UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE UN SISTEMA DE TRADUCCIÓN AUTOMÁTICA

Miguel Gragera Ariño
Tutor: Pablo Alfonso Haya Coll

Junio 2017

IMPLEMENTACIÓN DE UN SISTEMA DE TRADUCCIÓN AUTOMÁTICA

AUTOR: Miguel Gragera Ariño
TUTOR: Pablo Alfonso Haya Coll

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen

Este Trabajo Fin de Grado consiste en el desarrollo de una aplicación web educativa que facilite a los alumnos de traducción e interpretación de la UAM la subida de memorias de traducción para posteriormente traducir textos. La aplicación usa el sistema Moses SMT (Traducción automática estadística).

Moses es un sistema estadístico de traducción que permite automáticamente entrenar modelos de traducción para cualquier par de lenguajes. Todo lo que necesita para funcionar correctamente es un corpus paralelo.

La aplicación web está centrada en los Proyectos. Una vez seleccionado el Proyecto, el usuario podrá traducir, subir corpus paralelos o añadir glosarios para usarlos en la traducción. La idea de la aplicación es que los usuarios vayan alimentando los modelos de traducción de los Proyectos hasta alcanzar una traducción que se adapte a sus necesidades. Para alimentar el modelo previamente el administrador tiene que crear un Proyecto, una vez creado los usuarios podrán subir los corpus que posteriormente serán procesados para hacer uso de ellos en una traducción.

Con esta herramienta se trata de facilitar la evaluación de sus traducciones, subiendo sus corpus. La aplicación actualmente consta de ocho lenguas: español, inglés, francés, italiano, portugués, alemán, japonés y chino.

Para el desarrollo se ha utilizado una Máquina Virtual con Ubuntu 16.04 dado que la instalación del sistema de traducción de Moses esta facilitada en Ubuntu. Dentro de la propia máquina se ha desarrollado la aplicación en .NET Core con un diseño gráfico basado en Bootstrap 3 y un motor de base de datos SQLite3.

Palabras clave

Traducción automática estadística: sistema de traducción automática

corpus paralelo: Moses, tecnología educativa

Abstract

This Bachelor Thesis consists of the development of a web application that facilitates the students of translation and interpretation of the UAM a way to upload translation memories to later translate texts. The application uses the Moses SMT system.

Moses is a statistical translation system that allows you to automatically train translation models for any couple of languages. All you need for its proper functioning is a parallel corpus.

The web application is focused on Projects. Once selected the Project, the user will be able to translate, upload parallel corpus or add glossaries to use in the translation. The idea of the application is that the users will feed the translation models of the Projects until a translation that suits their needs is reached. To feed the model previously the administrator has to create a Project, once created the users will be able to upload the corpus that will later be processed to make use of them in a translation.

With this we try to facilitate the possibility of uploading their corpus, helping them to evaluate the quality of the translations derived from them. The application currently consists of eight languages: Spanish, English, French, Italian, Portuguese, German, Japanese and Chinese.

For development, I have used a Virtual Machine with Ubuntu 16.04 because the installation of the translation system of Moses is facilitated in Ubuntu. Inside the machine I have developed the application in .NET Core with graphic design based on Bootstrap 3 design and a SQLite3 database engine.

Keywords

SMT: *Stadistical Machine Translation*

parallel corpus: Moses, educational technology

Agradecimientos

Quiero agradecer a mi familia por el apoyo recibido y por brindarme la oportunidad de estudiar esta carrera.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

En las clases de traducción e interpretación de la UAM se utiliza *Google Tranlator Toolkit* [2] (GTT) para realizar traducción automática. Siendo un sistema bastante potente que permite subir memorias de traducción, generar modelos de lenguaje y traducir textos, presenta algunas limitaciones desde el punto de vista docente. Esta aplicación ha sido diseñada para cubrir estas limitaciones.

Los alumnos de traducción e interpretación de la UAM actualmente utilizan GTT para poner a prueba la calidad de sus traducciones de textos. Al igual que el GTT este TFG es un traductor online que permite modificar las traducciones y colaborar en ellas.

En el GTT no es posible agrupar las traducciones bajo un título para reconocer el ámbito para el que se han desarrollado. Por ejemplo, para traducir textos de medicina se necesita un tipo de vocabulario muy distinto al que se usaría en un ámbito de noticias. GTT actualmente es un sistema gratuito para todos aquellos usuarios que tenga cuenta de Google, pero en el futuro puede cambiar. Esta aplicación pone al alcance de los alumnos de traducción e interpretación una herramienta gratuita para poner a prueba sus traducciones tal y como hacían con GTT.

Desde el punto de vista docente GTT no ofrece un papel de administrador para gestionar las traducciones y ver su progresión. Con esta aplicación se cubre esta laguna permitiendo a los profesores acceder como administradores para ver y evaluar el progreso de sus alumnos.

1.2 Objetivos

El objetivo de este trabajo es diseñar e implementar una aplicación web que permita a los usuarios traducir textos mediante modelos de traducción generados por ellos mismos. La aplicación está enfocada desde el punto de vista docente, para los alumnos de traducción e interpretación de la UAM.

La traducción automática no genera unas traducciones perfectas, pero a base de entrenar los modelos de lenguajes con memorias de traducción se puede alcanzar una traducción casi humana. Con el uso de Moses se quiere poner esta herramienta accesible de manera sencilla a través de la aplicación web.

Moses es un sistema estadístico de traducción automática que permite entrenar automáticamente modelos de traducción para cualquier par de idiomas. [3]

Al igual que el traductor de Google [1], cuenta con la posibilidad de traducir textos sin iniciar sesión en la aplicación. En esta página, sin inicio de sesión el usuario puede elegir el proyecto que desea y lanzarse a traducir textos introduciéndolos en las áreas de texto, exactamente como ocurre en el traductor de Google.

El procesamiento de las memorias de traducción es muy costoso. Google en el GTT genera los modelos de lenguaje en segundo plano. En esta aplicación ocurre lo mismo, dejando al usuario la aplicación totalmente operativa a la espera de que el procesamiento termine.

"Una traducción satisfactoria no siempre es posible, pero un buen traductor nunca se satisface con ella. Normalmente se puede mejorar" – Peter Newmark, Manual de Traducción

1.3 Solución propuesta

Se ha desarrollado un sistema de traducción automática accesible online. La aplicación está basada en un sistema de traducción automática denominado Moses el cual ha sido desarrollado con el apoyo del proyecto EuroMatrix y liberado como código libre. [3]

La idea general de la funcionalidad de la aplicación es la siguiente. Los traductores suben sus memorias a un Proyecto anteriormente creado por el administrador. Una vez subidas se genera un modelo de lenguaje acorde a las memorias. A continuación, los traductores que han generado las memorias podrán traducir textos usando el modelo de lenguaje generado. Con esta funcionalidad se permite al traductor ver el resultado de sus memorias y cómo se comporta ante distintos tipos de textos. Si no está conforme con las traducciones puede volver a subir las memorias con mejoras y así optimizarlas.

1.4 Limitaciones

La generación de modelos de lenguaje es un proceso muy costoso y lento. Un usuario no puede traducir textos mientras se está generando un nuevo modelo de lenguaje. Por este motivo los administradores de la aplicación son los únicos que pueden generar modelos de lenguaje. La idea es que los modelos se generen fuera del horario de uso de la aplicación.

Aunque el mismo corpus paralelo permite crear dos modelos de lenguaje es precioso, que en la versión actual se generen por separado. Esto significa que si se desea crear un modelo para español-inglés e inglés-español, es necesario volver a subir los corpus y generar un modelo distinto. Esto es debido al sistema de generación del modelo de lenguaje con Moses que se explica más adelante en la memoria.

1.5 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1:** Introducción: breve descripción del proyecto
- **Capítulo 2:** Estado del arte: situación actual del ámbito de la traducción automática.
- **Capítulo 3:** Diseño: herramientas de desarrollo, diseño de la aplicación, base de datos e UI.
- **Capítulo 4:** Desarrollo: metodología utilizada para implementar el proyecto e integración con distintas herramientas.
- **Capítulo 5:** Integración, pruebas y resultados: puesta a punto de la aplicación para desplegarse y pruebas para comprobar su buen funcionamiento.
- **Capítulo 6:** Conclusiones y trabajo futuro

2 Estado del arte

En esta sección se explica cómo está actualmente la traducción automática estadística y las herramientas disponibles en la actualidad.

2.1 Traducción automática estadística

La Traducción Automática Estadística es un paradigma de traducción automática donde las traducciones se generan a partir de modelos estadísticos cuyos parámetros se derivan del análisis de un corpus de dos pares de lenguas. [18]

La Traducción Automática Estadística como área de investigación comenzó a finales de los años 80 con el proyecto *Candide* [4] en IBM. El proyecto *Candide* se nutrió de textos paralelos en inglés y francés. Adopto una perspectiva para mejorar la traducción automática basada en intentar encontrar el estimador máximo a posteriori (MAP) de una frase en ingles partiendo de una en francés. Esta primera aproximación de IBM a la traducción automática mapea palabra con palabra y permitía la inserción y eliminado de las palabras. [15]

Más adelante se descubrió que la mejor forma de conseguir una traducción de mayor calidad era con la traducción a nivel de frases. En la actualidad los sistemas de traducción más eficientes usan esta metodología, como CMU, IBM, ISI y Google.

La traducción basada en frases funciona de la siguiente manera. La entrada es fragmentada en conjuntos de palabras (frases), cada frase es traducida al lenguaje destino con la sintaxis correcta.

El teorema de *Bayes* es usado para reformular la probabilidad de traducción para traducir una frase en un lenguaje **f** a inglés **e** de esta forma:

$$\text{argmax}_{\mathbf{e}}\, p(\mathbf{e}|\mathbf{f}) = \text{argmax}_{\mathbf{e}}\, p(\mathbf{f}|\mathbf{e})\, p(\mathbf{e})$$

Durante la decodificación, la frase de entrada **f** se fragmenta en *I* frases $\mathbf{f}_1^{\mathbf{I}}$. Se asumime una probabilidad uniforme distributiva sobre todos los fragmentos. Cada frase del lenguaje origen $\mathbf{f_i}$ en $\mathbf{f}_1^{\mathbf{I}}$ es traducida a una frase en el idioma destino $\mathbf{e_i}$**.** Las frases en el lenguaje destino pueden ser reordenadas. La traducción basada en frases es modelada por una probabilidad distributiva $\boldsymbol{\varphi(f_i|e_i)}$**.**

El reordenamiento de las frases de salida del lenguaje destino esta modelado por una probabilidad de deformación $\mathbf{d(comienzo_i, fin_{i-1})}$, donde $\mathbf{comienzo_i}$ es la posición inicial de la palabra del lenguaje origen que se traduce a la *i* frase del inglés, y $\mathbf{fin_{i-1}}$ es la posición final de la frase del lenguaje origen que se traduce a la frase *i-1*.

Para calibrar el tamaño de la salida, se introduce un factor **ω** (coste de la palabra) para cada palabra generada en inglés (lenguaje destino). La fórmula queda de la siguiente forma:

$$\mathbf{e}_{best} = \text{argmax}_{\mathbf{e}}\, p(\mathbf{e}|\mathbf{f}) = \text{argmax}_{\mathbf{e}}\, p(\mathbf{f}|\mathbf{e})\, p(\mathbf{e})\, \boldsymbol{\omega}^{\text{length}(\mathbf{e})}$$

donde p(**e**|**f**) es:

$$p(f_1^I|e_1^I) = \Phi_{i=1}^I\, \varphi(f_i|e_i)\, d(\text{comienzo}_i, \text{fin}_{i-1})$$

2.2 Google Translator Toolkit

Como se menciona anteriormente en la Introducción, esta aplicación está basada en el *Google Translator Toolkit* ofreciendo una funcionalidad similar pero orientada al Proyecto.

Google Translator Toolkit es una aplicación pensada para permitir a los traductores subir sus traducciones y corregirlas posteriormente. Es un servicio gratuito para todos aquellos usuarios que tengan cuenta de Google, simplemente hay que acceder desde esta página web: https://translate.google.com/toolkit/ .

Fue lanzado en 2009, en sus orígenes contaba únicamente con inglés como lenguaje origen y 47 lenguajes destino. Actualmente admite más de 100.000 pares de lenguajes y cuenta con 345 lenguajes.

