Universitat Autònoma de Barcelona

Trabajo de fin de máster

Dragos Manager: Desarrollo de un programa de gestión de proyectos para traductores

Daniel Pérez Atienza

Tutor: Ramón Piqué

Máster en Tradumática: Tecnologías de la Traducción
Facultad de Traducción e Interpretación
Universitat Autònoma de Barcelona
Curso 2019/2020

Resumen

En este proyecto hemos diseñado y desarrollado una herramienta informática a la que hemos llamado Dragos Manager con el objetivo de facilitar y agilizar ciertos procesos y tareas de la gestión de proyectos de traducción. El público al que va dirigido el programa son traductores autónomos y empresas de traducción. Para su desarrollo hemos utilizado el lenguaje de programación Python y el lector podrá seguir paso a paso los métodos utilizados para su creación desde el diseño de la interfaz a la escritura del código, además de los problemas surgidos y su solución. Planteamos también las posibles mejoras basadas en las propuestas de herramienta que establecimos en un comienzo y los resultados obtenidos.

Palabras clave: Herramienta de traducción, gestión de proyectos, tecnologías de la traducción, Python.

Abstract

In this project we have designed and developed a computer tool called Dragos Manager with the aim of facilitating and speeding up certain processes and tasks in the management of translation projects. The target audience of the program is freelance translators and translation companies. For its development we have used the Python programming language and the reader will be able to follow step by step the methods used for its creation from the design of the interface to the writing of the code, as well as the problems encountered and their solution. We also propose possible improvements based on the tool proposals we established at the beginning and the results obtained.

Keywords: Translation tool, project management, translation technologies, Python.

Resum

En aquest projecte hem dissenyat i desenvolupat una eina informàtica que hem anomenat Dragos Manager amb l'objectiu de facilitar i agilitzar alguns processos i tasques de la gestió de projectes de traducció. El públic al qual va dirigit el programa són traductors autònoms i empreses de traducció. Per al seu desenvolupament hem utilitzat el llenguatge de programació Python i el lector podrà seguir pas a pas els mètodes utilitzats per a la seva creació des del disseny de la interfície a l'escriptura del codi, a més dels problemes sorgits i la seva solució. Plantegem també les possibles millores basades en les propostes d'eina que vam establir en el seu començament i els resultats obtinguts.

Paraules clau: Eina de traducció, gestió de projectes, tecnologies de la traducció, Python.

1. Introducción

La tecnología va avanzando inexorablemente y la figura del traductor tradicional está quedando atrasada, a pesar de que esta tecnología no hace más que simplificar la tarea de la traducción ofreciéndonos herramientas con las que mejorar cada vez más la productividad.

El traductor de hoy en día ya no trabaja con papel y bolígrafo (aunque puede haber casos excepcionales), sino que trabaja directamente con Microsoft Word o procesadores de texto similares porque objetivamente aumenta muchísimo la productividad.

A pesar de la mejora exponencial que supuso cambiar de papel a procesadores de texto, a medida que avanza la tecnología podemos comprobar que tenemos opciones mejores para aumentar aún más la productividad, es decir, las herramientas TAO que nos permiten crear proyectos, gestionar memorias de traducción y bases de datos, pretraducir con TA y poseditar, etc.

A pesar de la existencia de herramientas tan útiles, no debemos confundirnos, muchos traductores no se han adaptado todavía a los nuevos tiempos y siguen traduciendo en Word, aunque esperamos que esto vaya cambiando con una formación actualizada a los nuevos tiempos. Entre las razones que pueden estar provocando que los traductores no actualicen sus herramientas de trabajo, según nuestro punto de vista, pueden encontrarse:

- Los traductores se han acostumbrado a procesadores de texto y aprender sobre un nuevo programa y cambiar su flujo de trabajo va a disminuir la productividad en un principio hasta aprender a dominar el programa y todas sus funciones. La idea de mejorar la productividad es muy atractiva pero si vives exclusivamente de la traducción, el tiempo de adaptación a diferentes herramientas puede suponer una reducción importante de los ingresos dependiendo de, claro está, el ritmo de adaptación de cada individuo.
- Muchos traductores desconocen la existencia de las herramientas TAO. Este caso no creo que se dé porque los profesionales de letras puras sean tecnófobos (como se ha considerado tradicionalmente por prejuicios), sino que más bien se trata de una ignorancia tecnológica, es decir, no se les ha enseñado o no se han preocupado por informarse de estos temas. En el caso de traductores con años de experiencia quizás se deba a que no se preocupan por actualizarse, o bien porque lo que tienen funciona y no quieren cambiarlo, o bien porque no conocen ni siguen habitualmente canales de información sobre tecnología de la traducción. Por otro lado, en el caso de traductores recién graduados, en nuestra opinión debería haber un cambio drástico en las asignaturas de las carreras y pasar a enseñar ya en primer año todas las herramientas TAO y no dejarlo para el último curso para que sea recordado de manera anecdótica.

En el caso del desconocimiento de las nuevas tecnologías de traducción, la responsabilidad recae en el traductor que no se preocupa de actualizarse, aunque podemos contribuir a la formación de estos usuarios mediante entradas de blog, conferencias, jornadas de traducción y cualquier otro evento que reúna a traductores.

En cuanto al caso de la disminución de la productividad de manera temporal nos planteamos si seríamos capaces de crear una herramienta que por sus utilidades pudiera mejorar la productividad y que, además, por su sencillez redujera al mínimo el tiempo de adaptación a la nueva herramienta.

Así pues, planteado este reto, nos propusimos desarrollar una herramienta que reúna tareas y simplifique procesos para ayudar al traductor. Pretendemos programar con Python 3 una aplicación de escritorio que automatice y organice ciertos procesos relativos a la gestión de los proyectos. En un principio, el sistema operativo al que estará destinado será Windows, aunque si el resultado es satisfactorio estudiaremos hacer un *port* a otros sistemas.

En los proyectos de traducción, tener un flujo de trabajo y una organización establecidos son la clave para lograr unos mejores resultados en un menor tiempo. Como consideramos que dentro del flujo de trabajo de un equipo de traducción la gestión de proyectos es el proceso que involucra realizar más tareas distintas, creemos que los traductores (y sobre todo los traductores novicios) podrían perder demasiado tiempo al comienzo pensando en qué tarea es la siguiente en su flujo de trabajo.

El tiempo utilizado para pensar qué debo hacer y cómo lo debo hacer puede llegar a suponer bastante tiempo a la larga, por lo que queremos cubrir algunas de las tareas más importantes que consigan ahorrar tiempo a los traductores, y especialmente a aquellos que cubren personalmente y por completo todas las fases de un proyecto.

En el siguiente punto y a lo largo de este trabajo veremos las tareas que pensamos que un programa de este estilo debería tener y las que al final conseguiremos que realice teniendo en cuenta el tiempo y los recursos de los que disponemos.

Como hemos explicado en los párrafos anteriores, crear un programa para ayudar en procesos de gestión de proyectos podría resultar el objetivo principal de este trabajo, y, aunque es cierto que este es un de los objetivos, no es el único que queremos plantear.

El segundo objetivo, y no menos importante, es el aprendizaje. Durante nuestra vida académica nos hemos sentido profundamente atraídos a la informática y siempre nos ha gustado aprender sobre ellas, pero como nuestra rama del conocimiento es más cercana a las letras que a las tecnologías, no tuvimos la oportunidad de profundizar tanto en esta área como nos habría gustado.

Siempre quisimos aprender sobre lenguajes de programación y con el Máster de Tradumática pudimos dar los primeros pasos en el lenguaje de programación Python, pero en nuestra búsqueda de conocimiento quisimos ir más allá y esta fue precisamente la razón para elegir el tema de estudio de este trabajo.

Python es un lenguaje de programación de alto nivel, es decir, las líneas de código son más parecidas al lenguaje natural (primordialmente inglés) que al lenguaje máquina y además es de los lenguajes más legibles, por lo que pensamos que, siendo los idiomas nuestro punto fuerte, podría ser más sencillo de aprender que otros lenguajes de programación.

En definitiva, la idea de aprender un lenguaje de programación y salir un poco de nuestra área de conocimiento para adentrarnos en nuevas aguas era muy atractiva y, para saciar nuestra ansia de conocimiento, creímos pertinente aprender mediante la práctica de la programación de una aplicación propia.

2. Propuesta de herramienta

Al plantearnos en un principio el desarrollo de una herramienta hubo ciertas preguntas que debíamos resolver para encaminar nuestro proyecto hacia alguna dirección en concreto.

En primer lugar, ¿a qué parte del público va dirigida nuestra herramienta? Esta es si no la pregunta más importante de todas, una de gran importancia. Nuestra herramienta en un principio va dirigida a profesionales de la traducción autónomos y también sería posible implementarla como apoyo para el gestor de proyectos de un equipo de traducción en una empresa. Algunas de las funciones que planeamos incluir en nuestra herramienta son muy generales, por lo que podrían utilizarla también profesionales de otros ámbitos, si vieran que esas funciones les podrían ser útiles, pero creemos apropiado establecer un público objetivo en el que pensar a la hora de desarrollar las tareas que pensamos que le podría ser útil.

En segundo lugar, los proyectos de traducción tienen muchos procesos y un texto pasa por muchas manos y varios equipos hasta que se puede considerar que un texto está acabado, entonces ¿dentro del flujo de trabajo de un proyecto de traducción, a qué figuras del equipo van dirigidas esta herramienta? Nuestra primera idea es que cubriera algunas tareas del gestor de proyectos para facilitarle el trabajo o, en el caso de los traductores autónomos, guiarles a lo largo del proceso desde que reciben el texto hasta su devolución traducido.

Esta respuesta nos lleva de manera directa a nuestra tercera pregunta, ¿qué fases del proceso de un proyecto de traducción cubrirá esta herramienta? Desde un principio no queríamos cubrir un punto en concreto de las fases de un proyecto de traducción, sino que queríamos aportar algunas utilidades en concreto que se incluyen en diversas fases dentro del marco de la gestión de proyectos que veremos a continuación con la siguiente pregunta.

Así pues, enlazándolo con la pregunta anterior y proponiendo la cuarta, ¿dentro de las fases del proceso de traducción qué tareas en concreto se piensan abordar con el desarrollo de esta herramienta? Como cualquier estudiante del máster de Tradumática podríamos afirmar que, sin duda alguna, hemos pasado como mínimo los cinco últimos años de nuestra vida académica haciendo traducciones de textos de diferentes ámbitos y lenguas a otra lengua transcreando y adaptando sin parar, por lo que creo que podemos entender cuáles son los retos y obstáculos que enfrenta cada día un traductor, y conocemos sus debilidades y fortalezas.

A continuación, veremos las tareas que podría cubrir una herramienta de este estilo y de aquí extraeremos las utilidades que finalmente tendrá nuestro programa:

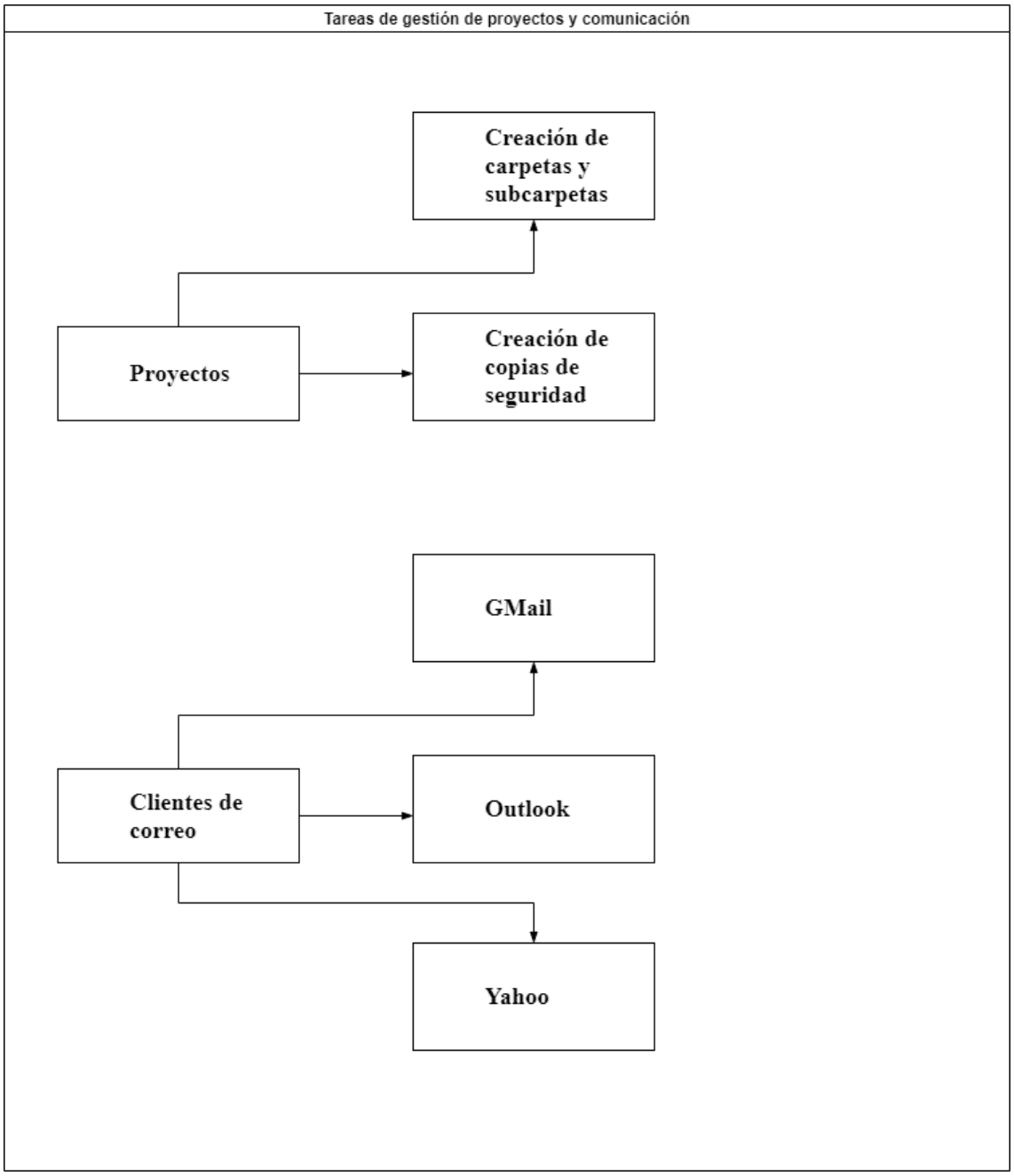

Figura 1: Diagrama de las funciones del gestor de proyectos y comunicación

La categoría de proyectos englobaría todo aquello relacionado con la organización de archivos en carpetas y subcarpetas. A esta categoría nos gustaría añadirle una utilidad de creación de copias de seguridad para archivos.

- Gestión de archivos: creación de una serie de carpetas y subcarpetas para facilitar la organización de los archivos originales, meta, memorias de traducción, glosarios, versiones anteriores y versiones modificadas.
- Creación de copias de seguridad: siguiendo con el punto anterior, contemplamos la posibilidad de que, al crear las carpetas y subcarpetas con el programa, se genere a su vez una carpeta de copias de seguridad con el objetivo de no perder los cambios hechos anteriormente en un archivo. El usuario es el que decidirá cuándo y de qué archivos se hacen las copias de seguridad.

En cuanto a la comunicación con el cliente, pretendemos ofrecer la conexión con una serie de clientes de correo para facilitar y agilizar su acceso en todo momento. También consideramos la opción de ofrecer un cliente de correo local para enviar y recibir correos escribiendo en unos cuadros de texto el correo del remitente, el del destinatario y el cuerpo del mensaje. Esta tarea serviría principalmente para asegurar que el canal se mantiene abierto y tener información actualizada sobre las necesidades del cliente en cuanto al proyecto. Junto con el sistema de notificaciones que nos gustaría que tuviera el calendario, sería interesante añadir una notificación de llegada de correos

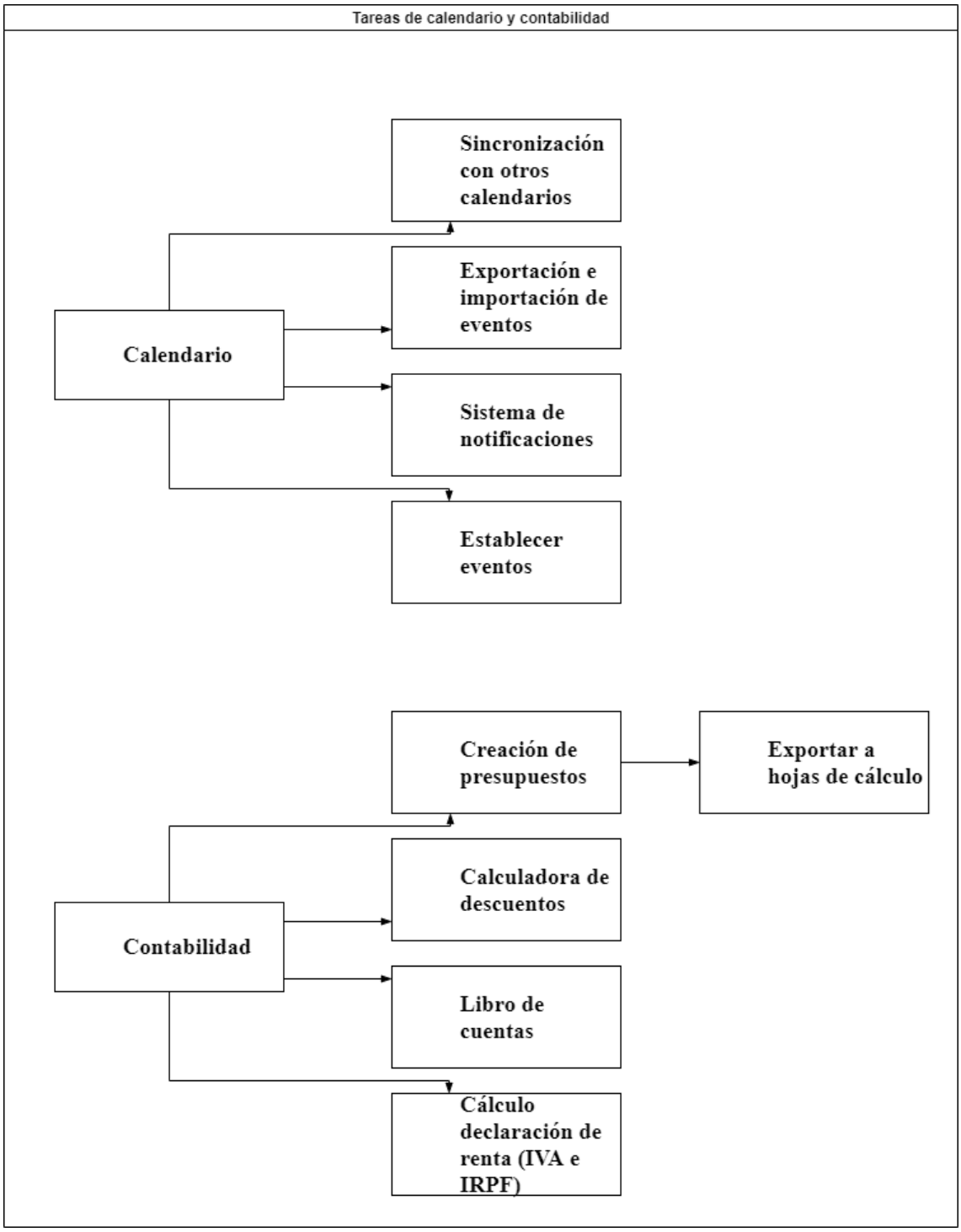

Figura 2: Diagrama de las funciones del calendario y contabilidad

Calendario: creación de un calendario con un sistema de notificaciones de las actualizaciones de los proyectos y fechas límite de entrega al iniciar el programa.

- Dependiendo del proyecto (hay algunos proyectos con fechas estrictas y otras con indicaciones sobre la fecha de entrega más laxa) podríamos dar la opción de poner como fecha límite un día o un intervalo de días.
- Opción de marcar un proyecto en el calendario con un color que indica la urgencia de la entrega. Rojo sería muy urgente, amarillo sería medianamente urgente y azul sería no urgente. Estos parámetros se podrán tener en cuenta más adelante para la elaboración del presupuesto, ya que la urgencia de un proyecto suele suponer un recargo.
- Se estudiará la opción de poder sincronizar el calendario del programa con servicios de calendario como el de Outlook, Google o similares que tenga el equipo, y la exportación de este calendario en formato ICS.

Una de las tareas que creemos que podría ser tremendamente útil para un traductor sería un asistente para la creación de presupuestos que trabaje con tablas y porcentajes, y que funcionaría introduciendo el número de palabras y el porcentaje por el que se te pagará (dependiendo de la presencia o no de las palabras en la memoria que nos envía el cliente) y se genera un presupuesto listo para adjuntarlo a un correo.

Aparte de la calculadora de descuentos, creemos que podríamos ofrecer un libro de cuentas donde apuntar facturas de proyectos, gastos y todo lo necesario para elaborar la declaración de la renta, como por ejemplo, bienes de inversión. Con todo ello podríamos ofrecer una serie de tareas para elaborar las cuentas y la declaración teniendo en cuenta el IVA y el IRPF.

