorage = true)]
        public string Description
        {
            get { return m_Description; }
            set { m_Description = value; }
        }
      }
```

Código Nº 10. Clase Project

Un simple ejemplo de cómo guardar un objeto Project en el siguiente código.

```
Project myproject = new Project(“project A”,” this is a test”);
myproject.store();
```

Código Nº 11. Guardado de un objeto Project

8.2.3. Comunicación con el motor de base de datos

Para la ejecución de comandos SQL se crearon dos clases: “Datamanager” y “Db”; la primera a cargo del mapping de los objetos y la segunda responsable de la ejecución de sentencias sql y procedimientos almacenados.

El diseño de la clase “Db” se realizó basándose en la librería “Data Access Application Block” [Entreprise2008]. A continuación sus principales métodos.

Tabla Nº 23. Métodos clase Db

Nombre	**Descripción**
ExecuteNonQuery	Ejecuta una sentencia sql o un procedimiento almacenado (insert, update, etc)
ExecuteScalar	Ejecuta una sentencia sql o un procedimiento almacenado

	devolviendo un único resultado.
ExecuteDataSet	Devuelve un conjunto de resultados encapsulados en un tipo de objeto Dataset.
ExecuteDataReader	Devuelve un conjunto de resultados encapsulados en un tipo de objeto DbDataReader.
ExecuteDataTable	Devuelve un conjunto de resultados encapsulados en un tipo de objeto DataTable.

Para mejorar la legibilidad del documento un extracto de la clase Db como se indica.

```
public class Db : IDb
{

        private static readonly DbProviderFactory m_factory = new
        MySql.Data.MySqlClient.MySqlClientFactory();

        //obtiene el string de conexión desde el archivo de configuracion
        private static readonly string m_connectionString =
        SystemSettings.ConnectionString;

        //permite ejecutar un procedimiento almacenado con parametros
        public  Int32 ExecuteNonQuery(string StoredProcedureName,
        ParameterCollection parameters)
         {
                //crea una conexion
                using (DbConnection DatabaseConnection =
                m_factory.CreateConnection())
                {
                DatabaseConnection.ConnectionString = m_connectionString;

                //crea un comando y ejecuta el proc. almacenado
                using (DbCommand DatabaseCommand = m_factory.CreateCommand())
                 {
                     DatabaseCommand.CommandText = StoredProcedureName;
                     DatabaseCommand.CommandType = CommandType.StoredProcedure;
                     DatabaseCommand.Connection = DatabaseConnection;
                     Db.PrepareCommand(DatabaseCommand, parameters);
                     DatabaseConnection.Open();
                     return DatabaseCommand.ExecuteNonQuery();
                 }
            }
          }


        //devuelve un objeto DBDatareader resultado de la ejecucion de un
        //prcedimiento almacenado con parametros
        public  DbDataReader ExecuteDataReader(string StoredProcedureName,
        ParameterCollection parameters)
         {
             MySqlConnection DatabaseConnection =
                       new      MySqlConnection(m_connectionString);
             DatabaseConnection.ConnectionString = m_connectionString;

             MySqlCommand DatabaseCommand= new MySqlCommand();
             DatabaseConnection.Open();
             DatabaseCommand.Connection = DatabaseConnection;
             DatabaseCommand.CommandText = StoredProcedureName;
             DatabaseCommand.CommandType = CommandType.StoredProcedure;
             PrepareCommand(DatabaseCommand, parameters);
             MySqlDataReader dr = DatabaseCommand.ExecuteReader();
             return dr;
         }
```

```
        //devuelve un objeto del dbparameter util en la ejecucion de proc
        //almacenados
         public  DbParameter GetDbParameter()
         {
             return m_factory.CreateParameter();
         }

         //agrega una lista de parametros a un comando.
         protected static void PrepareCommand(DbCommand DatabaseCommand,
        ParameterCollection parameters)
         {
             if (parameters != null && parameters.Count > 0)
             {
                 foreach (DbParameter parameter in parameters)
                 {
                     if (parameter.Value == null)
                     {
                         parameter.Value = DBNull.Value;
                     }
                     DatabaseCommand.Parameters.Add(parameter);
                 }

             }
         }`

    }
```

Código Nº 12. Clase Db

Por otra parte, la clase “DataManager” contiene los siguientes métodos:

Tabla Nº 24. Métodos clase DataManager

Nombre	**Descripción**
Insert	Inserta un nueva tupla en la base de datos, usando como parámetros los valores de las propiedades de un objeto del tipo “BusinessObject”.

Update	Actualiza un tupla en la base de datos, usando como parámetros los valores de las propiedades de un objeto del tipo “BusinessObject”.
Delete	Borra una tupla en la base de datos tomando como parámetro el valor en la propiedad “Id” de un objeto del tipo “BusinessObject”.
FillById	Setea cada una de las propiedades de un objeto de acuerdo a los valores almacenados en la base de datos. Toma como parámetro el Id almacenado en la tabla que representa el objeto del tipo “BusinessObject”.
FillAll	Setea una colección de objetos del tipo “BussinessObject” con la totalidad de tuplas almacenadas en la tabla que representa.

Un extracto de la clase DataManager se muestra en el código 13, donde se puede apreciar el uso del patrón “Singleton” para el acceso a única instancia de la clase. Más información del patrón en el anexo número A “Patrones de diseño usados”

```
public class DataManager
    {
        private static DataManager m_Instance;
        private static readonly IDb DbHelper = Db.Instance();

        protected DataManager()
        {
        }

       //Devuelve una instancia de la clase DataManager
        public static DataManager Instance()
        {
            if (m_Instance ==  null)
            {
                m_Instance = new DataManager();
            }
            return m_Instance;
        }


       //Espera un objeto del tipo BusinessObject y lo inserta a la base de
       //datos
        public void Insert(IBusinessObject item)
        {
            ParameterCollection parameters = new ParameterCollection();

            //mapea las propiedades del objeto y las coloca en una lista de
            //parametros.
             SetMappedProperties(item, parameters);

            parameters.Add(SetReturnValue());

           //obtiene el procedimiento almacenado asociado a la operación
           //Insert del objeto  por: l10nCommunity_Project_Insert
            string StoredProcedureName =
            GenerateStoredProcedureName(item.ObjectName, SqlOperation.Insert,
            parameters);

           //ejecuta la operación de insercion
            DbHelper.ExecuteNonQuery(StoredProcedureName, parameters);

             //se retorna el Id autogenerado
            item.Id = GetReturnValue(parameters);

        }
```

```
        //Actualiza un objeto del tipo IBusinessObject
        public void Update(IBusinessObject item)
        {
            ParameterCollection parameters = new ParameterCollection();

            parameters.AddNewParameter(CreateParameter(), "Id", item.Id);

            SetMappedProperties(item, parameters);

               string StoredProcedureName =
               GenerateStoredProcedureName(item.ObjectName,
               SqlOperation.Update, parameters);

            DbHelper.ExecuteNonQuery(StoredProcedureName, parameters);

        }

        //borra de la base de datos un objeto del tipo IBusinessObject
        public void Delete(IBusinessObject item)
        {
            ParameterCollection parameters = new ParameterCollection();

            parameters.AddNewParameter(CreateParameter(),"Id",item.Id);

            string StoredProcedureName =
            GenerateStoredProcedureName(item.ObjectName, SqlOperation.Delete,
            parameters);

            DbHelper.ExecuteNonQuery(StoredProcedureName, parameters);
        }

}
```

Código Nº 13. Clase Datamanager

Como se observa, métodos como insert, update y delete reciben como parámetro un objeto que implemente la interfaz IBusinessObject (implementada por BusinessObject). Objetos válidos aceptados como parámetros pueden ser, por ejemplo, la clase que representa la entidad “Project”. Con esto se resalta la ventaja que ofrece esta clase frente a duplicidad de código que se puede

originar en la creación de clases de acceso a datos por cada una de las clases del dominio de la herramienta.

8.2.4. Localización de la aplicación Web

Para llevar esta herramienta a múltiples idiomas, se utilizó el estándar de localización de aplicaciones .NET, el cual está basado en archivos de recursos con formato XML, donde cada archivo de recursos contiene las traducciones para cada lenguaje. Así por ejemplo, esta herramienta en su etapa inicial soportará tres idiomas: inglés, español y portugués, los que se encuentran representados por los archivos resource.en.resx, resource.es.resx y resource.pt.resx respectivamente. Un extracto del contenido de un archivo de recursos para el idioma español.

```
<data name="owner" xml:space="preserve">
  <value>Propietario</value>
</data>
<data name="password" xml:space="preserve">
  <value>contraseña</value>
</data>
<data name="possibletranslation" xml:space="
  <value>Posible Traduccion</value>
</data>
```

Figura Nº 21. Contenido de un archivo de recursos

La forma de cómo usar estos archivos en el siguiente código.