El flujo de GTT se puede describir de la siguiente manera. Primero, los usuarios suben un archivo que quieren traducir. GTT pre-traduce el documento dividiéndolo en segmentos, frases comunes y encabezados. A continuación, busca en la base de datos traducciones humanas que se adapten a los segmentos. Si encuentra alguna traducción humana, escoge la que tenga el resultado de búsqueda más alto y traduce ese segmento. Si no encuentra ninguna traducción que se adapte traduce el segmento basándose en una traducción automática. A posteriori, una vez obtenida la traducción los traductores pueden corregir las frases necesarias ayudando a GTT a optimizar el modelo de lenguaje.

Además de admitir memorias de traducción, GTT también acepta la subida de glosarios para mejorar las traducciones. Es posible subir e utilizar glosarios para traducir ciertas palabras por otras siempre que se quiera.

2.3 Moses

Moses es un sistema estadístico de traducción automática que permite entrenar automáticamente modelos de traducción para cualquier par de idiomas. Es un sistema de código abierto bajo licencia LGPL y está apoyado por la unión europea bajo los siguientes proyectos:

- EuroMatrix and TC-STAR (Framework 6)
- EuroMatrixPlus, LetsMT, META-NET, MosesCore and MateCat (Framework 7)

Además, ha recibido apoyo adicional de University of Edinburgh, Charles University, Fondazione Bruno Kessler, RWTH Aachen, University of Maryland, Massachusetts Institute of Technology y DARPA.

Moses ofrece dos tipos de sistemas de traducción, a base de scripts que se explican en el apartado siguiente o con el EMS (Experiment Management System). La primera opción ofrece mayor control sobre el proceso de generación del modelo de lenguaje. EMS funciona como una caja negra, esto podía suponer un riesgo al generar modelos de lenguaje muy costosos.

Para su correcto funcionamiento, Moses necesita un corpus paralelo con frases lo suficientemente variadas como para sacar un análisis sintáctico correcto, pero no muy extensas, esto es, menores de cien palabras.

Los corpus paralelos son colecciones de frases en dos idiomas distintos, que están alineadas. El alineamiento de las frases consiste en que cada frase del corpus coincida con su correspondiente en el idioma elegido. Los corpus deben prepararse antes de ser utilizados en el entrenamiento.

Moses funciona sobre Linux y es posible hacerlo funcionar también en *Cygwin*. Este proyecto se ha implementado sobre una máquina virtual con Linux Ubuntu 16.04.

2.4 Apertium

Apertium es un sistema de traducción automática que ha sido desarrollado en conjunto del gobierno de España, Generalidad de Cataluña en la Universidad de Alicante. [17]

Es un software libre con licencia GNU GPL diseñado para traducir entre lenguas relacionadas. Se puede probar desde esta página web: https://www.apertium.org .Originalmente fue creado para traducir textos de español a catalán, pero en la actualidad proporciona 43 pares de idiomas estables.

Proporciona una herramienta para traductores que quieran construir su propio par de idiomas para contribuir en el proyecto. Para ello tienen que subir tres diccionarios:

- Diccionario morfológico para la primera lengua que contenga las reglas de cómo se flexionan las palabras de ese lenguaje
- Diccionario morfológico para la segunda lengua igual que el anterior
- Diccionario bilingüe que contenga las correspondencias entre palabras de las dos lenguas.

3 Diseño

3.1 Moses

Matizo que el funcionamiento de Moses es unidireccional. Todo el proceso de entrenamiento que explico a continuación únicamente sirve para traducir lenguajes en una sola dirección, por ejemplo, español-inglés pero no inglés-español. Por este motivo en la aplicación al subir unas memorias para español como lengua origen e inglés como destino, solo permite las traducciones en esa única dirección. Si se desea que la traducción sea bidireccional habría que subir las memorias de traducción en el orden inverso y volver a generar el modelo de lenguaje.

3.1.1 Preparación del corpus

Para entrenar un sistema de traducción son necesarios datos paralelos alineados a nivel de frases. Una vez se tienen los corpus paralelos son necesarios los siguientes pasos para crear un modelo de datos.

- ***Tokenisation***: consiste en añadir espacios entre palabras y signos de puntuación.

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l en \
    < ~/corpus/training/news-commentary-v8.fr-en.en    \
    > ~/corpus/news-commentary-v8.fr-en.tok.en
```

- ***Truecasting***: las palabras iniciales de cada frase se asignan a su contenido más probable. Esto ayuda a reducir la escasez de datos.

```
~/mosesdecoder/scripts/recaser/train-truecaser.perl \
     --model ~/corpus/truecase-model.en --corpus     \
     ~/corpus/news-commentary-v8.fr-en.tok.en
~/mosesdecoder/scripts/recaser/truecase.perl \
   --model ~/corpus/truecase-model.en         \
   < ~/corpus/news-commentary-v8.fr-en.tok.en \
   > ~/corpus/news-commentary-v8.fr-en.true.en
```

- ***Cleaning***: frases largas, mal alineadas o mal formadas se eliminan, por ejemplo, 80 caracteres como longitud máxima de cada frase.

```
~/mosesdecoder/scripts/training/clean-corpus-n.perl \
    ~/corpus/news-commentary-v8.fr-en.true fr en \
    ~/corpus/news-commentary-v8.fr-en.clean 1 80
```

3.1.2 Creación del modelo de datos

El modelo de datos se usa para asegurar una salida fluida y correcta, se construye a raíz del idioma objetivo de la traducción

```
~/mosesdecoder/bin/lmplz -o 3 <~/corpus/news-commentary-v8.fr-en.true.en
> news-commentary-v8.fr-en.arpa.en
```

Usando la herramienta KenLm se convierte el fichero resultante a binario para una carga de datos más rápida.

```
~/mosesdecoder/bin/build_binary \
   news-commentary-v8.fr-en.arpa.en \
   news-commentary-v8.fr-en.blm.en
```

3.1.3 Entrenamiento del sistema de traducción

Una vez se tiene el modelo de datos generado toca entrenar el sistema de traducción. Para ello se ejecuta la alineación de palabras (GIZA++), extracción de frases y puntuación, creación de tablas de reordenación y el fichero de configuración de Moses.

```
nohup nice ~/mosesdecoder/scripts/training/train-model.perl -root-dir
train \
 -corpus ~/corpus/news-commentary-v8.fr-en.clean
\
 -f fr -e en -alignment grow-diag-final-and -reordering msd-
bidirectional-fe \
 -lm 0:3:$HOME/lm/news-commentary-v8.fr-en.blm.en:8
\
 -external-bin-dir ~/mosesdecoder/tools >& training.out &
```

Debido al elevado coste de la operación se ejecuta en un proceso nuevo. Esta es la parte más costosa del proceso. El tuneado necesita un nuevo corpus distinto a los anteriores con su respectivo corpus paralelo en el idioma objetivo. Esta fase del proceso se encarga de mejorar la tasa de error mínimo generada por el entrenamiento, básicamente reajusta los pesos de las palabras teniendo en cuenta el nuevo corpus introducido.

```
nohup nice ~/mosesdecoder/scripts/training/mert-moses.pl \
  ~/corpus/news-test2008.true.fr ~/corpus/news-test2008.true.en \
  ~/mosesdecoder/bin/moses train/model/moses.ini --mertdir
~/mosesdecoder/bin/ \
  &> mert.out &
```

Como ocurre con el entrenamiento del sistema de traducción, se ejecuta este comando en proceso nuevo.

3.1.4 Traducción

Finalmente, la tabla de pesos se ha formado correctamente. Para traducir un texto se usa el siguiente comando:

```
~/mosesdecoder/bin/moses -f ~/working/mert-work/moses.ini
```

Para optimizar la búsqueda de las palabras se puede convertir la tabla a binario. La conversión de la tabla de frases es necesaria para disponer de una traducción rápida en un modelo de lenguaje de cierto tamaño. Actualmente el proyecto cuenta con tablas de frases de 3gb y una traducción a estos modelos tardaba unos 5 min. Tras la conversión el tiempo de traducción se ha visto reducido a un intervalo entre 1-3 segundos. El proceso es el siguiente:

```
mkdir ~/working/binarised-model
 cd ~/working
 ~/mosesdecoder/bin/processPhraseTableMin \
   -in train/model/phrase-table.gz -nscores 4 \
   -out binarised-model/phrase-table
 ~/mosesdecoder/bin/processLexicalTableMin \
   -in train/model/reordering-table.wbe-msd-bidirectional-fe.gz \
   -out binarised-model/reordering-table
```

Una vez creadas las nuevas tablas se tiene que cambiar el fichero de configuración *moses.ini* para que apunte a las nuevas tablas. Primero se cambia el nombre *PhraseDictionaryMemory* a *PhraseDictionaryCompact*, luego su ruta (*path*) para que apunte a las nuevas tablas, además hay que cambiar la ruta de *LexicalReordering* también para que apunte a la nueva tabla correspondiente.

3.2 Aplicación Web

Actualmente las aplicaciones web son muy populares, ofrecen muchas funcionalidades al cliente accediendo únicamente a un servidor web. Funcionan con independencia del sistema operativo del equipo desde el que se accede o desde el navegador desde que se conecta.

Las aplicaciones web utilizan una combinación de scripts del lado del servidor (ASP) para manejar el almacenamiento y recuperan la información que se muestra a los usuarios a través de los scripts del cliente (JavaScript y HTML). Esto permite a los usuarios interactuar con la aplicación con total libertad.

Suelen estar codificadas en un lenguaje compatible con el navegador, como HTML o JavaScript. La aplicación web requiere un servicio web para gestionar las solicitudes del cliente, un servidor de aplicaciones para realizar las tareas solicitadas por el cliente. En este caso la tecnología utilizada es ASP.NET.

3.2.1 Funcionamiento

Figura 3-1 Esquema aplicación web [7]

- El usuario realiza una solicitud al servidor web a través de Internet mediante un navegador web.
- El servidor web reenvía la solicitud al servidor de aplicaciones.
- El servidor de aplicaciones lleva a cabo la tarea solicitada, por ejemplo, consultar en base de datos.
- El servidor de aplicaciones interpreta los datos y envía los resultados de vuelta al servidor web.
- El servidor web responde al cliente con la información solicitada.

3.3 Herramientas de desarrollo

3.3.1 Patrón de arquitectura: Modelo Vista Controlador

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de desarrollo de software que separa los datos, la lógica de negocio y la interfaz de usuario de una aplicación. La aplicación se separa en tres componentes principales: el modelo, la vista y el controlador.

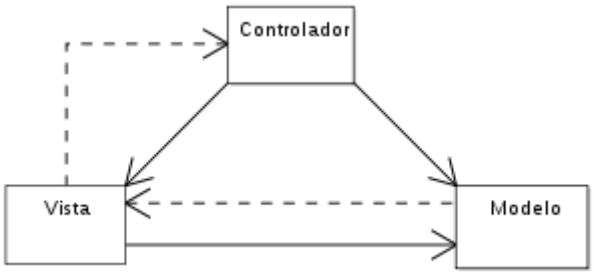

Figura 3-2 Esquema MVC [8]

En el modelo se trabajan los datos. Contiene funciones para acceder a la información y si es oportuno modificarla.

Las vistas contienen el código que crea la interfaz gráfica que se va a visualizar. En las vistas se muestran los datos que nos facilita el modelo, además es el encargado de enviar las peticiones del usuario al controlador.

Los controladores son los encargados de responder a las acciones que se solicitan en la aplicación. Sirven de enlace entre las vistas y los modelos para nutrir de información la interfaz gráfica de la aplicación.

La fragmentación de los niveles del patrón MVC permite un desarrollo más rápido y eficiente de las aplicaciones, mientras un desarrollador hace cambios en la capa de negocio el otro puede modificar la vista. Otra ventaja es que puede haber varias vistas de un único modelo, esto evita la redundancia de código y facilita la creación de nuevas vistas.

3.3.2 Framework: ASP.NET Core

ASP.NET Core es un *framework* de código abierto desarrollado por Microsoft compatible con Linux, Microsoft y Mac. Con esta versión de .NET se ha reducido la carga de componentes y librerías de los proyectos. Ofrece una infraestructura ligera, modular y multiplataforma. [9]

Una gran ventaja de .NET Core respecto a los demás *frameworks* .NET es que puede desplegarse de manera "*side-by-side*" permitiendo que las aplicaciones integradas tengan distintas versiones del CLR. CLR es un entorno de ejecución para los códigos que corren sobre .NET. [16]

Otra ventaja es que las dependencias se pueden integrar de forma independiente. Por ejemplo, *System.Collections* ya no depende de *System.XML*. Con esta mejora los proyectos son más ligeros y funcionan de manera más eficiente.

3.3.3 Entorno de desarrollo: Visual Studio Code

Visual Studio Code es un editor de código desarrollado por Microsoft. A diferencia de los Visual Studio IDE (*Integrated Development Environment*), este está disponible para Microsoft, Mac OS y Linux. Ha sido elegido por su compatibilidad con Linux.

Respecto a los demás Visual Studio este no tiene la mayoría de utilidades a través de la interfaz de usuario, pero las ofrece modificando los ficheros JSON del proyecto. Esto es algo más costoso para el desarrollador, pero permite que el programa sea mucho más ligero.

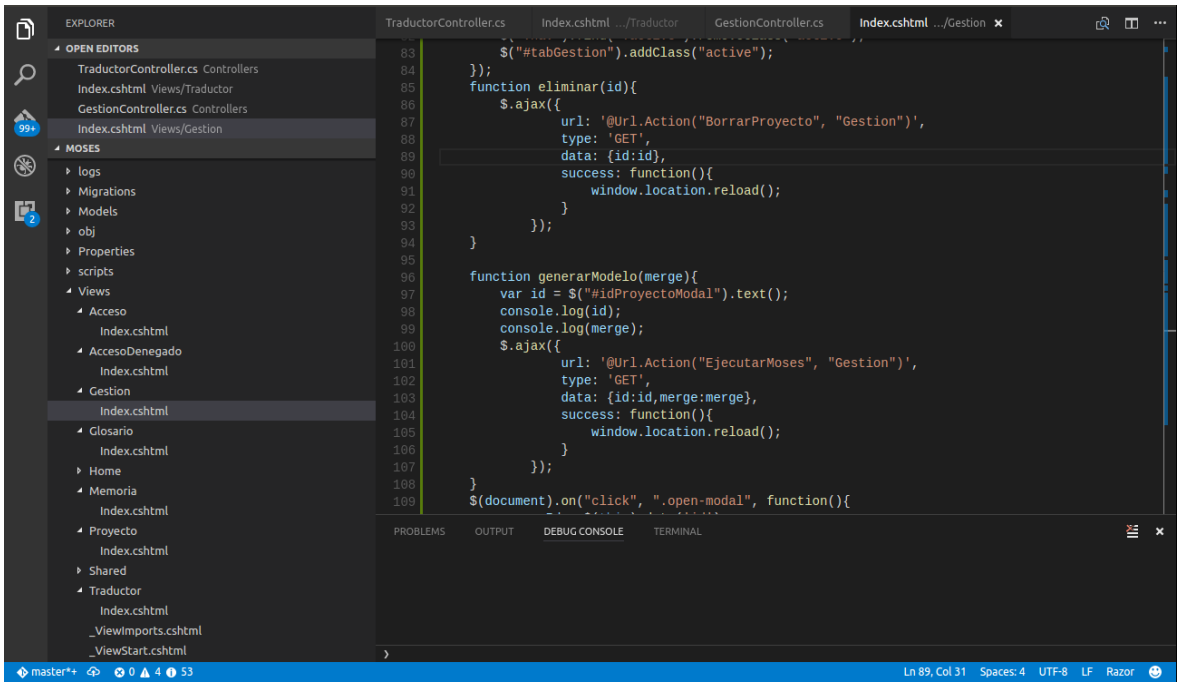

Figura 3-3 Captura VS Code

3.3.4 Servidor: Nginx

Nginx es un servidor web/proxy inverso de código abierto compatible con Linux. En este caso se utiliza para redirigir el tráfico del puerto en el que se despliega la aplicación a uno visible desde el exterior (Puerto 500 a Puerto 80).

3.3.5 Motor de base de datos: SQLite3

SQLite3 es un motor de base de datos SQL. Es óptimo para aplicaciones web con poca carga de base de datos y es compatible con multitud de sistemas operativos como por ejemplo Linux.

3.3.6 Sistema Operativo: Ubuntu 16.04 LTS

Ubuntu 16.04 es el sistema operativo elegido porque el sistema de traducción automática Moses está desarrollado para este sistema operativo. Ubuntu 16.04 cuenta con multitud de paquetes necesarios para la ejecución de los scripts de Moses, cuenta con: *g++, subversión, make, libtool, gcc, g++, libboost-dev, tcl-dev, tk-dev, zlib1g-dev, libbz2-dev y Python-dev.* Además, tiene compatibilidad con todas las herramientas de desarrollo nombradas. [10] [11]

3.3.7 Diseño gráfico: Bootstrap 3

Bootstrap es un conjunto de herramientas de código abierto para el diseño de aplicaciones web. Está formado por botones, tablas, botones… diseñados en HTML y CSS. También cuenta con librerías de JavaScript que permiten utilizar opciones adicionales. Ofrece un diseño responsivo que se ajusta a cualquier dispositivo y es compatible con todos los navegadores convencionales. Actualmente es muy popular y páginas web como Microsoft cuentan con ello.

La página web se adapta perfectamente a cualquier tipo de resolución. Para dispositivos móviles se agrupan las opciones de la barra de navegación en un solo botón

Figura 3-4 Captura de móvil

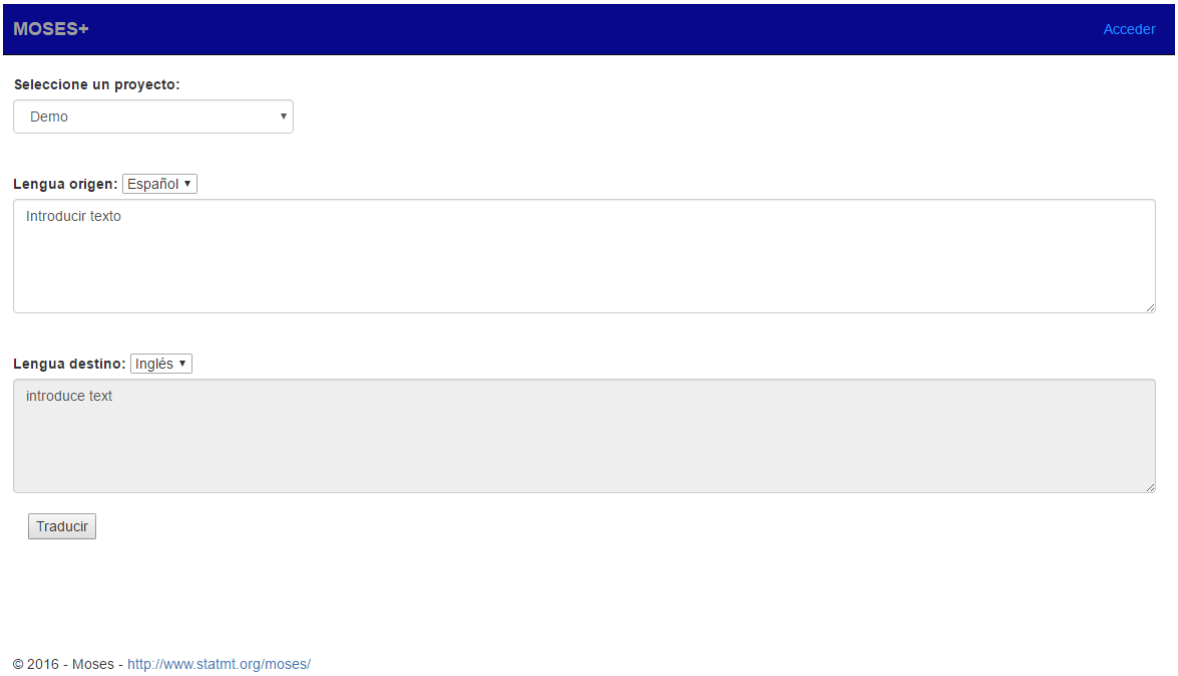

Figura 3-5 Captura de ordenador

3.4 Diagrama de casos de uso

A continuación, se muestra un diagrama de casos de uso que resume el comportamiento de la aplicación. Están definidos los tres roles de usuario y sus casos de uso correspondientes.

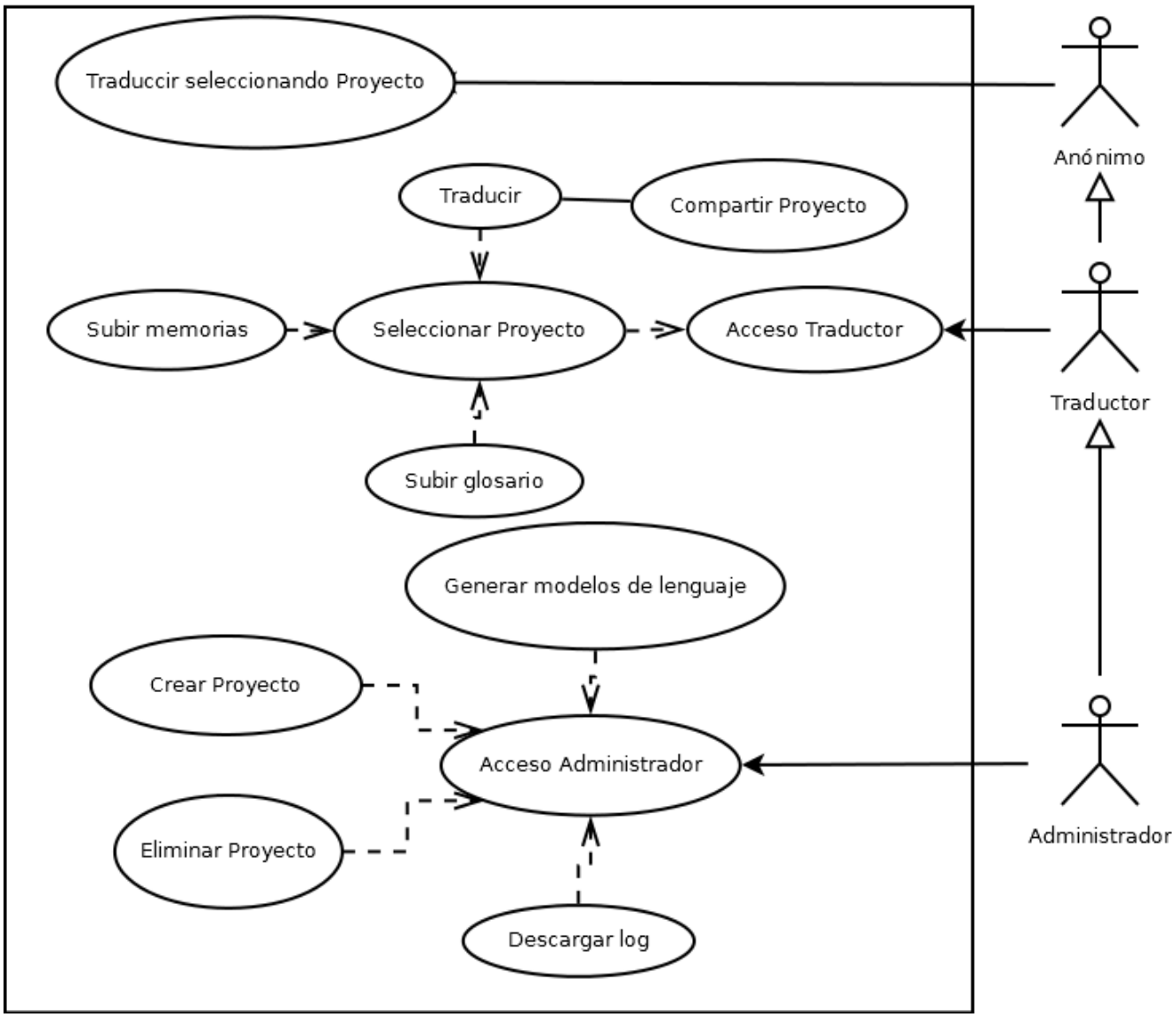

Figura 3-6 Diagrama de casos de uso

Existen 3 roles de usuario en la aplicación:

- **Usuario anónimo**: no inicia sesión en la aplicación.
- **Traductor:** accede como traductor a la aplicación y hereda todas las funcionalidades del usuario anónimo, además puede colaborar en la mejora de los modelos de traducción.