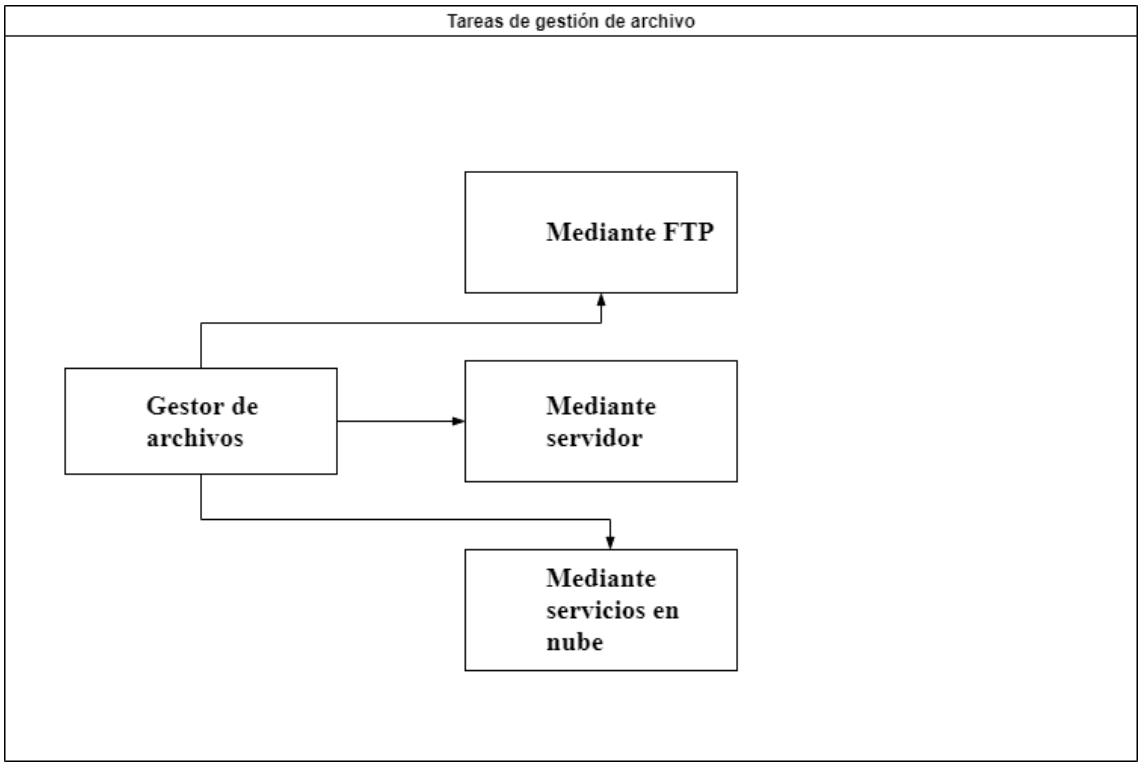

Figura 3: Diagrama de la función de gestión de archivos

Quizás en algunos proyectos pequeños los archivos que debamos tratar sean solo textos que se puedan enviar por correo, pero si tratamos productos digitales como por ejemplo videojuegos, películas o programas, el peso es tan grande que enviarlos por correo es imposible, además de que es un método más bien inseguro para tratar archivos que el público no debería conocer todavía.

Para evitar esta situación queríamos que nuestra aplicación pudiera abrir programas que permitan la gestión de archivos grandes y su transferencia entre cliente y traductor. En un principio queremos tratar un programa de gestión de archivos mediante FTP como Filezilla o similares. Aparte de FTP hay otros medios para intercambiar archivos como un servidor local, del que no disponemos y, por lo tanto será complicado intentar desarrollarlo en estos momentos, y servicios de almacenado en la nube, que podríamos intentar conectar con el programa. Estas tres opciones son válidas para su estudio e implementación en nuestra herramienta.

A continuación, presentamos algunas utilidades que pensamos que podría tener nuestro programa para ayudar al traductor y que nos gustaría implementar, pero lo más probable es que por falta de tiempo de investigación y de recursos tengamos que posponerlo a próximas actualizaciones del software.

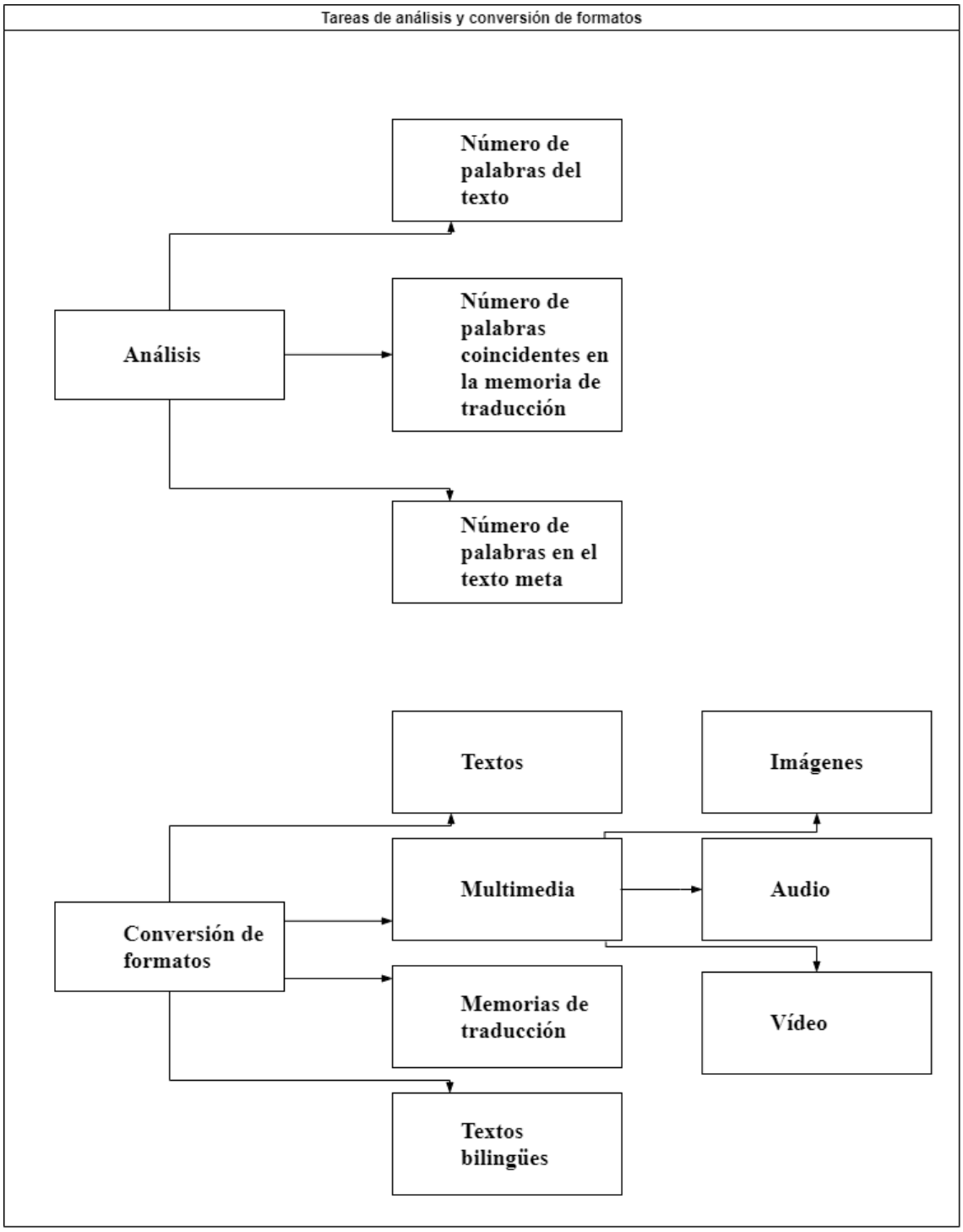

Figura 4: Diagrama de las funciones de análisis textual y conversión de formatos

Una de las características que más nos gustan de las herramientas TAO que hemos probado es la elaboración de análisis cuantitativos para obtener el número de palabras de un texto, y en algunos casos incluso encontramos un análisis comparativo que nos indica qué palabras del texto están ya contenidas en la memoria de traducción. Sin embargo, en algunas herramientas TAO, nos parece que los procesos que hay que seguir para llevar a cabo un análisis son demasiados en la mayoría de los casos. En muchas herramientas TAO por ejemplo es obligatorio crear un proyecto y tienes que pasar por innumerables ventanas de diálogo que parecen no acabar nunca.

No creemos necesario que, si el usuario solo quiere generar un informe que contenga el número de palabras, tenga que tardar un tiempo que podría ser bastante menor. Así pues, pretendemos simplificar este proceso y que se haga un análisis tomando archivos y dándole a un botón, un método bastante simple que ayudará al traductor a ahorrar unos valiosos minutos.

Uno de los problemas persistentes en la industria de la traducción tiene que ver con los formatos de programas privativos, y es que las empresas de herramientas TAO crearon sus propios formatos para trabajar con sus archivos.

Esta situación supone que si un traductor no tiene una licencia de cierto programa pero su cliente le pide que la extensión del archivo bilingüe o de la memoria tienen que ser el de un programa privativo, entonces se vería en la situación de comprar una licencia que no cuesta poco precisamente o buscar una nueva vía como pedir a alguien con licencia que se lo transforme o descargar versiones de prueba que le haría perder bastante tiempo. Nuestra intención es solventar los problemas de compatibilidad de formato entre archivos generados con herramientas TAO privativas diferentes con un conversor de archivos.

En principio en nuestra herramienta trataremos la conversión de formatos privativos a formatos tratados como los estándares, es decir, TMX para memorias de traducción y XLIFF para textos bilingües.

Cabría tratar también la conversión de un archivo de extensión privativa de una empresa a la extensión de otra empresa, aunque podría suponer un reto demasiado grande si tenemos en cuenta que el código de sus programas no es abierto y buscar información podría convertirse en una causa perdida. Aun así, creo que es un problema del día a día y que merece la pena su investigación.

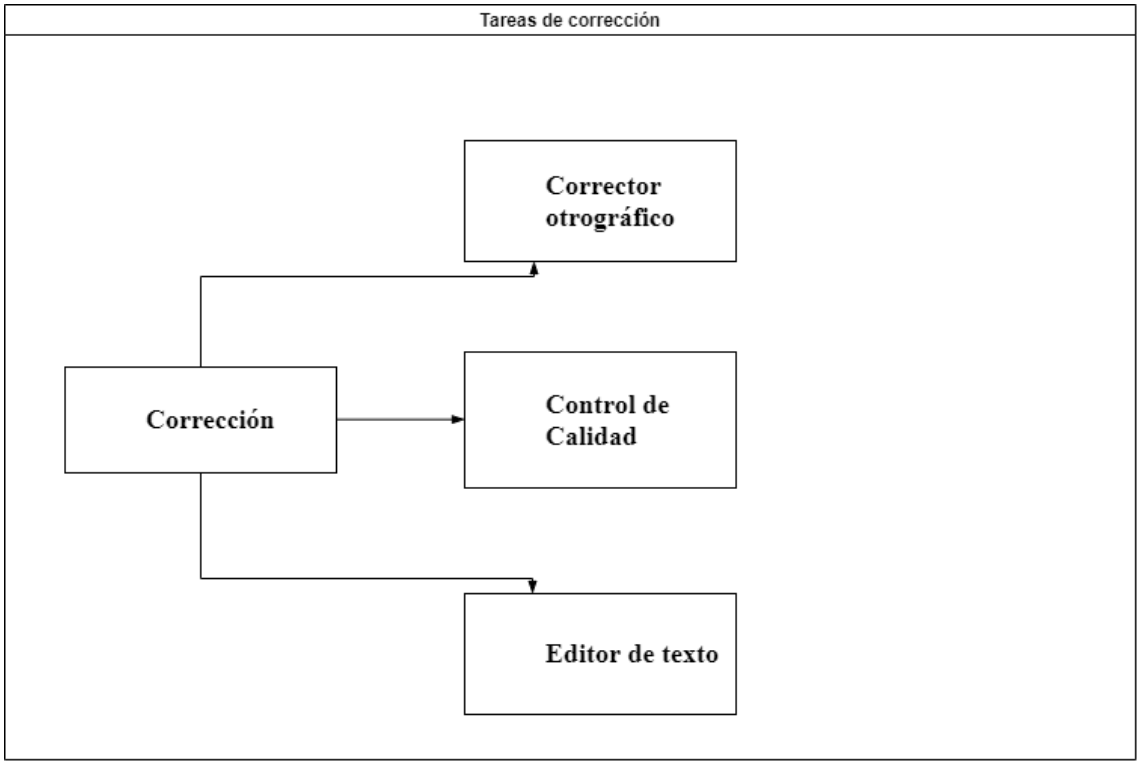

Figura 5: Diagrama de las funciones de corrección

Si el lector está familiarizado con la labor de la traducción y su industria, sabrá que un error se paga muy caro y dependiendo de la especialización del texto las consecuencias podrían ser hasta legales.

La recomendación general para empresas y agencias de traducción es que, aparte del mismo traductor, un tercero revise las traducciones para dar una visión crítica apartada del punto de vista del traductor. No es ningún secreto que en un considerable número de ocasiones la figura del revisor independiente no existe en los proyectos de traducción para ahorrar en costes y, aunque nos pese, si se comete un fallo todo el mundo apunta al traductor.

Para evitar que se dé esta situación y que quede clara la responsabilidad del trabajo de revisión que seguramente no te pagarían de igual manera, la solución es bien sencilla: no cometer fallos.

Quizás suene demasiado radical si tenemos en cuenta que el humano se equivoca constantemente, pero no tenemos por qué depender siempre de nuestra habilidad, sino que nosotros mismos hemos creado herramientas de corrección y control de calidad que facilitan mucho este trabajo.

El uso de correctores automáticos y de control de calidad está altamente recomendado para detectar errores que en muchas ocasiones pasamos por alto, como el archiconocido espaciado doble. Estos programas, aunque no son infalibles, están bastante perfeccionados hoy en día y creemos que no usarlos sería un error, y por ello queremos implementar las funciones de corrector ortotipográfico y control de calidad.

Además, para no tener que abrir otros programas y ahorrar tiempo sería adecuado incluir un editor de texto dentro del programa para hacer cambios rápidos. Esta parte del programa podría utilizarse para la posedición de la traducción automática de la que hemos hablado anteriormente. Sería interesante añadir a este editor de posedición ciertas herramientas como búsqueda y reemplazo, creación de macros, uso de expresiones regulares, etc.

Si bien las herramientas TAO actuales poseen ya correctores, como hemos comentado, no son infalibles, por lo que pasar un segundo filtro de corrección podría darnos una mayor seguridad.

Aun así, queremos dejar claro que, aunque estas herramientas son de gran ayuda, no pueden sustituir la revisión manual ni la labor del revisor, por lo que, aunque son un gran apoyo para traductores, no podemos eludir nuestra responsabilidad.

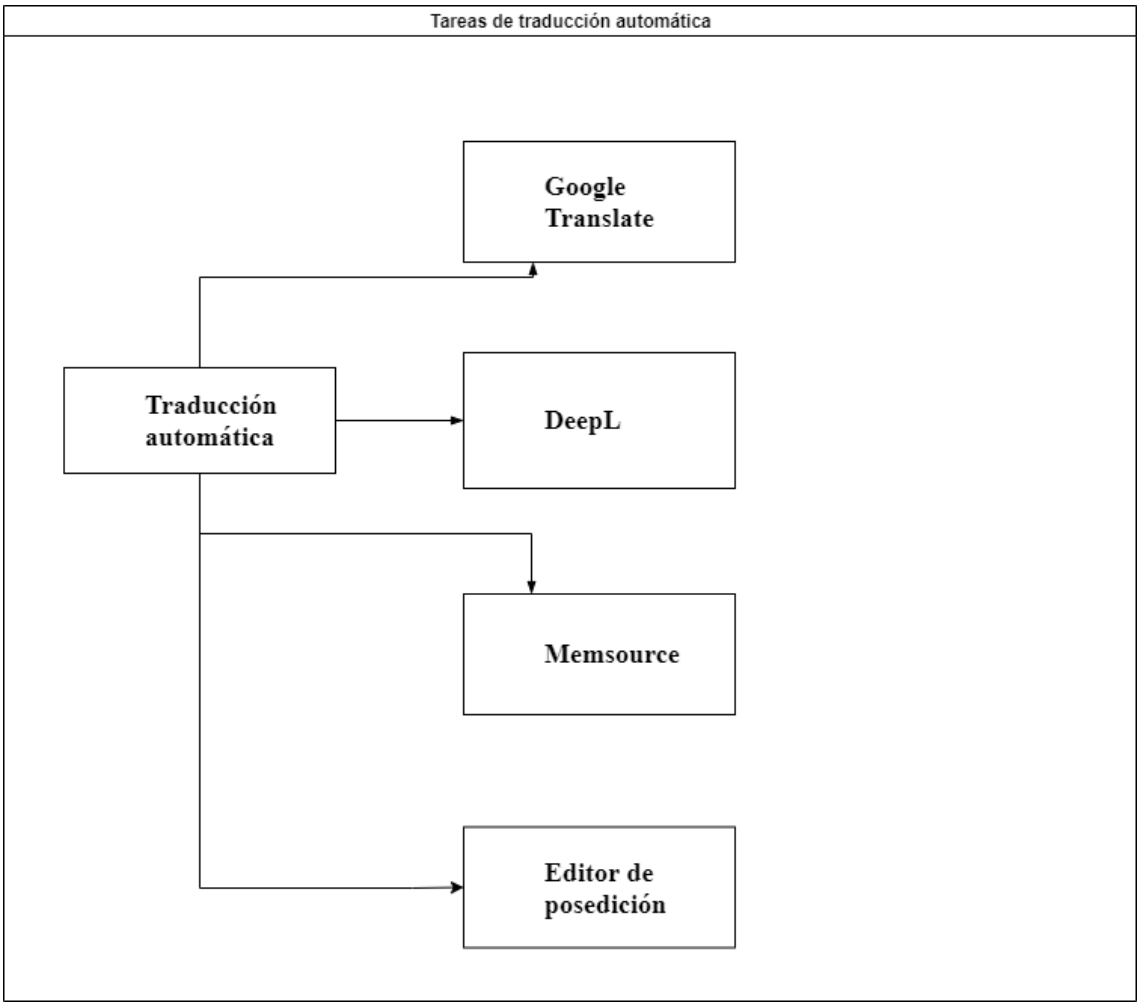

Figura 6: Diagrama de funciones de traducción automática

Gracias a las redes neuronales los programas de traducción automática funcionan cada día mejor y han surgido trabajos como la posedición, que permite tener textos traducidos más rápidamente y de una calidad variable dependiendo si la posedición es simple o completa.

La posedición genera buenos resultados en menos tiempo y es algo a lo que el traductor contrario a la traducción automática actual deberá hacerse a la idea en los próximos años.

Como con los avances de las redes neuronales aplicadas a la traducción se prevé que la calidad sea cada vez mayor y que el contexto no suponga un obstáculo tan grande, pensamos que sería coherente con nuestro discurso pensar en añadir traductores automáticos de textos que, además, combina muy bien con las tareas de corrección y control de calidad de las que hemos hablado.

Hoy en día, los traductores automáticos neuronales más conocidos y que pensamos que trabajan mejor son el de Google y el de DeepL. Google es el buscador de internet más utilizado en occidente y, como es de esperar, controla una cantidad ingente de datos, por lo que es bastante previsible que con tantísimo material para aprender su traductor neuronal no haga más que mejorar.

En cuanto a DeepL, hicimos una comparación con las traducciones de Google en la plataforma TAUS y nos pareció interesante que el resultado indicara que funciona igual e incluso ligeramente mejor que el traductor neuronal de Google.

Para acabar, quizás el usuario se pregunte por qué hemos añadido Memsource en este apartado si no es un traductor automático en sí, sino una herramienta TAO. Memsource, aunque efectivamente es una herramienta TAO, también posee un traductor automático que consideramos que funciona bastante bien, pero esta no es la única razón. La principal razón por la que queremos trabajar para intentar implementar el traductor automático de Memsource es porque existe una API para desarrolladores y pensamos que haciendo uso de Postman podríamos implementarlo. Así pues, la razón por la que la incluimos es porque pensamos que podría llevarse a cabo, es decir, por mero interés de investigación; aparte de, por supuesto, darle una utilidad al programa que ayude al usuario.

2.1. Identidad

Para poder hablar más fácilmente y que pudiera ser reconocido por gente que no está envuelta en el desarrollo de nuestro programa, decidimos dotar a nuestro programa de cierta identidad propia que lo hiciera destacar más allá de ser un producto de la investigación académica.

Para dotarlo de identidad planeamos mostrar un logo personal en la pantalla de bienvenida como veremos en el diseño, una página dentro de nuestro sitio web o un sitio web propio para el programa y, por último, un nombre propio.

2.2. Logo

Figura 12: Logo personal

Al diseñar nuestro logo pensamos en que debía estar relacionado con las letras y la traducción, así que representamos nuestro logo con la letra del silabario japonés あ (que corresponde a nuestro sonido “a”) enmarcado en una letra omega del griego.

Tenemos conocimientos de japonés y de griego antiguo, pero no elegimos estas letras únicamente porque las hayamos estudiado o simplemente nos guste, sino que hay una razón de fondo que aporta algún significado a nuestro logo.

En la tradición artística cristiana en Europa (principalmente en obras del medievo) puede observarse que a cada lado de algunas representaciones de Jesucristo aparece una letra alfa y una omega. La letra alfa es en el abecedario griego la primera y por ello representa el principio, mientras que la omega, siendo la última, representa el fin. Este significado fue tomado por la religión cristiana como acto de sincretismo para dar a entender un mensaje; que Dios es el principio y fin de todo.

En nuestro caso, lejos de querer dar un mensaje de omnipotencia, como es el caso del dios cristiano, queríamos recoger alguna referencia a los cimientos de nuestra cultura y dar a entender al usuario que se podrían realizar las tareas más importantes relacionadas con la gestión de proyectos con un programa que aúne estas tareas y agilizar el flujo de trabajo de principio a fin.

Una vez explicada la breve historia de nuestro logo, pasaremos a hablar del nombre de nuestro programa.

2.3. Nombre

Como último punto del diseño previo, nos gustaría mostrar el nombre de nuestro programa y el porqué de llamarlo así. Queríamos un nombre que llamara la atención, así que debía ser atractivo sonoramente, y también que pudiera reconocerse según las tareas que realizaba y el público al que está dirigido principalmente.

Después de barajar varios nombres decidimos bautizarlo finalmente como Dragos Manager. El nombre podría asemejarse a la raíz de la palabra "dragón", por lo que, si el usuario cree que proviene directamente de esta palabra, quizás le dé una sensación de fortaleza y cierta reminiscencia a la épica, y, aunque no son atributos que consideramos negativos, lo cierto es que tomamos está palabra como un acortamiento de la palabra del griego medieval δραγομάνος ("dragomanos"), que sería el equivalente a "dragomán" en castellano.