```
protected void localize()
{
        lblNumberOfPoFiles.Text = Resource.numberofpofiles;
        lblCompletePercentage.Text = Resource.completepercentage;
        lblname.Text = Resource.name;
        Title = Resource.titlemyprojetedit;
}
```

Código Nº 14. Localización en el code-behind

Otra forma alternativa de llamada a los archivos de recursos es desde el "Markup" de la página como muestra el siguiente código.

```
<asp: Label ID="lblProjectName" runat="server" text="<%$ Resources:Resource, projectname %>" />
```

Código Nº 15. Localización desde el markup de la página

Para más información respecto a como localizar aplicaciones en ASP.NET, este encuentra disponible en [Johnson2005].

8.2.5. Validación de Objetos

Las validaciones de los objetos fueron implementadas a través de tres métodos virtuales (ApplyValidationsOnInsert, ApplyValidationsOnUpdate, ApplyValidationsOnDelete), los que son llamados desde el método store de la

clase base “BusinessObject”. De esta forma cualquier objeto que herede de esta última clase podrá sobrescribir estos métodos y agregar código personalizado para las reglas de validación de la clase “hija” de “BusinessObject” (por ejemplo Project, Messages, etc). Abajo el método store desde donde se puede notar resaltado estos tres métodos sobreescribibles.

```
public void Store()
        {
            switch(GetState)
            {
                case ObjectState.NotAddedToDataStorage:
                    ApplyValidationsOnInsert();
                    DataProvider().Insert(this);
                   this.m_objectState =
               ObjectState.LoadedFromDataStorage;
                    break;
                 case ObjectState.LoadedFromDataStorage:
                    ApplyValidationsOnUpdate();
                    DataProvider().Update(this);
                      this.m_objectState =
               ObjectState.LoadedFromDataStorage;
                    break;
                 case ObjectState.MarkedForDeletion:
                    ApplyValidationsOnDelete();
                    DataProvider().Delete(this);
                    this.m_objectState
               =ObjectState.DeletedFromDataStorage;
                    break;
            }
        }
```

Código Nº 16. Métodos virtuales de validación

Junto con la validación de objetos desde el lado del servidor, se utilizaron los objetos de validación por el lado del cliente provistos por ASP.NET. Entre ellos se encuentran:

- RequiredFieldValidator, útil para validar que el usuario ha escrito un valor en un control.
- RegularExpresionValidator, útil para comprobar que los datos ingresados por el usuario coinciden con una expresión regular.
- ValidationSummay, necesario para mostrar la posible lista de errores que se han producido al tratar de enviar la información de un formulario al servidor.
- CustomValidator, usado para realizar validaciones personalizadas.

8.3. Pruebas

8.3.1. Creación de pruebas usando NUnit

Siguiendo las consideraciones descritas para las pruebas en la fase de elaboración, se crearon cuatro proyectos de pruebas para cada biblioteca de clases que compone la solución en mono. La siguiente tabla presenta una breve descripción de cada proyecto.

Tabla Nº 25. Proyectos de pruebas

Nombre	Descripción
L10nCommunity.DataAccess.Tests	Contiene las pruebas para las clases DataManager,Db
L10nCommunity.Businesslayer.Tests	Contiene las pruebas para cada clase del dominio (project,messages,etc)
L10nCommunity.Gettext.Tests	Contiene las pruebas para todos los archivos de manipulación y generación de archivos de localización PO.
L10nCommunity.WebManagment.Tests	Contiene los pruebas para las clases de administración de usuario (profileprovider, roleprovider, membershipprovider)

Para que las clases contenedoras de las pruebas puedan ser ejecutadas se hace necesario tener claro los atributos más utilizados por NUnit.

- TestFixture, es una colección de pruebas que por lo general están agrupados en una clase.

- Assert, que corresponde a una comprobación. Por ejemplo, comprobar que un valor es verdadero, que un número es mayor que otro, que un objeto es nulo, etc.
- Test, que es una colección de asserts que intenta comprobar una serie de acciones dentro del código de producción.
- Setup, permite crear un ambiente adecuado para cada prueba dentro de la clase (por ejemplo asegurarse tener datos iniciales de prueba para que un objeto pueda instanciarse).
- TeardDown, permite ejecutar código arbitrario en la finalización de cada método de pruebas.

Complementado con lo anterior algunas consideraciones respecto al paradigma de trabajo utilizado en este framework de pruebas de unidad.

- Para que una clase sea llamada clase de prueba, debe ser etiquetada con el atributo “[TestFixture]”.
- Para que cada método dentro de la clase de prueba sea ejecutado por NUnit es necesario etiquetarlo con un atributo “[Test]” (dentro del framework existen otros atributos para elegir, de acuerdo a lo que se esté probando).
- Por lo general atributos [Setup] y [Teardown] son utilizados para realizar pruebas contra una base de datos.

- Cada método etiquetado con “[Test]” debe ser público. Otro alcance relacionado con la ejecución de estos métodos de prueba es que cada uno de ellos es independiente del otro, por lo que no debe “suponerse” un orden concreto en la ejecución de los métodos de prueba.

A continuación se detallan algunos métodos de pruebas de la clase “Catalog”, responsable de cargar y guardar los cambios ocurridos en un archivo de localización.

```
[TestFixture()]
public class CatalogTests:  TestBase
	{

		[Test()]
		public void LoadTest()
		{
			//Se carga un archivo PO y se verifica que el parsing
			//estuvo ok
			Catalog MyCatalog= new Catalog();
			bool isOk=MyCatalog.Load(poFile);
			Assert.IsTrue(isOk,"fail loading the file");

		}

		[Test()]
		public void SaveTest()
		{
			//se carga y se guarda sin hacer modificaciones al archivo
			//PO
			Catalog MyCatalog= new Catalog();
			bool isOk=MyCatalog.Load(poFile);
			bool saved =MyCatalog.Save(poFile);

			Assert.IsTrue(saved,"test failing");

		}

		[Test()]
		public void TranslateTest()
		{
			//Se Traduce una cadena extraida desde el archivo PO
			Catalog MyCatalog= new Catalog();
			bool isOk=MyCatalog.Load(poFile);
			string key=MyCatalog[0].String;
			bool Translated =MyCatalog.Translate(key,"translated");


			Assert.IsTrue(Translated,"failing");
		}

		[Test()]
		public void FindItemTest()
		{
			//se prueba buscando un string conocido dentro de una
			//archivo de localizacion, asi se comprueba
			Catalog MyCatalog= new Catalog();
			bool isOk=MyCatalog.Load(poFile);
			string key=MyCatalog[0].String;
			l10nCommunity.Gettext.CatalogEntry
			FoundEntry=MyCatalog.FindItem(key);
			Assert.IsNotNull(FoundEntry,"failing");
		}


}
```

Código Nº 17. Pruebas clase catalog

Como se nota en la sección del código la clase hereda de “TestBase”, cuyo contenido es el que sigue.

```
public class TestBase
{
        protected  string poFile="./TestPofiles/es.po";


        protected Catalog GetCatalog()
        {
                Catalog MyCatalog= new Catalog();
                bool isOk=MyCatalog.Load(poFile);
                return MyCatalog;
        }
}
```

Código Nº 18. Clase base de pruebas

8.3.2. Ejecución de las pruebas de unidad en Linux

La realización de las pruebas pudieron ser ejecutadas de las siguientes formas:

- Al mismo tiempo que se fue construyendo la aplicación se invocó la aplicación NUnit-console.exe desde la consola entregando como parámetro el assembly a probar. La figura 22 muestra la ejecución de un proyecto de pruebas.
- Utilizando NAnt se creó un script para correr todos los proyectos de pruebas de la solución. Una explicación más detallada en el capítulo “Integración Continua”.

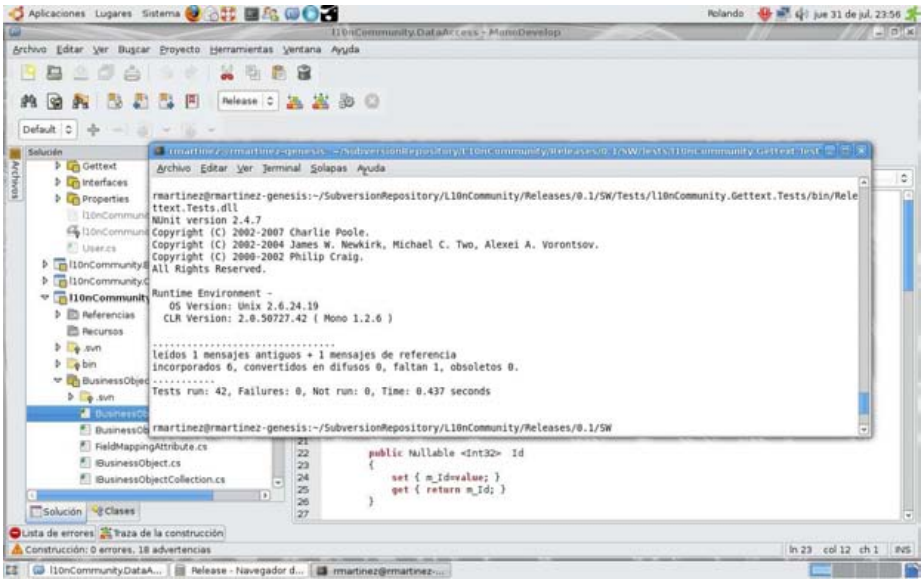


Figura Nº 22. Ejecución de proyecto L10nCommunity.Gettext.Tests

8.4. Deployment

8.4.1. Instalación de la base de datos

La instalación de la base de datos se lleva a cabo ejecutando un script bash, el cual a su vez llama una serie de script con extensión “.sql”. En estos archivos se encuentran el esquema de datos junto con los procedimientos almacenados de la herramienta.

Los archivos script que contienen el esquema de la base de datos pueden ser encontrados en la ruta:

“$workingdirectory/L10nCommunity/Releases/$Version/SW/I10nCommunity.DataAccess/MySqlDataBase/Scripts”. Un extracto del script a continuación.