- **Administrador**: accede a la aplicación como administrador y hereda las funcionalidades del usuario anónimo y del traductor. Puede gestionar los proyectos.

Funcionalidades:

- **Traducir seleccionando Proyecto:** traduce un texto insertado según el proyecto seleccionado en la misma página, el lenguaje origen y el lenguaje destino. Devuelve el resultado de la traducción en la misma página. Además, permite utilizar un glosario previamente subido a la aplicación.
- **Seleccionar Proyecto**: el usuario debe seleccionar un proyecto para luego subir memorias de traducción, glosarios o simplemente traducir.
- **Traducir:** una vez haya seleccionado el proyecto podrá traducir como se ha mencionado anteriormente. La gran diferencia respecto a la traducción sin inicio de sesión es la posibilidad de utilizar los glosarios.
- **Compartir Proyectos:** desde la pantalla de traducción es posible copiar la dirección URL y compartirla. Esta dirección contiene un parámetro llamado idProyecto que habilita únicamente el Proyecto seleccionado. Por ejemplo, se genera una dirección así http://maria.lllf.uam.es/?idProyecto=1 . Es posible acceder sin iniciar sesión si se dispone de la dirección y al igual que la anterior traducción se puede aplicar el glosario.
- **Subir memorias**: las memorias de traducción (anexo C) que el usuario suba al proyecto se guardaran en una carpeta con el nombre del proyecto y una subcarpeta con los códigos de los lenguajes seleccionados. Estas memorias se usarán al generar los modelos de traducciones.
- **Subir glosarios**: los glosarios subidos al proyecto serán utilizados en la traducción. El formato del glosario debe ser un .csv separado por comas. Una vez subido se almacena en base de datos en su respectiva tabla. Para usar el glosario el usuario debe marcar la casilla de "Usar glosario" al traducir un texto.
- **Crear Proyecto:** el administrador puede crear proyectos para que los traductores suban sus memorias y glosarios.
- **Eliminar Proyecto**: el administrador además de crear, puede eliminar los proyectos que desea, eliminando así todas sus relaciones en base de datos y carpetas de memorias de traducción.
- **Generar modelos de lenguaje:** una vez nutrido el proyecto con memorias de traducción el administrador podrá generar modelos de lenguaje. En el caso de que un proyecto esté actualizado no se podrá generar un modelo nuevo. Una vez seleccionado el proyecto el administrador podrá elegir si quiere generar el modelo con las últimas memorias de traducción subidas o con todas. Este proceso funciona ejecutando un script que genera un modelo de traducción por cada par de lenguajes. Los modelos que genera se corresponden a las memorias subidas.
- **Descargar log**: el administrador podrá descargar un log para ver las acciones de los usuarios en la aplicación.

3.5 Diagrama de secuencia

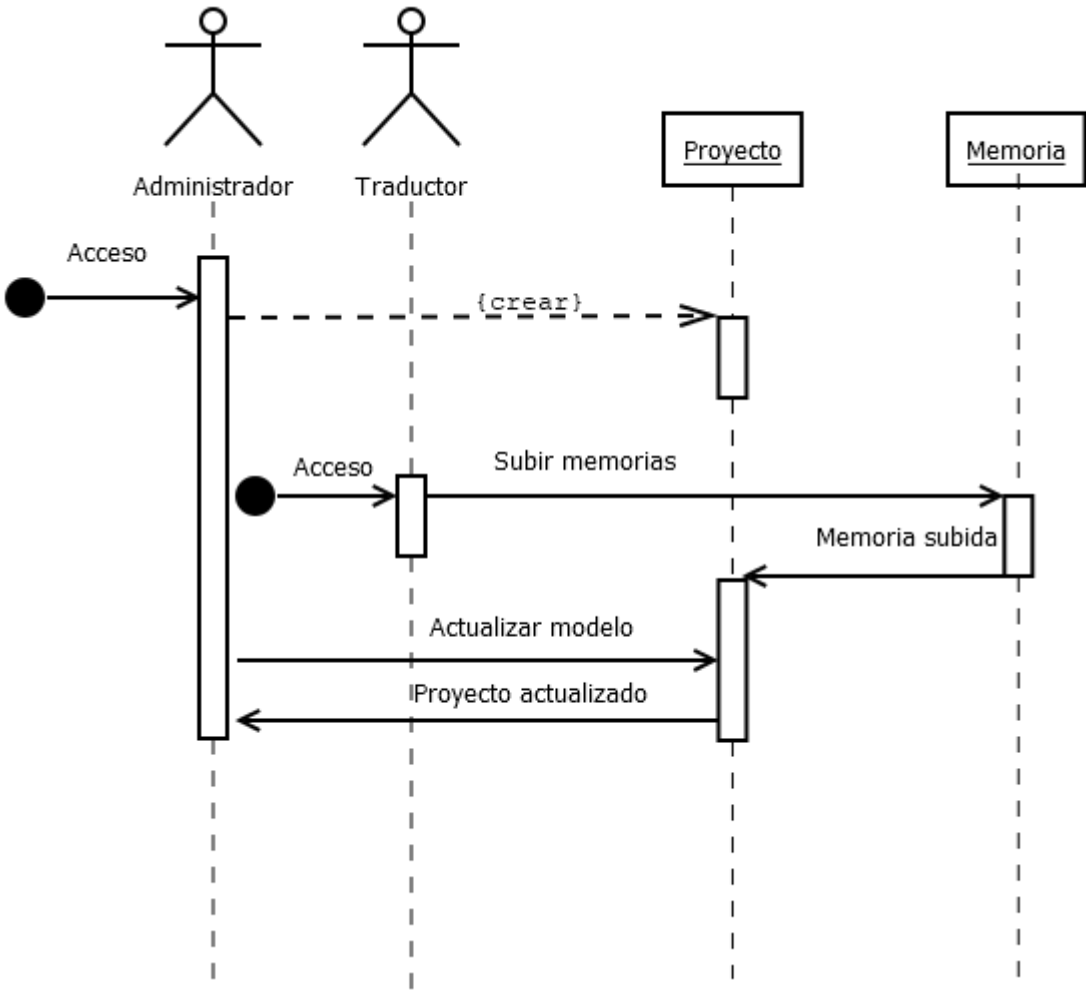

Figura 3-7 Diagrama de secuencia generación modelo de lenguaje

El diagrama de secuencia de la generación de un modelo de lenguaje para posteriores traducciones muestra los pasos necesarios para realizarse. Primero el Administrador tiene que crear un proyecto. Una vez creado estará disponible para que el Traductor o el propio Administrador suban un par de memorias de traducción seleccionando los lenguajes correspondientes. Cuando la memoria esta subida el Administrador podrá generar un modelo nuevo o actualizar el anterior usando memorias de traducción antiguas. Tras un largo proceso de generación del modelo, el proyecto aparecerá como actualizado y estará listo para usarse para traducir textos.

3.6 Diseño de base de datos

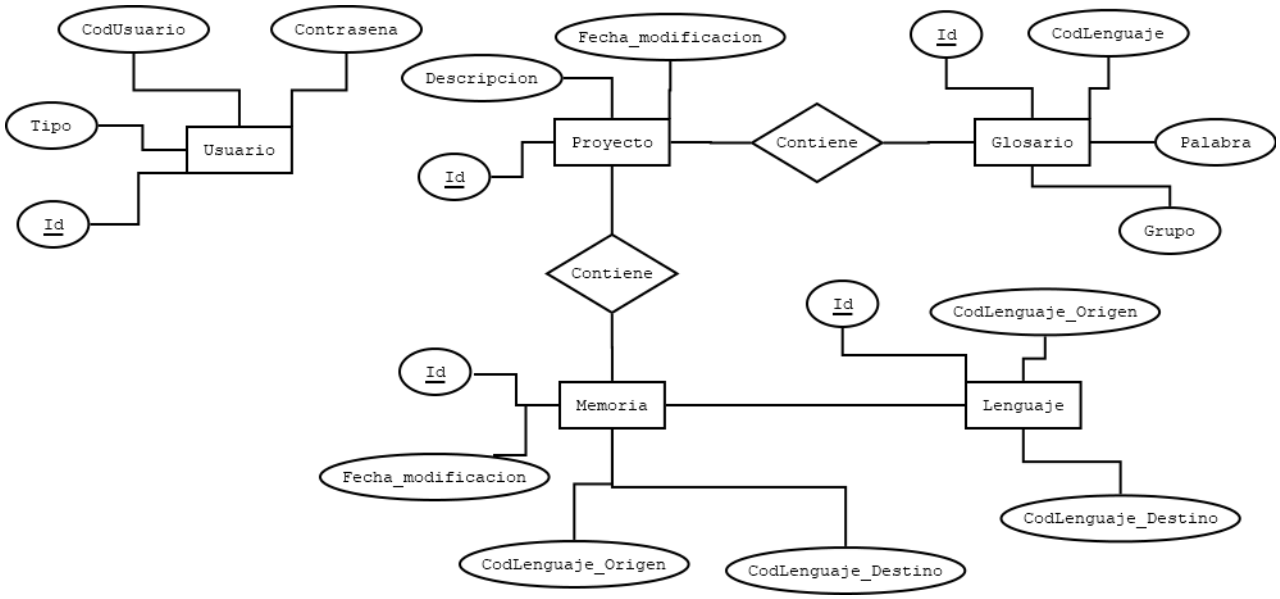

Figura 3-8 Diagrama Entidad Relación de la base de datos

Usuario

Campo	Tipo	Descripción
Id	INTEGER	Identificador de usuario autogenerado secuencial
Tipo	INTEGER	Indica el rol del usuario
CodUsuario	TEXT	Usuario para acceder a la aplicación
Contrasena	TEXT	Contraseña para acceder a la aplicación

Tabla 3-1 Tabla Usuario de la BD

Proyecto

Campo	Tipo	Descripción
Id	INTEGER	Identificador de proyecto autogenerado secuencial
Descripcion	TEXT	Nombre del proyecto
Fecha_modificacion	TEXT	Fecha de la última modificación del proyecto

Tabla 3-2 Tabla Proyecto de la BD

Memoria

Campo	Tipo	Descripción
Id	INTEGER	Identificador de memoria autogenerado secuencial
Fecha_modificacion	TEXT	Fecha de subida de las memorias
CodLenguaje_Origen	TEXT	Código del lenguaje origen de la memoria
CodLenguaje_Destino	TEXT	Código del lenguaje destino de la memoria

Tabla 3-3 Tabla Memoria de la BD

Glosario

Campo	Tipo	Descripción
Id	INTEGER	Identificador de proyecto autogenerado secuencial
CodLenguaje	TEXT	Código del lenguaje de la palabra
Palabra	TEXT	Palabra que se usará en el glosario
Grupo	TEXT	Indica el grupo de palabras similares

Tabla 3-4 Tabla Glosario de la BD

Proyecto_Memorias

Campo	Tipo	Descripción
ProyectoId	INTEGER	Relación con el Proyecto
MemoriaId	INTEGER	Relación con la Memoria

Tabla 3-5 Tabla Proyecto_Memorias de la BD

Proyecto_Glosarios

Campo	Tipo	Descripción
ProyectoId	INTEGER	Relación con el Proyecto
GlosarioId	INTEGER	Relación con la Glosario

Tabla 3-6 Tabla Proyecto_Glosarios de la BD

Lenguaje

Campo	Tipo	Descripción
Id	INTEGER	Identificador de lenguaje autogenerado secuencial
CodLenguaje_Origen	TEXT	Código del lenguaje origen del par de lenguajes de la memoria
CodLenguaje_Destino	TEXT	Código del lenguaje destino del par de lenguajes de la memoria

Tabla 3-7 Tabla Lenguaje de la BD

Como se puede apreciar en el Diagrama ER el esquema de la aplicación cuenta con 5 tablas principales y 2 relaciones. La tabla Usuario indica los usuarios que pueden acceder a la aplicación y dependiendo de su Tipo (rol) tendrán más o menos permisos. Los roles que se contemplan son Administrador (1) y Traductor (2).

Un Proyecto está formado por Memorias y por Glosarios. Las Memorias se utilizan para generar los modelos de lenguaje con los que se traducirán los textos, y los Glosarios se aplican sobre la traducción para traducir las palabras que se hayan insertado. A su vez las Memorias contienen una referencia a la tabla Lenguaje, que indica en que lenguajes está el par de memorias subido.

4 Desarrollo

4.1 Integración con Moses

La gestión de los Usuarios, Proyectos, Memorias, Glosarios y Lenguajes se lleva a cabo en la aplicación .NET, pero las laboriosas tareas que permiten generar modelos de lenguaje y traducir posteriormente se ejecutan mediante scripts.

Se han desarrollado varios scripts según la tarea que se deba ejecutar. Los scripts son bash corren sobre Ubuntu 16.04 para ejecutar los scripts instalados en la implementación de Moses.

4.1.1 Scripts

La aplicación cuenta con cuatro scripts para realizar las siguientes tareas:

a) Generación del modelo de lenguaje
b) Concatenación de las memorias subidas
c) Traducción de textos.

La generación del modelo es la tarea más costosa y por este motivo se realiza en segundo plano. Para su ejecución en segundo plano se usa otro script que tiene un único comando para realizar la tarea en segundo plano y volcar su salida en un fichero de texto.

Script ejecutarMoses.sh