"Dragomán" es una palabra antigua que se utilizaba para nombrar el oficio de lo que hoy en día conocemos como traductor. Quizás el resultado final sea demasiado enrevesado para entenderlo a primera vista, pero con esta palabra conseguimos un nombre sonoramente potente y su raíz está relacionada con las labores del público al que está destinado el programa.

3. Preparaciones previas

Antes de comenzar con el diseño y la programación de nuestra herramienta debíamos preparar nuestro equipo para poder llevar a cabo todos los procesos que veremos más adelante.

Como ya hemos anunciado anteriormente, la programación se llevaría a cabo con Python 3, un lenguaje de programación que puede descargarse desde la página oficial de Python. En nuestro caso hemos utilizado la última versión disponible cuando comenzamos el proyecto, es decir, Python 3.8.2.

En la sección de descargas en la parte inferior de la página podemos observar diferentes versiones dependiendo de la arquitectura de nuestro equipo y la marca de nuestro sistema operativo. Como nuestro equipo es un Windows 10 con una arquitectura de 64 bits, descargaremos la opción "Windows x86-64". Si la arquitectura de nuestro sistema fuera de 32 bits descargaríamos la opción "Windows x86".

Al iniciar el ejecutable nos aparecerán dos opciones en la parte inferior del cuadro de diálogo, una que aparece marcada de manera predeterminada y la otra desmarcada. La primera es la opción de instalar el lanzador para todos los usuarios, que podemos dejar marcada, y por otro lado aparece la opción de añadir Python 3.8 al PATH.

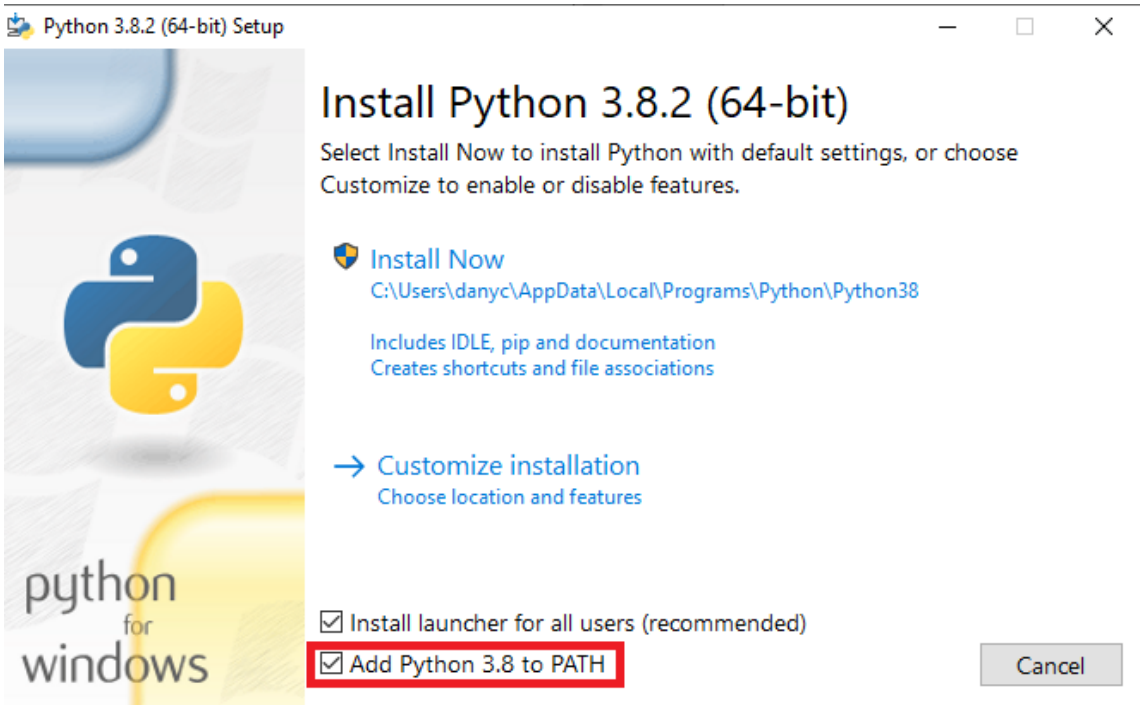

Figura 7: Asistente de instalación de Python 3.8.2

Esta segunda opción aparece desmarcada, pero es una opción de gran utilidad, ya que nos permite ejecutar python.exe escribiendo únicamente "python", mientras que, de otra manera, tendríamos que especificar la ruta en la que se encuentra el ejecutable del programa en el símbolo del sistema, así que esta opción nos ahorrará tiempo.

A continuación, podemos continuar con la instalación o seleccionar qué queremos instalar con una instalación personalizada. Aquí nos aparecen opciones que consideramos que deberían quedar marcadas a excepción de la documentación, que podría quitarse si ya se conoce cómo funciona el programa.

Entre las opciones opcionales que se nos muestran aparece “pip”, que es de bastante utilidad porque nos permite instalar y, posteriormente, importar en nuestro programa paquetes de Python. Los paquetes de Python son de gran utilidad, si no imprescindibles, porque, dependiendo del paquete, nos ofrecen definiciones específicas.

Entre los paquetes de Python más útiles que hemos utilizado hasta ahora se encuentran:

- NumPy: paquete con funciones matemáticas de alto nivel.
- Matplotlib: creado como extensión de NumPy. Sirve para generar gráficos.
- Pandas: creado también como extensión de NumPy. Biblioteca para generar tablas de contenidos.

A lo largo de este trabajo de fin de máster mostraremos otros paquetes que nos ayudarán en gran medida en la programación.

4. Entorno de desarrollo integrado (IDE)

Los entornos de desarrollo integrado o IDE (del inglés *Integrated Development Environment*) son un tipo de aplicación dedicada a los programadores o desarrolladores que reúne en una única herramienta varias utilidades para facilitarles el trabajo y automatizar procesos.

Entre las utilidades y ventajas que nos pueden ofrecer los entornos de desarrollo integrado podemos encontrar:

- Búsqueda y reemplazo de código.
- Marcar la sintaxis con diferentes colores para hacerlo más intuitivo.
- *Debugger*.
- Autocompletar códigos.
- Refactorizar código.

Existe un gran número de entornos de desarrollo para Python y, dependiendo de las funcionalidades que nos ofrezcan respecto a otros o, simplemente, basándonos en usar el que nos resulte más cómodo, utilizaremos unos u otros.

Como existe una gran cantidad de entornos y explicar las características de cada uno no es una prioridad para este proyecto, ofrecemos al lector un enlace de la Wikipedia en el que se muestra mediante tablas y de manera bastante representativa las características de muchos de ellos.

Entre los entornos de desarrollo para compatibles con Python podemos encontrar Microsoft Visual Studio Code, Eclipse + Pydev, ATOM, IDLE (entorno de desarrollo predeterminado de Python), Geany, Sublime Text o Pycharm entre otros.

En nuestro caso ya habíamos utilizado anteriormente Visual Studio Code y Geany, pero en este proyecto, por recomendación de un programador especializado en Big Data, probaremos a desarrollar nuestra aplicación en Pycharm, de la que hablaremos más en detalle en el siguiente punto.

4.1. Pycharm

Pycharm es un entorno de desarrollo integrado multiplataforma de la empresa de desarrollo de software JetBrains y se utiliza para la programación y el desarrollo web principalmente. En la actualidad existen tres versiones:

- Versión Community: bajo licencia libre permisiva (Apache). Posee prácticamente todas las funciones necesarias para programar.
- Versión para estudiantes: también de código abierto. Acceso a todo el paquete de JetBrains de manera gratuita.
- Versión Professional: versión de pago de Pycharm, con un precio de 89 euros al año para particulares. Además de las herramientas de programación que nos ofrece la versión Community, incluye herramientas científicas, herramientas de desarrollo web, marcos de trabajo web para Python, perfilador Python, capacidades de desarrollo remoto y soporte para bases de datos y SQL.

Como hemos comentado en el apartado de los entornos de desarrollo, nuestra opción a la hora de desarrollar nuestro programa será Pycharm. En primer lugar, porque nos lo recomendaron profesionales de la programación que trabajan con Python, y, en segundo lugar, porque, de acuerdo con sus características, parece ser uno de los mejores programas en hoy en día para la programación con Python.

Entre las características que pueden encontrarse en la página de Pycharm, podemos destacar las siguientes características de la versión Community que nos serán de utilidad para nuestro proyecto:

- Editor de código inteligente: funcionalidad para autocompletar código partiendo del lenguaje, detección de errores y corrección del código sobre la marcha. De gran utilidad para ahorrar tiempo a la hora de introducir código predecible según el contexto dado al programa. En muchas ocasiones pequeños fallos a la hora de escribir código hace que el programa no funcione o ignore un método, por lo que la detección de errores ahorra muchísimo tiempo, ya que ir buscando manualmente por todo el código qué podría estar mal es tedioso y completamente agotador (especialmente en proyectos grandes).
- Depuración visual y ejecutor de pruebas: la depuración, también conocida como *debugging*, consiste en localizar y solventar los problemas que contenga el código. Con Pycharm esta depuración es más sencilla porque proporciona una imagen más visual al proceso. En cuanto a la ejecución de pruebas, también conocido como *testing* o testeo, nos permite comprobar que un programa se ejecuta correctamente, y esta acción se puede realizar directamente desde el IDE.
- Navegación y refactorización: Pycharm permite la navegación rápida entre archivos de vista y plantilla o desde símbolos de una plantilla al propio código Python de ese símbolo. La refactorización o limpieza de código consiste en la modificación del código fuente del programa sin cambiar la función que realiza. La función de la refactorización no es la de corregir errores, esa es la del *debugging* como hemos comentado anteriormente, su función es la de hacer el código más legible y mejorar su comprensión.
- Inspecciones de código: búsqueda y corrección rápida de errores en el código.
- Compatibilidad con control de versiones: el control de versiones o VCS (del inglés *Version Control System*), consiste en la administración y gestión de los cambios realizados en el código fuente del programa.

Para comprobar que el código que íbamos escribiendo funcionaba teníamos que ejecutar nuestro script de Python, por lo que podíamos hacerlo desde el símbolo del sistema o desde el propio Pycharm. En este trabajo comprobaremos que aparecerá la mayoría de las veces el símbolo del sistema porque creemos que los fallos aparecen bastante bien detallados, pero durante nuestro trabajo hemos estado utilizando tanto el intérprete de Python con Pycharm como el símbolo del sistema.

Para que funcione la opción “Run” de Pycharm hay que especificarle al programa el intérprete de Python que queremos utilizar. Si queremos configurar el intérprete debemos ir a File>Settings>Project>Project Interpreter y aquí elegiremos la versión de Python que usaremos y que tendrá instalados todas las bibliotecas que hemos instalado previamente. En nuestro caso utilizamos Python 3.8.2 y como podemos ver en la siguiente imagen, nos carga todas las bibliotecas:

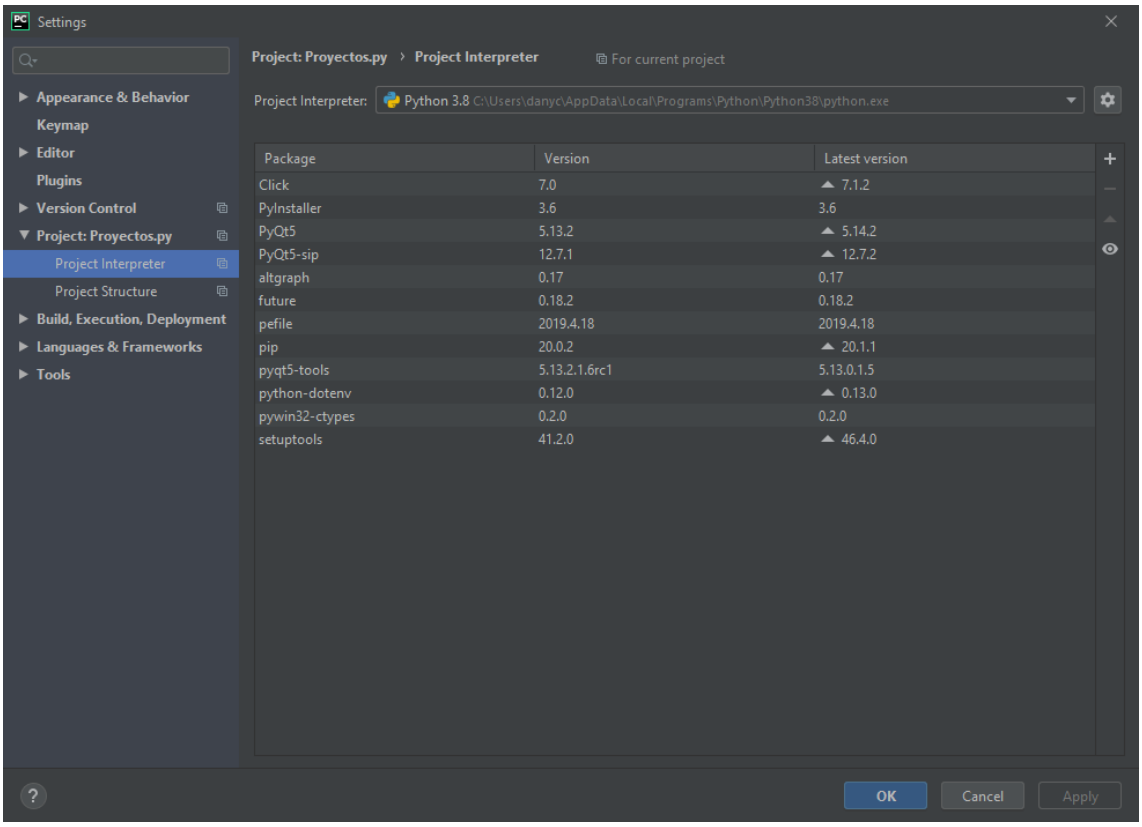

Figura 8: Establecer un intérprete de Python en Pycharm

4.2. ¿Por qué usar un IDE?

Hasta ahora todo lo que se ha expuesto sobre los IDE en cuanto a la programación han sido ventajas, pero ¿realmente son todo ventajas?

Los IDE, como hemos comentado anteriormente, nos ayudan a aumentar la productividad en gran medida gracias a funciones como el autocompletado de código, pero también pueden ayudarnos en exceso.

Para los programadores con experiencia esta ayuda no supone ningún coste, solo un aumento de la productividad, pero para los estudiantes, o incluso programadores de nivel intermedio, los IDE suponen una amenaza para su aprendizaje.

En primer lugar, si no se conoce bien el lenguaje de programación con el que se está trabajando se acabará abandonando u olvidando el estudio de la sintaxis, y con la función de autocompletado esto es más fácil de que ocurra.

En segundo lugar, el hecho de que muchos IDE te digan dónde está el problema puede ser incluso beneficioso para el aprendizaje, pero si te muestran la solución directamente o si se corrigen automáticamente sin ni siquiera poder haber visto y evaluado el error puede suponer, por razones obvias, un obstáculo en el aprendizaje.

Para evitar que esto ocurra y que estos programas puedan llegar a ser perjudiciales en el proceso de aprendizaje se recomienda utilizar editores de texto como Notepad++ en vez de los IDE.

Después de estas dos visiones que hemos ofrecido acerca de los IDE, ¿por qué vamos a utilizar un IDE de todas maneras?

La primera razón por la que se me ocurrió utilizar Pycharm es porque no lo habíamos utilizado anteriormente. Desde nuestro punto de vista, todo programa merece

un tiempo de estudio y aprendizaje hasta que se puede utilizar de manera habilidosa y, como ya hemos dicho, este trabajo nos servirá para explorar una nueva área de conocimiento distinta de la traducción y las letras puras. Así pues, la primera de las razones es el aprendizaje sobre nuevas herramientas y la búsqueda de conocimiento nuevo.

La segunda razón es la productividad. El tiempo que tenemos es limitado y estos programas pueden ayudarnos a lograr el objetivo deseado para las fechas previstas.

La tercera razón es la de corrección de errores. Que un programa te diga cuáles son los errores potenciales, además de la visión negativa que hemos dado antes, debería poder tomarse como una ayuda para comprender nuestros fallos y nosotros los plasmaremos en este trabajo como errores habituales para aprender de ellos y mejorar.

5. Diseño de la interfaz gráfica

Por lo general, el primer paso a la hora de crear un programa con interfaz gráfica de usuario o GUI (siglas del inglés *graphical user interface*) es crear esta misma antes de ponerse a programar.

Así pues, comenzamos a indagar e investigar acerca de los posibles programas para diseñar la interfaz de nuestro programa, pero nos topamos con un problema, y es que, con las tecnologías de la informática avanzando a un ritmo desproporcionado, todo lo escrito acerca de programas o desarrollo hace un tiempo es bastante susceptible de quedar obsoleto con el desarrollo de distintas versiones de software.

En el momento de redacción de este TFM, para la última versión de Python, la 3.8.2, hemos podido encontrar dos opciones diferentes para la creación de interfaces gráficas que no están obsoletas y funcionan bastante bien: Tkinter y PyQt5.

5.1. Diseño previo

Antes de comenzar a trabajar con el programa de diseño de la interfaz gráfica, hicimos con Photoshop algunos bocetos para pasar a tener una imagen real de la forma que tendría nuestra interfaz y saber así cuántas ventanas debíamos trabajar.

Para empezar, pensamos en poner una ventana de inicio que mostrara nuestro logo, una marca personal que hiciera reconocible nuestro programa que ya habíamos diseñado anteriormente y que también está presente en el inicio de nuestra página web.

A parte de nuestro logo, queríamos que hubiera tres botones; uno que da paso a la siguiente ventana, en la que aparecerían las tareas que realiza el programa; otro que redirija al usuario a un manual online; y otro que redirija a nuestra página web, dónde podrá consultar más información sobre la versión del programa y las notas de las actualizaciones. La imagen mental que teníamos sobre esta ventana sería algo aproximado a lo siguiente:

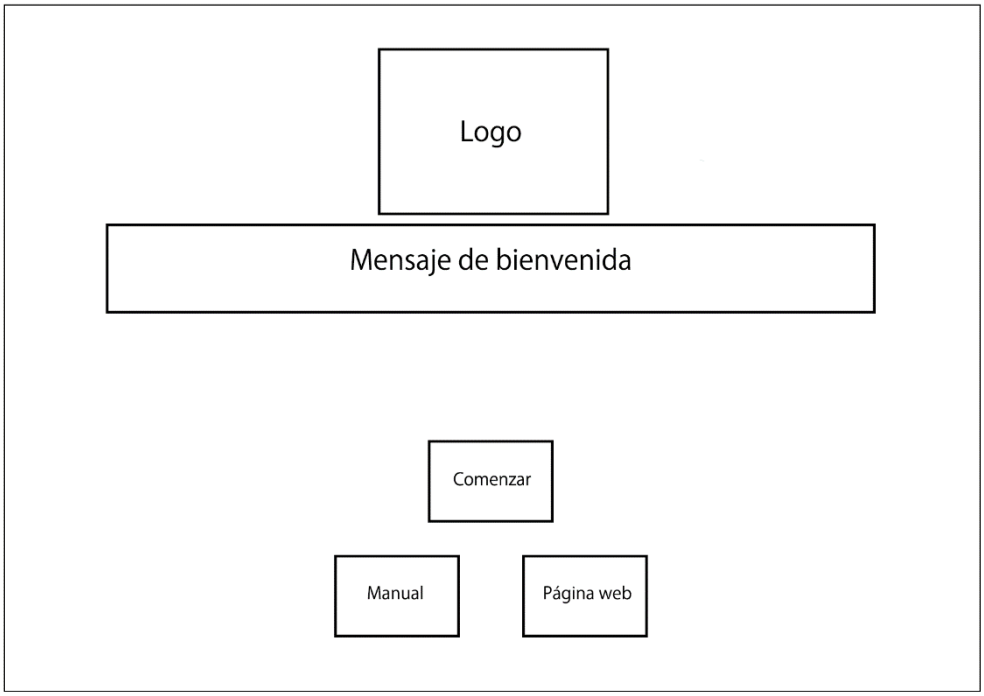

Figura 9: Boceto de la ventana de inicio

La siguiente ventana, a la que llegas después de hacer clic el botón que dice “comenzar”, es la ventana de las tareas que hará el programa y pensamos que podría tener la siguiente estructura:

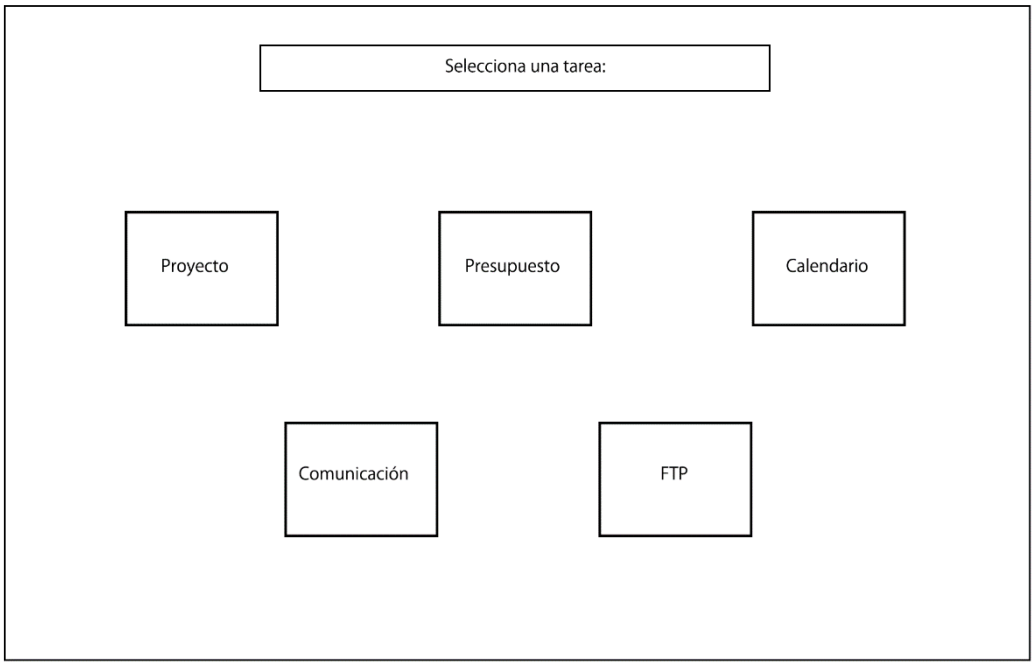

Figura 10: Boceto de la ventana de tareas

Los espacios de las tareas estarán ocupados por imágenes intuitivas de cada tarea y también aparecerá el nombre de la tarea debajo de la imagen, que podrían ser un *label* o un botón con una hoja de estilo.