```
#!/bin/bash
scripts_path='Scripts/'
username="root"
password="holamundo"
host="localhost"

cd $scripts_path

sqlfile="CreateDataBase.sql"
mysql -h $host -u $username -p$password < $sqlfile
sqlfile="Populatelookups.sql"
mysql -h $host -u $username -p$password < $sqlfile
sqlfile="SecurityTables.sql"
mysql -h $host -u $username -p$password < $sqlfile
sqlfile="PopulateSecurityTablesTest.sql"
mysql -h $host -u $username -p$password < $sqlfile
sqlfile="IsTranslatorFor.sql"
mysql -h $host -u $username -p$password < $sqlfile
sqlfile="Language.sql"
```

Figura Nº 23. Script bash de instalación de base de datos.

Dentro del script se puede ver el nombre de usuario y password del usuario mysql con privilegios necesarios para la creación de una base de datos mysql. El valor de estas variables, debiera cambiar de acuerdo al servidor de base de datos donde se quiera instalar la aplicación.

8.4.2. Instalación de módulo Mono en Apache para la ejecución de la Aplicación ASP.NET

Si bien durante toda la etapa de desarrollo de la herramienta se utilizó el servidor web “XSP” incluido con mono, al finalizar esta etapa se instala el complemento necesario para la ejecución de paginas “aspx” utilizando el servidor web apache. Los pasos necesarios para la instalación, se encuentran en el anexo B “Instalación y Configuración de la aplicación en Apache”.

Para la instalación de este módulo se hicieron necesarios seguir los siguientes pasos:

- Descargar los paquetes para la ejecución de aplicaciones ASP.NET.
- Editar el archivo de configuración de Apache para registrar el módulo de mono.
- Hacer una primera prueba ejecutando los ejemplos para ASP.NET incluidos con mono.
- Por último registrar la herramienta editando el archivo de configuración de Apache.

8.4.3. Actualización del esquema físico de datos

Luego que el esquema físico haya sido creado se hace necesario poder tener algún procedimiento para poder actualizarlo. Para el caso de esta herramienta, el procedimiento comprende de la creación de archivos script con extensión sql con los correspondientes cambios aplicados al modelo y una tabla que contenga la actual versión de la base de datos. Más información relacionada con versionamiento de base de datos se encuentra en [Allen2008].

9. Transición

Durante esta etapa se creó un espacio de trabajo en la web alojado en un servidor de la carrera de Ingeniería en Computación de la misma Universidad que contiene tantos archivos ejecutables como fuentes para que cualquier interesado pueda extenderlo.

En términos de aceptación del producto, la implementación de los requerimientos fueron validados por el profesor patrocinante Sr. Moisés Coronado.

Por otra parte, uno de los primeros pasos pensados para la difusión de este proyecto son: ponerse en contacto con personas involucradas en el sistema operativo Famelix, los cuales podrán ocupar este software para poder localizar aplicaciones creadas en este sistema operativo.

10. Integración Continua

Dentro de la fase de construcción de la metodología empleada en este seminario, se investigó el proceso de “integración continua”, practica ampliamente usada en las metodologías ágiles y que se describe a continuación.

La integración en el desarrollo de software consiste en la unión de cada uno de los componentes que conforman una aplicación. Estos componentes, los cuales pueden ser esquema de la base de datos, procedimientos almacenados, archivos de código fuente y scripts de instalación necesitan trabajar de forma adecuada (por ejemplo el esquema de la base de datos este correctamente sincronizado con la definición de las clases de acceso a datos) desde la concepción de ciclo de desarrollo de la aplicación. Ahora bien, teniendo como base lo anteriormente comentado, integración continua es una práctica de desarrollo de software donde los miembros de un grupo integran su trabajo frecuentemente, usualmente cada persona integra al final del día, idealmente teniendo más de una integración en el día. Cada integración es verificada por un “build” o construcción automática (que incluyen tests) que detecta errores tan rápido como sea posible, haciendo posible que el equipo de desarrollo construya software con mayor rapidez.

Algunos alcances sobre esta práctica son:

- Los desarrolladores ejecutan construcciones (builds) privados sobre sus propias máquinas antes de subir sus cambios al repositorio de código.
- Los desarrolladores suben sus cambios al repositorio de código al final del día.
- Cada build de integración (integration build) ocurre muchas veces al día sobre una máquina separada (el trabajo realizado por el grupo de desarrolladores se centraliza en un única máquina).
- 100% de los tests deben pasar para cada build.
- Problemas de integración deben ser parchados con la rapidez posible.
- Desarrolladores revisan los resultados de cada build fijándose en tests, recubrimiento de código y alguna métrica dispuesta por el equipo de desarrollo.
- La práctica de integración continua permite una rápida retroalimentación debido a que permite entregar información rápida y confiable a los desarrolladores acerca de posibles errores en la integración de un producto software.

10.1 Proceso de Integración Continua

Un escenario de integración continua comienza con un desarrollador haciendo un “commit” de sus cambios hacia el repositorio de código, estos cambios pueden ser en alguna clase, script de definición de datos, archivos de configuración, entre otros. A partir de aquí se gatilla una serie de pasos que el servidor de integración ejecutará.

Los pasos dentro de un escenario de integración continua es como sigue:

- Primero, un desarrollador sube sus cambios al repositorio.
- Tan pronto como se suben los cambios, el servidor de integración continua detecta los cambios que han sido subidos al repositorio. De esta manera el servidor de integración continua recupera la última versión de código desde el repositorio y entonces ejecuta un build script, el cual integra el software.
- El servidor de integración entrega información sobre los resultados de la integración ocurrida. El estado de esta última integración puede ser enviada por medio del e-mail o revisada a través un visor web.
- El servidor de integración continua espera por los próximos cambios en el repositorio de código.

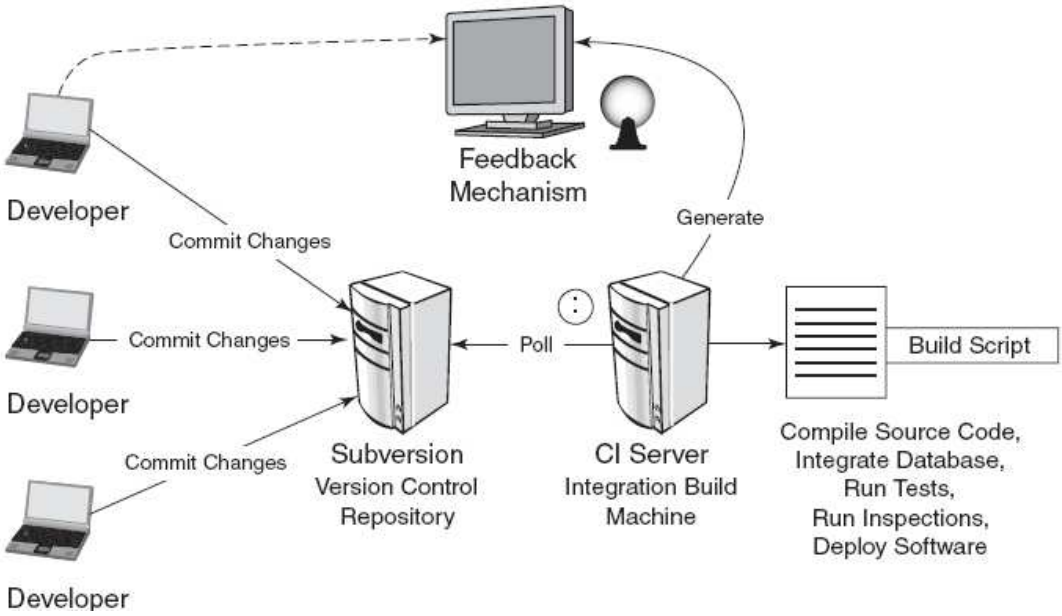


Figura Nº 24. Proceso de integración continua

10.2 Herramientas utilizadas dentro integración continua

Dependiendo del framework de desarrollo, se utilizarán unas u otras herramientas. Para el caso de aplicaciones construidas sobre la plataforma .Net., pueden ser utilizadas las siguientes listadas en la tabla.

Tabla Nº 26. Aplicaciones involucradas en el proceso

Nombre Herramienta	Descripción
Subversión, SourceSafe, CVS, Racional Clear Case, otros.	Servidores de repositorio de códigos.
CruiseControl.Net Draco.Net	Servidor de integración continua
Nant, FinalBuilder, MSBuild, Visual Studio.	Herramientas para la construcción de un software.
Nunit, Mbunit, NCover, MBunit, NDepend, MSTest.	Framework de pruebas de unidad, cobertura y otras métricas.

10.3 Actores involucrados en el proceso

10.3.1. Desarrollador

Es la persona que se encarga de crear o modificar código fuente existente que van hacia el cumplimiento de un requerimiento específico. Antes que el desarrollador suba sus cambios al repositorio de control de versiones, es necesario correr un “build” privado dentro de su propia máquina. Herramientas

que apoyan la ejecución de build privados son NAnt para el caso de .Net y Ant para el caso proyectos de desarrollados en Java. Una vez comprobado que el desarrollador no romperá el proceso de integración (problemas de compilación o tests que fallen) los cambios son subidos al repositorio de control de versiones.

10.3.2. Repositorio de Control de Versiones

El propósito de un repositorio de control de versiones es administrar los cambios realizados en el código fuente y otros archivos relacionados (documentación, dll's de terceros) usando un control administrado al repositorio. Esto provee a los desarrolladores un único punto de acceso al código, de tal forma que se disponible desde un único lugar primario. Un sistema de control de versiones permite regresar atrás en el tiempo y obtener distintas versiones del código fuente. Algunos de los sistemas de control de versiones más conocidos son: Subversión, CVS, Visual Source, PVCS y MKS. Documentación exhaustiva relacionada con teoría de sistemas de control de versiones se encuentra en [Sink2004] y [Subversion2008].

10.3.3. Servidor de Integración Continua

Un servidor de integración o servidor IC, correrá una integración siempre y cuando un cambio haya sido introducido al repositorio de código fuente. De manera típica el servidor IC se configurará para revisar periódicamente el repositorio de control de versiones. El servidor IC traerá los archivos de código fuente y correrá un “build script”. Junto con eso el servidor IC contendrá un sincronizador para construir una integración cada cierto tiempo (todas las noches a cierta hora), donde los resultados de cada integración serán anunciadas utilizando algún medio, como correo electrónico o un visor web.

10.3.4. Build Script

Este puede ser uno o un set de archivos de script escritos utilizando un lenguaje descriptivo como XML. Las principales funciones que tiene este script son:

- Compilar. Necesario para compilar el conjunto de proyectos que compone el producto software.
- Testear. Utilizando un framework de pruebas de unidad como NUnit o Mbunit se corren las pruebas de unidad.
- Inspeccionar. A través de frameworks de cobertura como NCover, las pruebas son inspeccionadas para revisar que tan cubierto se encuentra

el código fuente de producción o código que finalmente será ejecutado por el cliente.

- Desplegar. Utilizando herramientas de construcción como Nant, se pueden construir paquetes que contengan todo lo necesario para instalar el software.

10.3.5. Mecanismo de publicación

Uno de las tareas de un servidor IC es poder publicar los resultados de una integración, de tal forma que cada uno de los integrantes del equipo de desarrollo sepa cual es el actual estado de una integración. Para el caso que hubieran problemas en la integración (problemas de compilación, pruebas que fallen o alguna métrica que no cumpla con los estándares implantados por el equipo de desarrollo), el desarrollador responsable del problema deberá darle solución tan pronto como sea posible.

Una guía teórica relacionada con integración continua se encuentra en [Duvall2007].

10.4. Integración continua para este desarrollo

A continuación se entrega un guía práctica al lector de cómo configurar un servidor de integración continua en el sistema operativo Linux.

10.4.1. Herramientas

Dentro de las herramientas necesarias para la configuración de servidor de integración continua se encuentran las siguientes:

- CCNet: Servidor de integración continua para los proyectos desarrollados en la plataforma .Net. Disponible en [Ccnet2008].
- Nant: Herramienta de construcción útil para la compilación y ejecución de prueba de unidad, además herramientas de diagnóstico y empaquetado que se puedan necesitar. Disponible en [Nant2008].
- Nantcontrib: Herramienta que cubre necesidades de construcción que no son soportadas por Nant. Disponible en [Nantcontrib2008].

10.4.2. Instalación de CCNet

Las instalación del servidor de integración continua CCNet consta de las descarga y copiado de los archivos binarios ejecutables a un lugar apropiado que el usuario elija (por ejemplo /home/ccnetuser/ccnet).

Luego de elegir desde donde la aplicación será ejecutada, se puede iniciar el servidor ejecutando la aplicación “ccnet.exe”, el cual hace lectura del archivo de configuración “ccnet.config” para poder ejecutar las tareas de integración. La figura muestra la iniciación del servidor.