```
./moses.sh $1 $2 $3 $4 $5 &> moses.out &
```

Moses.sh es el script que contiene todos los pasos para generar el modelo de lenguaje. Los pasos son los que se han comentado en el *apartado 3.1* de este documento con alguna modificación para mejorar el rendimiento. A continuación, se muestran las mejoras que se han introducido en el script para facilitar una traducción más rápida.

Script moses.sh

```
echo "Binarising model"
 cd ~/$var2/wwwroot/moses/working/$var1/
 mkdir binarised-model

~/mosesdecoder/bin/processPhraseTableMin -in
~/$var2/wwwroot/moses/working/$var1/train/model/phrase-table.gz -
nscores 4 -out binarised-model/phrase-table

~/mosesdecoder/bin/processLexicalTableMin -in
~/$var2/wwwroot/moses/working/$var1/train/model/reordering-table.wbe-
msd-bidirectional-fe.gz -out binarised-model/reordering-table

cp ~/$var2/wwwroot/moses/working/$var1/mert-work/moses.ini binarised-
model
cd binarised-model
```

```
sed -i 's/PhraseDictionaryMemory/PhraseDictionaryCompact/g' moses.ini
sed -i
"s,/home/operador/$var2/wwwroot/moses/working/${var1}/train/model/phra
se-
table.gz,/home/operador/$var2/wwwroot/moses/working/$var1/binarised-
model/phrase-table.minphr,g" moses.ini
sed -i
"s,/home/mgragera/$var2/wwwroot/moses/working/${var1}/train/model/reor
dering-table.wbe-msd-bidirectional-
fe.gz,/home/mgragera/$var2/wwwroot/moses/working/$var1/binarised-
model/reordering-table,g" moses.ini
```

Se convierten las tablas de traducciones a binario para encontrar más rápido las palabras equivalentes a nuestra traducción. Una vez convertidas se copia la configuración a la carpeta que contiene las nuevas tablas y se cambian los valores para que apunten a estas nuevas tablas. Con esto se reduce el tiempo muy notablemente. Si convertir, un texto de 10 palabras en el modelo de lenguaje de demostración tardaba 5 minutos, tras este cambio la misma traducción tarda escasos 3 segundos en obtener un resultado.

La llamada desde la aplicación se realiza mediante la función *Process.Start()* de la librería *System.Diagnostics*, a continuación, muestro un ejemplo de mi código en C#:

```
/*Primero creamos las carpetas para almacenar los ficheros generados
por Moses*/
CrearDirectorios(folderName1);

ProcessStartInfo startInfo = new ProcessStartInfo();
startInfo.FileName="ejecutarMoses.sh";
startInfo.UseShellExecute = false;
startInfo.RedirectStandardOutput = true;
startInfo.Arguments = file1 + " " + lang1 + " " + file2 + " " + lang2
+ " " + folderName1;
Process proc = Process.Start(startInfo);
string outputTerminal = proc.StandardOutput.ReadToEnd();
proc.WaitForExit();
return outputTerminal;
```

Se ha creado un script para unir las memorias que se suben para el mismo par de lenguajes dentro del mismo proyecto porque resulta mucho más eficiente y rápido que leer el texto en el controlador de la aplicación y concatenarlo ahí.

Script memorias.sh

```
cd $1
cat *.$2 > merged-file.$2
cat *.$3 > merged-file.$3
```

Finalmente se llega al script que ejecuta las traducciones, desde este script se le pasa el fichero de configuración *moses.ini* al script de Moses que se encarga de traducir.

Script traducción.sh

```
var1="MosesApp"

echo "${@:2}" | ~/mosesdecoder/bin/moses -f
~/$var1/wwwroot/moses/working/$1/binarised-model/moses.ini
```

4.1.2 Gestión de archivos

La generación de modelos de lenguajes necesita y crea gran cantidad de archivos. Para ello, basándome en la estructura de la aplicación en la que el centro son los Proyectos, genero una distribución de archivos que gira en torno a él.

Cuando se da de alta un Proyecto se crea automáticamente una carpeta llamada *uploads*, en la cual se subirán las memorias clasificadas por los pares de lenguajes. Al generar el modelo de lenguaje se crean otras tres subcarpetas para ir almacenando los archivos que genera la ejecución de los scripts de Moses. Estas carpetas son: *corpus*, *lm* y *working*

Dentro de cada una de las cuatro carpetas nombradas en este apartado se encuentra la misma distribución:

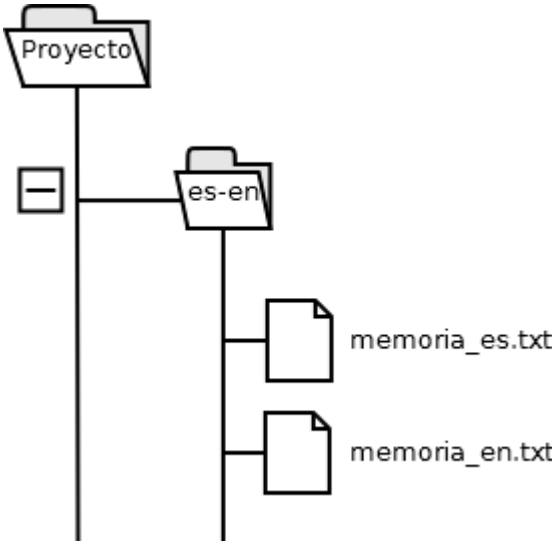

Figura 4-1 Diagrama de distribución de archivos

La figura 4.1 se corresponde exactamente a la distribución de la carpeta *uploads*. En las otras carpetas únicamente cambia el contenido de la carpeta del par de lenguajes, que en este ejemplo es español-inglés.

4.1.3 Aplicación web

Como se ha explicado anteriormente en la parte de diseño, se ha utilizado el patrón de arquitectura MVC. La aplicación está formada por seis clases principales, con su respectivo modelo, vista y controlador. Las clases principales son: Traductor, Acceso, Proyecto, Memoria, Glosario y Gestión.

La conexión entre las tres piezas de la arquitectura se realiza de distinta manera. Desde la Vista al Controlador se usan formularios ASP.NET o llamadas AJAX. AJAX no es un lenguaje de programación, es una combinación de peticiones *XMLHttpRequest* para solicitar información al servidor, y JavaScript y HTML para mostrar la información al cliente [5]. Permite que las páginas web se actualicen asíncronamente intercambiando información con el servidor sin que la página entera se recargue. A continuación, un ejemplo de una llamada con AJAX.

```
function bindlenguajesdestino(){
        console.log($("#langOrigen").val());
        $("#langDestino").empty();
            $.ajax({
                url: '@Url.Action("BindLenguajesDestino",
"Traductor")',
                type: 'GET',
                data: {idProyecto:$("#slctProyectos").val(),
codLenguaje:$("#langOrigen").val()},
                dataType: 'json',
                success: function(data){
                    console.log(data);
                    data.forEach(function(leng){
                        $("#langDestino").html($('<option>',{
                            value: leng.codigo,
                            text: leng.nombre
                            }));
                    })

                }
            });
    }
```

En el ejemplo se utiliza AJAX para obtener los lenguajes destino disponibles al seleccionar un Proyecto, una vez obtenidos en *success* se rellena el desplegable con los lenguajes devueltos. De esta forma cada vez que se hace un cambio de Proyecto los lenguajes disponibles cambian inmediatamente sin necesidad de refrescar la página.

Para la comunicación Modelo-Vista se utiliza Razor. Razor es una sintaxis que permite usar código del servidor (C#) en el cliente (HTML), para empezar a usarlo simplemente hay que empezar el código con un @. A continuación, un ejemplo del uso de Razor en una Vista.

```
@foreach (var proyecto in @Model.Proyectos){
      <tr id=@proyecto.Id>
      <td>@Html.DisplayFor(modelItem  => proyecto.Id)</td>
´…
}
```

En el ejemplo anterior se muestra cómo se accede al modelo con Razor y se hace uso de un bucle *foreach* propio de C# en la Vista. En este caso se obtienenen los Proyectos directamente del modelo para mostrarlos en la Vista.

La aplicación cuenta con una sesión que se genera al acceder a la aplicación (figura 0-2). Cuando se accede se genera una *HttpContext.Session* [13] que guarda los datos del usuario desde que se ha accedido. Como se menciona anteriormente, dependiendo del tipo de usuario se pueden acceder a unas determinadas páginas. Otro dato importante que se guarda en la sesión es el Proyecto, cuando se selecciona en la pantalla de menú de Proyectos se almacena en la sesión para utilizarlo como modelo de lenguaje de una traducción, subir memorias o glosarios. Para cambiar de Proyecto simplemente hay que seleccionar otro. La sesión en la aplicación dura 30 minutos, si no se hace uso de la misma durante ese tiempo se cierra sesión automáticamente. También existe la posibilidad de cerrar sesión de forma manual pulsando en el texto "Cerrar Sesión" en la parte superior derecha de la pantalla.

4.1.4 Integración con la base de datos: Entity Framework

La integración con la base de datos que se usa en la aplicación web corre a cargo de *Entity Framework.*

"Entity Framework es un conjunto de tecnologías de ADO.NET que permite el desarrollo de aplicaciones de software orientadas a datos" [6]. Permite a los desarrolladores trabajar a un nivel más alto de abstracción para manejar los datos. Más concretamente para este proyecto se ha utilizado *Entity Framework Core* que está integrado sobre *.NET Core*.

En primer lugar *Entity Framework* permite crear y modificar las bases de datos a través de migraciones *Code First* [14]. De esta manera se puede generar una base de datos a través de nuestras clases del Modelo. En *EF Core* para Linux las migraciones se crean con el siguiente comando:

```
dotnet ef add migration {nombreMigracion}
```

Cuando existen cambios en el modelo de datos y se desea añadir una nueva migración, la nueva se crea con los cambios que detecta ayudando a no tener que restaurar la base de datos entera. Esto significa que la última migración creada engloba todos sus precedentes. Para actualizar la base de datos con la última migración se utiliza este comando:

```
dotnet ef database update
```

A continuación, se muestra un ejemplo del código de una migración para crear la base de datos de este proyecto:

```
migrationBuilder.CreateTable(
    name: "Proyectos",
    columns: table => new
    {
       Id = table.Column<int>(nullable: false)
       .Annotation("Autoincrement", true),
        Actualizado = table.Column<bool>(nullable: false),
        Descripcion = table.Column<string>(nullable: true),
                    Fecha_modificacion =
table.Column<string>(nullable: true),
        },
        constraints: table =>
        {
           table.PrimaryKey("PK_Proyectos", x => x.Id);
        });
```

El ejemplo anterior muestra el código generado automáticamente para crear la tabla Proyecto mediante Code First.

4.1.5 Fichero de log: Log4net

Para el desarrollo de cualquier tipo de aplicación es recomendable guardar registro de operatividad de la aplicación. Dado que en un entorno de producción no es posible depurar el código, es necesario dejar trazas de todo lo que sucede en el sistema para poder detectar con facilidad los posibles errores y buscar posibles mejoras.

Log4net es una librería de .NET compatible con Core que permite monitorizar todo lo que se desea de la aplicación. Para su funcionamiento únicamente necesita estar referenciada en el proyecto y en un fichero de configuración *log4net.config* que indica el formato de escritura en el log que se va a usar. [12]

Como se ha mencionado anteriormente en el diagrama en los casos de uso (apartado 3.3), el Administrador puede descargarse el fichero de log para seguir las trazas de la aplicación.