Al clicar en la imagen de proyecto se abrirá la ventana de proyectos que incluiría las tareas relativas al proyecto que queremos que realice nuestro programa. El diseño previo que realizamos basada en la imagen mental que teníamos de esta ventana se vería así:

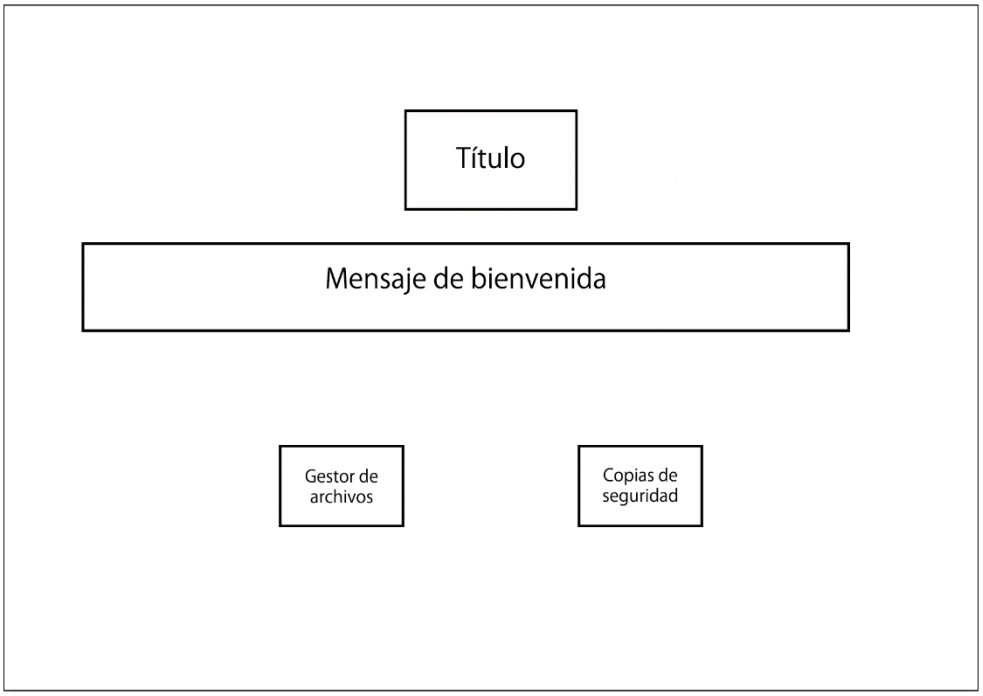

Figura 11: Boceto de la ventana de proyectos

5.2. PyQt5

PyQt5 es una biblioteca de Python que permite la creación de programas con interfaces gráficas para usuarios. Junto a Tkinter, Pyqt5 es de las bibliotecas más utilizadas para crear interfaces, y el hecho de poder encontrar en internet una mayor cantidad de información nos hizo decidirnos por esta biblioteca. La forma de crear interfaces con el programa incorporado Qt Designer hace muy sencillo crear una interfaz porque permite colocar *widgets* simplemente arrastrándolos a la ventana.

Aunque en un primer momento pensamos en utilizar Tkinter para hacer nuestra interfaz, tras revisar ejemplos de programadores en vídeos, las aplicaciones que desarrollaban con Tkinter, aunque eran sencillas, parecían visualmente algo antiguas.

Podríamos haber investigado cómo crear hojas de estilos y cómo aplicarlas en nuestro código, pero el hecho de que Qt Designer nos ofreciera un programa para hacerlo directamente con un diseño medianamente moderno, nos ahorró un tiempo muy preciado que pudimos usar más adelante en otras fases del proyecto.

5.2.1. Instalación

Para instalar y comenzar a usar PyQt5, deberemos tener el pip para instalar paquetes y que deberíamos tener al instalar Python a no ser que lo desmarcáramos en la opción de instalación personalizada.

Como comentamos con brevedad en puntos anteriores, pip es un gestor de paquetes que se utiliza para gestionar e instalar paquetes de software de Python.

En nuestro caso, aunque teníamos instalado el pip, lo teníamos desactualizado. Para conseguir la última versión del pip podemos hacerlo accediendo al símbolo del sistema y teclear:

-python -m pip install --upgrade pip

En la siguiente imagen podemos ver cómo queda actualizado el pip:

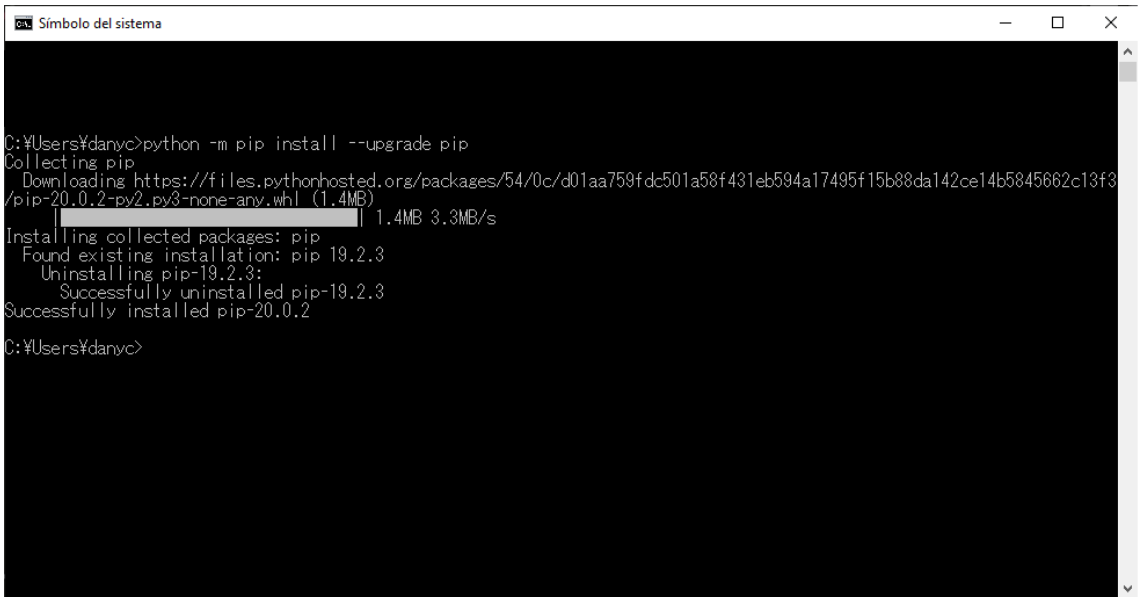

Figura 2: Actualización del pip

Una vez actualizado el pip procedimos a instalar PyQt5 con la siguiente línea en el símbolo del sistema:

pip install pyqt5

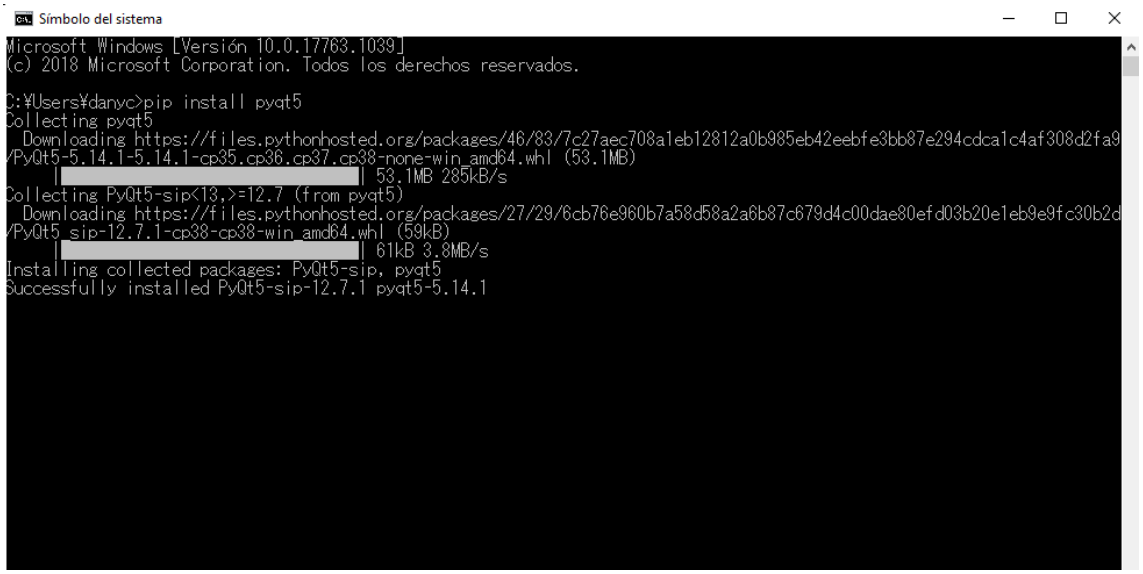

Figura 13: Instalación de PyQt5

Tras instalar PyQt5 nos quedó por realizar un último paso, instalar PyQt5-tools. Este paquete incluye una serie de programas entre los que se encuentra Qt Designer, un programa que veremos en el punto siguiente y que sirve para la creación visual de una

interfaz gráfica de usuario que utilizaremos en nuestra aplicación. Para instalar PyQt5-tools deberemos escribir la siguiente línea en el símbolo del sistema:

```
pip install pyqt5-tools
```

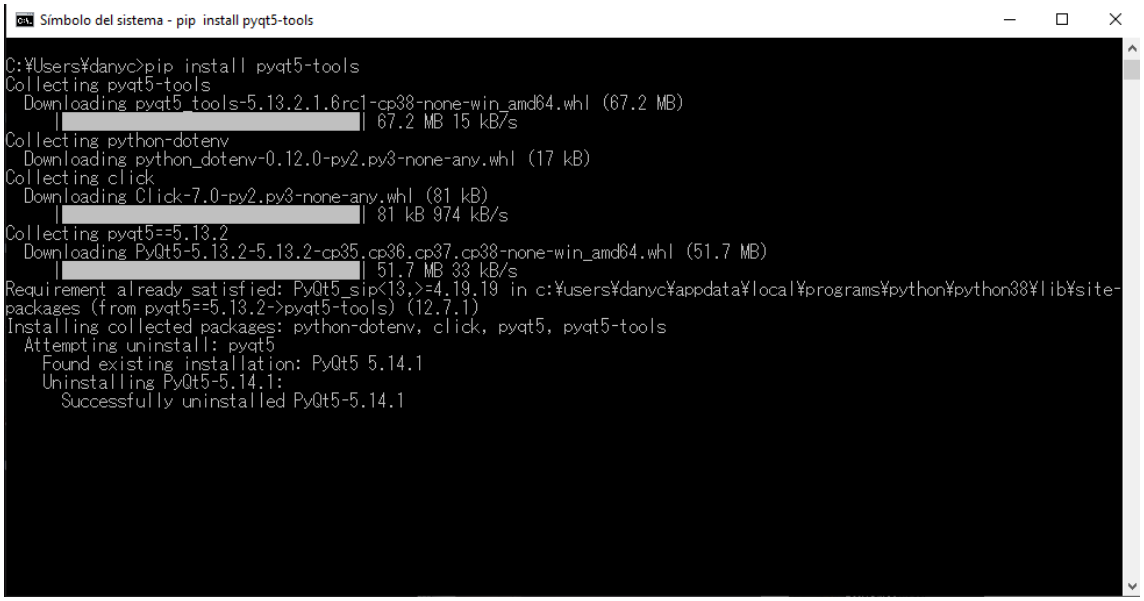

Figura 14: Instalación de PyQt5-tools

5.2.2. Qt Designer

Al instalar con el símbolo del sistema PyQt5-tools como hemos mostrado en el apartado anterior, se nos instala el programa Qt Designer, que será el programa que utilizaremos para crear visualmente nuestra interfaz.

Las interfaces pueden crearse escribiendo código manualmente, pero gracias a este programa podemos diseñarlo visualmente y el resultado será que el programa ha escrito el código por nosotros según cómo hemos diseñado la interfaz.

Para poder acceder a este programa primero debemos buscar la ruta en la que tenemos Python instalado. Para encontrarlo podemos buscarlo manualmente en nuestro disco C o buscarlo en el explorador de archivos de Windows. Cuando lo encontremos debemos seguir la siguiente ruta:

```
Python\Python38\Lib\site-packages\pyqt5_tools\Qt\bin
```

En nuestro caso, Qt Designer aparecía en esta ruta, pero, si no apareciera en esta ruta, comprobamos que a algunos usuarios les aparecía el programa en la siguiente ruta:

```
Python\Python38\Lib\site-packages\pyqt5_tools
```

Una vez localizado el programa creamos un acceso directo al escritorio para no tener que hacer este proceso de búsqueda cada vez que quisiéramos utilizar Qt Designer.

Para nosotros Qt Designer era un programa completamente nuevo cuando lo empezamos a usar por lo que, si el lector tuviera más preguntas acerca de los procesos

que narraremos a continuación para crear nuestra interfaz, le recomendamos consultar el manual de Qt Designer que nos ayudó bastante a nosotros en más de una ocasión.

5.2.3. Diseño con Qt Designer

Así pues, una vez hechas las preparaciones previas para tener a mano Qt Designer, y teniendo ya en mente una idea previa de cómo queríamos que luciera nuestro programa gracias a nuestros bocetos, comenzamos a diseñar nuestra interfaz por la ventana de inicio:

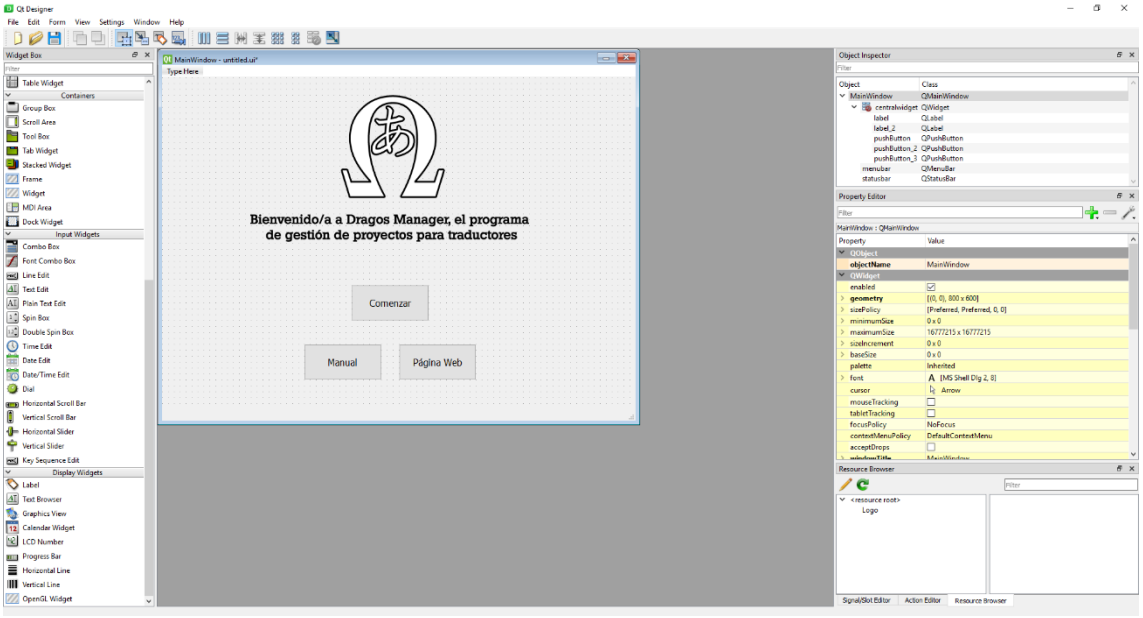

Figura 15: Diseño de ventana de inicio en Qt Designer

Para diseñar esta ventana lo que hicimos fue crear una nueva ventana principal, la redimensionamos para que tuviera el tamaño que nosotros quisiéramos y añadimos dos *labels* y tres *push buttons*. Los *labels* los usaremos para el logo y el mensaje de bienvenida y con los *push buttons* añadiremos las opciones de “comenzar”, “manual” y “página web” como planeamos anteriormente en el diseño previo y como puede ya observarse en la figura anterior.

A continuación, mostramos la ventana principal con las tareas:

Figura 16: Diseño de ventana principal de tareas

Para añadir las imágenes que hemos utilizado como el logo o los iconos de las tareas tuvimos que crear archivos de recursos. En un principio no sabíamos muy bien a qué se refería el programa con archivos de recursos, pero tras indagar un poco acerca del tema descubrimos que para obtenerlos simplemente hay que crear un nuevo archivo de texto plano y cambiarle la extensión a .qrc.

Una vez tuvimos los archivos de recursos fuimos jugando con las dimensiones en Photoshop y conseguimos guardar en un tamaño adecuado para nuestra interfaz los iconos que incluimos en el programa. Para agregar recursos al archivo .qrc deberemos crear un *label*, hacer clic derecho y acceder a "Change rich text…". En este punto veremos cómo se nos abre una ventana en la que hay un icono de insertar imagen, si le clicamos se nos abrirá la pantalla de recursos y aquí podemos abrir el archivo de recursos y editar sus contenidos:

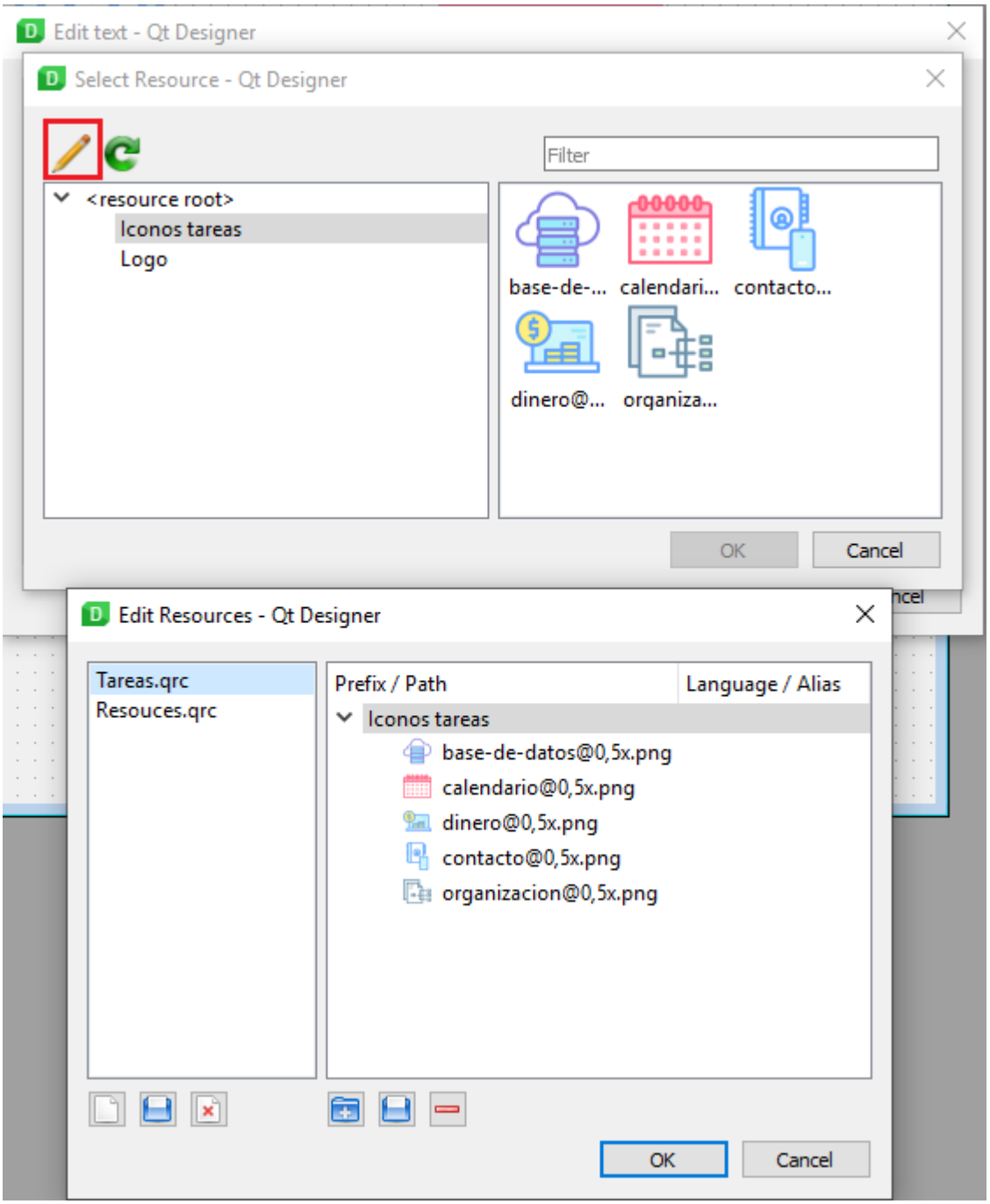

Figura 17: Gestión de los recursos en Qt Designer

Siguiendo con los principios anteriores creamos un nuevo archivo de recursos con imágenes intuitivas para crear nuestra ventana de proyecto basándonos en el boceto que dibujamos anteriormente:

Figura 18: Diseño de ventana de proyectos

Al clicar en estos dos iconos el programa abrirá una nueva ventana para escribir el nombre de la carpeta. La opción de gestor de archivos nos permitiría crear una serie de carpetas y subcarpetas donde almacenar los archivos que se utilizarán en el proyecto. Con la opción de creación de copias de seguridad podremos crear copias de los archivos conforme vayamos avanzando en nuestro proyecto para evitar perder los datos en caso de incidentes comunes como es que se apague nuestro equipo o se caigan los servidores. Estas dos opciones podrían combinar bien con otras de las tareas que queremos ofrecer con nuestro programa, que es la conexión con servidores FTP para compartir archivos, por lo que estudiaremos esta opción.