```
rmartinez@rmartinez-desktop:~/ccnet/server$ mono ccnet.exe
CruiseControl.NET Server 1.4.0.3524 -- .NET Continuous Integration Server
Copyright © 2008 ThoughtWorks Inc.  All Rights Reserved.
.NET Runtime Version: 2.0.50727.42 [Mono]       Image Runtime Version: v2.0.5072
7
OS Version: Unix 2.6.24.19      Server locale: es-CL

log4net:ERROR XmlHierarchyConfigurator: Cannot find Property [level] to set obje
ct on [log4net.Appender.ConsoleAppender]
[CCNet Server:DEBUG] The trace level is currently set to debug.  This will cause
 CCNet to log at the most verbose level, which is useful for setting up or debug
ging the server.  Once your server is running smoothly, we recommend changing th
is setting in /home/rmartinez/ccnet/server/ccnet.exe.config to a lower level.
[CCNet Server:INFO] Reading configuration file "/home/rmartinez/ccnet/server/ccn
et.config"
[CCNet Server:INFO] Registered channel: tcp
[CCNet Server:INFO] Starting CruiseControl.NET Server
[l10ncommunity:INFO] Starting integrator for project: l10ncommunity
```

Figura Nº 25. Iniciación del servidor

10.4.3. Configuración CCNet

El archivo de configuración para el servidor de integración continua consta de una serie de bloques, los cuales serán llamados o ejecutados por CCNet de una forma secuencial. La figura 26 muestra el contenido del archivo de configuración para el servidor de integración continua.

```
<cruisecontrol>
    <project name="l10ncommunity">
        <sourcecontrol type="svn">
            <trunkUrl>http://192.168.100.23/svn/myrepo</trunkUrl>
            <workingDirectory>/home/rmartinez/l10ncommunityci
            </workingDirectory>
            <executable>/usr/bin/svn</executable>
        </sourcecontrol>
        <nant>
            <executable>
                nant
            </executable>
            <baseDirectory>
                /home/rmartinez/l10ncommunityci/L10nCommunity/Releases/0.1/Deployment/
            </baseDirectory>
            <buildfile>
                solution.build
            </buildfile>
        </nant>
    </project>
</cruisecontrol>
```

Figura Nº 26. Archivo de configuración de CCNet.

La descripción de cada bloque a continuación:

- sourcecontrol: Este bloque específica que tipo de repositorio que será utilizado por CCNet. Además, se especifican la dirección del repositorio, directorio de trabajo y el cliente utilizado para descargar el código fuente.

- nant: Terminada la descarga y compilación del código, se llama a NAnt para que ejecute tareas como creación de la base de datos y ejecución de las pruebas de unidad.

10.4.2. Archivo de configuración para NAnt

El archivo de configuración llamado “solution.build” se divide en bloques llamados “targets”, donde cada target puede contener una o más tareas. La descripción de cada target a continuación.

- DeployDataBase: Utilizando el tag “MySql”, NAnt permite ejecutar cualquier script mysql. Este “target” carga el esquema y cualquier objeto dentro de la base de datos (procedimientos almacenados, vistas, etc.). Un ejemplo de cómo utilizar este tarea se muestra a continuación:

```
<MySql connstring="${Project.ConnectionString}"
source="${Project.DataBaseDirectory}\CreateDataBase.sql" />
```

- UnitTest: Este target es responsable de copiar cada una de las asemblies a un directorio arbitrario. Luego de lo anterior se llama al programa de consola de NUnit, el cual tomará los assemblies y buscará clases que contengan pruebas de unidad.