```
INFO  13/06/17 23:44:55 Acceso-LoginCorrecto: admin
INFO  13/06/17 23:45:06 Gestion-CrearProyecto:
{"Id":2,"Descripcion":"Prueba","Fecha_modificacion":"13/06/2017
23:45:05","Actualizado":false}
INFO  13/06/17 23:45:20 Gestion-BorrarProyecto:
{"Id":2,"Descripcion":"Prueba","Fecha_modificacion":"13/06/2017
23:45:05","Actualizado":false}
```

En el log anterior se muestra que el usuario Administrador ha accedido correctamente a la aplicación, luego ha creado un Proyecto para borrarlo posteriormente.

5 Integración, pruebas y resultados

5.1 Integración en el servidor

Tras terminar la fase de desarrollo llega la fase de integración. Para esta fase se utilizó una máquina virtual en un servidor con el mismo sistema operativo que la que use en la fase anterior para crear la aplicación

La aplicación se desplego en un maquina con Ubuntu 16.04. Primero, se instaló todo lo necesario tal y como se hizo en la fase anterior (anexo A). Una vez la máquina estaba a punto tocaba la tarea de publicar la aplicación web. Es muy sencillo publicar aplicaciones web con *.NET Core*, simplemente hay que usar el siguiente comando sobre la carpeta raíz:

```
dotnet publish
```

La publicación de una aplicación web se encarga de encapsular todo el código y librerías utilizadas en la parte del servidor en nuevas librerías .dll. Una vez ejecutado el comando aparece una carpeta llamada *publish* en la carpeta *bin* del proyecto. Para compilar la publicación solamente hay que llamar a la dll principal del proyecto mediante el comando *dotnet* de esta forma:

```
dotnet Moses.dll
```

Tras publicar correctamente la aplicación en la máquina de desarrollo, únicamente había que desplegarlo en el servidor. Para copiar un archivo de forma remota de manera sencilla se utiliza SCP. Una vez copiado al servidor, se ejecuta en segundo plano para que la aplicación se mantenga desplegada. El comando *nohup* facilita la ejecución de tareas en segundo plano, el comando utilizado es el siguiente:

```
nohup dotnet Moses.dll >& moses.out &
```

Toda la información de la aplicación se guarda en el fichero moses.out, al que se puede acceder de forma remota mediante SSH en caso de que suceda algún imprevisto.

5.2 Pruebas

El servidor sobre el que se ha desplegado la aplicación tiene 8gb de memoria RAM y 2 CPU.

Tras desplegar la aplicación correctamente en el servidor había que comprobar su funcionamiento. Para ello se han llevado a cabo dos tipos de pruebas. En las pruebas no se evalúa la calidad de la tradición dado que depende de las memorias de traducción utilizadas, sin embargo, si se evalúa cómo se comporta la aplicación ante distintas situaciones.

La página más solicitada de la aplicación previsiblemente será la pantalla de traducción inicio de sesión. Esta pantalla se tomará como objetivo de las pruebas.

Para llevar a cabo estas pruebas se han usado dos aplicaciones distintas pero compatibles: *BadBoy* y *JMeter*.

BadBoy es un software libre que permite grabar acciones sobre una página web para luego recrearlas repetidas veces. La tarea a grabar es la selección de un proyecto, carga de sus lenguas disponibles y traducción.

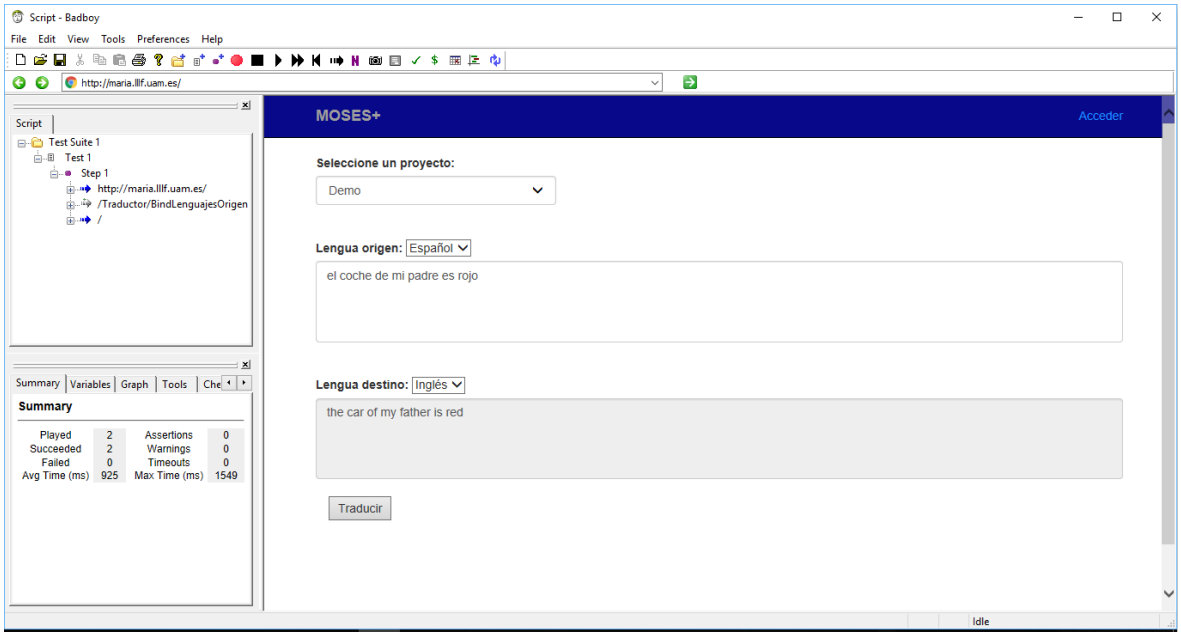

Figura 5-1 Captura de pantalla de BAdBoy

Una vez grabada la funcionalidad se va ha repetir, se descarga en formato .jmx que es compatible con *JMeter*.

JMeter es un software libre encargado de replicar la anterior funcionalidad en numerosos hilos paralelos. Al insertar el .jmx generado con *BadBoy* se eligen el número de hilos y si se desea, un pequeño retraso entre sus ejecuciones.

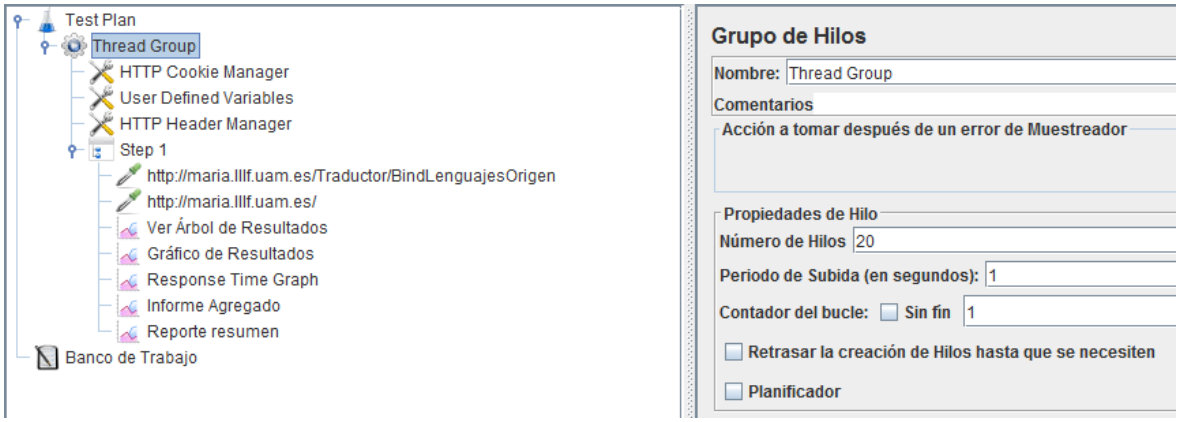

Figura 5-2 Captura de pantalla de JMeter

La primera prueba es la más sencilla, una traducción de 7 palabras en un único hilo

Etiqueta	# Muestras	% Error	Rendimiento	Kb/sec	Media de Bytes
http://maria.lllf.uam.es/Traductor/BindLenguajesOrigen	1	0,00%	21,3(/sec)	5,45	264,0
http://maria.lllf.uam.es/	1	0,00%	1,1(/sec)	14,97	13461,0
Total	**2**	**0,00%**	**2,2(/sec)**	**14,49**	**6862,5**

Tabla 5-1 Resultados de tercera prueba de JMeter

La Tabla 5.1 muestra que la aplicación responde de manera eficaz, la primera petición carga los lenguajes disponibles y la segunda envía los datos para traducir. La carga de los lenguajes disponibles tiene un alto rendimiento de 21,3 peticiones por segundo con un 100% de peticiones correctas, la frecuencia de peticiones es mayor dado a que se envían menos bytes como se indica en la última columna. La traducción es algo más costosa como se puede apreciar en su rendimiento, el rendimiento está directamente relacionado con la cantidad de bytes enviados. La traducción del texto “el coche de mi padre es rojo” se ha ejecutado a una frecuencia de 1,1 peticiones por minuto, lo que significa que la petición ha tardado menos de un segundo en procesarse de manera correcta.

La siguiente prueba va a aumentar la carga de la petición, subiendo de 7 a 500 palabras. Los resultados son los siguientes.

Etiqueta	# Muestras	% Error	Rendimiento	Kb/sec	Media de Bytes
http://maria.lllf.uam.es/Traductor/BindLenguajesOrigen	1	0,00%	12,8(/sec)	3,31	264,0
http://maria.lllf.uam.es/	1	0,00%	2,6(/min)	0,81	19403,0
Total	**2**	**0,00%**	**5,1(/min)**	**0,82**	**9833,5**

Tabla 5-2 Resultados de segunda prueba de JMeter

Aquí se puede apreciar que la carga de la petición (bytes) es mucho mayor y por eso su frecuencia de peticiones se ha reducido a 2,6 peticiones por minuto, equivale a que la petición ha tardado 23,1 segundos en ejecutarse y devolver la traducción correcta del texto.

5.2.1 Pruebas de estrés

Al tratarse de una aplicación web desplegada en un servidor accesible a todas horas, es recomendable someterla a pruebas de estrés para ver cómo se comporta.

Con el *JMeter* al insertar el .jmx generado con *BadBoy* se elegen el número de hilos y si se queiere un pequeño retraso entre sus ejecuciones. La prueba que se ha realizado está formada por 20 hilos que simulan 20 usuarios distintos traduciendo el texto de la primera prueba simultáneamente.

Etiqueta	# Muestras	% Error	Rendimiento	Kb/sec	Media de Bytes
http://maria.lllf.uam.es/Traductor/BindLenguajesOrigen	20	0,00%	19,6 (/sec)	5,05	264,0
http://maria.lllf.uam.es/	20	0,00%	48,2(/min)	56,35	13461,0
Total	**40**	**0,00%**	**8,5(/sec)**	**32,32**	**68682,5**

Tabla 5-3 Resultados de tercera prueba de JMeter

Con esta prueba de estrés se puede reseñar que la aplicación responde rápidamente a un flujo en paralelo de peticiones pequeñas. Comparando con los resultados de la primera prueba se ha disminuido el rendimiento de 1,1 peticiones por segundo a 48,2 peticiones por minuto, esto significa que el tiempo de cada petición ha aumentado de 0,9 segundos a 1,24 segundos.

A continuación, hice la misma prueba de estrés con el texto de la segunda prueba. Esta prueba no superó las expectativas y el servidor no aguantó la alta carga. Para que el servidor supere esta última prueba habría que aumentar la capacidad de procesamiento de la CPU. Actualmente tiene 2 núcleos, duplicándolo se conseguirían pasar las pruebas.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Este proyecto tenía como objetivos proporcionar una herramienta de traducción automática para los alumnos de traducción e interpretación. Se ha desarrollado una aplicación web que permite a los alumnos subir sus memorias de traducción para posteriormente generar modelos de lenguaje y ver los frutos de sus memorias. Esta utilidad les permite corregir sus traducciones hasta acercarse a la traducción óptima que ellos desean.

La aplicación proporciona una herramienta docente para convertir traducciones humanas en modelos de lenguaje para traducir todo tipo de textos. Además, permite compartir los modelos de lenguaje de manera libre para que otras personas puedan utilizar el modelo para traducir.

Se ha desarrollado para que sea fácil e intuitiva de manejar, consiguiendo una interfaz sencilla y ajustable a todo tipo de dispositivos o navegadores web.

La aplicación ha sido desarrollada con tecnologías novedosas (.NET Core en Linux) que permiten optimizar los recursos utilizados. Una vez terminada se ha validado su desarrollo mediante pruebas. Se ha detectado que es preciso disponer de un preciso hardware para no sufrir ningún imprevisto en una clase real.

6.2 Trabajo futuro

Derivado de la implementación realizada en este TFG, se ha propuesto como un proyecto de Innovación Docente de la UAM. El objetivo es que los alumnos de la asignatura Traducción Automática de Cuarto del Grado de Traducción e Interpretación puedan hacer uso de la aplicación. Para ello hay que llevar a cabo algunas mejoras. El código del proyecto es FYL_007.17_INN.

La asignatura cuenta con aproximadamente 25 alumnos. Como se ha comentado en el apartado de pruebas de estrés (5.2.1), el servidor no ha aguantado 20 peticiones simultáneas de 500 palabras cada una. Habría que aumentar la capacidad del servidor para no sufrir caídas en su momento más alto de uso.

Además de mejorar el servidor, al introducirse 25 usuarios, sería necesaria una pantalla de gestión de usuarios. Con esta pantalla se podrían dar de alta con facilidad usuarios para que ejerzan como traductores. Otra mejora que se podría implementar es la posibilidad de descargarse las memorias de traducción subidas, para mejorarlas y volver a subirlas a la aplicación. Con esta mejora se agilizaría el trabajo del traductor al buscar en su ordenador la última memoria subida a la aplicación.

Referencias

[1] Traductor de Google https://translate.google.com/
[2] Google Translator Toolkit https://translate.google.com/toolkit
[3] Moses Statistical Machine Translation System http://www.statmt.org/moses/
[4] "Stadistical Machine Translation" http://www.cs.cmu.edu/~aberger/mt.html
[5] https://www.w3schools.com/xml/ajax_intro
[6] Información general de Entity Framework https://msdn.microsoft.com/es-es/library/bb399567(v=vs.110).aspx
[7] *Developing Web Applications* http://www.lansa.com/products/web-development.htm
[8] MVC https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
[9] *Commands .NET Core* https://docs.microsoft.com/en-us/dotnet/articles/core/tutorials/using-with-xplat-cli
[10] https://www.tutorialspoint.com/sqlite/sqlite_drop_table.htm
[11] https://www.sqlite.org/cli.html
[12] http://dotnetthoughts.net/how-to-use-log4net-with-aspnetcore-for-logging/
[13] http://www.hossambarakat.net/2016/02/03/configuring-redis-as-asp-net-core-1-0-session-store/
[14] Migraciones .NET Core http://benjii.me/2016/05/dotnet-ef-migrations-for-asp-net-core/
[15] https://www.taus.net/translate/mosescore/machine-translation-and-moses-tutorial#principles-of-mt
[16] CLR https://es.wikipedia.org/wiki/Common_Language_Runtime
[17] Apertium https://es.wikipedia.org/wiki/Apertium
[18] Traducción automática estadística https://es.wikipedia.org/wiki/Traducci%C3%B3n_autom%C3%A1tica_estad%C3%ADstica
[19] Petr Homola y Vladislav Kubon, "Improving Machine Translation Between Closely Related Romance Languages" http://www.mt-archive.info/EAMT-2008-Homola.pdf

Glosario

LGPL	GNU Lesser General Public License
GTT	Google Translator Toolkit
EMS	Experiment Management System
API	Application Programming Interface
BD	Base de datos
MVC	Modelo Vista Controlador
AJAX	Asynchronous JavaScript And XML
EF	Entity Framework
SSH	Secure Shell
SCP	Secure Copy Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
UI	Interfaz de Usuario
CMU	Carnegie Mellon University
IBM	International Business Machines

Anexos

A Manual de instalación

Instalar en Ubuntu 16.04:

1. Instalar Moses:
   a. `sudo apt-get install build-essential git-core pkg-config automake libtool wget zlib1g-dev python-dev libbz2-dev`
   b. `sudo apt-get install libsoap-lite-perl`
   c. `git clone https://github.com/moses-smt/mosesdecoder.git`
      `cd mosesdecoder`
   d. `make -f contrib/Makefiles/install-dependencies.gmake`
   e. `./compile.sh`

2. Instalar dot NET CORE:
   a. `sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ xenial main" > /etc/apt/sources.list.d/dotnetdev.list'`
   b. `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893`
   c. `sudo apt-get update`
   d. `sudo apt-get install dotnet-dev-1.0.4`

3. Instalar SQLite3:

```
sudo apt install sqlite3
```

Código de la aplicación en github: https://github.com/mgragera/MosesApp

B Manual de uso

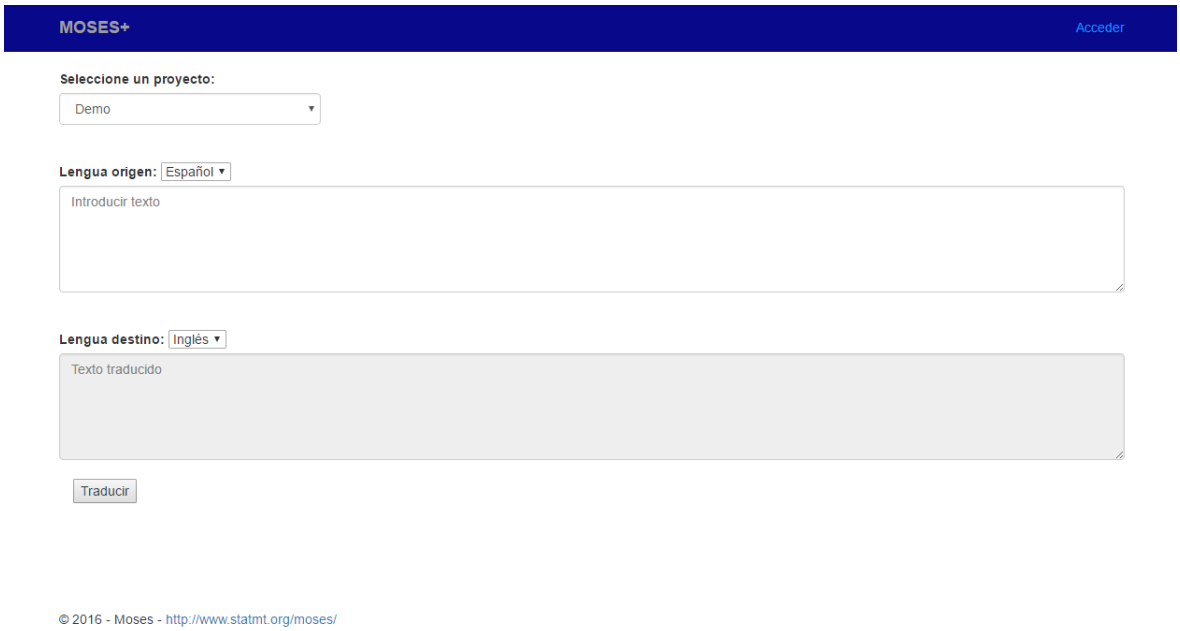

Figura 0-1 Captura Traductor sin Acceso

Accediendo a la dirección http://maria.lllf.uam.es/ se accede a la página de la figura anterior. Desde esta pantalla se puede ir a la pantalla de la figura siguiente pulsando en el texto **Acceder** de la esquina superior derecha. Esta página ofrece la posibilidad de elegir un proyecto y traducir un texto instruido en el cuadro de texto dejado de Lengua origen. Al seleccionar un proyecto se rellenarán los desplegables de manera automática con las lenguas disponibles para ese proyecto.

Figura 0-2 Captura Acceso

Desde esta pantalla es posible iniciar sesión en la aplicación con un usuario y sus credenciales. Dependiendo con el tipo de usuario que se acceda tendrá habilitadas unas funcionalidades u otras.

Figura 0-3 Captura Menú de proyectos

Una vez iniciada la sesión en la aplicación aparece la pantalla de Proyectos (figura 0-3). Esta página es común para todos los tipos de usuario. Ofrece tres funcionalidades distintas que son accesibles desde los iconos de cada proyecto. Al deslizar el cursor por encima de los iconos se muestra su funcionalidad. En orden son **Traductor, Memorias y Glosario**. Al pulsar en uno de ellos nos selecciona la funcionalidad para el proyecto elegido como muestro a continuación.

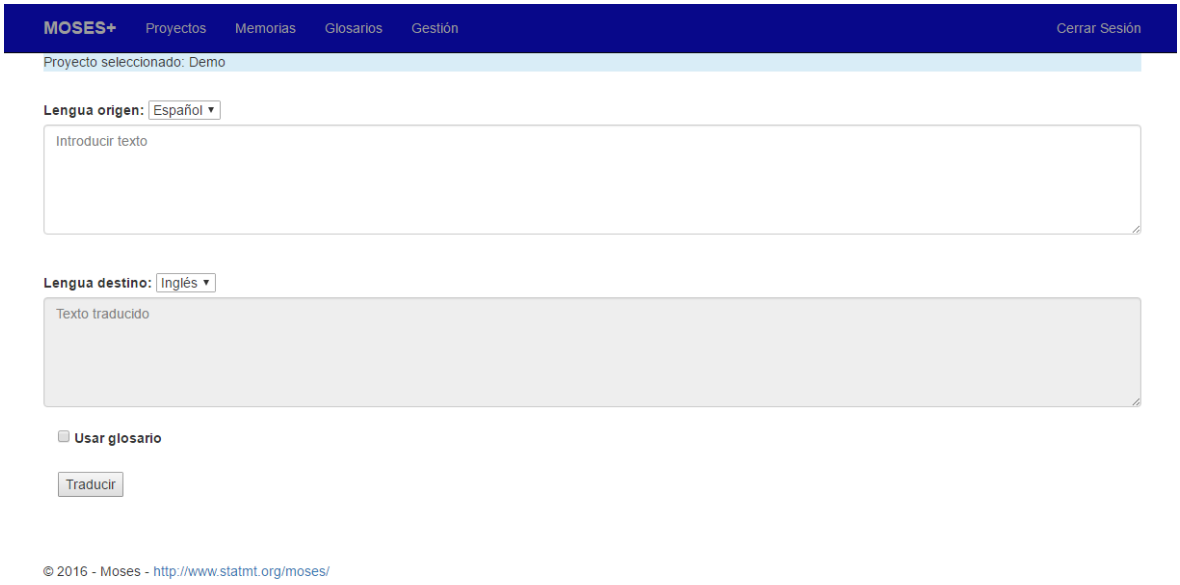

Figura 0-4 Captura Traductor tras seleccionar Proyecto

La figura 0-4 reprensenta la traducción tras haber seleccionado un proyecto en la pantalla anterior. Existen varias direrencias entre esta pantalla y la de la figura 0-1. Esta pantalla tiene en la dirección url variable según el proyecto seleccionado, y es accesible sin inicio de sesión. Con esto consigo que la página sea compatirble por el traductor y así más usuarios pueden hacer uso de únicamente su Proyecto sin tener que seleccionarlo. Un ejemplo de la dirreción url que se genera es: http://maria.lllf.uam.es/?idProyecto=1

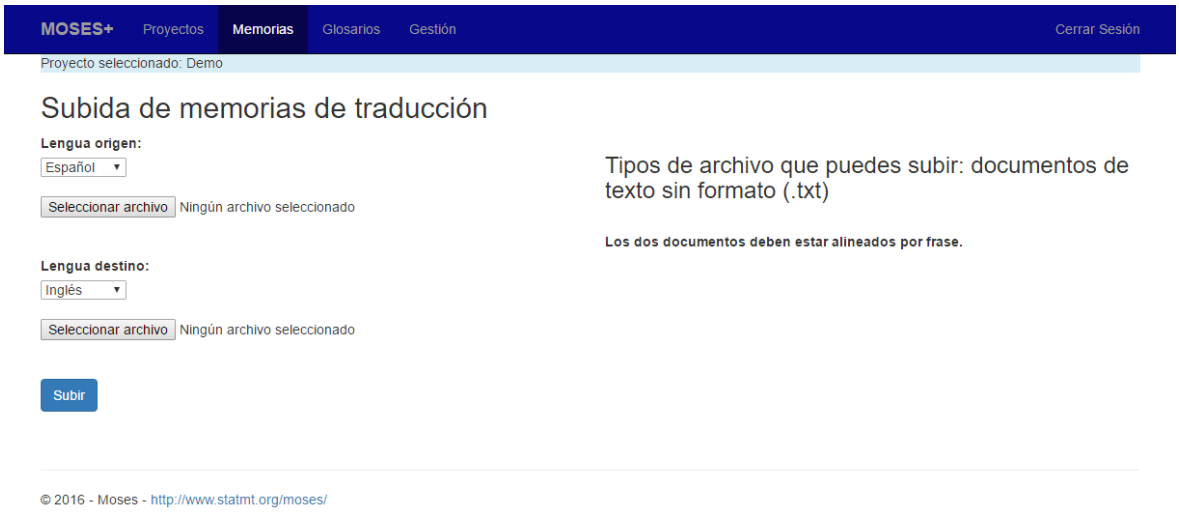

Figura 0-5 Captura Subida de Memorias de traducción

La figura 0-5 es accesible pulsando el segundo icono de la pantalla de Proyectos (figura 0-3). Desde aquí se suben las memorias de traducción. Como se indica en la página los archivos que se deben subir son documentos de texto sin formato. Cuando la subida se realiza con éxito aparece un mensaje indicándolo.

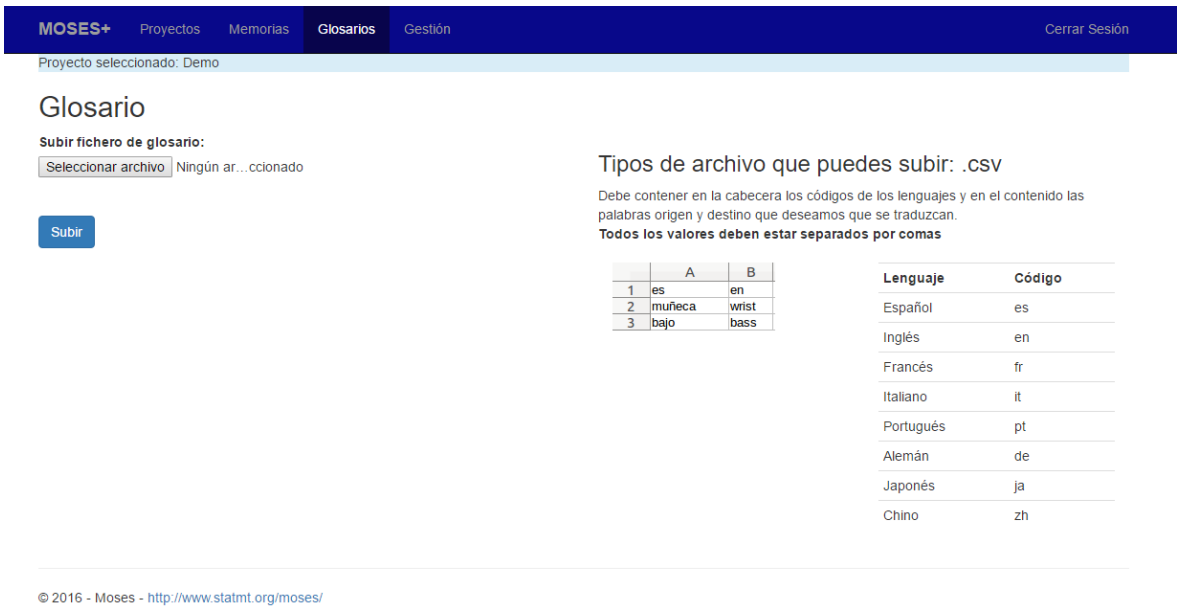

Figura 0-6 Captura Subida de Glosario