En la tarea de contacto queremos facilitar a nuestros usuarios la comunicación entre clientes y proveedores. Como podemos ver en la siguiente imagen, ofrecemos las opciones de clientes de correo más utilizadas y una cuarta opción para elegir un correo alternativo:

Figura 19: Diseño de la ventana de contacto

Nuestra primera idea es conectar los iconos de los clientes de correo con el navegador predeterminado del usuario y que se abriera su correo directamente en internet o en las aplicaciones del equipo. En cuanto al último, pensábamos en crear otra ventana para introducir el correo del remitente, el del destinatario y el cuerpo del mensaje para enviarlo si el usuario tiene un correo diferente a los que aparecen en esta ventana.

La siguiente ventana de tarea que diseñamos fue la del calendario. Algo que nos gustaría apuntar es que Qt Designer tiene un *widget* que es un calendario en sí, así que aprovechamos la utilidad de este programa para añadirlo al nuestro.

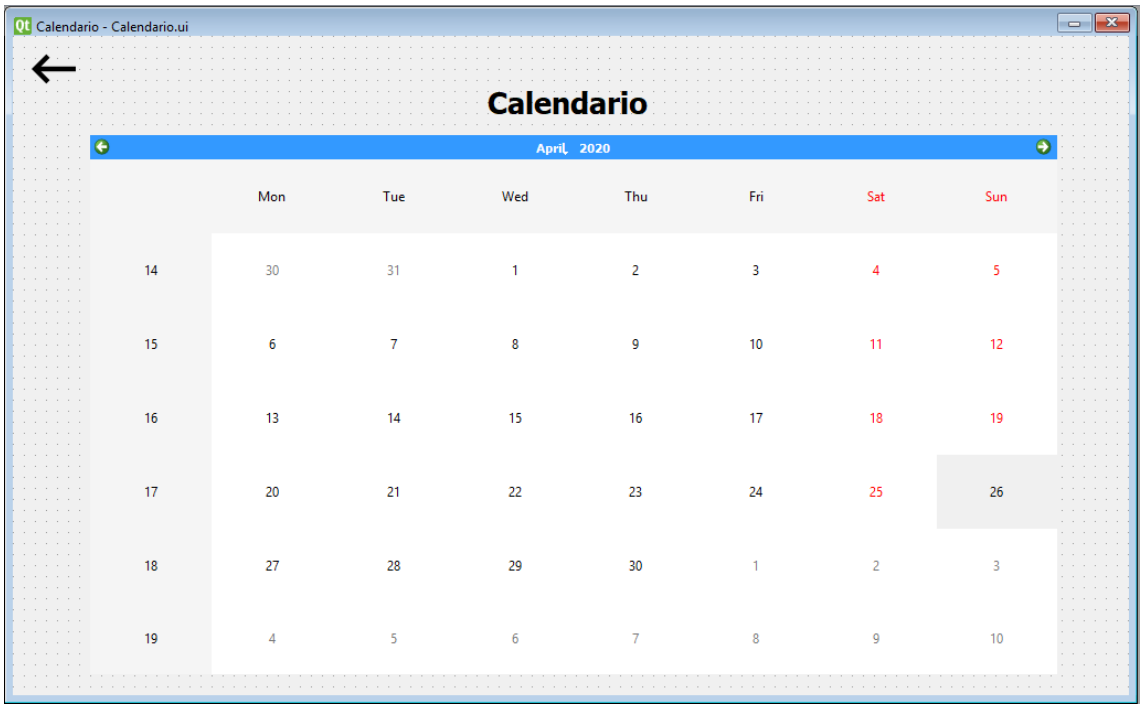

Figura 20: Diseño de la ventana de calendario

En principio esta ventana nos ofrece la utilidad que buscábamos en un principio pero estamos estudiando cómo añadir ciertas opciones que podrían ser de mucha utilidad para el usuario.

La idea de añadir un sistema de notificaciones, marcado de los días de entrega de proyectos con un color según la urgencia de los proyectos, y poder sincronizar este calendario con los calendarios de otras cuentas como para marcarlos como eventos en Windows, Google, Outlook o cualquier otro calendario que utilice el usuario.

Aunque son muchas tareas posibles, investigar cómo hacerlo y llevarlo a cabo nos llevaría demasiado tiempo, por lo que, como hemos dicho ya anteriormente, estudiaremos más adelante cómo implementarlo, pero no será a corto plazo.

Una de las tareas que creemos que pueden ser de mayor utilidad de este programa para sus usuarios es la de crear presupuestos. Al diseñar la siguiente ventana pensamos en los casos que se dan cada vez más en estos días, y es que muchos clientes ofrecen al traductor una memoria de traducción y una tabla de descuentos aplicable según las palabras contenidas en dicha memoria. Con nuestra tarea de presupuestos podemos obtener de manera rápida resultados con descuentos que nos muestran la cantidad facturable al cliente.

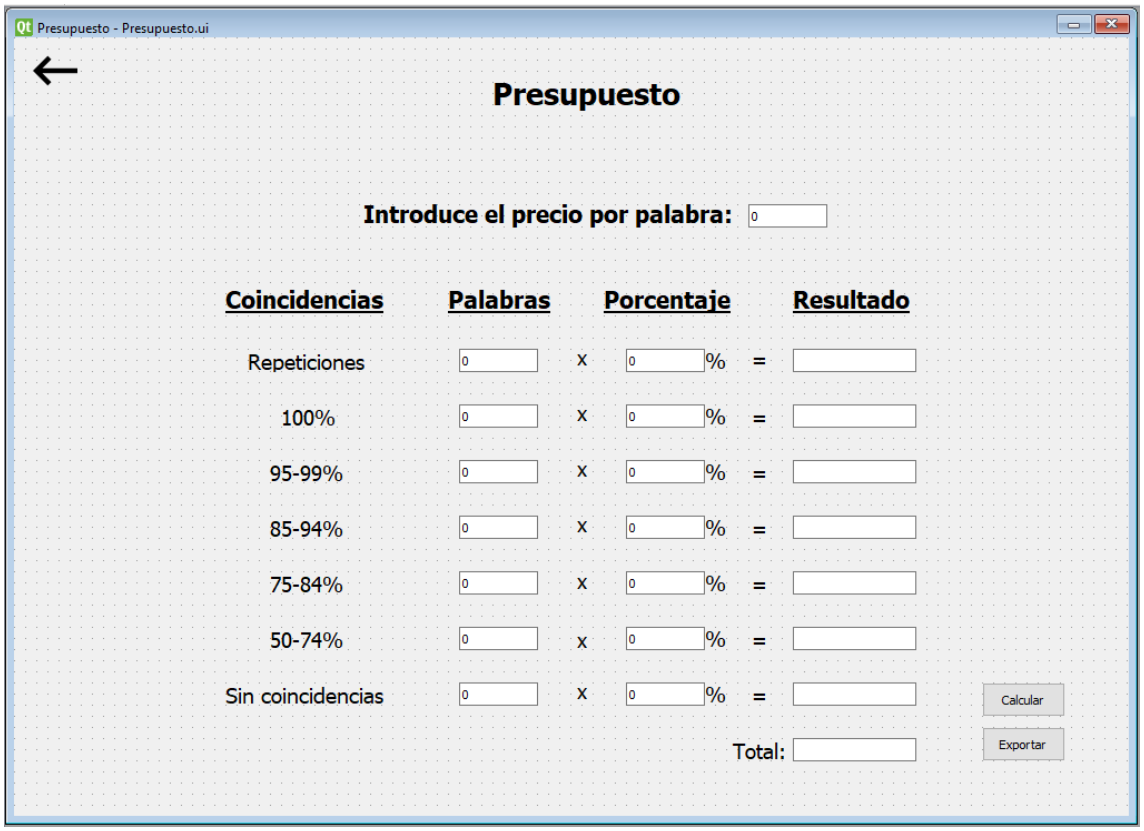

Figura 21: Diseño de la ventana de presupuesto

Para el diseño de esta ventana utilizamos principalmente *widgets* de QLabel y de QLineEdit. Las QLabel nos sirvieron principalmente para escribir los nombres de las columnas y los que le hemos dado a las coincidencias, y para los símbolos matemáticos. Una utilidad del programa que nos ayudó mucho para centrar los símbolos matemáticos que no quedaban bien alineados fue la opción de mover los objetos disminuyendo o aumentado el número de X o Y en el editor de propiedades.

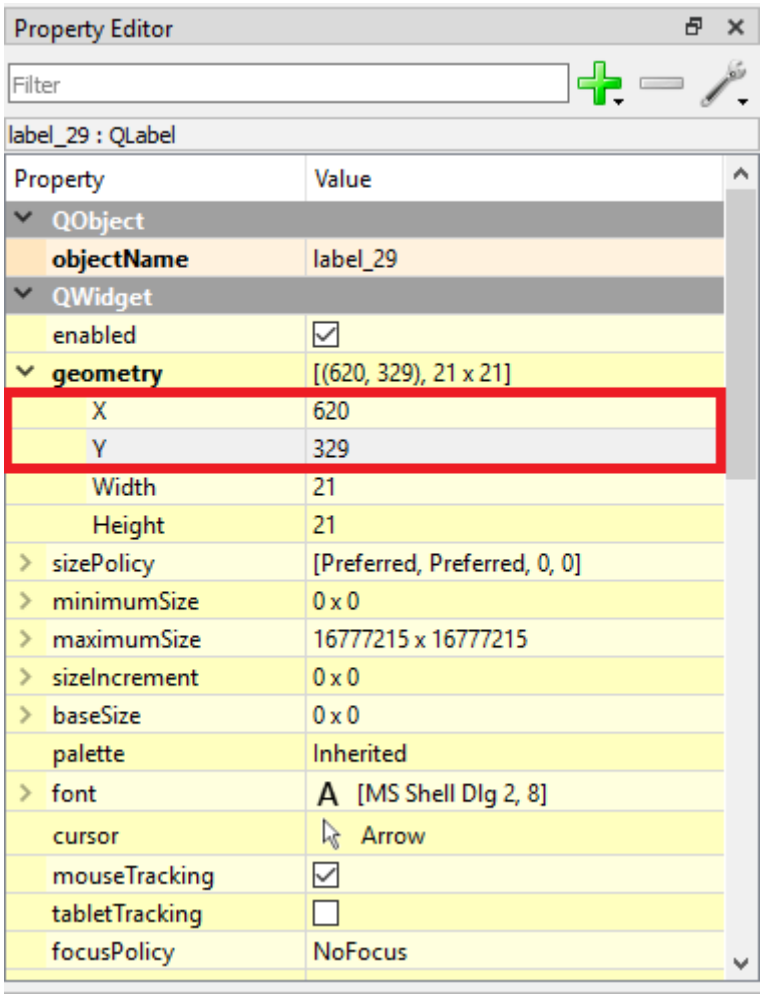

Figura 22: Centrado de objetos modificando los ejes X e Y.

Como en las demás ventanas debíamos darles utilidad a algunos *widgets*, dentro del código de Python necesitábamos cambiar el nombre de estos objetos dentro del programa para saber cuál era cada uno y asignarle una función a cada uno. En este caso, aparte de al botón para volver atrás, necesitábamos asignarle funciones principalmente a los QLineEdit, por lo que les cambiamos el nombre como vemos en la siguiente figura.

cien_palabra	QLineEdit
cien_porcentaje	QLineEdit
cien_resultado	QLineEdit
cincuenta_palabra	QLineEdit
cincuenta_porcentaje	QLineEdit
cincuenta_resultado	QLineEdit
ncinco_palabra	QLineEdit
ncinco_porcentaje	QLineEdit
ncinco_resultado	QLineEdit
no_coin_palabra	QLineEdit
no_coin_porcentaje	QLineEdit
no_coin_resultado	QLineEdit
ocinco_palabra	QLineEdit
ocinco_porcentaje	QLineEdit
ocinco_resultado	QLineEdit
pushButton	QPushButton
rep_palabra	QLineEdit
rep_porcentaje	QLineEdit
rep_resultado	QLineEdit
scinco_palabra	QLineEdit
scinco_porcentaje	QLineEdit
scinco_resultado	QLineEdit
total	QLineEdit

Figura 23: Nombres de los *widgets* de la ventana de presupuesto

5.3. Conversión a formato Python

Para comprobar que el diseño del programa funciona y se ve adecuadamente, deberemos ejecutarlo en el símbolo del sistema, pero podemos ver que los archivos generados por Qt Designer están en formato .ui, por ello deberemos pasar nuestras interfaces a formato .py.

Para lograrlo podemos abrir el símbolo del sistema, situarnos en la carpeta en la que estén nuestras interfaces y escribir el siguiente comando:

pyuic5 -x [nombre].ui -o [nombre].py

En la siguiente captura puede verse cómo escribiendo estos comandos pudimos conseguir estos dos archivos de interfaz en formato Python:

Figura 24: Acceso a la carpeta y conversión de archivos .ui a .py

Una vez tuvimos los dos archivos en formato .py procedimos a comprobar si funcionaban y nos surgió un programa al intentar ejecutarlo. El símbolo del sistema nos insistía en que era un problema de módulo en cuanto a nuestros archivos de recursos.

Al comprobar en entradas del foro sobre programación Stack Overflow acerca de este problema, nos dimos cuenta de que los archivos de recursos también debían pasarse a formato Python para que funcionaran con la interfaz.

Así pues, utilizamos un comando de nuevo para convertirlos a formato Python:

```
pyrcc5 Resources.qrc -o Resources_rc.py
pyrcc5 Tareas.qrc -o Tareas_rc.py
```

Ya convertidos los archivos a formato .py, volvimos a ejecutar nuestra interfaz con el comando "python Inicio.py", y esta vez funcionaba perfectamente.

5.4. Creación de las conexiones entre interfaces

Una vez creadas todas las interfaces, modificamos los códigos fuente para conectarlos entre sí y crear una navegación eficaz para el usuario.

Como en el punto anterior hemos tomado los archivos .ui de nuestra interfaz y los hemos convertido a formato .py, ahora podemos trabajarlos con PyCharm, el entorno de desarrollo que vimos en los primeros puntos de este trabajo de investigación.

Antes de explicar cómo conectamos los diferentes objetos de la interfaz, cabe destacar que los nombres de los objetos que aparecerán en las descripciones pueden cambiarse de manera manual en los archivos .py o directamente desde Qt Designer como podemos observar en la siguiente figura:

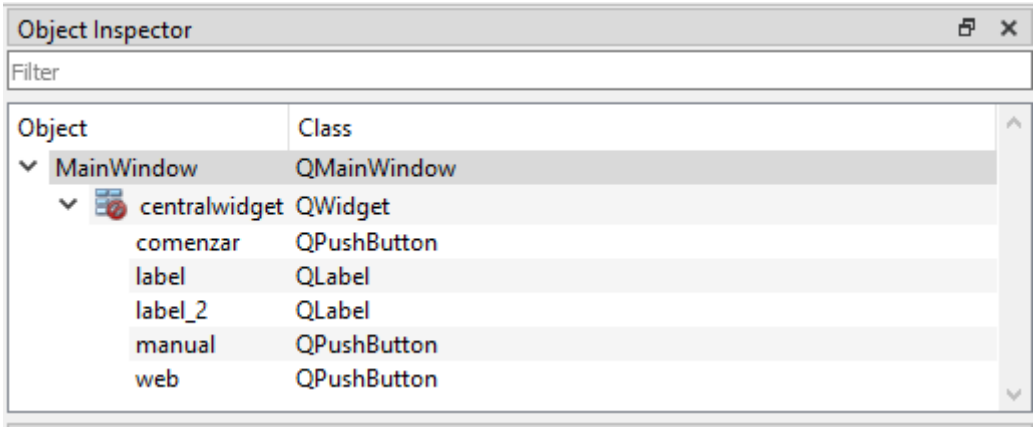

Figura 25: Nombres de objetos de la interfaz

En cuanto a los pasos seguidos para modificar el código fuente, comenzaremos explicando el proceso seguido para unir el botón que nombramos "comenzar" en Qt Designer (un objeto de tipo QPushButton) con la ventana de tareas, es decir, nuestro archivo Principal.py (la segunda ventana de la interfaz).

En primer lugar, cambiamos el nombre de la clase de nuestra interfaz Principal.py para hacerlo más accesible al escribir código. En este caso, cambiaremos el nombre predeterminado de "Ui_Form" por "Tareas"; y debemos hacerlo tanto en la clase que podemos observar al principio del archivo como en la importación final.

```
13      class Tareas(object):
14          def setupUi(self, Form):
15              Form.setObjectName("Form")
16              Form.resize(943, 685)
```

```
101 if __name__ == "__main__":
102     import sys
103     app = QtWidgets.QApplication(sys.argv)
104     Form = QtWidgets.QWidget()
105     ui = Tareas()
106     ui.setupUi(Form)
107     Form.show()
108     sys.exit(app.exec_())
109
```

Figuras 26 y 27: Cambio del nombre de la clase

Seguidamente, volveremos al código fuente de nuestro archivo "Inicio.py" (nuestra primera ventana) e importaremos de archivo "Principal.py" la clase "Tareas". Para ello simplemente escribimos lo siguiente:

```
10  from PyQt5 import QtCore, QtGui, QtWidgets
11  from Principal import Tareas
```

Figura 28: Importación en Python con from

El siguiente paso es crear la función y el atributo para conectar el botón con la siguiente ventana. Para ello escribimos dentro de la función "setupUi" un nuevo atributo y fuera de esta función escribimos una nueva función como podemos ver a continuación:

```
48          self.comenzar.clicked.connect(self.openTareas)
49      def openTareas(self):
50          self.Form = QtWidgets.QWidget()
51          self.ui = Ui_Form()
52          self.ui.setupUi(self.Form)
53          self.Form.show()
```

Figura 29: Nueva función y atributo

Como podemos ver, el nombre de nuestra función es el que nosotros hemos escogido, y en este caso lo llamamos "openTareas".

5.5. Problemas encontrados con el diseño

Como en cualquier ámbito de investigación, nos podemos llegar a encontrar ciertos obstáculos y retos que debemos intentar superar para culminar un proyecto. Teniendo en cuenta que el desarrollo de aplicaciones es un área nueva en la que nos adentramos con este Trabajo de Fin de Máster, los problemas que surgieron fueron variados y tratamos de resolver todos los que pudimos valorando en todo momento que el tiempo que teníamos para hacerlo no era ilimitado.

Hemos recogido algunos ejemplos de los problemas que nos surgieron en torno a la interfaz, aunque la mayoría de los casos aparecieron ya cuando estábamos trabajando con el código. Este fue un problema sustancial porque aunque pareciera que la interfaz estaba acabada, si alguna función entraba en conflicto con partes de la interfaz, debíamos eliminarlas o modificarlas, repitiendo así procesos que ya hemos mencionado y afectando a nuestro tiempo de investigación y escritura de código.

5.5.1. Problemas con las conexiones entre ventanas

En un principio, el principal problema relativo a conexiones que queríamos resolver tenía que ver con el punto que hemos visto anteriormente, las conexiones entre archivos .py. Al principio queríamos hacer que clicando encima de las imágenes pudiéramos redirigir al usuario a otras ventanas, páginas web, etc.

Para hacerlo investigamos varios métodos, pero ninguno se adecuaba a lo que buscábamos exactamente, por lo que buscamos una alternativa diferente con los recursos que teníamos a disposición, en este caso, la biblioteca de PyQt5.

La primera opción que pensamos que podría funcionar fueron los botones. Con un *Push Button* podríamos conseguir dirigir al usuario donde queríamos de manera bastante fácil porque era algo que ya habíamos hecho anteriormente y nos resultó bastante sencillo encontrar información al respecto en internet.

El problema de poner cada conexión ligada a un botón era que cambiaría completamente nuestra idea previa del diseño de la interfaz llenándolo todo de botones y el usuario podría encontrarlo demasiado caótico.

Al final, para resolver este dilema sobre por qué deberíamos cambiar los botones, llegamos a la conclusión de que podríamos poner sobre las imágenes un QCommandLinkButton, que le da un aspecto moderno más acorde con lo que queríamos crear.

Figura 30: Uso de un *Command Link Button*

Aquí podemos observar que en vez de acceder a la tarea de abrir el cliente de mensajería de Outlook con un botón, lo podemos hacer directamente presionando en cualquier lugar comprendido dentro del cuadrado que enmarca la imagen.

En un primer momento, descartamos la opción de utilizar un *Command Link Button* porque de manera predeterminada viene acompañado de una flecha azul que no nos gustaba demasiado, pero al revisarlo como una opción por segunda vez nos dimos cuenta de que podía dimensionarse ese icono, por lo que lo cambiamos a dimensiones a 0x0 y vimos que la flecha desaparecía.

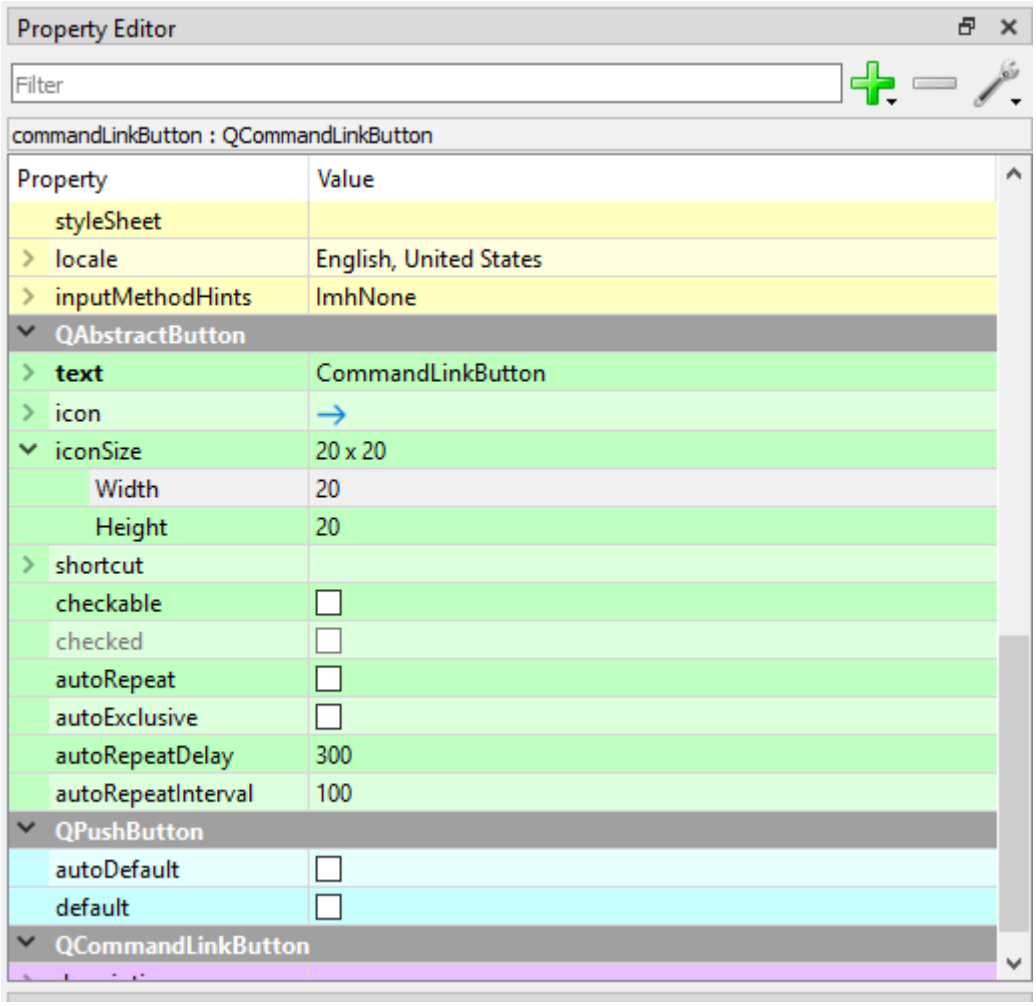

Figura 31: Ajuste de las propiedades de un QCommandLinkButton

Por lo demás, como el código que debíamos escribir era el mismo, no debíamos hacer más investigaciones acerca de cómo conectar las ventanas, así que, aunque cambiamos el diseño, no perdimos mucho más tiempo adicional intentando averiguar cómo debíamos hacer las conexiones.