```
<target name="UnitTests">
    <echo message="UnitTest report is found in ${UnitTestOutputDir}" />
        <property name="UnitTest.BinDir" value="UnitTestBinDir" overwrite="false" />
    <mkdir dir="${UnitTest.BinDir}" />
    <copy flatten="true" todir="${UnitTest.BinDir}">
        <fileset>
            <include name="${assembliesfolder}\*.dll" />
            <include name="${assembliesfolder}\*.pdb" />
            <include name="${assembliesfolder}\\*.dll.config" />
        </fileset>
    </copy>
    <call target="RunUnitTests" />
</target>
<target name="RunUnitTests">
    <exec program="${nunitconsole}" failonerror="false" resultproperty="testresult.unittestassembly">
        <arg value="${UnitTest.BinDir}\l10nCommunity.DataAccess.Tests.dll" />
        <arg value="/xml=${UnitTestOutputDir}\UnitTestAssembly-Results.xml" />
    </exec>
</target>
```

Figura Nº 27. Archivo de configuración para nant

Al finalizar la instalación y configuración se tiene un servidor de integración continua trabajando en un equipo linux. Si bien la instalación del visor gráfico web para ver el estado de las construcciones no pudo ser instalada debido a problemas de compatibilidad con el runtime de Mono, el alumno logró que el servidor CCNet permita crear build de acuerdo a las actualizaciones en el repositorio de código.

11. Proyecto Mono

Mono es el proyecto de código abierto impulsado por Novell para crear un grupo de herramientas libres, basadas en GNU – Linux y compatibles con la plataforma de desarrollo .NET según lo especificado en el estándar ECMA. Uno de los objetivos originales para la creación de este proyecto fue facilitar la creación de aplicaciones de escritorio para aplicaciones Linux reduciendo el tiempo y los costes de desarrollo, pero con el tiempo esta plataforma ha entregado la posibilidad de ejecutar en Linux aplicaciones que han sido diseñadas en Microsoft .NET.

Mono implementa las siguientes partes de la tecnología .NET:

- Common Language Runtime
- Compilador / Desensamblador IL
- Compilador C#
- Compilador Visual Basic .NET
- Librería de clases
- Otras librerías

Con el framework de aplicaciones Mono se pueden escribir aplicaciones en múltiples lenguajes de programación, entre ellos Python, Object Pascal, Nemerle C# y Visual Basic .NET. Una vez escritas las aplicaciones se traducen

a CIL (Common Intermediate Language), que es un lenguaje intermedio que no tiene particularidades de ninguna arquitectura. Una vez compilado en CIL la aplicación se traduce al lenguaje específico de la arquitectura final donde será ejecutado, de esta forma se distribuye un único programa binario para todas las arquitecturas en vez de un programa específico para cada una de ellas. En la actualidad el proyecto Mono soporta las siguientes arquitecturas:

- X86
- SPARC
- StrongARM
- S390

Como la figura 28 muestra, Mono presenta una atractiva pila de lenguajes y arquitecturas para la creación y ejecución de aplicaciones que a diferencias de programas tradicionales que se ejecutan sobre el sistema operativo de forma directa, los programas en la plataforma Mono se ejecutan sobre un entorno controlado de ejecución conocido como máquina virtual. Este entorno proporciona ventajas tales como: gestión de memoria automática, un entorno seguro de ejecución y un sistema de control de errores.

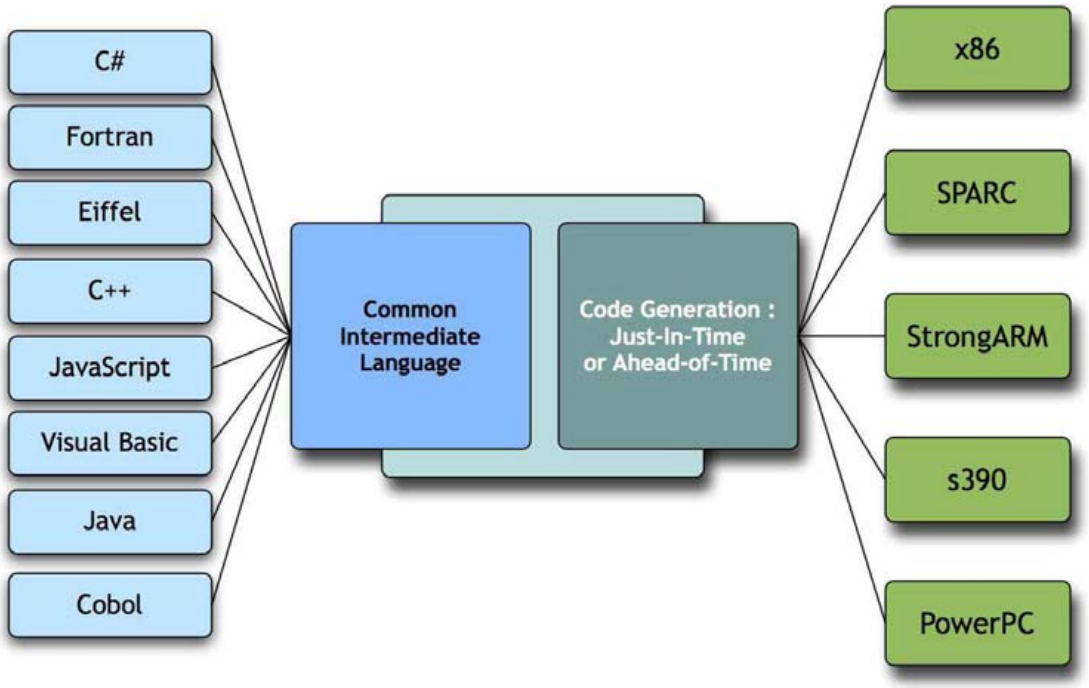


Figura Nº 28. Pila de lenguajes y plataformas

Otro punto importante en Mono es la completa implementación de los espacios de nombres más importantes en .NET como son XML y ASP.NET para aplicaciones web y ADO.NET para acceso a base de datos.

11.1. Mono y aplicaciones ASP.NET

Para la creación de aplicaciones webs, Mono tiene a disposición dos formas de servir las paginas web; la primera es a través del servidor XSP escrito

íntegramente en C# y la segunda es configurando Apache para poder utilizar el modulo “mod_mono”. Ambos se explican a continuación.

11.1.1. Servidor XSP

XSP es un servidor web “liviano” que fue escrito en lenguaje de programación C# usando las clases del espacio de de nombres System.Web. El código fuente para este servidor está disponible y puede ser utilizado en sistemas operativos Windows como en Linux. Para usar este servidor web se debe seguir los siguientes pasos:

- Instalación. Por lo general este servidor viene incluido en el paquete estándar de Mono para Windows. Para el caso de linux, se hará necesario utilizar algún programa de descarga como apt-get o el mismo código fuente.
- Ejecución del servidor. XSP está configurado para correr sobre un puerto específico usando el parámetro –port en la línea de comandos cuando se ejecuta. El puerto por defecto para un servidor web de test en Linux es el 8080. La figura 29, muestra la ejecución de XSP en Linux.

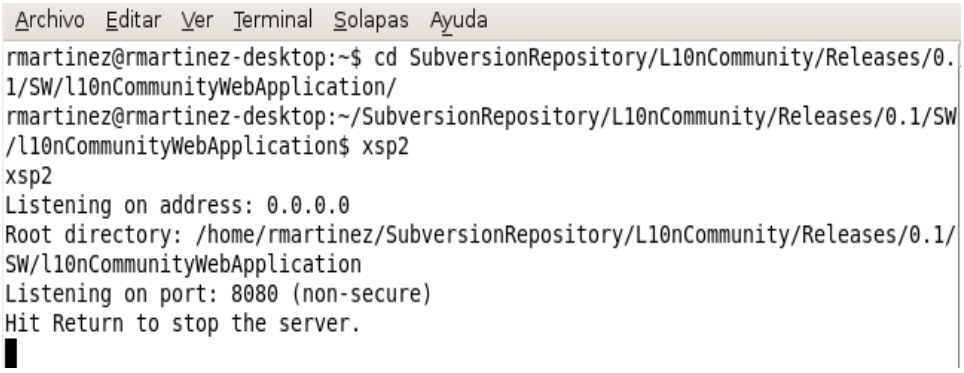


Figura Nº 29. Inicio del servidor xsp

11.1.2. Módulo Mono para Apache

El modulo de “mod_mono” corre dentro de apache y pasa cada requests que se dirijan a aplicaciones ASP.NET a un proceso externo llamado “mod-mono-server”, el cual es parte del modulo XSP. La comunicación entre Apache y “mod-mono-server” es establecida usando unix socket o tcp socket. Para usar esta alternativa se requiere:

- Instalación del servidor web Apache e instalación del módulo mod_mono.
- Edición de archivo de configuración del servidor Apache, el cual se encuentra en la carpeta donde fue instalado el servidor Web. La figura 30, muestra una entrada dentro del archivo “Apache2.conf”.