La figura 0-6 es accesible pulsando el tercer icono de la pantalla de Proyectos (figura 0-3). Desde esta pantalla se suben los glosarios a la aplicación. Al igual que en la captura anterior en la propia página se muestran los tipos de fichero y el formato de ficheros que estan admitidos para ser procesados. Si el fichero se sube de forma correcta o incorrecta se le notifica al usuario.

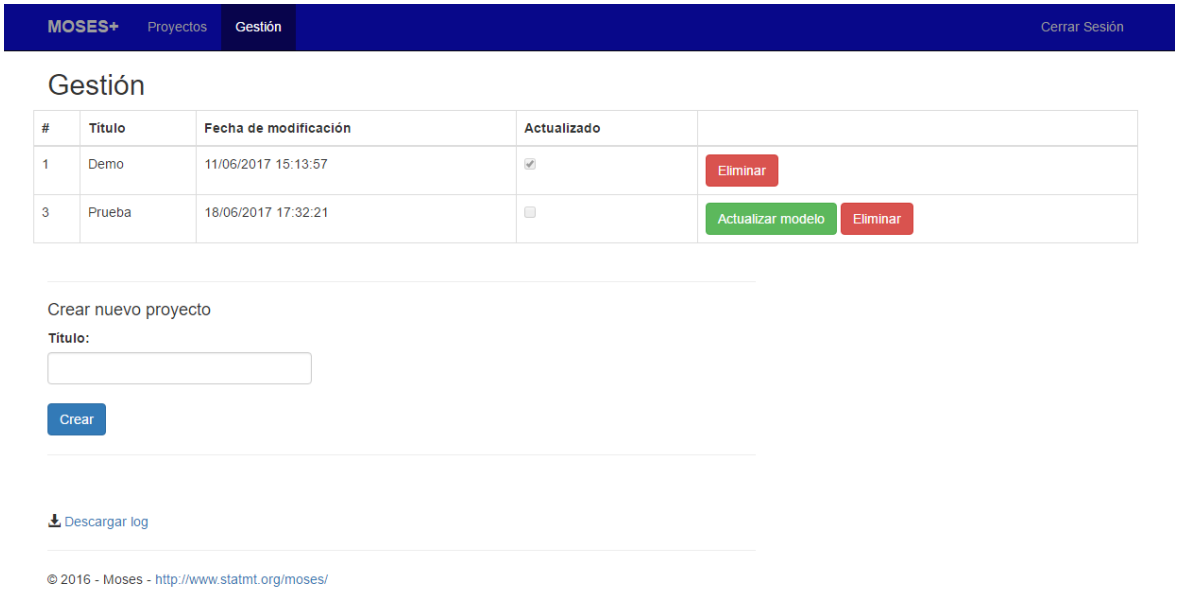

Figura 0-7 Captura Gestión

Esta pantalla (figura 0-7) al contrario que las anteriores es únicamente accesible por aquellos usuarios que posean privilegios de administrador (tipo de usuario 1). Desde aquí se dan de alta nuevos Proyectos, se eliminan Proyectos o se generan los modelos de lenguajes. Además, se puede descargar el log de la aplicación.

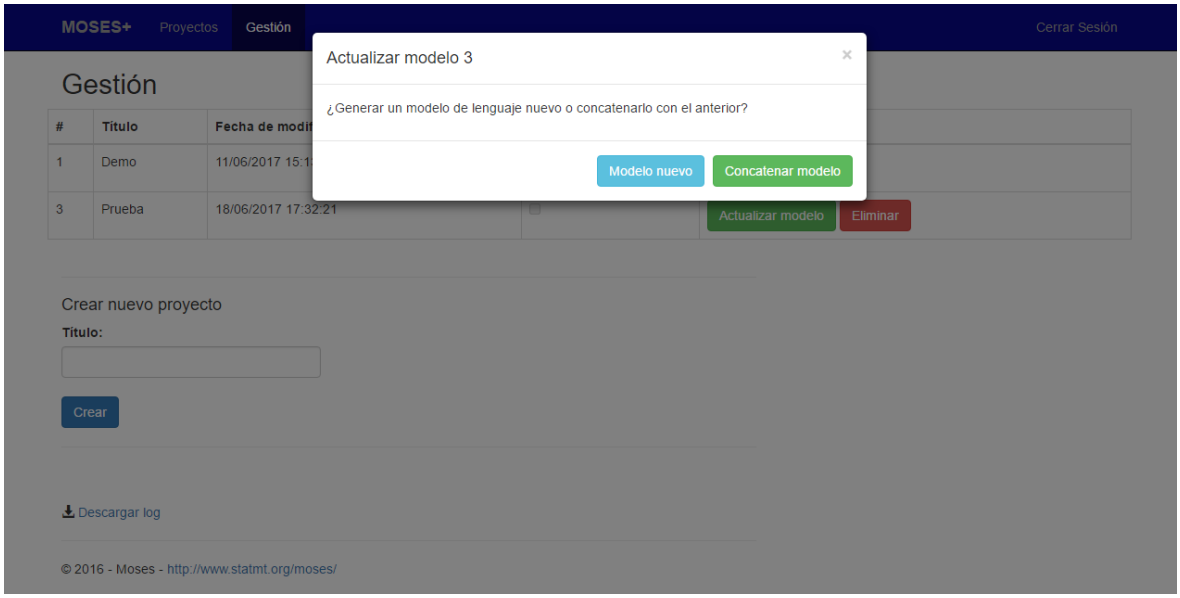

Figura 0-8 Captura panel Actualizar Modelo

Dentro de la página anterior (figura 0-7) al hacer click en generar modelo aparece un panel dando dos opciones al administrador. Si se selecciona la primera opción el modelo de lenguaje se genera usando únicamente las últimas memorias subidas al proyecto. Seleccionando la segunda opción se usan todas las memorias subidas al proyecto.

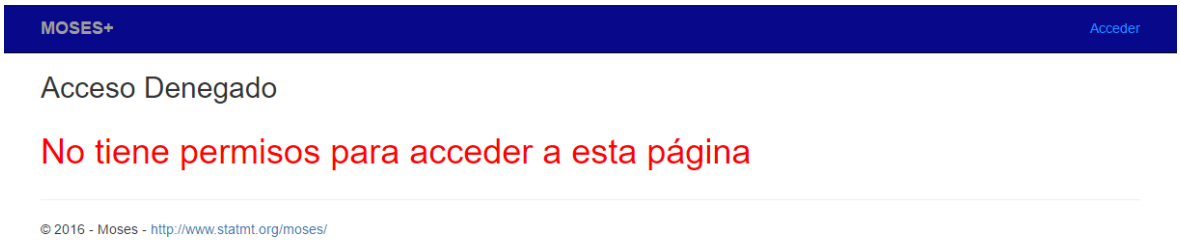

Figura 0-9 Captura Acceso denegado

Esta pantalla (figura 0-9) es la que se muestra cuando se intenta acceder a una página sin los permisos suficientes. Por ejemplo, al acceder mediante la dirección http://maria.lllf.uam.es/Gestion sin haber accedido anteriormente con un usuario con permisos de administrador.

C Ejemplo de Memoria de traducción

Fragmento de una memoria de traducción en español:

```
Triunfo de la derecha populista en Austria, juntos forman el 29 %
Las elecciones anticipadas al parlamento austriaco han traÃdo
segÃ°n los primeros resultados aproximados, un apreciable
debilitamiento de ambos partidos de la gran coaliciÃ³n actual y un
expresivo fortalecimiento del partido de la derecha populista.
Unas fuertes y especiales pÃ©rdidas ha sufrido el Partido Popular
AustrÃaco (OVP), donde se tambalea seriamente la posiciÃ³n del
actual presidente, Wilhelm Molterer.
```

Fragmento de una memoria de traducción en inglés:

```
Right-wing populists triumph in Austria, have total of 29 percent
According to the first preliminary results of the early parliament
elections in Austria have brought about a perceptible weakening of
both parties in the present large coalition and a significant
boost for the right-wing populist parties.
The Austrian People's Party (Ã–VP), where the position of the
current head, Wilhelm Molterer, is being severely jolted, suffered
particularly great losses.
```