En este punto todo parecía funcionar correctamente, pero decidimos que para mejorar el viaje entre ventanas crearíamos un botón que conectara la ventana actual con la anterior y que pudiera visualizarse en la parte superior izquierda de las ventanas. Para crearla utilizamos el mismo método de conexión con un *Command Link Button* que utilizamos en el ejemplo anterior.

Al principio parecía funcionar correctamente, pero cuando presionabas accedías y volvías hacia atrás dos veces ocurría un error de importación circular que hacía cerrarse el programa y, por ende, hacía esta función completamente inutilizable si no cumplía su cometido.

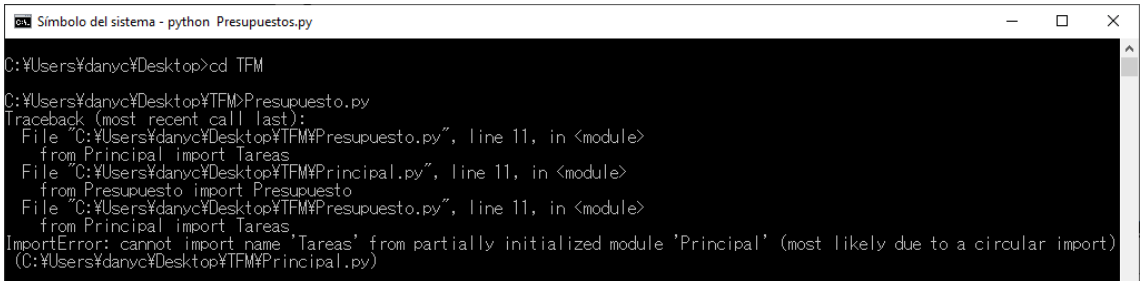

Figura 32: Mensaje de error debido a una importación circular

Con la pequeña aportación de información que nos ofrecía el símbolo del sistema sobre este error, lo único que sabíamos es que era un error relacionado con las importaciones pero no sabíamos por qué sucedía exactamente.

Buscamos información al respecto en internet pero no encontrábamos nada que lo resolviera, por lo que decidimos investigar por nuestra cuenta a partir de prueba y error.

Aunque en un principio pareciera que era poca información, sabíamos que el error se debía a las importaciones que escribimos al principio del documento. No podía deberse a los módulos porque estaban bien escritos y las tareas del programa funcionaban adecuadamente, así que solo podía deberse a las conexiones que hicimos nosotros *a posteriori*.

Hicimos pruebas cambiando y borrando estas importaciones, y con la información que obtuvimos probamos a cambiar el lugar donde escribimos las importaciones del otro archivo .py a la definición que conectaba la flecha con otra ventana.

```
    self.atras.clicked.connect(self.atrasTareas)
    self.calcular.clicked.connect(self.calculadora)

def atrasTareas(self):
    from Principal import Tareas
    self.Form = QtWidgets.QWidget()
    self.ui = Tareas()
    self.ui.setupUi(self.Form)
    self.Form.show()
```
Figura 33: Importación situada en la definición de conexión

De esta manera evitamos que se importe este archivo a todo el documento y conseguimos que se importe solo en esta definición. De esta manera conseguimos que las conexiones de vuelta a la ventana anterior funcionaran correctamente y que no volviera a suceder el error de importación circular entre archivos.

5.5.2. Establecer el logo como icono de la aplicación

Como ya explicamos anteriormente, quisimos darle cierta identidad a nuestro programa para que fuera reconocido, y para ello utilizamos un logo diseñado por nosotros mismos.

El logo aparece en la primera ventana del programa, en la ventana de inicio, pero nos fijamos que aunque apareciera ahí, por coherencia con lo que se hace tradicionalmente con los programas, ese icono debería aparecer también en la parte superior izquierda de la ventana.

Encontramos varias formas de hacerlo pero, ya que estamos utilizando PyQt5, por coherencia buscamos información sobre cómo incluir nuestro icono con el código que ofrece esta librería.

Antes de mostrar las líneas de código que incluimos, cabe destacar que, después de informarnos, observamos que los iconos de las aplicaciones de los programas suelen tener unas dimensiones de 32x32, por lo que abrimos nuestro archivo .psd con Photoshop, accedimos a archivo>exportar>exportar como…, exportamos con unas dimensiones parecidas nuestro logo y lo guardamos con el nombre de dragos-manager.png y dragos-manager.ico, porque en un principio no sabíamos que extensión podríamos necesitar.

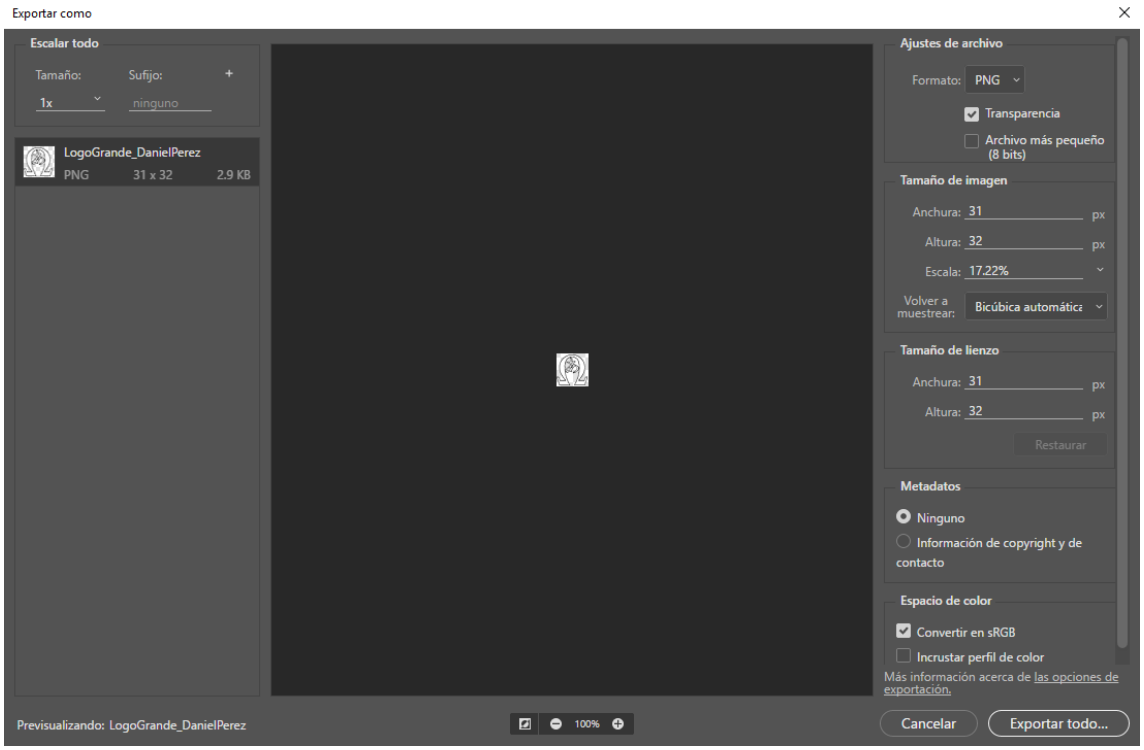

Figura 34: Escalado del logo a dimensiones de icono

Una vez teníamos nuestro icono con las dimensiones que queríamos solo debíamos escribir en nuestro código las siguientes líneas:

```
icon = QtGui.QIcon()
icon.addPixmap(QtGui.QPixmap("icono_dragos.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
Form.setWindowIcon(icon)
```

Figura 35: Establecer una imagen como icono de la aplicación

Después de guardar y volver a ejecutar nuestro archivo .py podemos observar que el icono del programa ahora es nuestro logo:

Figura 36: Icono del programa en la parte superior izquierda

Para que nuestro icono apareciera en el resto de las ventanas, lo único que hicimos fue repetir estas líneas de código en los demás archivos .py a excepción de la ventana de Inicio, que como es una ventana de tipo MainWindow en vez de Widget, PyQt5 tiene una variación propia para el tipo de ventana y tuvimos que alterarlo un poco para que funcionara correctamente.

```
icon = QtGui.QIcon()
icon.addPixmap(QtGui.QPixmap("icono_dragos.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
MainWindow.setWindowIcon(icon)
```

Figura 37: Establecer icono de una MainWindow

Como podemos ver en esta imagen cambiamos el tipo de ventana, es decir, sustituimos Form por MainWindow y funcionaba bien.

5.5.3. Mostrar el título de cada ventana

Al comienzo, cuando creamos nuestras ventanas, aparte de que el icono no aparecía, otro de los factores relativos a la interfaz que podría causar problemas y confundir al usuario era que en el título de la ventana aparecía un nombre predeterminado que no reflejaba la función que desempeñaba el programa en ese punto.

Como es un problema que consideramos bastante importante, tratamos de resolverlo lo antes posible buscando en la documentación de PyQt5 y QtDesigner. Al buscar las posibles soluciones, volvimos a nuestros archivos y vimos que en la definición retranslate.ui que aparecía casi al final del documento aparecía una línea de código con

dos frases entrecomilladas, es decir, eran *strings*, y una de ellas coincidía con el nombre predeterminado de la ventana.

Así pues, lo que hicimos fue cambiar esa *string*, guardamos y comprobamos si surtía efecto y, efectivamente, de esta manera conseguimos cambiar el nombre de la ventana por el que quisimos.

```
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "Dragos Manager"))
```

Figura 38: Cambio del título de la ventana Inicio

5.5.4. Cierre de ventana anterior al producirse una conexión

Un problema estético y práctico relacionado con las conexiones entre interfaces era que al acceder a una nueva ventana pinchando un botón o un *Command Link Button* se abría una ventana nueva pero la anterior permanecía abierta.

No es un problema grave que impida que el programa funcione correctamente o se cierre debido a un error, pero estéticamente no es agradable que haya varias ventanas apiladas y prácticamente no es útil si tenemos en la barra de herramientas un programa con muchas ventanas.

Como solución a este problema, probamos a escribir una nueva línea, MainWindow.hide() y como alternativa MainWindow.close() en las definiciones de conexión.

En principio parecía funcionar normalmente, pero cuando se producía más de una conexión, como nos ocurría en el caso del error de importación circular, ocurría un error y el programa se cerraba.

```
C:¥Users¥danyc¥Desktop¥TFM>Principal.py
Traceback (most recent call last):
  File "C:¥Users¥danyc¥Desktop¥TFM¥Presupuestos.py", line 217, in atrasTareas
    Form.close()
NameError: name 'Form' is not defined
```

Figura 39: Error relacionado con que Form no está definido en el archivo .py

El problema tiene que ver con que ese Form no está definido en los dos archivos que estamos tratando. Esto creemos que podría solucionarse importando el documento donde sí está definido, pero quizás caeríamos de nuevo en el error de la importación circular que hacía que el programa fallara. Para evitar un mal mayor decidimos descartar la idea de cambiar de nuevo las importaciones que por fin habíamos conseguido que funcionara correctamente.

Probamos varios métodos y buscamos más información sobre este problema, pero nada parecía funcionar bien en nuestro caso, por lo que decidimos dejarlo apartado de nuestras prioridades de momento teniendo en cuenta el tiempo que nos quedaba.

Por el momento dejaremos las ventanas sin cerrarse al abrir una nueva, pero lo consideramos una prioridad y será de las primeras cosas que investiguemos y arreglemos en próximas actualizaciones del software.

5.5.5. Cambio de las dimensiones de las ventanas

Uno de los problemas que no hemos podido resolver hasta ahora es el ajuste de los contenidos de una ventana en cuanto a las dimensiones de la ventana. Por su naturaleza estética y no tanto práctica, decidimos dejar la investigación de este punto para próximos parches.

Aunque es cierto que es un problema de características estéticas y no nos hemos querido centrar tanto en ese aspecto como en la utilidad que pueda aportar este programa al usuario, siempre intentamos que fuera visualmente agradable y consideramos que esta característica es imprescindible.

En este punto proponemos dos posibles soluciones que abordaremos en el futuro. Como primera opción podemos estudiar cómo escalar el contenido a las dimensiones de la ventana y, por otro lado, como segunda opción podemos estudiar cómo eliminar el botón de maximizar y minimizar para establecer el tamaño de ventana predeterminado como la única opción.

Nos gustaría poder ofrecer al usuario que tenga la opción de elegir la escala de la ventana, por lo que estudiaremos en primer lugar la primera opción y pasaremos a la segunda si no encontramos una solución viable.

6. Programación con Python 3

La parte de la programación de nuestro programa fue la que más tiempo de investigación nos llevó porque, por un lado, los conocimientos de base que adquirimos durante nuestra formación siendo alumnos de letras puras es escaso y, por otro lado, las ideas sobres las tareas que realizaría nuestro programa iban cambiando conforme íbamos avanzando en el proyecto y eso suponía tener que volver a editar partes que considerábamos terminadas, como el diseño de la interfaz.

Por el momento, dejaremos los problemas que nos surgieron a la hora de programar y las soluciones que aportamos (si las hubiera) para resolverlos para los puntos siguientes a la explicación de las tareas, por lo que comenzaremos a continuación con las explicaciones acerca de la programación.

6.1. Inicio

La programación de la primera ventana que da paso al resto del programa es una ventana de inicio con algunas funcionalidades sencillas. Todas las funciones son de redirección, por lo que llevarlas a cabo no nos llevó mucho tiempo ni de investigación ni de puesta en práctica.

En primer lugar, preparamos las importaciones que se iban a llevar a cabo en el archivo para que funcionaran ciertas definiciones.

```
from PyQt5 import QtCore, QtGui, QtWidgets
import webbrowser
```

Figura 40: Importaciones de la ventana Inicio

Como podemos ver, a parte de las importaciones habituales de PyQt5 que veremos en el resto de los archivos .py, tenemos la importación de webbrowser, que nos ayudará a que se busque una dirección web cuando se ejecute nuestra definición.

Así pues, como en nuestra ventana de inicio pusimos redirecciones a nuestra página web para conocer el estado actual del desarrollo del programa y a un apartado de nuestra página web que contiene el manual del programa.

```
    self.comenzar.clicked.connect(self.openTareas)
    self.web.clicked.connect(self.openPaginaWeb)
    self.manual.clicked.connect(self.openManual)

def openTareas(self):
    from Principal import Tareas
    self.Form = QtWidgets.QWidget()
    self.ui = Tareas()
    self.ui.setupUi(self.Form)
    self.Form.show()
    MainWindow.hide()

def openPaginaWeb(self):
    webbrowser.open("https://tradumatica.net/perezatienza/")

def openManual(self):
    webbrowser.open("https://tradumatica.net/perezatienza/dragos-manager/")
```

Figura 41: Redirección a página web con webbrowser

Como podemos ver, con estas simples líneas de código cuando el usuario clique en los botones de la página web o del manual se le abrirá su explorador de internet predeterminado y aparecerá la dirección escrita.

También se puede observar que hay otro botón que redirige al resto del programa, que es como concebimos desde un comienzo la idea de crear una pantalla de inicio, una ventana que dé la bienvenida al usuario y abra paso al resto del programa.

6.2. Proyectos

Aunque lo tratemos todo en un solo punto, lo cierto es que en la tarea de "Proyectos" tratamos dos utilidades que a nosotros nos parecen bastante interesantes para mejorar la organización de los archivos y la protección de los avances que se han llevado a cabo.

En primer lugar, hablaremos de la tarea de creación de carpetas y subcarpetas, una función que permite tener cada archivo en su lugar; y, en segundo lugar, una tarea para crear copias de seguridad de archivos.

Como ya hemos dicho anteriormente, una de las funciones que queríamos crear desde un principio era un gestor de carpetas y subcarpetas para poder generar con un solo clic las carpetas que podemos encontrar de manera habitual en un proyecto y no tener que hacerlo manualmente.

En muchas ocasiones, si no se lleva una disciplina espartana en cuanto a la organización de los archivos con los que trabajamos, podemos acabar con archivos y copias por todos lados desde el escritorio hasta la carpeta de descargas. En el peor de los casos, con esta desorganización podríamos enviar a un cliente un documento inacabado, que no sea el esperado o que no sea la versión final del archivo y podríamos enfrentarnos

a problemas y consecuencias bastante serias dependiendo del cliente y la importancia del proyecto.

Para evitar la desorganización, por lo general, creamos una serie de carpetas para guardar nuestros archivos y le cambiamos el nombre a cada archivo para saber cuál es cuál más o menos. A lo largo del tiempo y si trabajamos en varios proyectos de ámbitos diferentes a la vez, la creación de carpetas y cambiarlas de nombre se volverá tedioso y nos daremos cuenta de que al poco tiempo el directorio estará lleno de carpetas irreconocibles de nuevo.

La solución a este problema nos pareció obvia: si la organización es esencial pero llevar a cabo esta organización es tedioso, ¿por qué no simplificamos esta tarea de modo que el usuario solo tenga que dar un clic y tener una serie de carpetas creadas?

Una vez planteado lo que queríamos hacer, solo quedaba investigar acerca de cómo hacerlo y ponerse manos a la obra.

Para comenzar, conectamos el archivo Proyectos con los archivos .py de Parametros y Copias:

```
    self.gestor_archivos.clicked.connect(self.openParametros)
    self.copia_seguridad.clicked.connect(self.openCopia)

def openParametros(self):
    self.Form = QtWidgets.QWidget()
    self.ui = parametros(self.ruta)
    self.ui.setupUi(self.Form)
    self.Form.show()

def openCopia(self):
    self.Form = QtWidgets.QWidget()
    self.ui = Copias_de_seguridad(self.ruta)
    self.ui.setupUi(self.Form)
    self.Form.show()
```

Figura 42: Conexión de Proyectos con Parámetros y Copias

Una vez conectados los *scripts* de Python pasamos a explicar la programación de la tarea de creación de carpetas.

Como podemos ver a continuación, para la creación de estas carpetas utilizamos Os una de las bibliotecas que vienen instaladas de serie con Python.

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox
import os
from Copias import Copias_de_seguridad
```

Figura 43: Importaciones de la tarea de creación de carpetas

Como podemos ver, también importamos, QMessageBox, una función de un módulo de PyQt5 que utilizamos para generar un mensaje y que explicaremos más en detalle después de hablar de la programación.

```
def __init__(self):
    self.ruta = os.getcwd()
```

Figura 44: Establecer la ruta en la que se encuentra el programa

El __init__ que vemos en la figura anterior nos sirven para que los otros scripts conectados mediante importación con el archivo Proyectos (Parametros y Copias) puedan heredar la función que indicamos. Con este código estamos estableciendo que ruta es igual a la ruta en la que se encuentra en el momento el programa ejecutado y esto nos servirá a continuación para crear las carpetas en el mismo directorio.