```
Include /etc/apache2/sites-enabled/
Alias /test "/usr/share/asp.net2-demos"
Alias /l10n "/var/www/l10nCommunity/l10nCommunityWebApplication"


AddmonoApplications default "/test:/usr/share/asp.net2-demos"
#AddmonoApplications default "/l10n:/var/www/l10nCommunity/l10nCommunityWebApplication"

<Location /test>
 SetHandler mono
</Location>

<Location /l10n>
 SetHandler mono
</Location>
```

Figura Nº 30. Archivo de configuración "apache2.conf"

Mas información relacionada con las distintas alternativas de instalación y ejecución de aplicaciones web se encuentran en [Mamone2007].

11.2. Proveedores de Datos

Si bien el diseño de la interfaz gráfica de una aplicación es importante para el éxito de la misma, igual de importante es la habilidad que tenga una aplicación de poder leer y escribir información en un medio persistente como lo es una base de datos. Es así como dentro plataforma .Net existe el paradigma de modelo de proveedor [Msdn2004] del cual ADO.NET hace uso. Dentro de Mono se puede hace uso de la siguiente lista de proveedores de datos:

- Microsoft SQL Server
- PostgreSQL (Npgsql)
- Oracle (Mono)
- ODBC (Mono)
- MySqlConnector (MySql)

12. Conclusiones

El desarrollo de aplicaciones que puedan ser utilizadas en múltiples localidades geográficas, ha traído consigo la necesidad de poder crear estándares de internacionalización y localización que puedan ser utilizados por programadores y traductores voluntarios. Dentro de este contexto es que los traductores necesitan de herramientas capaces de poder acoger sus necesidades tales como tener un repositorio de archivos de localización centralizado que sea capaz de poder editar estos archivos. Es por ello que la herramienta creada en este seminario puede llegar a constituir una gran ayuda a lo que se refiere a la localización de aplicaciones Linux.

Junto con lo anterior, se puede establecer que los requerimientos y/o funcionalidades esperados por esta herramienta se han cumplido satisfactoriamente de acuerdo a lo esperado por el experto del dominio que en este caso fue el profesor patrocinante de este seminario. Sin embargo, se debe ser crítico en que el sistema aún no ha sido lanzado de forma pública hacia la comunidad “Open Source” de tal forma de recibir retroalimentación de los usuarios finales, es decir, los traductores.

Por otra parte, dentro de la elección de la metodología se entregan prácticas que son poco conocidas dentro del ambiente académico como son el uso de un

repositorio de código, un servidor de integración continua y la utilización de un framework de pruebas de unidad. Estas prácticas son altamente beneficiosas tanto para el desarrollo de una aplicación como también en términos profesionales para quienes lo practiquen.

Otro punto importante dentro de desarrollo de este seminario fue la elección del ambiente de desarrollo, el cual en su primera etapa fue un ambiente Windows utilizando herramientas de desarrollo Microsoft, pero que finalmente decantaron al uso de herramientas libres como el proyecto Mono y su entorno de desarrollo integrado MonoDevelop. Algunas observaciones relacionadas a estos dos últimos proyectos de software libre fueron la buena disponibilidad de los voluntarios en poder dar solución a problemas encontrados durante el desarrollo de esta herramienta. Si bien al comienzo de este seminario había un grado de incertidumbre sobre la capacidad del framework Mono, estas dudas quedan resueltas al comenzar a trabajar sobre Monodevelop y su conjunto de herramientas.

Finalmente, relacionada con la extensibilidad de la herramienta, queda abierta la posibilidad de agregar nuevas funcionalidades como el soporte a nuevos estándares de localización, como son el estándar para aplicaciones ASP.NET el cual se basa en archivos XML. Otra posible extensión a hacer a esta herramienta es la posibilidad de entregar al usuario sugerencias en sus

traducciones utilizando como fuente o “diccionario” el sistema de traducción de google, o cualquier otra fuente similar (babelfish, systran, etc).

13. Bibliografía

[Allen2008] Allen, Scott: Versioning Databases. Disponible en http://odetocode.com/Blogs/scott/archive/2008/01/31/11710.aspx

[Ambler2005] Ambler, Scott W. The Agile Unified Process (AUP). Disponible en http://www.ambysoft.com/unifiedprocess/agileUP.html, Septiembre 2005.

[Ambler2003] Ambler, Scout W. Agile Database Techniques. Willey. Primera Edición. 2003.

[Ccnet2008] Servidor de Integración Continua, disponible en http://ccnet.thoughtworks.com

[Connolly2005] Connolly y Begg. Sistema de bases de datos. Un enfoque práctico para diseño, implementación y gestión. Addison Wesley, Cuarta Edición. 2005.

[Duvall2007] Paul M. Duvall. Continuous Integration. Primera Edición. 2007.

[Eguíluz2008] Javier Eguíluz Perez. Introducción a CSS. Primera Edición. Febrero 2008.

[Entreprise2008] Enterprise Library, Patterns and Practices. Disponible en http://www.codeplex.com/entlib.

[Esposito2006] Dino Esposito. Programming Microsoft ASP.NET 2.0. Core Reference. Primera Edición.

[Fowler2002] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford. Patterns of Enterprise Application Architecture. Addison Wessley. Noviembre 2002.

[Fowler1997] Fowler Martin. Uml gota a gota. Addison Wesley. 1999

[Gnu1995] Gnu-Gettext Utilities. Disponible en http://www.gnu.org/software/gettext/manual/gettext.html.

[Johnson2005] Glenn Jonson, Tony Northrup. .NET Framework 2.0 Web Based Client Development. Microsoft Press. 2005.

[Kruchten2003] Kruchten Phillipe. Rational Unified Process. Tercera Edición. Addison Wesley. Diciembre 2003.

[Lhotka2006] Rockford Lhotka. Expert C# 2005 Bussines Objects. Segunda Edición.

[Lisa2008] Localization Industry Standards Association. Disponible en http://www.lisa.org/.

[Mamone2007] Mamone Mark. Practical Mono. Apress, Primer Edición.

[Mono2008] Pagina del proyecto Mono. Disponible en http://www.mono-project.com/Main_Page.

[Msdn2004] Especificación del Modelo Proveedor, Libreria MSDN. 2004. Disponible en http://msdn.microsoft.com/en-us/library/ms972319.aspx

[Nant2008] Herramienta para automatización de tareas. Disponible en http://nant.sourceforge.net/.

[Nantcontrib2008] Extensiones para la herramienta para automatización de tareas Nant. Disponible en http://nantcontrib.sourceforge.net/

[NewKirk2004] James W. Newkirk: TestDriven Development in Microsoft .Net. Microsoft Press. Primera Edición. 2004.

[Pressman2001] Pressman Roger. Software Engineering. MacGraw-Hill. Quinta edición, 2001.

[Sink2004] Sink Eric. Source Control, How to. Disponible en http://www.ericsink.com/scm/source_control.html, Agosto 2004.

[Subversion2008] Pagina del proyecto Subversión. Disponible en http://subversion.tigris.org/.

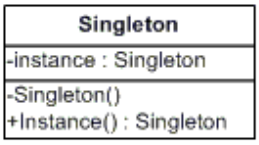
14. Anexos

Anexo A. Patrones de diseño usados

14.1. Singleton

Asegura que una clase tenga una única instancia dentro del sistema y provea un punto global de acceso a ella.

Diagrama UML



Participantes:

- Singleton:
  - ✓ Responsable de crear y mantener su única instancia.
  - ✓ Define un método llamado “instance” que permite a los clientes acceder a su única instancia. “Instance” es una operación de clases o estática.

Un ejemplo del patrón se encuentra en el siguiente código, donde fácilmente se puede notar que el cliente, en este caso MainApp, solo puede acceder a una instancia de la clase singleton utilizando el método estático “instance”.

```
class MainApp
{

        static void Main()
        {
        // Constructor is protected -- cannot use new
        Singleton s1 = Singleton.Instance();
        Singleton s2 = Singleton.Instance();

                if (s1 == s2)
                {
                        Console.WriteLine("Objects are the same instance");
                }
        }
}

class Singleton
{
        private static Singleton instance;

        protected Singleton()
        {

        }

        public static Singleton Instance()
        {
                if (instance == null)
                {
                        instance = new Singleton();
                }

                return instance;
        }
}
```

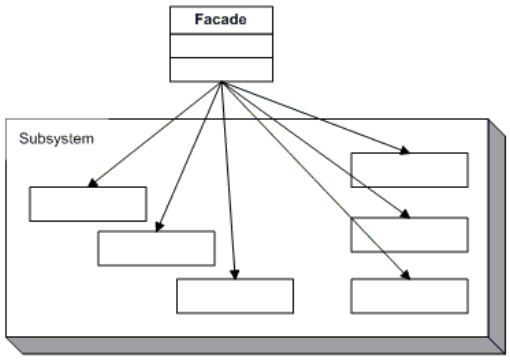
En el siguiente ejemplo se puede apreciar el uso del patrón singleton con una clase de acceso a datos. De esta forma la aplicación contendrá una única instancia de esta clase durante todo su ciclo de vida.

```
public class DataManager
    {
        private static DataManager m_Instance;


        protected DataManager()
        {
        }


        public static DataManager Instance()
        {
            if (m_Instance ==  null)
            {
                m_Instance = new DataManager();
            }
            return m_Instance;
        }
       //Codigo de manipulacion de objetos del tipo BusinessObject..
}

public class BusinessObject: IBusinessObject
{


       public static DataManager DataProvider()
        {
            return DataManager.Instance();
        }

        public void Store()
        {
            switch(GetState)
            {
                case ObjectState.NotAddedToDataStorage:
                    ApplyValidationsOnInsert();
                    DataProvider().Insert(this);
                    this.m_objectState =
ObjectState.LoadedFromDataStorage;
                    break;
                case ObjectState.LoadedFromDataStorage:
                    ApplyValidationsOnUpdate();
                    DataProvider().Update(this);
                    this.m_objectState =
ObjectState.LoadedFromDataStorage;
                    break;
                case ObjectState.MarkedForDeletion:
                    ApplyValidationsOnDelete();
                    DataProvider().Delete(this);
                    this.m_objectState =
ObjectState.DeletedFromDataStorage;
                    break;
            }
        }

       //codigo de manipulacion de atributos del objeto..

}
```

14.2. Facade

Provee una interface unificada para un conjunto de interfaces dentro de un subsistema. Facade define una interface de alto nivel para hacer el subsistema fácil de usar.

Diagrama UML



Participantes

- Facade
  - ✓ Sabe que clases del subsistema son responsables para una tarea específica.
  - ✓ Delega las tareas a los diferentes objetos dentro del subsistema.
- Clases del subsistema
  - ✓ Implementa la funcionalidad del subsistema.

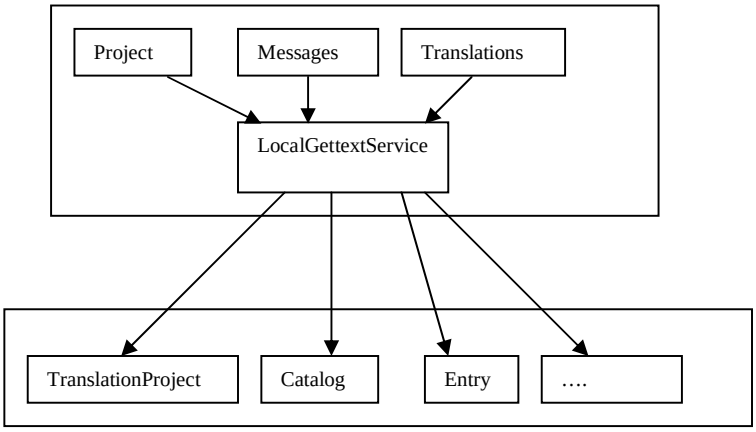
- ✓ Maneja el trabajo asignado por Facade.
- ✓ No tiene conocimiento de la existencia de Facade y no mantiene referencias a esta clase.

El código de ejemplo muestra a Facade delegando responsabilidades a las distintas clases del subsistema.

```
// "Facade"

 class Facade
 {
   SubSystemOne one;
   SubSystemTwo two;
   SubSystemThree three;
   SubSystemFour four;

   public Facade()
   {
     one = new SubSystemOne();
     two = new SubSystemTwo();
     three = new SubSystemThree();
     four = new SubSystemFour();
   }

   public void MethodA()
   {
     Console.WriteLine("\nMethodA() ---- ");
     one.MethodOne();
     two.MethodTwo();
     four.MethodFour();
   }

   public void MethodB()
   {
     Console.WriteLine("\nMethodB() ---- ");
     two.MethodTwo();
     three.MethodThree();
   }
 }
```

```
// Mainapp test application

class MainApp
{
  public static void Main()
  {
    Facade facade = new Facade();

    facade.MethodA();
    facade.MethodB();

    // Wait for user
    Console.Read();
  }
}

// "Subsystem ClassA"

class SubSystemOne
{
  public void MethodOne()
  {
    Console.WriteLine(" SubSystemOne Method");
  }
}

// Subsystem ClassB"

class SubSystemTwo
{
  public void MethodTwo()
  {
    Console.WriteLine(" SubSystemTwo Method");
  }
}

// Subsystem ClassC"

class SubSystemThree
{
  public void MethodThree()
  {
    Console.WriteLine(" SubSystemThree Method");
  }
}

// Subsystem ClassD"

class SubSystemFour
{
  public void MethodFour()
  {
    Console.WriteLine(" SubSystemFour Method");
  }
}
```

El siguiente ejemplo muestra el uso del patrón facade en la interacción entre el componente de L10nCommunity.BusinessLayer y el componente l10nCommunity.Gettext.

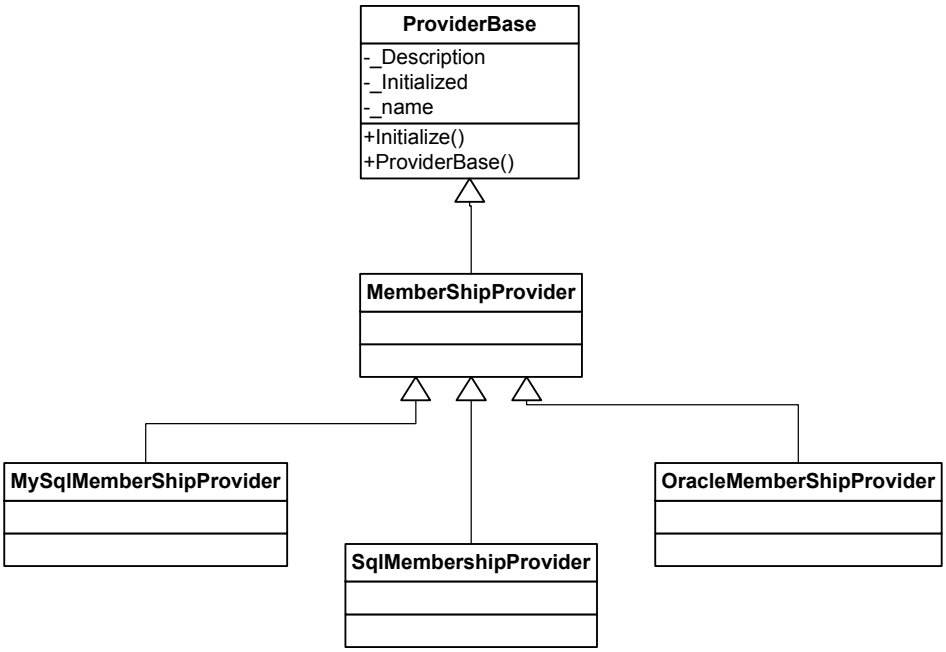


```
namespace l10nCommunity.BusinessLayer.Gettext
{

   //Clase responsable de comunicarse con el componente l10nCommunity.Gettext.
    public class LocalGettextService
   {

       //crear los archivo MO para el proyectos
       //recorre todos los archivos pot del proyecto y va generando los
       //correspondientes archivo MO
       public void BuildProject(Project projectDB)
        {

              TranslationProject tp = new
              TranslationProject(projectDB.ProjectName);
              string templatefileNamewithoutExtension;

           foreach (TemplateFile potfileDB in projectDB.PotFiles)
           {
               templatefileNamewithoutExtension = potfileDB.Name.Substring(0,
               potfileDB.Name.Length - 4);