```
def __init__(self, ruta):
    self.ruta=ruta
```

Figura 45: __init__ de Parámetros y Copias

```
    self.crear.clicked.connect(self.creacion_carpetas)
    self.crear.clicked.connect(self.mensaje_creacion)


def creacion_carpetas(self):
    nombre_directorio = str(self.nombre_carpeta.text())

    os.chdir(self.ruta)
    os.mkdir(nombre_directorio)
    os.chdir(nombre_directorio)
    os.mkdir("Original")
    os.mkdir("Meta")
    os.mkdir("Copias de seguridad")
    os.mkdir("Memorias de traducción")
    os.mkdir("Glosarios")

def mensaje_creacion(self):
    msg = QMessageBox()
    msg.setWindowTitle("Creación completada")
    msg.setText("Se han creado las carpetas de tu proyecto.")
    msg.setIcon(QMessageBox.Information)
    ejecucion = msg.exec_()
```

Figura 46: Código de la tarea de creación de carpetas y subcarpetas

En la primera imagen tenemos una definición parecida al primer __init__ pero en vez de escribirlo de nuevo, podemos hacer que herede la misma función de Parametros y Copias en Proyectos.

En la imagen anterior tenemos el código que utilizamos para ejecutar la creación de las carpetas y subcarpetas. Lo primero que observamos son dos líneas de código que

nos ayudan a que cuando pulsemos en el botón crear de nuestra interfaz, se inicien las dos definiciones que escribimos a continuación: creacion_carpetas y mensaje_creacion.

La definición creacion_carpetas incluye en la primera línea un código que indica que lo escrito en el objeto nombre_carpeta (un QLineEdit) se almacene en nombre_directorio.

A partir de este momento seguiremos utilizando la biblioteca Os para crear las carpetas. Os.chdir indica que la ruta en la que se creará la carpeta será en "ruta" (la ruta en la que se encuentra nuestro programa y que obtuvimos gracias a os.getcwd(), como comentamos anteriormente). El os.mkdir siguiente nos ayuda a crear una carpeta con el nombre almacenado en nombre_directorio, es decir, el nombre que el usuario haya indicado en la línea de texto. De nuevo, escribimos un os.chdir indicando la ruta en la carpeta con el nombre que acabamos que crear o, dicho de otro modo, lo que hagamos en estos momentos sucederá dentro de esta nueva carpeta. Para terminar con la creación de las carpetas, indicamos varios os.mkdir para crear tantas carpetas como indicamos y con el nombre que ponemos en forma de *string*.

La definición que vemos justo a continuación de la que utilizamos para crear carpetas se trata de un mensaje informativo que aparecerá cuando creemos las carpetas. Cuando probamos si la definición para crear carpetas funcionaba correctamente nos dimos cuenta de que el usuario no sabía si se habían creado correctamente las carpetas y podría seguir pulsando el botón. Así pues, investigamos acerca de otras funciones de la biblioteca de PyQt5 y vimos que se puede generar un mensaje importando QMessageBox desde los *widgets* de PyQt5 y creando una definición. De esta manera informamos al usuario con un mensaje al que le pusimos el título en la ventana y el cuerpo del mensaje escribiendo las *strings* como argumentos, y también añadimos el .Information para que apareciera un icono informativo. A continuación mostramos una captura del mensaje del programa en funcionamiento:

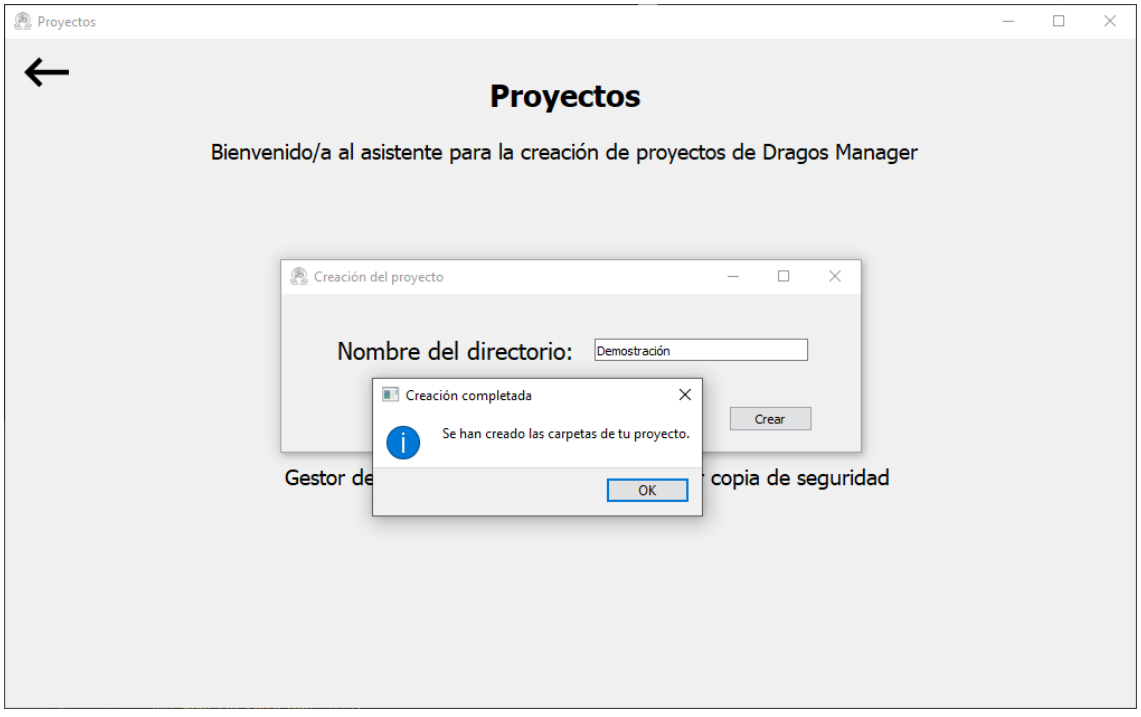

Figura 47: Demostración del funcionamiento de la creación de proyectos

Con esto concluye la explicación del gestor de archivos y pasamos a la explicación de la otra tarea que desarrollamos relacionada con los proyectos.

A parte de la desorganización, y estando también tremendamente relacionada con ella, otro de los grandes problemas de los traductores son las copias de seguridad. Por miedo a que hagamos algún cambio no deseado a un archivo importante o haya algún fallo que haga que nuestro ordenador se apague y que perdamos información a la que hemos dedicado mucho tiempo, creamos copias de seguridad. La mayoría de las veces les cambiamos el nombre a algo así como "Definitivo.txt" y acabamos teniendo tres o cuatro definitivos que no hacen más que confundirnos y generarnos una gran sensación de incertidumbre.

Para evitar el sudor frío que supone abrir y comprobar uno a uno los archivos que podrían ser realmente el archivo final y tener en la mente el pensamiento constante de que quizás tu trabajo se ha perdido, pensamos que podríamos añadir a nuestro programa una función que evitara pasar un mal rato a muchos profesionales, y no solo traductores, porque intuyo que todo el mundo se ha visto envuelto en esta peliaguda situación en alguna ocasión.

Cuando hicimos esta reflexión pensamos que podría ser realmente útil incorporarlo a nuestro programa y así lo seguimos creyendo, pero lo complicado que fue conseguir resultados con nuestros conocimientos iniciales y el contenido de no demasiada calidad publicado en internet, supuso momentos de completa desesperación. Comentaremos los problemas que tuvimos a la hora de desarrollarlo en el siguiente punto, a continuación hablaremos del código que utilizamos.

```
from PyQt5 import QtCore, QtGui, QtWidgets
import os
from datetime import datetime
from PyQt5.QtWidgets import QApplication, QWidget, QInputDialog, QLineEdit, QFileDialog, QMessageBox
from shutil import copyfile
```

Figura 48: Importaciones realizadas en Copias

En la figura anterior podemos comprobar que en el archivo Copias importamos Os, la biblioteca que utilizamos anteriormente para crear carpetas, pero además importamos algunas bibliotecas que no habíamos mencionado anteriormente.

Con el módulo Datetime podremos tener a nuestra disposición la fecha y la hora de nuestro sistema. Esto lo utilizamos para ponerle el nombre a las copias de seguridad, porque pensamos que tener un punto de referencia temporal ayudaría mucho al usuario a saber qué archivo está buscando.

Una de las bibliotecas que no habíamos utilizado anteriormente pero que nos fue de bastante ayuda para hacer las funciones que copian archivos en nuestro código fue Shutil.

```
    self.crear.clicked.connect(self.seleccionar_carpeta)
    self.crear.clicked.connect(self.crear_copias_seguridad)
    self.crear.clicked.connect(self.mensaje_copias)

def seleccionar_carpeta(self):
    self.nombre_directorio = str(self.nombre_carpeta.text())

def crear_copias_seguridad(self):
    ruta_origen=os.path.join(self.ruta, self.nombre_directorio, "Original")
    ruta_salida = os.path.join(self.ruta, self.nombre_directorio, "Copias de seguridad")
    fecha=datetime.now()
    fecha_formateada=fecha.strftime(" %d-%m-%Y %H-%M")
    for archivo in os.listdir(ruta_origen):
        nombre_separado=archivo.split(".")
        extension=nombre_separado[1]
        nombre_no_ext=nombre_separado[0]
        nombre=nombre_no_ext+fecha_formateada+"."+extension
        copyfile(os.path.join(ruta_origen, archivo), os.path.join(ruta_salida, archivo))
        os.rename(os.path.join(ruta_salida, archivo), os.path.join(ruta_salida, nombre))

def mensaje_copias(self):
    msg = QMessageBox()
    msg.setWindowTitle("Copias de seguridad creadas")
    msg.setText("Se han creado las copias de seguridad de los archivos.")
    msg.setIcon(QMessageBox.Information)
    ejecucion = msg.exec_()
```

Figura 49: Código de la tarea de copias de seguridad

Como hemos hecho también en el archivo Proyectos, las primeras líneas de código nos servirán para conectar tres definiciones a un botón, en este caso también lo llamamos "crear".

La primera definición, seleccionar_carpeta, nos sirve para escribir el nombre de la carpeta principal del proyecto que creamos con la función de creación de carpetas y que recibirá los efectos de la próxima definición, crear_copias_seguridad. Para esta ventana utilizamos la misma interfaz de Parámetros, incluyendo el cuadro de texto y el botón.

Una vez le indicamos al programa que la carpeta del proyecto con el nombre especificado es con la que vamos a tratar, se ejecutaría la segunda definición para copiar un archivo en una ruta con un nombre en específico y, aunque parece más sencillo de lo que parece, nos supuso unos cuantos quebraderos de cabeza.

Las dos primeras líneas que vemos en esta definición nos ayudan a establecer la ruta de origen del archivo, que en este caso será la subcarpeta creada automáticamente, "Original", y la ruta de destino del archivo, en la carpeta, "Copias de seguridad". Como podemos comprobar almacenaremos ambos códigos en ruta_origen y ruta_salida respectivamente para tener más claro qué hace cada uno y acortar el código.

A continuación, con la biblioteca Datetime, escribimos datetime.now() para indicar que queremos obtener la fecha y hora actual de nuestro sistema y lo almacenamos en "fecha", como podemos ver.

En la siguiente línea almacenamos en fecha_formateada la fecha con el formato que queremos. Quizás en este punto el lector se pregunte por qué separa un guion cada dato individual de la fecha y la hora y no algo más común como una barra inclinada para la fecha y dos puntos para la hora. El cambio de estos símbolos por guiones se debe a que al ejecutar el programa nos daba un fallo que no podíamos arreglar, pero entraremos en más detalles en el punto siguiente sobre las problemáticas a la hora de desarrollar estas tareas.

A continuación, podemos ver el código que utilizamos para crear las copias de seguridad y para ello nos servimos de un bucle. Como no podía ser de otra forma, los fallos no vienen nunca de uno en uno, y aquí nos surgió otro relativo al nuevo nombre del archivo pero, de nuevo, lo trataremos más adelante. Sin entrar en mucho detalle digamos que las líneas sirven para separar el nombre del archivo e insertar antes de la extensión la fecha y horas actuales.

Por último, con Copyfile y Os.rename conseguimos copiar los archivos contenidos en la carpeta "Original", pegarlos en la carpeta "Copias de seguridad" y renombrar estos nuevos archivos.

También podemos ver que hay una definición para generar un mensaje informativo con un QMessageBox cuando se pulsa el botón crear, pero al ser algo muy parecido a lo que hemos visto en la tarea anterior no entraremos en detalles.

6.3. Problemas de programación de proyectos

En cuanto a los problemas que encontramos mientras programábamos las dos tareas del apartado de proyectos, el gestor de archivos para crear las carpetas no nos dio demasiado problema, sin embargo con la creación de copias de seguridad encontramos un par de problemas al ejecutarlo.

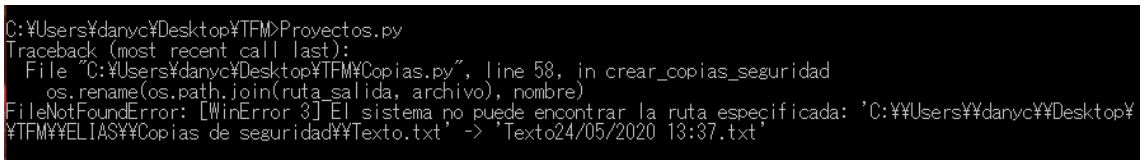

Figura 50: Problema con el nombre del archivo

El primer problema que encontramos nos surgió cuando escribimos las líneas de código para cambiar el nombre del archivo. Para establecer el formato de la fecha y la hora utilizamos en un principio fecha=strftime("%d/%m/%Y %H:%M"). Esto nos generaba un error que no podíamos llegar a comprender, pero después de hacer prueba y error una y otra vez nos fijamos en que Windows no permite poner ciertos símbolos en el nombre de los archivos, entre los que se incluyen los dos puntos y la barra inclinada.

Así pues, para solucionarlo hicimos de tripas corazón y tuvimos que optar por poner un símbolo que sí se pudiera poner en los nombres de los archivos y nos decidimos por el guion, aunque sepamos que la opción óptima sería ponerlos con barras inclinadas y dobles puntos. Además de estos cambios, comprobamos que la fecha se pegaba al nombre original del archivo, por lo que añadimos un espacio en blanco antes del día.

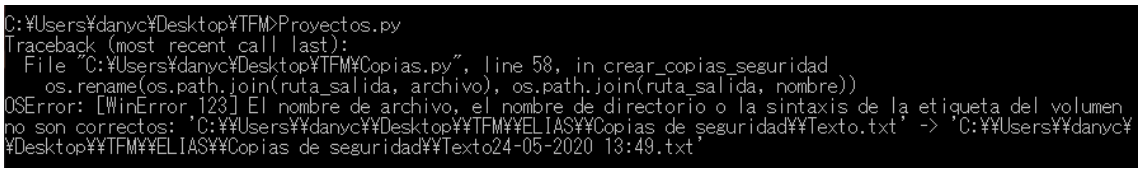

Figura 51: Problema con el nombre y la extensión de los archivos

A continuación, abordamos el segundo problema que tuvimos que hacer frente durante el desarrollo de esta tarea. En un principio cuando se cambiaba el nombre del archivo con Os.rename la fecha se situaba al final de la extensión de los archivos dejando los archivos completamente inservibles, es decir, que si tuviéramos un archivo de Microsoft Word que se llamara "Proyecto", el archivo quedaría como "Proyecto.docxFecha" siendo "Fecha" la fecha actual.

Para corregir este fallo utilizamos la función .split(".") afectando a "archivo", por lo cual le indicamos que queremos que nos separe en dos partes el nombre de los archivos a partir de un punto. Los resultados se generaron en forma de lista que almacenamos en nombre_separado, ocupando así la parte anterior al punto la posición 0, y la extensión la posición 1, que almacenamos a su vez en nombre_no_ext y extensión respectivamente.

El último paso que tuvimos que seguir fue indicar el nombre que queríamos que tuvieran los archivos al final, para ello sumamos nombre_no_ext más fecha_formateada más una *string* con un punto más extensión. Con este último paso obtuvimos el nombre con fecha y extensión correctamente.

6.4. Contacto

Una de las rutinas que prácticamente toda persona del ámbito profesional y, si me apuras, también del no profesional, es revisar el correo cada cierto tiempo para estar en contacto con los demás. En el caso de los profesionales, me atrevería a decir que prácticamente toda la comunicación con los clientes se produce mediante correos electrónicos.

No sería descabellado decir que el lector de este trabajo tiene más de una cuenta de correo y, seguramente, en clientes de correo diferentes entre sí. Como tener que revisar el correo es imprescindible hoy en día, pensamos que podríamos ahorrar algo de tiempo de esta tarea que puede hacerse tediosa por repetirlo constantemente, así que pensamos ofrecer a los profesionales una herramienta que pudiera ofrecer un acceso rápido a los principales clientes de correo.

Llevar a cabo el desarrollo de esta tarea fue bastante sencillo porque apenas son unas líneas de código y además pudimos encontrar bastante documentación. Para esta tarea conectamos cada uno de los botones representados con los logos de los clientes de correo con las direcciones web de estos clientes, por lo que el método utilizado es el mismo que con la ventana de inicio, es decir, importando webbrowser.

```
    self.gmail.clicked.connect(self.openGmail)
    self.outlook.clicked.connect(self.openOutlook)
    self.yahoo.clicked.connect(self.openYahoo)
    #self.otro.clicked.connect(self.openGmail)

    self.atras.clicked.connect(self.atrasTareas)

def atrasTareas(self):
    from Principal import Tareas
    self.Form = QtWidgets.QWidget()
    self.ui = Tareas()
    self.ui.setupUi(self.Form)
    self.Form.show()

def openGmail(self):
    webbrowser.open("https://mail.google.com/mail/u/0/#inbox")

def openOutlook(self):
    webbrowser.open("https://outlook.live.com/mail/0/inbox")

def openYahoo(self):
    webbrowser.open("https://mail.yahoo.com/")
```

Figura 52: Conexión con clientes de correo

Podemos ver que las direcciones que escribimos llevan directamente a las bandejas de entrada porque queríamos que el usuario perdiera el menor tiempo posible para consultar sus correos. Testeamos estas definiciones y, efectivamente, si has entrado en tu cuenta con el navegador anteriormente y tienes tus datos guardados para no tener que escribir tu dirección de correo y contraseña cada vez que entras, te muestra tu bandeja de entrada directamente.

Esta opción es muy útil para equipos propios, es decir, si confías en guardar las contraseñas en ese equipo, entrarás directamente a la bandeja y el proceso será más rápido. Sin embargo, si el equipo en el que estás entrando a tu correo no tiene guardados tus datos porque no confías plenamente en su seguridad, pedirá al usuario que escriba sus credenciales.

Por esta razón, aunque también es útil para equipos en empresas que no permiten guardar sus credenciales porque evitas los pasos de abrir el navegador y escribir el nombre del cliente de correo, para profesionales autónomos que tengan su propio ordenador y guarden sus contraseñas es realmente útil porque, además de saltar los pasos que hemos mencionado en equipos públicos, evitas entrar en el resultado de la búsqueda y escribir las credenciales.

Para comprobar cuánto tiempo se puede ahorrar con esta utilidad de nuestro programa, hicimos una pequeña prueba con cinco sujetos. Les pedimos a cinco personas que desde su escritorio abrieran a una ventana de incógnito y accedieran a la bandeja de entrada de su cliente de correo favorito y los cronometramos. Los resultados que arrojaron estas pruebas fueron los siguientes:

- Sujeto 1: 42”
- Sujeto 2: 38”
- Sujeto 3: 30”
- Sujeto 4: 45”
- Sujeto 5: 37”

Debemos remarcar que en esta prueba todos los usuarios conocían su dirección de correo y contraseña perfectamente, y además utilizan equipos informáticos todos los días. Teniendo en cuenta que no son legos en ofimática y las credenciales las conocían bien y las usaban constantemente, concluimos en que el tiempo podemos ahorrar con nuestra utilidad es de entre 30 y 45 segundos, aunque, por supuesto, puede variar entre usuarios.

Este intervalo de tiempo puede no parecer mucho a primera vista, pero si combinamos el tiempo que hemos perdido haciendo este proceso varias veces todos los días y teniendo en cuenta que en labores de gestión de proyectos se tiene el canal de contacto con el cliente monitorizado constantemente, el tiempo que obtenemos puede ser notable.

6.5. Calendario

En un proyecto debemos tener en cuenta siempre las tres variables para lograr un resultado satisfactorio: coste, tiempo y calidad.

De estas tres variables nosotros consideramos que el tiempo es la que genera más incertidumbre y requiere más planificación porque solo hace falta vivir la vida para saber que el tiempo pasa para todo el mundo incondicionalmente y no se detiene por nadie.

En los proyectos de traducción, como en proyectos de otros oficios, existen plazos de entrega y no tenerlos en cuenta es un grave error.

Como dificultad añadida en cuanto a otros oficios, en el mundo de la traducción no es raro encontrar casos de proyectos en los que el equipo de traducción debe adaptarse al huso horario del cliente y las traducciones son de un día para otro y a contrarreloj, como por ejemplo el caso de traducciones de juegos de móvil que actualizan el juego con cada parche para añadir nuevo contenido.

Es por eso por lo que, entre las opciones que barajamos en un principio sobre las utilidades que podría tener nuestro programa, decidimos incluir un calendario porque medir el tiempo que queda hasta la fecha de finalización de un proyecto es clave para su éxito.

En un principio propusimos algunas herramientas añadidas que podría tener este calendario como un sistema de notificaciones, marcado de fechas y exportación del calendario para su sincronización, pero, como hemos comentado en este último punto, el tiempo nos traicionó.

Teníamos planeado desde un primer momento el tiempo que nos llevaría cada punto de la programación, pero no tuvimos en cuenta los problemas que pudieran surgir y que cada tarea tomaría un tiempo distinto dependiendo de su dificultad. En este caso lo que más afectó a su desarrollo fue que no supimos medir bien la dificultad, y aunque simplemente hicimos el calendario, no hemos descartado en absoluto estas utilidades para su desarrollo en un futuro.

```
self.calendarWidget = QtWidgets.QCalendarWidget(Form)
self.calendarWidget.setGeometry(QtCore.QRect(70, 90, 876, 489))
self.calendarWidget.setObjectName("calendarWidget")
```

Figura 53: Creación de un calendario con PyQt5

Para conseguir mostrar un calendario que se basa en la fecha del equipo, utilizamos un *widget* que viene incluido en PyQt5 y que, como podemos ver en la imagen se puede crear fácilmente con la primera línea de código y establecer sus dimensiones en la segunda línea.

Aunque pueda parecer un calendario bastante simple es completamente funcional, aunque aun así no estamos plenamente satisfechos con esta utilidad del programa, por lo que definitivamente este será uno de los puntos de mejoría que estableceremos para la siguiente actualización del programa.

Aún no sabemos si los recursos de los que disponemos en este momento sean suficientes para llevar a cabo todo lo que propusimos en este punto, pero no daremos nuestras ideas como un caso perdido y las estudiaremos en detalle.

6.6. Cálculo de presupuestos

Una de las situaciones que vimos durante el máster y que se daban cada vez más habitualmente en el sector de la traducción es la aplicación de un descuento al valor total del precio por palabra dependiendo de si una palabra está o no incluida en la memoria de traducción.

Por lo general, es el cliente quien pide estos descuentos por cuestiones de rentabilidad en un proyecto, pero también podría ofrecérselo un traductor autónomo a un cliente para conseguir más encargos, por ejemplo.

Independientemente de quién solicite u ofrezca tarifas con descuentos, es un hecho que se da cada vez más porque cada vez más traductores se están adaptando a los nuevos tiempos y utilizando herramientas TAO que incluyen una función para usar memorias de traducción.

Con las herramientas TAO podemos hacer análisis para obtener el número de palabras totales en un texto, y el porcentaje de coincidencia y exactitud de las palabras con las palabras contenidas en la memoria.

Dependiendo de la herramienta podríamos obtener un resultado diferente en el nombre de las coincidencias con las que numerar las coincidencias, por lo que elegimos los nombres que creíamos que podrían definir bien las coincidencias. Los nombres de las coincidencias que usamos para nuestro programa fueron:

- Repeticiones
- 100%
- 95-99%
- 85-94%
- 75-84%
- 50-74%
- Sin coincidencias

Como podemos ver, creamos una lista descendente teniendo en cuenta el precio de las palabras, es decir, el porcentaje de descuento será menor en las primeras coincidencias e irá incrementando conforme vamos bajando en la lista.

En el caso de que fuera el cliente quien nos pidiera los descuentos, el propio cliente nos enviaría una matriz de descuentos para aplicar un porcentaje de descuento a las palabras, pero si fuera el propio traductor quien ofertara este descuento debería tener bien en cuenta qué descuentos va a aplicar para que su trabajo sea rentable teniendo en cuenta el tiempo que se le dedica a un proyecto de traducción.

Así pues, explicada la necesidad de esta herramienta pasemos a la parte más técnica sobre la programación.

```
self.calcular.clicked.connect(self.calculadora)
```

Figura 54: Conectar botón con la función de calculadora

Con esta línea de código podemos conectar el botón que llamamos en nuestro caso "calcular" con la definición "calculadora" que es la que realiza todas las tareas de la parte de presupuestos. Como podemos ver Python es un lenguaje de alto nivel y su sintaxis es fácil de entender porque se asemeja al lenguaje natural, por lo que cualquiera que lo lea tiene una idea de lo que hace.

```
def calculadora(self):
    precio = float(self.precio_palabra.text())

    palabras_1 = int(self.rep_palabra.text())
    descuento_1 = int(self.rep_porcentaje.text())
    resultado_1 = palabras_1 * (precio - (descuento_1 / 100) * precio)
    texto_resultado_1 = str(resultado_1)
    self.rep_resultado.setText(texto_resultado_1)

    palabras_2 = int(self.cien_palabra.text())
    descuento_2 = int(self.cien_porcentaje.text())
    resultado_2 = palabras_2 * (precio - (descuento_2 / 100) * precio)
    texto_resultado_2 = str(resultado_2)
    self.cien_resultado.setText(texto_resultado_2)

    palabras_3 = int(self.ncinco_palabra.text())
    descuento_3 = int(self.ncinco_porcentaje.text())
    resultado_3 = palabras_3 * (precio - (descuento_3 / 100) * precio)
    texto_resultado_3 = str(resultado_3)
    self.ncinco_resultado.setText(texto_resultado_3)

    palabras_4 = int(self.ocinco_palabra.text())
    descuento_4 = int(self.ocinco_porcentaje.text())
    resultado_4 = palabras_4 * (precio - (descuento_4 / 100) * precio)
    texto_resultado_4 = str(resultado_4)
    self.ocinco_resultado.setText(texto_resultado_4)

    palabras_5 = int(self.scinco_palabra.text())
    descuento_5 = int(self.scinco_porcentaje.text())
    resultado_5 = palabras_5 * (precio - (descuento_5 / 100) * precio)
    texto_resultado_5 = str(resultado_5)
    self.scinco_resultado.setText(texto_resultado_5)

    palabras_6 = int(self.cincuenta_palabra.text())
    descuento_6 = int(self.cincuenta_porcentaje.text())
    resultado_6 = palabras_6 * (precio - (descuento_6 / 100) * precio)
    texto_resultado_6 = str(resultado_6)
    self.cincuenta_resultado.setText(texto_resultado_6)

    palabras_7 = int(self.no_coin_palabra.text())
    descuento_7 = int(self.no_coin_porcentaje.text())
    resultado_7 = palabras_7 * (precio - (descuento_7 / 100) * precio)
    texto_resultado_7 = str(resultado_7)
    self.no_coin_resultado.setText(texto_resultado_7)
```

Figura 55: Definición de calculadora

Aquí podemos ver que “calculadora” es una función definida porque va precedido de “def” y es a lo que hacemos la conexión con el botón que hemos visto anteriormente. Todo lo que vemos indentado debajo de la función es lo que le da su utilidad como programa.

Así pues, lo primero que vemos es el precio, que solo escribimos una vez para cada cálculo posterior. Como podemos ver esta línea de código está modificada por la función float() que hace que el resultado pueda ser tanto números decimales como números enteros.

A partir de este momento, el patrón de las líneas de código se va repitiendo porque su cometido al fin y al cabo es el mismo, pero en diferentes cuadros de texto. Estuvimos pensando algún método para ahorrarnos líneas de código para hacer el código más sencillo pero por falta de tiempo decidimos conservarlo como presentamos porque funciona, aunque no descartamos simplificarlo más adelante.

Como podemos observar la función int() nos dice que el contenido en los cuadros de texto será transformado a números enteros.

Lo más interesante de esta función es cómo he calculado el resultado, que se puede visualizar en las líneas de resultado_X (siendo X cada uno de los números de las líneas). Así pues, el número de palabras en cada coincidencia (palabras_X) se multiplica por el precio menos el descuento entre cien por precio. El resultado de cada línea se almacenará en su resultado_X correspondiente.

Como en nuestro programa pusimos el número del descuento con un símbolo de porcentaje al lado, supusimos que el usuario entendería que el número aparecería como un número porcentual, así que el programa hace internamente el cálculo del número del porcentaje entre cien sin que el usuario lo note en la interfaz.

Por último, para conseguir la suma total de las palabras descontadas, sumamos todos los cuadros de texto de resultado_X y mostramos el resultado en el cuadro de texto que llamamos suma_total.

```
suma_total = resultado_1 + resultado_2 + resultado_3 + resultado_4 + resultado_5 + resultado_6 + resultado_7
resultado_total = str(suma_total)
self.total.setText(resultado_total)
```

Figura 56: Suma total de los resultados

6.7. Problemas de programación de la calculadora de presupuestos

El primer problema lo notamos al comprobar si nuestro programa funcionaba correctamente. En un principio nos percatamos de que los cuadros de texto que el usuario debería rellenar con los números de palabras y porcentajes de descuento estaban en blanco. Esto hacía que si le dábamos a calcular con los cuadros de texto vacíos el programa se cerraba y el símbolo del sistema nos daba un mensaje de error.

Esto se debía a que no se podían hacer las operaciones si no había información en los cuadros de texto. Como no podíamos permitirnos que el programa diera un error si el usuario no rellenaba alguno de los recuadros, decidimos poner un cero escrito por defecto en los cuadros de palabras, porcentaje y precio.

Los espacios en blanco, que eran reconocidos como un valor *string* mientras que le estábamos indicando que los queríamos convertir en *integer*, estaban haciendo que el programa fallara, así que, ofreciendo un cero que sí que es un número, podíamos conseguir que el programa funcionara correctamente.

Al cambiarlo y hacer otra prueba de funcionamiento comprobamos que, efectivamente, lo que ocasionaban los fallos eran los espacios en blanco y ahora funcionaba correctamente.

El segundo problema que encontramos a la hora de programar nuestra calculadora de descuentos se dio a la hora de comprobar el funcionamiento del cuadro de texto del precio por palabra.

Cuando hicimos las primeras pruebas comprobamos que funcionaba correctamente pero nos surgió una hipótesis acerca del comportamiento del usuario.

No es nada extraño de hablar de céntimos en el mundo de la traducción porque tradicionalmente en el mundo de la traducción se ha pagado siempre céntimos por palabra. Si el usuario pusiera los céntimos como un número entero en el cuadro de texto del precio obtendría un resultado fiable, pero tendría que hacer una operación matemática más para obtener el resultado en euros, por lo que seguramente optara por escribir los céntimos como número decimal.

Teniendo eso en cuenta, comprobamos si nuestro programa nos daría un resultado tanto si escribimos los céntimos como número entero, es decir, por ejemplo 7 céntimos; como si escribimos los céntimos como números decimales, como 0,07 euros.

Al hacer la prueba comprobamos que, de nuevo, nos saltaba un error y el programa se cerraba. El símbolo del sistema nos indicaba que se debía a un error relacionado con la función int() en la línea en la que tratábamos el precio.

Nos dimos cuenta de que el error se debía a que si el usuario escribe un número decimal el programa se cerraba porque habíamos indicado que convirtiera lo que hubiera escrito en el cuadro de texto del precio a *integer*.

Para evitar este error cambiamos el int() por float(), con lo que le indicamos al programa que el texto contenido en el cuadro de texto del precio puede ser un número decimal.

Temiendo que el programa no funcionara con números enteros, probamos si funcionaba con números enteros, y como funcionaba perfectamente, pusimos fin a esta tarea para calcular el presupuesto de manera exitosa.

6.8. FTP

Con el paso del tiempo las necesidades del mercado de la traducción van cambiando y los traductores ya no traducen única y exclusivamente archivos de texto plano y, aunque se estén traduciendo textos, los formatos de trabajo son tan variados que hay que buscar nuevos métodos para tratarlos.

Habiendo nuevos formatos alternativos a los archivos de texto plano, las características de los archivos también difieren de estos últimos y, siendo más concretos, el peso de los archivos es un tema fundamental que hay que abordar a la hora del envío en caso del cliente y de su devolución en el caso del traductor.

Si hablamos de archivos de imagen y de audio, el problema no es tan grande (a no ser que la cantidad sea ingente), pero en el caso de vídeos o proyectos con archivos

grandes que tienen que traducirse al completo como los videojuegos, estamos hablando de archivos que en su totalidad llegan a los gigabytes.

Para el envío y recepción de estos archivos de gran tamaño, una de las posibles soluciones es utilizar un servidor FTP, que facilita bastante la transferencia de archivos.

De momento realizamos la conexión con el servicio FTP de Filezilla por ser uno de los más conocidos y utilizados hoy en día, pero más adelante introduciremos compatibilidad con más programas de este tipo.

También existen otros métodos de transferencia de archivos que estudiaremos en otra ocasión, pero creemos que ofrecer al usuario al menos una posibilidad de recibir y enviar archivos ya es útil *per se*.

Así pues, a continuación mostramos al lector el código que utilizamos para conectar un programa de transferencia FTP, en este caso Filezilla, con el botón de nuestro programa.

```
def openFTP(self):
    import os
    os.startfile("filezilla.exe")

    self.FTP.clicked.connect(self.mensaje_Filezilla)
def mensaje_Filezilla(self):
    msg = QMessageBox()
    msg.setWindowTitle("Conexión con FTP")
    msg.setText("Se está abriendo Filezilla, espera unos segundos.")
    msg.setIcon(QMessageBox.Information)
    ejecucion = msg.exec_()
```

Figura 57: Abrir Filezilla con Os

Podemos encontrar esta definición en el mismo archivo Python de la ventana de tareas porque como solo queríamos abrir Filezilla no necesitábamos crear una ventana adicional, algo que cambiará si en un futuro añadimos más opciones.

Como podemos observar, importamos Os y utilizamos os.startfile y en forma de string el nombre del ejecutable de Filezilla y en cuestión de 5-10 segundos el programa se ha iniciado, aunque podría variar dependiendo de la potencia y rendimiento del equipo.

Al testear si funcionaba esta definición, notamos que si el tiempo que pasa desde que el usuario pulsa el botón hasta que se abre el programa es más de lo que se espera el usuario se impacienta y pulsa varias veces el botón, resultando en la apertura de varias ventanas de Filezilla.

Como esto puede resultar bastante molesto, utilizamos una definición del paquete de PyQt5 que habíamos mostrado anteriormente en el punto de la programación relativa a creación de proyectos. En la imagen puede verse que utilizamos un QMessageBox y cambiamos las *strings* del título de la ventana y el mensaje que muestra para darle información al usuario sobre el proceso que se está llevando a cabo.

7. Próximas actualizaciones

A medida que íbamos avanzando en nuestro proyecto, vimos la posibilidad de añadir nuevas funciones que podrían beneficiar en gran medida al usuario pero que por falta de tiempo o recursos no podían llevarse a cabo para el término de este trabajo.

Así pues, como no queremos que este programa al que hemos dedicado tanto tiempo de investigación y creación quede apartado en un rincón olvidado, pretendemos actualizarlo cada cierto tiempo para conseguir un programa cada vez más cercano a la perfección en cuanto a las necesidades del usuario.

Entre las prioridades que consideramos que deberán resolverse para los próximos parches encontramos:

A. En la parte de diseño:

- Conseguir solucionar el error que impide que una ventana se cierre al producirse una conexión.
- Escalado de contenidos en cuanto a las dimensiones de la ventana.

B. En la parte de programación:

- En las funciones de contacto por correo ofrecer escribir el correo en el propio programa como habíamos planeado en un principio.
- Crear un calendario con funciones complejas como un sistema de notificaciones interno al programa, exportación de calendarios, sincronización con otros servicios de calendario, marcado de fechas de entrega, etc.
- Exportación de presupuestos/facturas a archivos de hojas de cálculo.

En cuanto a nuevas funciones que nos gustaría implementar en un futuro podemos contemplar:

- Añadir mejoras a la parte de contabilidad y además de tener una calculadora de descuentos para elaborar presupuestos y facturas como ya tenemos, queremos implementar un libro de cuentas para hacer un seguimiento de costes y beneficios, y una función para realizar de manera sencilla y junto con el libro de cuentas la declaración de la renta con el cálculo de IVA e IRPF.
- Añadir otros métodos de transferencia de archivos aparte de FTP, como con un servidor o en la nube.
- Añadir tareas de análisis de textos, es decir, llevar a cabo un análisis cuantitativo del número de palabras del texto origen y meta o el número de palabras coincidentes en la memoria de traducción.
- Añadir funciones de cambio de formato de textos, archivos multimedia, memorias de traducción y archivos de texto bilingüe.
- Añadir un corrector de ortográfico para comprobar los textos antes de su envío.
- Estudio de la implementación de traducción automática mediante la llamada a algunas API con Postman, además de un editor para la posedición.

8. Conclusiones

Una vez expuestas todas las premisas que pretendíamos abordar en este trabajo, procederemos a terminar con las conclusiones y resultados obtenidos en último término.

Como explicamos al comienzo de este trabajo, los objetivos principales de este trabajo eran dos; por un lado conseguir resultados en la programación de una aplicación para traductores que fuera útil para el usuario y, por otro lado, la obtención de nuevo conocimiento y aprendizaje con este mismo trabajo de programación.

No podemos decir que al comienzo de este proyecto fuéramos completamente legos en cuanto a Python, pero nuestro conocimiento era realmente básico. Como el lector de este trabajo seguramente entienda, la programación no es una habilidad que se adquiera de la noche al día y hemos tenido que investigar y practicar bastante para conseguir obtener resultados.

Aunque pensamos que en estos momentos no podríamos compararnos en absoluto con profesionales del mundo de la programación, creemos que nuestras habilidades en programación en Python y en abstracción del lenguaje han mejorado notablemente en estos últimos meses, y en este aspecto estamos plenamente satisfechos.

En cuanto al programa, se distinguen claramente dos grupos de funciones, las que forman parte del diseño conceptual teórico, y las que se han implementado en la fase relacionada con este trabajo. En este sentido, aunque nos hubiera gustado añadir más funciones para que fuera incluso más útil de lo que ya es para el usuario, creemos que el número de tareas que realiza teniendo en cuenta el tiempo y los recursos que teníamos en un comienzo es bastante satisfactorio.

Aun así, todas las ideas que hemos tenido y que hemos plasmado en este trabajo no serán olvidadas, es más, pretendemos continuar el desarrollo de este programa, no solo por autosatisfacción o para aprender más sobre su lenguaje de programación, sino para crear una aplicación que ofrezca una utilidad real al usuario.

Partiendo desde la experiencia propia del traductor escogimos las tareas que podrían resultar de más ayuda para los trabajadores de este gremio y, si tenemos en cuenta todas las tareas que hemos tratado en este trabajo, como podrían ser las relacionadas con la gestión de proyectos, la comunicación mediante clientes de correo y el presupuesto, creemos que hemos acertado en la importancia y ayuda que podrían ser para los traductores.

La idea con la que concebimos nuestro programa era la creación de un entorno gráfico sencillo de entender en el que se podrían realizar diversas tareas de gestión de proyectos con el objetivo de simplificar las tareas de manera eficiente y ahorrar tiempo.

La interfaz, a la que prestamos bastante atención a la hora de su creación, creemos que es lo suficientemente sencilla e intuitiva para que cualquier usuario pueda entenderlo a simple vista. Esto provoca que cualquier nuevo usuario tenga una ligera idea de lo que hace el programa y no perderá tanto tiempo en la fase de aprendizaje de la herramienta.

En cuanto a las funciones en sí, sabemos que muchos desarrolladores llenan sus programas de ventanas de opciones que no hacen más que alargar los procesos que llevan a cabo, por lo que queríamos que las nuestras fueran lo más sencillas y dinámicas posibles. Poniendo en práctica nuestras funciones pensamos que son lo suficientemente intuitivas y se ejecutan bastante rápido. De esta manera creemos haber conseguido que el tiempo de adaptación a esta herramienta por parte de nuevos usuarios sea reducido y la cantidad de tiempo que se puede ahorrar en la gestión de proyectos es considerable.

Bibliografía

Aggarwal, Nikhil. *__init__ in Python*. Geeks for Geeks. Último acceso: 28 de abril de 2020 en https://www.geeksforgeeks.org/__init__-in-python/

B.Downey, Allen. 2012. *Think Python. How to Think like a Computer Scientist.* O'Reilly: Estados Unidos.

Chazallet, Sébastien. 2015. *Python 3. Los fundamentos del lenguaje.* ENI: Barcelona

Costa Guzmán, Héctor. 2019. *Paquetes*. Hektor Profe. Último acceso: 26 de febrero de 2020 en https://docs.hektorprofe.net/python/modulos-y-paquetes/paquetes/

Covantect. Programación en Python - Nivel básico. Último acceso: 10 de marzo de 2020 en https://entrenamiento-python-basico.readthedocs.io/es/latest/index.html

D. Hunter, John. 2003. *Matplotlib*. Último acceso: 26 de febrero de 2020 en https://matplotlib.org/

Dawson, Michael. 2003. *Python Programming for absolute beginner*. Boston: Premier Press.

Fernández de Sevilla Vellón, María Ángeles. 2016. *Introducción práctica a la programación Python.* Alcalá de Henares. Universidad de Alcalá de Henares

Freepik. *Bussiness Icon Pack.* Flaticon. Último acceso: 10 de marzo de 2020 en https://www.flaticon.es/packs/business-532

Geek University. 2019. *Add Python to the Windows Path.* Último acceso: 26 de febrero de 2020 en https://geek-university.com/python/add-python-to-the-windows-path/

Google. Hardware Icon Pack. Flaticon. Último acceso: 4 de abril de 2020 en https://www.flaticon.es/packs/hardware-3

Khalid, Yasoob. 2014. *ImportError: No module named resource_rc.* Yasoob Khalid's Blog. Último acceso: 17 de abril de 2020 en https://yasoob.me/2014/03/08/importerror-no-module-named-resource_rc/

Martín-Mor, Adrià; Piqué, Ramón; Sáchez-Gijón, Pilar. 2016. *Tradumàtica. Tecnologies de la traducción.* Barcelona. Eumo: Universidad Autónoma de Barcelona.

Matheus Simpao, Carl. 2020. *10 Best Python IDE & Code Editors*. Último acceso: 1 de marzo de 2020 en https://hackr.io/blog/best-python-ide

McKinney, Wes. 2002. *Pandas*. Último acceso: 26 de febrero de 2020 en https://pandas.pydata.org/

MrNtou. 2018. *Python: How to add an image in pyqt designer and convert to py.* Youtube. Último acceso 20 de marzo de 2020 en https://www.youtube.com/watch?v=TAJwrIO_ctI

Oliphant, Travis. 1995. *Numpy*. Último acceso: 26 de febrero de 2020 en https://numpy.org/

Pixel Buddha. Logotypes Icon Pack. Flaticon. Último acceso: 20 de marzo de 2020 en https://www.flaticon.es/packs/logotypes

Pixel Perfect. *Logo Icon Pack.* Flaticon. Último acceso: 20 de marzo de 2020 en https://www.flaticon.es/packs/logo-502

Programiz. *Python Directory and Files Management.* Último acceso: 16 de mayo de 2020 en https://www.programiz.com/python-programming/directory

Programiz. *Python strftime().* Último acceso: 20 de mayo de 2020 en https://www.programiz.com/python-programming/datetime/strftime

Pycharm. 2019. *Funcionalidades de Pycharm.* Último acceso: 1 de marzo de 2020 en https://www.jetbrains.com/es-es/pycharm/features/

PyPi. 2020. *Pip 20.0.2*. Último acceso: 04 de marzo de 2020 en https://pypi.org/project/pip/

Python. 2020. *Os - Miscellaneous operating system interfaces.* Último acceso: 14 de mayo de 2020 en https://docs.python.org/3.7/library/os.html

Python. 2020. *Python 3.8.2*. Último acceso: 26 de febrero de 2020 en https://www.python.org/downloads/release/python-382/

Python. *Webbrowser — Convenient Web-browser controller*. Último acceso: 29 de marzo de 2020 en https://docs.python.org/3.8/library/webbrowser.html

Qt. 2020. *QMessageBox Class*. Último acceso: 5 de mayo de 2020 en https://doc.qt.io/qt-5/qmessagebox.html

Qt. 2020. *Qt Designer Manual*. Último acceso: 05 de marzo de 2020 en https://doc.qt.io/qt-5/qtdesigner-manual.html

Qt. 2020. *Qt for Python Documentation*. Último acceso: 5 de marzo de 2020 en https://doc.qt.io/qtforpython/contents.htmlhttp://acodigo.blogspot.com/2016/10/pyqt-5-estilos-qss.html

Ramalho, Luciano. 2015. *Fluent Python*. O'Reilly: Estados Unidos.

Smashicons. Essential Set Icon Pack. Flaticon. Último acceso: 20 de marzo de 2020 en https://www.flaticon.es/packs/essential-set-2

Stechies. 2019. *Valueerror: Invalid literal for int() with base 10.* Último acceso: 5 de mayo de 2020 en https://www.stechies.com/valueerror-invalid-literal-int-base-10/

W3schools. *Python String split() Method*. Último acceso: 29 de mayo de 2020 en https://www.w3schools.com/python/ref_string_split.asp

Wikipedia. *Comparison of integrated development environments. Python.* Último acceso: 27 de febrero de 2020 en https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Python