               TemplateCatalog potfile =
               tp.AddNewTemplate(templatefileNamewithoutExtension);
               foreach (FileByLanguage poFileDB in potfileDB.PoFiles)
               {
                     string pofileWithoutExtension =
                     poFileDB.Name.Substring(0, poFileDB.Name.Length - 3);

                     potfile.AddNewTranslation(poFileDB.IsoCode,
                     pofileWithoutExtension);
               }
               potfile.BuildMoFiles();
           }
       }
    }
```

14.3 Modelo Proveedor

Patrón estructural creado por Microsoft y que consiste en una fusión de patrones de diseño “strategy” y “abstract factory”. El modelo de proveedor permite exponer una interface común con métodos sobreescribibles por las clases que hereden de la clase Provider. La forma en que una u otra clase es cargada y utilizada por el cliente del patrón se basa en archivos de configuración donde se especifica que clase debe ser cargada.

Diagrama UML



Participantes

- ProviderBase: Clase interna de .Net que es usado para definir el provider concreto (en este caso cualquiera de las implementaciones xxxMemberShip). El método “Initialize” toma la información de un archivo de configuración (web.config o app.config) y luego inicializa la clase concreta correspondiente.
- MemberShipProvider: Clase abstracta que contiene métodos sobreescribibles, tales como AddUser, DeleteUser, entre otros. Esta clase tiene la propiedad de poder sobrescribir el método initialize de la clase ProviderBase.
- MySqlMemberShip – SqlMemberShip – OracleMemberShip: Sobrescriben los métodos creados en MemberShipProvider de acuerdo a los motores correspondientes.

Dentro de .Net existen además los siguientes proveedores abstractos para las siguientes tareas:

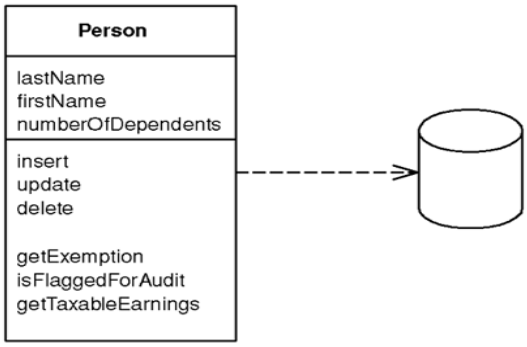
- Session State Providers
- Site Map Providers
- Web Events Providers

- Web Parts Personalization Providers

Junto con la lista anterior, este patrón esta abierto para agregar nuevos proveedores de acuerdo a las necesidades de desarrollo de cada aplicación.

14.4 Patrón de Registro Activo

Un objeto que representa una fila en una tabla de la base de datos, encapsula el acceso a la base de datos y agrega lógica del dominio.



La esencia de este patrón es que cada instancia de una clase “active record” debe coincidir con la estructura de un registro en la base de datos. Cada objeto de este tipo es responsable de guardarse y cargarse desde la base de datos y además agregar lógica del dominio según corresponda.

Clases active records contienen típicamente los siguientes métodos:

- Construye una instancia de Active Record desde un set de resultados provenientes de una consulta SQL.
- Construye una nueva instancia para más tarde insertarla en la base de datos.

- Actualizar o insertar los valores correspondientes a sus propiedades en la base de datos.
- Contiene propiedades get y set.
- Implementa algunas piezas de lógica.

Como punto a favor este patrón se desprende lo siguiente:

- Se comporta bien para aplicaciones donde la lógica no es demasiada compleja.
- Es simple de entender para un proyecto pequeño a lo que base de datos se refiere.

Anexo B: Instalación y configuración de la aplicación en Apache

14.5 Instalación Modulo Apache

- Descarga de paquetes necesarios

```
apt- get install mono-runtime
apt-get install libapache2-mod-mono
apt-get install mono-apache-server2
apt-get install libmono-i18n2.0-cil
```

- Activar el módulo de mono

```
a2enmod mod_mono
```

- Agregar el siguiente segmento de código al archivo de configuración ubicado en “/etc/apache2/site-available/default”.

```
Alias /devsite /var/www/ l10nCommunityWebApplication
MonoApplications xsp2 “/devsite:/var/www/l10nCommunityWebApplication”
MonoServerPath xsp2 “/usr/bin/mod-mono-server2″
<Directory /var/www/l10nCommunityWebApplication>
   Allow from all
   Order allow,deny
   SetHandler mono
    MonoSetServerAlias xsp2
</Directory>
```

- Copiar la herramienta de traducción a la carpeta /var/www
- Probar el sistema accediendo a http://localhost/devsite/default.aspx

14.6 Editar el archivo de configuración “web.config”

El archivo de configuración contendrá las siguientes claves necesarias para la correcta ejecución de la aplicación:

- TemporalDirectory. Indica el directorio temporal usado para subir imágenes y archivos POT y PO.
- BaseDirectory. Indica el directorio base usado para los archivos de localización.
- UrlSite. Indica la dirección del sitio.
- MoFilesDirectory. Indica la ruta donde se encuentran los archivos MO generados.
- TrashDirectory. Indica la ruta del directorio usado para dejar los archivos y directorios eliminados.
- ImagesDirectory. Necesario para indicar al sistema donde debe dejar las imágenes subidas.
- LocalMySqlServer. String de conexión con el motor de base de datos MySql.

A continuación una imagen del archivo de configuración “web.config”.

```
  <appSettings>
    <add key="TemporalDirectory" value="/var/www/l10nCommunityWebApplication/App_Data/" />
    <add key="BaseDirectory" value="/var/www/l10nCommunityWebApplication/App_Data/BaseDirectory/" />
    <add key="UrlSite" value="http://localhost:8080/Mofiles/" />
    <add key="MoFilesDirectory" value="/var/www/l10nCommunityWebApplication/Mofiles/" />
    <add key="TrashDirectory" value="/var/www/l10nCommunityWebApplication/App_Data/TrashDirectory/" />
    <add key="ImagesDirectory" value="/var/www/l10nCommunityWebApplication/Images/" />

  </appSettings>
  <connectionStrings>
      <remove name="LocalMySqlServer"/>
      <add name="LocalMySqlServer" connectionString="Server=localhost;Database=l10nCommunity;User
ID=root;Password=holamundo;Pooling=false"/>

  </connectionStrings>
```

14.7 Crear la base de datos

Cargar los scripts de creación de base de datos y procedimientos almacenados de la aplicación.

Para la creación de los objetos asociados a la base de datos (tablas, procedimientos almacenados), se dispondrá de un script bash ubicado en la carpeta “/var/www/l10nCommunityWebApplication/MySqlDatabase/Scripts”. Dos alcances para la ejecución de este script:

- Se deberá editar el archivo para agregar el nombre y password del usuario autorizado para creación de base de datos.
- El script contiene una variable host, la cual indica el servidor de base de datos destino. Este valor no debe ser cambiado.

Anexo C: Uso de Gettext utilizando C#

Para usar la librería Gettext#, se necesita agregar el espacio de nombre Gnu.Unix y entonces llamar al método estático “Init” de la clase “Catalog”, la cual buscara el correspondiente catálogo con los mensajes traducidos de acuerdo a la variable de lenguaje seteada en el entorno Linux. Por otra parte el método “GetString”, permitirá reemplazar las cadenas de texto traducible de acuerdo a los valores provenientes desde los archivos de localización PO.

```
using System;
using Mono.Unix;

public class Example
{
	public static void Main (string[] args)
	{
		Catalog.Init("i18n","./locale");
		System.Console.WriteLine (Catalog.GetString ("Hello world!"));
	}
}
```

En el código anterior hay una sola cadena marcada como traducible: “Hello world!”. Por otra parte usando el método “GetString” se marcará cualquier cadena de caracteres como traducible.

Luego el código será compilado, y se extraerán las cadenas traducibles utilizando el comando “xgettext”.

```
$ mcs -r GNU.Gettext Example.cs
```

Una vez extraídas las cadenas, estas serán guardadas en un archivo de plantilla llamado “example.pot”.

```
$ xgettext --from-code=UTF-8 Example.cs -o example.pot
```

Luego para agregar la traducción al idioma español se invocará el comando “msginit” de la librería Gntu-Gettext.

```
$ msginit --locale=es --input=example.pot
```

Al abrir el archivo generado por el comando “msginit”, se encontrará la cadena traducible proveniente desde el código fuente de ejemplo.

```
#: Example.cs:11
msgid "Hello world!"
msgstr  ""
```

Viendo el código anterior se notara tres secciones:

- Número de línea y código fuente.

- Cadena traducible: Cadena de caracteres extraída desde el código fuente.
- Traducción del string: vacío por defecto, representa la nueva cadena que será mostrada cuando la aplicación se este ejecutando.

Después que el archivo haya sido traducido, se verá como sigue:

```
# 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: 2004-08-13 04:08-0500\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"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: Example.cs:11
msgid "Hello world!"
msgstr "¡Hola mundo!"
```

Luego para que esta traducción al español pueda ser utilizada se ejecutará los siguientes comandos: el primero crear un directorio hijo dentro del directorio “bin” de la aplicación y el segundo ejecutará el comando “msgfmt” de la librería Gnu-Gettext, el cual transforma el archivo PO a un archivo binario MO.

```
$ mkdir -p locale/es/LC_MESSAGES/
$ msgfmt es.po -o locale/es/LC_MESSAGES/i18n.mo
```

Finalmente, para que el programa de ejemplo pueda mostrar las cadenas traducibles al español, se ejecutará el siguiente comando que asignará la variable de entorno “LANGUAGE” a español.

```
$ LANGUAGE=es mono Example.exe
```

La salida del programa a continuación.

```
¡Hola mundo!"
```